Finite Difference Techniques for Body of Revolution Radar Cross Section

by

Joe Pacheco, Jr.

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Bachelor of Science in Electrical Science and Engineering

and

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2000
(June 2000)

Author .....
Department of Electrical Engineering and Computer Science
May 22, 2000

Certified by .....
Dr. Robert G. Atkins
MIT Lincoln Laboratory
Thesis Supervisor

Certified by .....
Professor Jin Au Kong
Electrical Engineering
Thesis Supervisor

Certified by .....
Dr. Y. E. Yang
Research Laboratory of Electronics
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Finite Difference Techniques for Body of Revolution Radar Cross Section

by

Joe Pacheco, Jr.

Submitted to the Department of Electrical Engineering and Computer Science
on May 22, 2000, in partial fulfillment of the
requirements for the degrees of
Bachelor of Science in Electrical Science and Engineering
and
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Although a number of radar cross section prediction techniques have been developed which exploit body of revolution symmetry, the use of finite-difference techniques with these geometries has not been throughly explored. This thesis investigates several finite-difference approaches which vary both in the approximations they introduce as well as the computational resources they require. These techniques include body of revolution finite-difference time-domain methods with both staircase and conformal grids, a hybrid FD-TD/geometrical optics method, and a body of revolution parabolic wave equation method. In addition, the use of the monostatic-bistatic equivalence principle is explored in approximating monostatic RCS at multiple angles from a single FD-TD simulation. Both canonical and more realistic BOR targets are modeled. The results from these techniques are compared, with each other and with method of moment predictions, physical theory of diffraction predictions, and analytic results. From these comparisons the tradeoffs possible between accuracy and computation with this collection of finite-difference tools is determined.

Thesis Supervisor: Dr. Robert G. Atkins
Title: MIT Lincoln Laboratory

Thesis Supervisor: Professor Jin Au Kong
Title: Electrical Engineering

Thesis Supervisor: Dr. Y. E. Yang
Title: Research Laboratory of Electronics

Acknowledgments

This thesis represents a major milestone in the course of my studies at M.I.T. over the past four years. It has been a challenging and rewarding experience that could not have been possible without the assistance and support of many people. To them, I am very grateful and would like to acknowledge their invaluable contributions.

First and foremost I would like to acknowledge my VI-A supervisor at Lincoln Laboratory, Dr. Robert Atkins, who first introduced me to the world of research and electromagnetics in the summer of 1997. I am grateful for the patience and dedication he has shown in his role as my advisor. He always made every effort to take the time to fully answer all my questions and encourage my work. I am also very appreciative of the extra efforts that he made to come to the lab or campus late at night and on weekends to help me improve my talks. It has been a pleasure working with him, and I look forward to working with him in the future as I continue my education at M.I.T.

There are many other people at Lincoln Laboratory whom I would like to thank for helping me on a variety of projects. I'd like to thank Barbara Mozzicato, Janet Quinn, Kristin Rocheleau, Heather Schiffer, and Irene Stapleford for making many of the charts I used as various talks as well as some of the figures in this thesis. I would also like to acknowledge Nathan Lundblad and Andrea Carleton for helping with various tasks along the way. I am also thankful to have had such good officemates and co-workers as a VI-A student in Group 49/106 at Lincoln Laboratory.

I am also grateful to Professor Jin Au Kong for serving as my thesis advisor and teacher of electromagnetics. He truly has made my study of electromagnetics an enjoyable experience, and has inspired me to strive harder.

I'd also like to thank the students and scientists in the CETA research group on campus for their helpful discussions and feedback during group meetings. In particular, I would like to acknowledge Dr. Eric Yang and Dr. Fernando Teixeira for their valuable technical expertise and advise. I look forward to working with the CETA research group on campus as I begin the next stage of my graduate studies.

In addition to those who helped with my academic endeavors, I would like to thank all my friends for making my M.I.T. experience more enjoyable. To all my friends from Conner 5, Burton 4, and G Entry, thanks for all the fun nights out, weekend trips, and sports we played.

And to all my friends from TAMS, thanks for keeping in touch and supporting me every way, especially to Danny Lee, who was in many ways like a brother to me.

Finally, I'd like to thank my family for all their love, support, and encouragement. I wouldn't be where I am today without them. To my brother Daniel, thanks for being such a good brother and friend. And to my parents, thank you for always being there and supporting me in countless ways. You've helped make me the person I am today, and for this I am eternally grateful.

To My Family

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

1.1.1 Definition of RCS

Radar technology continues to find widespread use in the long range, all weather detection of airborne, space-borne, or land moving targets. Important in the analysis of such detection systems is a knowledge of the electromagnetic characteristics of the target to be detected. Of the power incident on the target, some will be absorbed as heat, and the remainder is scattered. The direction and magnitude of the scattered fields is described by the target's radar cross section, or RCS, defined as that area intercepting that amount of power, which, when scattered isotropically, produces an echo at the radar equal to that from the target [31]. Mathematically the RCS, σ, is defined as,

$$\sigma(\phi,\theta) = \lim_{R\to\infty} 4\pi R^2 \frac{|E_s(R,\phi,\theta)|^2}{|E_i(R,\phi,\theta)|^2} \tag{1.1}$$

where E_s is the electric scattered by a target illuminated by an incident electric field, E_i. Monostatic RCS is the radar cross section of a target when the receiver and source are in the same location. Bistatic RCS is the radar cross section of a target when the receiver and source are located at two different points.

Several factors influence a target's RCS, including the target's size and shape, the frequency of the pulse, the incident and receiver polarizations, and the orientation of the target with respect to the incident field. Radar cross section may be determined by either direct mea-

surement or by theoretical prediction. Direct measurement, however, is often expensive and requires specially constructed radar measurement facilities. In contrast, prediction of a target's RCS involves modeling the electromagnetic fields scattered by the target using analytical or numerical techniques. Prediction avoids the need for costly measurements, and allows estimation of target cross sections even in cases where the actual target is unavailable. Because of the importance of target signatures in radar system analysis, a variety of approaches to RCS prediction have been developed in the past.

1.1.2 RCS Prediction Methods

The methods used to model targets vary widely depending on the electrical size of the target. The Rayleigh or low frequency scattering regime corresponds to situations in which the object is much smaller than one wavelength. In these cases, the incident field changes very slowly compared to the time required for propagation across the target, and the problem can be treated by electrostatic methods. For larger objects, in the resonant scattering regime where the target is approximately one wavelength in size, the dynamic nature of the fields can no longer be neglected. For a small number of these geometries, such as a sphere or a cylinder, Maxwell's equations can be solved analytically to obtain an exact series solution for the scattered fields. For other geometries, however, Maxwell's equations must be solved numerically, using techniques such as the Method of Moments [16, 17, 18] or the Finite Difference-Time Domain [49, 50, 58] approach.

The Method of Moments (MoM) approach determines the scattered fields by solving Maxwell's equations in integral form. An integral equation in the unknown surface currents is formulated, and the currents are represented by a weighted series of basis functions. The integral equation is then tested with another series of testing functions to produce a matrix equation which may be solved for the unknown basis function weights. Because this method solves Maxwell's equations numerically, it gives a solution which is exact within the limits of the geometry modeling accuracy. However, because the currents must be sampled at a spacing of one fifth of a wavelength or less, the resulting matrix equation quickly becomes intractable for all but electrically small objects. In addition, the integral equation is generally formulated in the frequency domain, requiring repeated application of the process if the RCS is desired over a band of frequencies.

The latter shortfall is overcome by the Finite-Difference Time-Domain (FD-TD) method, which solves the differential form of Maxwell's equations in the time domain. The equations are

discretized in time and space, and the resulting difference equations stepped forward in time. With appropriate excitation, the RCS over a band of frequencies can be calculated from a single simulation. However, unlike MoM, only one aspect angle is obtained from each simulation, and calculation of RCS over a range of angles again requires multiple simulations. Also, the spatial grid required in FD-TD is again small, restricting the approach to electrically small objects.

Prediction of RCS for larger objects requires some approximation of Maxwell's equations. One approach which employs such an approximation, but still solves numerically for the fields is the Parabolic Wave Equation (PWE) technique [33, 38, 59]. This approach has found extensive past use in the modeling of propagation, but has more recently been explored as a RCS prediction technique. The PWE uses finite-difference techniques to solve a modified Helmholtz wave equation in which an explicit spatial phase dependence has been assumed. The advantage of this approach is that less memory is required to model an electrically large target since the fields only need to be stored at one range step in contrast to MoM and FD-TD methods, which require that all the fields in the computational domain be stored.

An alternate approach to approximating the scattered fields is embodied in the high frequency RCS prediction techniques, such as geometrical optics (GO), physical optics (PO), the geometrical theory of diffraction (GTD), and the physical theory of diffraction (PTD). Inherent in all of these approaches is the assumption that the wavelength is small compared to the size or curvature of the target. Geometrical optics [23] models electromagnetic scattering as optical ray reflection by the target, and ray tracing techniques are used to determine the specular points where reflection is in the direction of the receiver. RCS is calculated by determining the change in power density in the reflected ray, arising from the spreading caused by the surface curvature of the reflection point. One of the main disadvantages in GO theory is that flat objects with infinite curvature cause caustics yielding unbounded results. In addition, since only specular reflections are assumed, GO does not account for diffraction, surface waves, or traveling waves.

Physical optics [23] overcomes the first problem of calculating scattering from flat surfaces, where GO would predict either no return, or an infinite return. The method applies a tangent plane approximation to calculate the induced currents at each point of the target surface. The surface currents on the target are then integrated to produce the scattered fields. As with GO, PO does not account for diffraction, surface waves, or traveling waves.

The geometrical theory of diffraction and physical theory of diffraction extend the validity of the geometrical optics and physical optics methods by including more scattering phenomena.

GTD, developed by Keller [29, 30] uses a variety of canonical problems to predict scattering due to diffraction from wedges, straight edges, and corners. The incorporation of fields due to surface and traveling waves is also possible. By predicting scattering due to diffraction, the RCS of targets with edges or corners can more accurately be determined without significantly increasing the amount of computation. The limitation of this approach, however, is that GTD solutions are only available for canonical geometries, and more complex geometries must be constructed from these canonical components. The physical theory of diffraction, developed by Ufimtsev [57], is very similar to Keller's GTD. Unlike the GTD, however, which calculates diffracted fields directly, the PTD uses the solution to the canonical wedge problem to find the induced non-uniform edge currents due to diffraction only. This current is then placed in the PO model and integrated as before. Consequently, PTD allows direct treatment of more arbitrary geometries. By modeling additional electromagnetic phenomena, both GTD and PTD can significantly improve the performance of GO and PO.

Despite the assortment of RCS modeling tools described above, there remain cases where high-frequency techniques fail to achieve the desired accuracy, yet numerical techniques are impractical. For example, consider an electrically large structure with a smaller structure attached to it. Clearly, numerical techniques are impractical due to the overall target's size; moreover, high-frequency techniques cannot accurately model the small attached scatterer. A hybrid concept, initially proposed by Thiele and colleagues [13, 19, 48, 55, 56] provides one method for analyzing geometries of this type. In his solution, the method of moments was used to model the small attached scatterer, while GTD was used to model the large structure. Still, even without a small attached structure, many electrically large targets can not be accurately modeled using high frequency techniques. One possible solution is to look for special geometries, such as a body of revolution, to simplify the numerical techniques. Both the MoM and FD-TD techniques have been specialized to model bodies of revolution (BOR) allowing for the rigorous modeling of large bodies of revolution.

In this thesis, both the hybrid and special geometry approaches are explored in the context of body of revolution (BOR) geometries. In particular, finite-difference approaches involving the FD-TD and PWE techniques are used to model electromagnetic scattering from BOR targets. A hybrid approach involving geometrical optics and a 2D FD-TD method is used to model large targets with small features. In addition, the BOR FD-TD algorithm and BOR PWE method are used to model scattering from large bodies of revolution. In order to provide a sufficient

understanding of the two primary methods to be used, background information on the FD-TD method and the PWE approach will be given in the following section. In section 1.3, work that has been done in the past on the body of revolution RCS problem will be discussed to motivate the need for more research. Finally, section 1.4 will discuss in detail the work done in this thesis.

1.2 Background

Since it's introduction in 1966 by K.S. Yee [58], the finite-difference time-domain method has been applied to a large number of electromagnetic problems [49, 50]. The method has gained recent popularity due to the availability of computers with faster processing speeds and larger memory capacities. The FD-TD method works by directly solving the time-dependent form of Maxwell's curl equations by discretizing in both time and space. Electric and magnetic fields are placed at interleaving spatial locations and are solved for in a leap-frog manner allowing the solution to be obtained by a marching in time approach.

One of the main advantages of the FD-TD method is that since the time-dependent form of Maxwell's equations are used, only one simulation is required to determine scattering at multiple frequencies, whereas frequency domain formulations require separate runs for each frequency of interest. Simultaneous analysis of multiple frequencies is accomplished with the FD-TD method by using a multi-frequency incident excitation, such as a Gaussian pulse. Frequency components are then extracted by a Fourier transform of the time domain fields.

One disadvantage to the FD-TD approach is that geometries of interest are defined in open regions where the spatial domain of the computed fields is unbounded in one or more coordinate directions. In order for the problem to be well-posed, the spatial domain must be truncated. One possible solution is to assume that the fields outside the region of interest, or computational domain, are zero. However, this approach, in effect, models the edges of the domain as perfect conductors that reflect all incident waves. Provided the domain is large enough, an arbitrary geometry can be modeled inside the domain for durations where the wave does not reach the boundary. In practice, this forces the computational domain to be very large to ensure no reflections, leading to long computation times and large memory requirements.

To reduce the size of the domain, an absorbing boundary condition, or ABC, can be used instead. An absorbing boundary condition attempts to reduce the number of reflections at the

edge of the computational domain, simulating the propagation of the electromagnetic fields out into free space beyond the computational domain. One ABC, popular until recently is the second order boundary condition formulated by Engquist and Majda [14]. This approach works well for waves that are normal or nearly normal incident upon the edges of the computational domain. Larger reflections occur for waves that are near grazing angles.

An alternate ABC introduced by Berenger [3] in 1994 , the perfectly matched layer, or PML, is able to absorb waves incident at a broader range of angles with little or no reflection. This approach is based upon a splitting of the electric and magnetic field components in the absorbing boundary region with possibility of assigning loss to the individual split field components. Field components incident upon a PML region are split into a component that is traveling normal to the absorbing medium, and a component that is traveling tangential to the absorbing medium. The normal component is attenuated as it travels through the absorbing medium while the tangential component is allowed to propagate normally. The tangential component will eventually be attenuated by additional PML regions. The net effect is to create a nonphysical medium that has a wave impedance independent of the angle of incidence and the frequency of the incoming scattered waves. Reflections at the interface between the PML region and the free space region are prevented by matching the wave impedance of the PML region to that of the free space region. Berenger originally introduced the PML for a two dimensional rectangular coordinate system, but it has since then been extended to more complex domains [24, 27, 44].

An alternate PML formulation approach based on a coordinate stretching viewpoint was later proposed by Chew *et al.* [8, 9]. Their approach involves the development of a modified set of Maxwell's equations via a complex coordinate transform. The additional degrees-of-freedom introduced by the complex coordinate stretching allow for the specification of a lossy material layer such that the interface between free space regions and PML regions is reflectionless for all frequencies, polarizations, and angles of incidence. Under the coordinate stretching transformation, Maxwell's equations inside the PML can be written in the same form as the original Maxwell's equation, but on a complex spatial domain. While the original PML formulated by Berenger applies only to rectangular coordinate systems, the generalized PML formulation can be applied to other coordinate systems to provide PML's on these systems. For example, in [54], PML formulations using complex stretching variables for a cylindrical coordinate system and a spherical coordinate system are developed.

A second drawback to the FD-TD approach is the difficulty of modeling surfaces which do

not lie along grid lines. One simple approach to the problem is to force the object to align with the grid lines creating a "staircase" approximation of the object. Accuracy of the results depends on the size of grid cells used, with higher accuracy possible with smaller cell sizes, at the expense of an increase in the the number of unknowns. More accuracy, however, may be obtained without increasing the computational overhead, by using a conformal gridding FD-TD approach [25, 26]. The conformal gridding FD-TD algorithm works by deforming the normally square or cubic cells along the boundary of the object being modeled. Special contour integrals are evaluated to determine alternate finite-difference equations valid for the new deformed cells.

While the FD-TD method provides a robust and rigorous method for predicting RCS, it is computationally intractable for electrically large objects. One alternative which has recently been proposed is the parabolic wave equation (PWE) approach. In the past, the PWE method was primarily used to study the propagation of electromagnetic waves in the troposphere [2, 11, 33, 46]. However, recently the PWE method has also been used in the prediction of scattering by acoustical and electromagnetic waves. Levy studied the prediction of acoustical scattering from soft and rigid cylinders in 2D, and soft and rigid spheres in 3D [38] using scalar wave equations. The methods developed there have been used to model objects ranging in size from a few wavelengths to hundreds of wavelengths. In addition studies involving the prediction of the radar cross section of arbitrary 2D and 3D targets has been carried out using vector parabolic wave equation techniques [4, 5, 36, 37, 59].

The parabolic wave equation method works by introducing an explicit spatial phase dependence for the scattered field. The time-harmonic Helmholtz equation is then rewritten in terms of the new field representation. The resulting exact equation that must be solved involves a pseudo-differential operator. The accuracy of the solution depends on the approximation used to represent this operator. A low order approximation, such as a two term Taylor expansion, of this operator yields a narrow-angle PWE method whose solutions are valid for angles of propagation less than $15^\circ - 20^\circ$. High order approximations, such as the split-step Fourier and Padé methods, result in a wide-angle PWE method whose solutions are valid for angles of propagation up to 90° [5]. Using either approximation, difference equations that relate two adjacent fields are formulated. Applying an initial condition, the full solution may be obtained by using a memory efficient marching in space approach, allowing for the possibility of modeling electrically large targets.

As with the FD-TD method, the PWE method also models objects that are in open regions

with a spatial domain unbounded in one or more directions. Hence, the computational domain must be truncated by an absorbing boundary condition. One possible ABC used by Levy [38] is Berenger's perfectly matched layer discussed above. A second approach, more suited to the PWE is the non-local boundary condition (NLBC). The method works by expressing the fields outside the computational domain in terms of the fields on the boundary. The NLBC has been developed for both narrow-angle and wide-angle 2D PWE approaches [35]. Since the formulation of the NLBC is exact, the upper and lower boundary of the computational domain can be placed arbitrarily close to the scatterer greatly reducing the computational overhead.

Generally, the FD-TD and PWE methods are used to solve for the near-fields. In order to calculate the radar cross section, which requires knowledge of the far-fields, a near-to-far field transformation is necessary. This can be accomplished by computing the scattered fields over a surface that completely encloses the object. Using Huygens' principle, the far-fields can then be calculated, and from them the RCS determined.

1.3 Past Work

Since the numerical approaches discussed above are computationally intensive, it is advantageous to look for special geometries whose features can be exploited, such as a body of revolution. A body of revolution is a three dimensional object that exhibits axial symmetry, which can be formed by rotating a two dimensional curve about one axis. Examples of bodies of revolution include cylinders, spheres, and cone shaped objects.

There have been several approaches in the past to model the RCS and scattering patterns of bodies of revolution. One popular approach to solving the problem is the method of moments. The specific case for scattering from bodies of revolution was treated by Andreasen [1] and Harrington [18] where the axial symmetry of the object was exploited by decomposing the electric and magnetic fields into Fourier modes. Since the modes are orthogonal, the problem decouples from one large three dimensional problem into a sequence of smaller two dimensional problems, one for each Fourier mode. This is advantageous both in terms of memory usage and speed since it is faster to invert several small matrices, than one large matrix. As noted previously, however, one of the drawbacks of the MoM method is that the technique must be repeated multiple times for each frequency of interest.

In contrast, the FD-TD method allows for wide-band analysis and has been used extensively

in the prediction of radar cross section [51, 52, 53]. As with the MoM method, a BOR FD-TD algorithm can be developed that takes advantage of the object's axial symmetry, resulting in a more efficient FD-TD algorithm. Such an algorithm was developed by Merewether and Fisher for use with electromagnetic pulse applications [41]. The BOR FD-TD algorithm was later used by Britt [6] to calculate the radar cross section for bodies of revolution. Britt studied the monostatic and bistatic cross sections of a sphere, a cylinder, a cone, and a cone-sphere by using staircase models of the targets. Each of the targets analyzed were assumed to be perfect electric conductors. He verified the BOR FD-TD method by comparing the scattering cross section of the sphere with the exact analytic results.

Since the introduction of the BOR FD-TD method, it has been used for a wide variety of applications [7, 10, 43, 47]. Of particular importance was the work done by Saewert and Jurgens in the development of a conformal BOR FD-TD code capable of modeling longitudinal and transverse wake fields and impedances of particle accelerator beam line structures. The code they developed used a perfectly matched layer absorbing boundary condition that was previously developed by Jurgens [24]. Adapting the BOR FD-TD to use a conformal gridding scheme and the introduction of the PML ABC for the BOR FD-TD method were both innovations that increased the usefulness of the method. However, no work has been done to study the use of the conformal BOR FD-TD method for the prediction of radar cross section.

As discussed in section 1.1, high-frequency techniques have been used in conjunction with MoM techniques to predict radar cross section. In particular, Medgyesi-Mitschang uses the BOR MoM formulation in combination with the physical optics technique to analyze conducting bodies of revolution [40]. In this technique, the object is subdivided into smooth convex and irregular surfaces. The currents induced along the smooth surfaces are obtained using physical optics methods, while the currents induced on irregular surfaces, such as discontinuities caused by protrusions or concavities, are modeled using a MoM expansion for the surface currents. However, there has been little work done to combine high-frequency techniques with the FD-TD method for the analysis of conducting bodies of revolution.

1.4 Thesis Work

The purpose of this thesis is to further explore finite-difference approaches for calculating the RCS of targets involving body of revolution geometries. There are three main parts to the

research: calculating the RCS of bodies of revolution using the BOR FD-TD method, calculating the RCS of BORs using a hybrid 2D FD-TD/GO method, and developing a BOR PWE algorithm for BOR RCS prediction. In each of the methods explored, results are compared to BOR MoM results, PTD results, and analytic results where possible.

The first part of this work involved implementing the BOR FD-TD algorithm in a computer code capable of analyzing perfectly electric conducting bodies of revolution. Both staircase and conformal gridding techniques are used to model the target of interest. The code is capable of automatic mesh generation for BOR structures with piecewise linear cross sections for both the staircase and conformal gridding methods. The results obtained from the BOR FD-TD code are verified by comparing them to analytic solutions, and BOR MoM results. In addition, the results from the staircase and conformal models are compared to determine the improvement from the conformal gridding approach. Next, the monostatic-bistatic equivalence principle [28] is used to generate monostatic data from BOR FD-TD calculated bistatic data. This approach is used as an attempt to overcome the fact the FD-TD method requires separate runs to produce monostatic data for multiple aspect angles. The fidelity of the results produced is compared to the exact solutions obtained from the repeated use of the BOR FD-TD method at each aspect angle of interest. In addition, the results are also compared to PTD results to determine what accuracy advantage the monostatic-bistatic equivalence approach provides over high-frequency approaches.

The second part of this work includes the development of a method that combines the FD-TD method with a geometrical optics high-frequency approach. A two-dimensional FD-TD method is used to calculate the field propagation along the two dimensional surface described by a cross section of the body of revolution. Using the field values predicted by the FD-TD method, and assuming a large radius for the BOR object, a geometrical optics type approach is used to calculate the resulting scattered fields where the contribution is assumed to arise from the stationary phase point in the plane of incidence. Since the FD-TD method is an exact numerical technique, propagation along the body of revolution will be modeled rigorously. Results from this approach are compared to the exact BOR FD-TD results to determine the loss of fidelity in accuracy and the limits of the hybrid approach. In addition, comparisons are made with the physical theory of diffraction to determine if the hybrid technique is more accurate than existing methods based purely on high-frequency approximations.

In the third part of this work, a body of revolution parabolic wave equation method is

developed as an alternate solution for cases where the hybrid approach is inappropriate and the full BOR FD-TD approach is too computationally expensive. The work done includes the development of the governing modal parabolic wave equations that allow the general three dimensional problem to be simplified into a sequence of two dimensional problems. In addition, a PML absorbing boundary condition for the body of revolution parabolic wave equation method is developed. The BOR PWE approach is then implemented in a computer code, and the results produced compared to the exact BOR MoM methods to determine the loss in accuracy from the PWE approximations. The results of the BOR PWE approach are also compared to PTD results, to determine if this numerical approach yields more accuracy than high-frequency methods.

Chapter 2

RCS Prediction Using the Body of Revolution Finite-Difference Time-Domain Method

In this chapter, a finite-difference time-domain (FD-TD) algorithm is presented for the modeling of objects with body of revolution (BOR) symmetry. The BOR FD-TD formulation exploits the rotational symmetry of the problem by expressing the azimuthal (ϕ) dependence of the fields in a Fourier series. Since the azimuthal variation is accounted for analytically, each Fourier mode can be solved independently, and there is no gridding in the ϕ direction. This results in a BOR FD-TD algorithm which is two-dimensional in terms of computer memory usage.

2.1 BOR FD-TD Algorithm

One element in any application of the FD-TD technique involves the discretization of Maxwell's equations. The FD-TD difference equations can be derived from the integral form of Maxwell's equations by applying the integrals to small grid cells and assuming the electric and magnetic fields remain constant over each cell. For example, Figure 2-1 illustrates the interlocking grid cells used in the derivation of the BOR FD-TD equations from the integral form of Maxwell's equations. The time-dependent integral equations for a source free region are,

$$\oint \vec{E} \cdot d\vec{l} = \iint_S \sigma^* \vec{H} \cdot d\vec{S} - \frac{\partial}{\partial t} \iint_S \vec{B} \cdot d\vec{S} \tag{2.1}$$

$$\oint \vec{H} \cdot d\vec{l} = \iint_S \sigma \vec{E} \cdot d\vec{S} + \frac{\partial}{\partial t} \iint_S \vec{D} \cdot d\vec{S} \tag{2.2}$$

$$\oint \vec{D} \cdot d\vec{S} = 0 \tag{2.3}$$

$$\oint \vec{B} \cdot d\vec{S} = 0 \tag{2.4}$$

where $\vec{D} = \epsilon \vec{E}$ and $\vec{B} = \mu \vec{H}$.

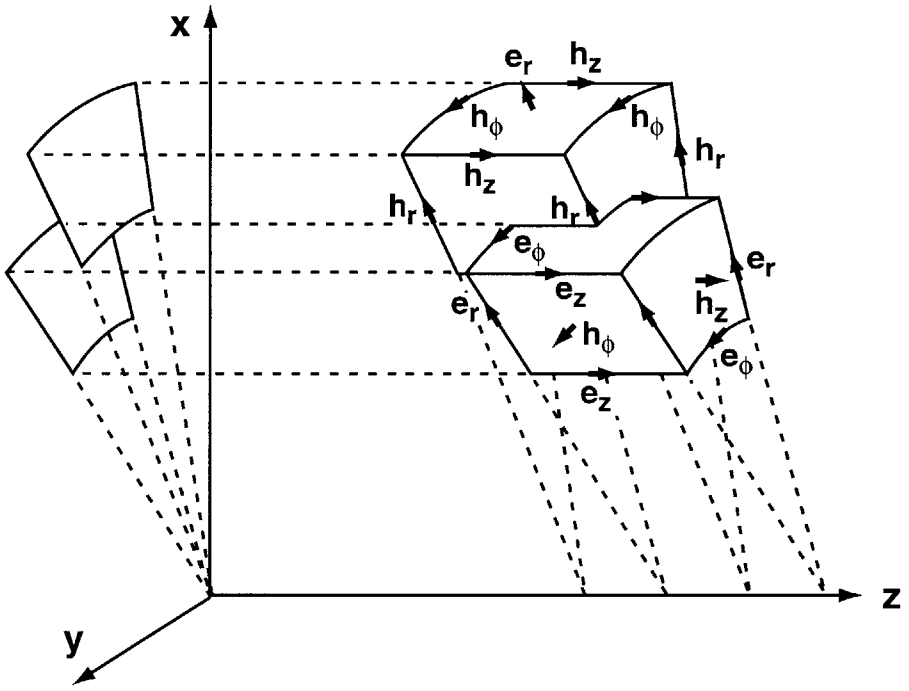

Figure 2-1: BOR 3D mesh showing interlocking grid cells

The FD-TD difference equations can also be derived by approximating the space and time derivatives in the differential form of Maxwell's equations with central difference expressions. In the following sections it is this later approach which will be applied, following the formulation by Davidson [10].

A second element in application of the FD-TD technique requires arranging the electric and magnetic fields in a grid structure. In three dimensions, the simplest grid a rectangular mesh, often called Yee's lattice. A significant advantage of this mesh is its simplicity,however, since objects are discretized with rectangular boundaries, curves and slanted lines must be approximated by staircases. Other grid systems are possible [15, 20, 21] including the cylindrical coordinate grid system that is well suited to modeling bodies of revolution. It is this grid which

is used in the BOR FD-TD algorithm.

A third element in application of the FD-TD technique is the time step solution for field values. Electric and magnetic fields are solved for in a "leap-frog" manner, where at each time step the electric fields are calculated in terms of the electric and magnetic fields of the previous time step. Magnetic fields are then updated in a similar manner. Since the fields are solved for one time step at a time, the method of solution is often referred to as a "marching in time approach."

2.1.1 Field Expansion

Maxwell's equations in vector differential form in a source free isotropic and homogeneous dielectric and magnetic material are,

$$\nabla \times \vec{E} = -\frac{\partial \vec{B}}{\partial t} + \sigma^* \vec{H} \tag{2.5}$$

$$\nabla \times \vec{H} = \frac{\partial \vec{D}}{\partial t} + \sigma \vec{E} \tag{2.6}$$

$$\nabla \cdot \vec{D} = 0 \tag{2.7}$$

$$\nabla \cdot \vec{B} = 0. \tag{2.8}$$

In order to exploit the rotational symmetry of the problem, the electric and magnetic fields are expressed as the following Fourier series,

$$\vec{E} = \sum_{m=0}^{\infty} (\vec{e}_{m,u} \cos m\phi + \vec{e}_{m,v} \sin m\phi) \tag{2.9}$$

$$\vec{H} = \sum_{m=0}^{\infty} \left(\vec{h}_{m,u} \cos m\phi + \vec{h}_{m,v} \sin m\phi\right) \tag{2.10}$$

where $\vec{e}_{m,u}$, $\vec{e}_{m,v}$, $\vec{h}_{m,u}$, and $\vec{h}_{m,v}$ are independent of ϕ. In practice, the Fourier series representing the electric and magnetic fields must be truncated at some finite number of terms. The number of modes needed to accurately represent the fields depends on the amount of variation in the azimuthal direction. One simple rule requires the number of modes to be at least $M = k\rho_{max} + 1$, where k is the wave number of the highest frequency of interest, and ρ_{max} is the maximum radius of the object being modeled.

Substituting the above expansions into (2.5) and (2.6) yields the following modal form of

Maxwell's equations,

$$\pm\frac{m}{\rho}\hat{\phi}\times\vec{e}_{v,u}+\nabla\times\vec{e}_{u,v} = -\mu\frac{\partial}{\partial t}\vec{h}_{u,v}+\sigma^{*}\vec{h}_{u,v} \tag{2.11}$$

$$\pm\frac{m}{\rho}\hat{\phi}\times\vec{h}_{v,u}+\nabla\times\vec{h}_{u,v} = \epsilon\frac{\partial}{\partial t}\vec{e}_{u,v}+\sigma\vec{e}_{u,v}. \tag{2.12}$$

Expanding the cross products and curl operators in equations (2.11) and (2.12) yields the following two decoupled sets of scalar equations governing the 12 field components.

$$\epsilon\frac{\partial}{\partial t}e_{u}^{\rho}+\sigma e_{u}^{\rho} = \frac{m}{\rho}h_{v}^{z}-\frac{\partial}{\partial z}h_{u}^{\phi} \tag{2.13}$$

$$\epsilon\frac{\partial}{\partial t}e_{v}^{\phi}+\sigma e_{v}^{\phi} = \frac{\partial}{\partial z}h_{v}^{\rho}-\frac{\partial}{\partial\rho}h_{v}^{z} \tag{2.14}$$

$$\epsilon\frac{\partial}{\partial t}e_{u}^{z}+\sigma e_{u}^{z} = -\frac{m}{\rho}h_{v}^{\rho}+\frac{1}{\rho}\frac{\partial}{\partial\rho}(\rho h_{u}^{\phi}) \tag{2.15}$$

$$\mu\frac{\partial}{\partial t}h_{v}^{\rho}+\sigma^{*}h_{v}^{\rho} = \frac{m}{\rho}e_{u}^{z}+\frac{\partial}{\partial z}e_{v}^{\phi} \tag{2.16}$$

$$\mu\frac{\partial}{\partial t}h_{u}^{\phi}+\sigma^{*}h_{u}^{\phi} = -\frac{\partial}{\partial z}e_{u}^{\rho}+\frac{\partial}{\partial\rho}e_{u}^{z} \tag{2.17}$$

$$\mu\frac{\partial}{\partial t}h_{v}^{z}+\sigma^{*}h_{v}^{z} = -\frac{m}{\rho}e_{u}^{\rho}-\frac{1}{\rho}\frac{\partial}{\partial\rho}(\rho e_{v}^{\phi}) \tag{2.18}$$

$$\epsilon\frac{\partial}{\partial t}e_{v}^{\rho}+\sigma e_{v}^{\rho} = -\frac{m}{\rho}h_{u}^{z}-\frac{\partial}{\partial z}h_{v}^{\phi} \tag{2.19}$$

$$\epsilon\frac{\partial}{\partial t}e_{u}^{\phi}+\sigma e_{u}^{\phi} = \frac{\partial}{\partial z}h_{u}^{\rho}-\frac{\partial}{\partial\rho}h_{u}^{z} \tag{2.20}$$

$$\epsilon\frac{\partial}{\partial t}e_{v}^{z}+\sigma e_{v}^{z} = \frac{m}{\rho}h_{u}^{\rho}+\frac{1}{\rho}\frac{\partial}{\partial\rho}(\rho h_{v}^{\phi}) \tag{2.21}$$

$$\mu\frac{\partial}{\partial t}h_{u}^{\rho}+\sigma^{*}h_{u}^{\rho} = -\frac{m}{\rho}e_{v}^{z}+\frac{\partial}{\partial z}e_{u}^{\phi} \tag{2.22}$$

$$\mu\frac{\partial}{\partial t}h_{v}^{\phi}+\sigma^{*}h_{v}^{\phi} = -\frac{\partial}{\partial z}e_{v}^{\rho}+\frac{\partial}{\partial\rho}e_{v}^{z} \tag{2.23}$$

$$\mu\frac{\partial}{\partial t}h_{u}^{z}+\sigma^{*}h_{u}^{z} = \frac{m}{\rho}e_{v}^{\rho}-\frac{1}{\rho}\frac{\partial}{\partial\rho}(\rho e_{u}^{\phi}) \tag{2.24}$$

Without loss of generality, the first set, (2.13)–(2.18), will be used in deriving the difference equations. The difference equations for the second set can be found by simply replacing m by $-m$ and interchanging the u and v subscripts in each of the six equations. Also, since only one set is being considered the u and v subscripts will be omitted in the following sections. Finally, since the object will be modeled in free space, it will be assumed that $\epsilon=\epsilon_0$, $\mu=\mu_0$, and that

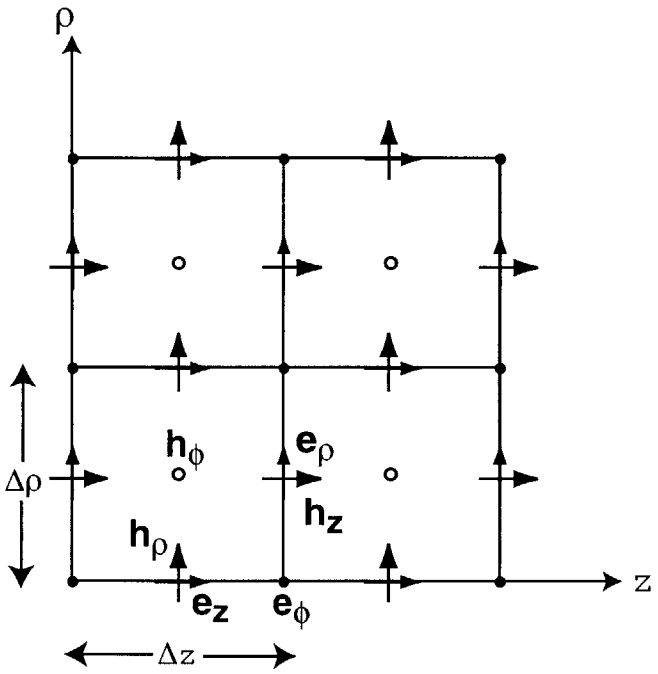

Figure 2-2: BOR 2D mesh showing interleaved field components

$\sigma = \sigma^* = 0$.

2.1.2 Difference Equations for Off-Axis Cells

As with the three dimensional FD-TD method, difference equations are found by replacing the space and time derivatives by a central-difference approximation. The second order accurate central-difference approximation for a first derivative is given by,

$$\frac{\partial f(\xi)}{\partial \xi} \approx \frac{f(\xi + \Delta\xi/2) - f(\xi - \Delta\xi/2)}{\Delta\xi}. \tag{2.25}$$

As shown in Figure 2-2, fields are arranged in an interleaving fashion similar to that of the Yee algorithm in Cartesian coordinates. Staggering the field components in both time and space leads to an efficient "marching in time" algorithm for solving for the scattered electric and magnetic fields. The following notation will be used for any function of time and space in the finite difference equations.

$$f(i\Delta\rho, k\Delta z, n\Delta t) = f|^n_{i,k} \tag{2.26}$$

The BOR FD-TD difference equations are found by applying the central difference approximation to the time and space derivatives in equations (2.13)–(2.18). For example, equation (2.13) becomes,

$$\epsilon_0 \frac{e_\rho|^{n+1/2}_{i+1/2,k+1/2} - e_\rho|^{n-1/2}_{i+1/2,k+1/2}}{\Delta t} = \frac{m}{(i+1/2)\Delta\rho} h_z|^n_{i+1/2,k+1/2} - \frac{h_\phi|^n_{i+1/2,k+1} - h_\phi|^n_{i+1/2,k}}{\Delta z}. \tag{2.27}$$

Rearranging terms,

$$\begin{aligned} e_\rho|^{n+1/2}_{i+1/2,k+1/2} &= e_\rho|^{n-1/2}_{i+1/2,k+1/2} + \eta_0 \frac{\Delta\tau}{\Delta z}\left(h_\phi|^n_{i+1/2,k} - h_\phi|^n_{i+1/2,k+1}\right) \\ &+ \eta_0 \frac{m\Delta\tau}{(i+1/2)\Delta\rho} h_z|^n_{i+1/2,k+1/2} \end{aligned} \tag{2.28}$$

where

$$\eta_0 = \sqrt{\frac{\mu_0}{\epsilon_0}} \tag{2.29}$$

$$\Delta\tau = c_0\Delta t = \frac{\Delta t}{\sqrt{\mu_0\epsilon_0}} \tag{2.30}$$

where η_0 is the free space impedance and c_0 is the speed of light in free space. The remaining BOR FD-TD difference equations are found by applying the central difference approximation to equations (2.14)–(2.18).

$$\begin{aligned} e_\phi|^{n+1/2}_{i,k+1/2} &= e_\phi|^{n-1/2}_{i,k+1/2} + \eta_0 \frac{\Delta\tau}{\Delta\rho}\left(h_z|^n_{i-1/2,k+1/2} - h_z|^n_{i+1/2,k+1/2}\right) \\ &+ \eta_0 \frac{\Delta\tau}{\Delta z}\left(h_\rho|^n_{i,k+1} - h_\rho|^n_{i,k}\right) \end{aligned} \tag{2.31}$$

$$\begin{aligned} e_z|^{n+1/2}_{i,k} &= e_z|^{n-1/2}_{i,k} + \eta_0 \frac{(i+1/2)\Delta\tau}{i\Delta\rho} h_\phi|^n_{i+1/2,k} - \eta_0 \frac{(i-1/2)\Delta\tau}{i\Delta\rho} h_\phi|^n_{i-1/2,k} \\ &- \eta_0 \frac{m\Delta\tau}{i\Delta\rho} h_\rho|^n_{i,k} \end{aligned} \tag{2.32}$$

$$h_\rho|^{n+1}_{i,k} = h_\rho|^n_{i,k} + \frac{1}{\eta_0}\frac{\Delta\tau}{\Delta z}\left(e_\phi|^{n+1/2}_{i,k+1/2} - e_\phi|^{n+1/2}_{i,k-1/2}\right) + \frac{1}{\eta_0}\frac{m\Delta\tau}{i\Delta\rho} e_z|^{n+1/2}_{i,k} \tag{2.33}$$

$$h_\phi|^{n+1}_{i+1/2,k} = h_\phi|^n_{i+1/2,k} + \frac{1}{\eta_0}\frac{\Delta\tau}{\Delta\rho}\left(e_z|^{n+1/2}_{i+1,k} - e_z|^{n+1/2}_{i,k}\right)$$

$$+ \quad \frac{1}{\eta_0}\frac{\Delta\tau}{\Delta z}\left(e_\rho|^{n+1/2}_{i+1/2,k-1/2} - e_\rho|^{n+1/2}_{i+1/2,k+1/2}\right) \tag{2.34}$$

$$\begin{aligned} h_z|^{n+1}_{i+1/2,k+1/2} &= h_z|^{n}_{i+1/2,k+1/2} + \frac{1}{\eta_0}\frac{i\Delta\tau}{(i+1/2)\Delta\rho}e_\phi|^{n+1/2}_{i,k+1/2} - \frac{1}{\eta_0}\frac{(i+1)\Delta\tau}{(i+1/2)\Delta\rho}e_\phi|^{n+1/2}_{i+1,k+1/2} \\ &- \frac{1}{\eta_0}\frac{m\Delta\tau}{(i+1/2)\Delta\rho}e_\rho|^{n+1/2}_{i+1/2,k+1/2} \end{aligned} \tag{2.35}$$

While the FD-TD method numerically solves equations (2.5) and (2.6) using the above difference equations, the two Gauss's Law relations, equations (2.7) and (2.8), are not explicitly enforced. However, the location of the $\vec{E}$ and $\vec{H}$ components in the grid and the central difference operations on these components implicitly enforce the two Gauss's Law relations so that all four of Maxwell's equations are satisfied [50].

2.1.3 Difference Equations for On-Axis Cells

Fields components that lie on the coordinate axis cannot be calculated using the difference equations presented in the previous section. Figure 2-2 shows that the e_z, e_ϕ, and h_ρ fields components lie on the axis. These cylindrical coordinate components can be expressed in terms of the Cartesian coordinate components as follows,

$$E_z(\rho,\phi,z,t) = E_z(x,y,z,t) \tag{2.36}$$

$$E_\phi(\rho,\phi,z,t) = -E_x(x,y,z,t)\sin\phi + E_y(x,y,z,t)\cos\phi \tag{2.37}$$

$$H_\rho(\rho,\phi,z,t) = H_x(x,y,z,t)\cos\phi + H_y(x,y,z,t)\sin\phi. \tag{2.38}$$

Along the z axis at any $z = z_0$, the $\hat{\rho}$ and $\hat{\phi}$ cylindrical coordinate components are not defined, but may be approximated by their values at $z = z_0$ and $\rho = \delta$ where δ is a small positive number. For $z = z_0$ and $\rho = \delta$ the Cartesian coordinate components can be approximated as constant and independent of ϕ. Thus, the ϕ dependence of the cylindrical coordinates can be seen to arise from the relations given in (2.36)–(2.38). Since the E_z field is the same in both coordinate systems, its only nonzero Fourier component will be that for $m = 0$. Similarly, since the only ϕ dependence for the E_ϕ and H_ρ fields arises from the $\cos\phi$ and $\sin\phi$ terms, their only nonzero Fourier components will be those for $m = 1$.

Difference Equation for the On-Axis e_z Field

The difference equation for the e_z field on the axis can be found by applying the integral form of Ampere's law (2.2) to a small loop of radius $\rho_0 = \Delta\rho/2$ centered at $\rho = 0$ and perpendicular to the z axis.

$$\epsilon\frac{\partial}{\partial t}\int_0^{\rho_0}\int_0^{2\pi}[e_{z,u}(0,z,t)\cos m\phi + e_{z,v}(0,z,t)\sin m\phi]\,\rho\; d\phi d\rho$$
$$= \int_0^{2\pi}[h_{\phi,u}(\rho_0,z,t)\cos m\phi + h_{\phi,v}(\rho_0,z,t)\sin m\phi]\,\rho_0\; d\phi \tag{2.39}$$

For the case $m = 0$, the integrals above can be evaluated to give the following relationship,

$$\epsilon\pi\rho_0^2\frac{\partial}{\partial t}e_{z,u}(0,z,t) = 2\pi\rho_0 h_{\phi,u}(\rho_0,z,t). \tag{2.40}$$

Discretizing the time derivative using a central difference approximation, and using the same indexing scheme as in the previous section produces the following difference equation for the $e_{z,u}$ term,

$$e_{z,u}|_{0,k}^{n+1/2} = e_{z,u}|_{0,k}^{n-1/2} + \frac{4\Delta t}{\epsilon\Delta\rho}h_{\phi,u}|_{1/2,k}^{n}. \tag{2.41}$$

A similar derivation can be carried out to yield a equation identical to (2.41) for the $e_{z,v}$ term along the axis [50].

Difference Equation for the On-Axis e_ϕ Field

The difference equation for the e_ϕ field component along the axis can be found by applying the integral form of Ampere's law (2.2) to a contour about the e_ϕ in the ρ–z plane.
Applying Ampere's law for the mode $m = 1$ fields to the contour illustrated in Figure 2-3 yields,

$$\epsilon\frac{\partial}{\partial t}\int_{z_1}^{z_2}\int_0^{\rho_0}[e_{\phi,u}(0,zz,t)\cos\phi + e_{\phi,v}(0,zz,t)\sin\phi]\;\; d\phi dz$$
$$= \int_{z_1}^{z_2}[h_{z,u}(0,zz,t)\cos\phi + h_{z,v}(0,zz,t)\sin\phi]\,dz$$
$$+ \int_0^{\rho_0}[h_{\rho,u}(0,z_2,t)\cos\phi + h_{\rho,v}(0,z_2,t)\sin\phi]\,d\rho$$
$$+ \int_{z_2}^{z_1}[h_{z,u}(\rho,zz,t)\cos\phi + h_{z,v}(\rho_0,zz,t)\sin\phi]\,dz$$
$$+ \int_{\rho_0}^{0}[h_{\rho,u}(0,z_1,t)\cos\phi + h_{\rho,v}(0,z_1,t)\sin\phi]\,dz \tag{2.42}$$

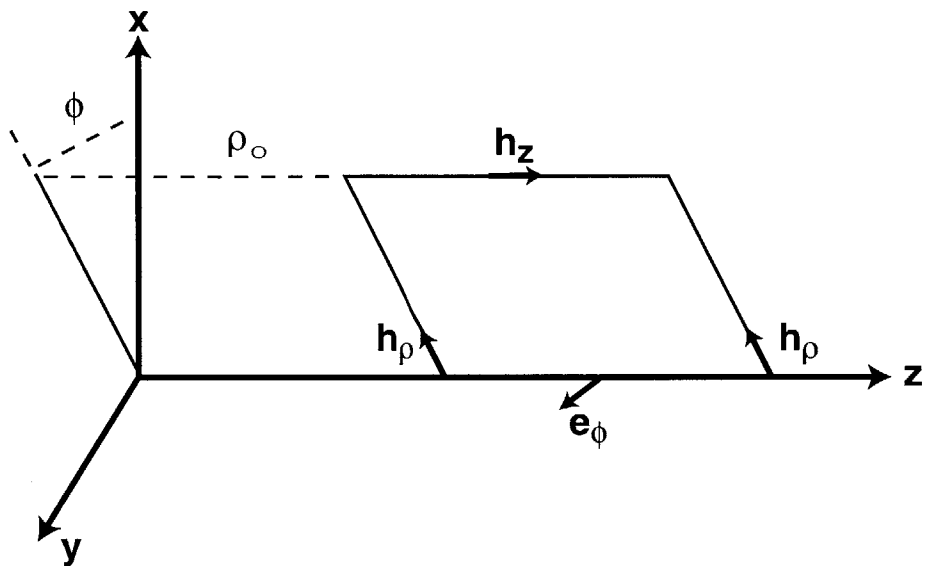

Figure 2-3: Contour integral used for calculation of on-axis e_ϕ term.

where $\rho = \Delta\rho/2$ and $zz = z_1 + \Delta z/2$. Integrating the above equation noting that $h_z = 0$ at $\rho = 0$ for $m = 1$ and separating the cosine and sine terms yields,

$$\left[\epsilon\Delta z\frac{\Delta\rho}{2}\frac{\partial}{\partial t}e_{\phi,u}(0,zz,t)\right]\cos\phi$$
$$= \left\{-\Delta z h_{z,u}(\rho_0,zz,t) + \frac{\Delta\rho}{2}\left[h_{\rho,u}(0,z_2,t) - h_{\rho,u}(0,z_1,t)\right]\right\}\cos\phi \tag{2.43}$$
$$\left[\epsilon\Delta z\frac{\Delta\rho}{2}\frac{\partial}{\partial t}e_{\phi,v}(0,zz,t)\right]\sin\phi$$
$$= \left\{-\Delta z h_{z,v}(\rho_0,zz,t) + \frac{\Delta\rho}{2}\left[h_{\rho,u}(0,z_2,t) - h_{\rho,v}(0,z_1,t)\right]\right\}\sin\phi. \tag{2.44}$$

Using the same indexing scheme as before, and using a central difference approximation for the time derivative yields the following difference equation for updating the $e_{\phi,u}$ and $e_{\phi,v}$ field components along the axis.

$$e_\phi|^{n+1/2}_{0,k+1/2} = e_\phi|^{n-1/2}_{0,k+1/2} - \frac{2\Delta t}{\epsilon\Delta\rho}h_z|^n_{1/2,k+1/2} + \frac{\Delta t}{\epsilon\Delta z}\left(h_\rho|^n_{0,k+1} - h_\rho|^n_{0,k}\right) \tag{2.45}$$

Difference Equation for the On-Axis h_ρ Field

The difference equations for the on-axis h_ρ fields are found by discretizing the scalar equations (2.16) and (2.22) for mode $m = 1$ since h_ρ is zero for all other modes. Although the e_z field

at the axis is zero for mode $m = 1$, the e_z term in these equations is actually a measure of the derivative of e_z with respect to ϕ and is not zero. Thus, the value of e_z from the cell above is used as an approximation to the ϕ derivative yielding the difference equations,

$$h_{\rho,v}|_{0,k}^{n+1} = h_{\rho,v}|_{0,k}^{n} + \frac{\Delta t}{\mu\Delta\rho} e_{z,u}|_{1,k}^{n+1/2} + \frac{\Delta t}{\mu\Delta z}\left(e_{\phi,v}|_{0,k+1/2}^{n+1/2} - e_{\phi,v}|_{0,k-1/2}^{n+1/2}\right) \quad (2.46)$$

$$h_{\rho,u}|_{0,k}^{n+1} = h_{\rho,u}|_{0,k}^{n} - \frac{\Delta t}{\mu\Delta\rho} e_{z,v}|_{1,k}^{n+1/2} + \frac{\Delta t}{\mu\Delta z}\left(e_{\phi,u}|_{0,k+1/2}^{n+1/2} - e_{\phi,u}|_{0,k-1/2}^{n+1/2}\right). \quad (2.47)$$

2.1.4 Numerical Concerns

Explicit finite-difference schemes have stability restrictions on choices for the space and time increments. The conditions necessary for stability impose an upper limit on the value of the time increment. The upper limit on the time step is the well-known Courant-Friedrichs-Lewy stability criterion [50]. The numerical stability bound for the two and three dimensional FD-TD methods are given by,

$$\Delta t_{2D} \leq \frac{1}{c\sqrt{\frac{1}{(\Delta x)^2} + \frac{1}{(\Delta y)^2}}} \quad (2.48)$$

$$\Delta t_{3D} \leq \frac{1}{c\sqrt{\frac{1}{(\Delta x)^2} + \frac{1}{(\Delta y)^2} + \frac{1}{(\Delta z)^2}}} \quad (2.49)$$

where Δx, Δy, and Δz are the space increments and Δt is the time increment. Although the BOR FD-TD algorithm works by solving a sequence of two dimensional problems, the above stability requirement cannot be used. Instead, the numerical stability bound for the BOR FD-TD algorithm depends and the space increments as well as the mode number [50], and is given by

$$\Delta t_{\mathrm{BOR}} \leq \frac{\Delta}{sc} \quad (2.50)$$

where $s \approx \max(\sqrt{2}, m+1)$ and Δ is the space increment. Note that for low order modes the stability requirement is comparable to the stability requirements for the 2D and 3D FDTD methods. However, as the mode number increases, the required BOR FD-TD time step becomes progressively smaller.

Another numerical concern is the potential for dispersion caused by errors in the phase velocities of waves traveling through the FD-TD lattice. While in free space the phase velocity should equal the group velocity for all frequencies and directions, a wave in the FD-TD lattice

will have a phase velocity that is slightly smaller than its group velocity, depending both on its frequency and the direction it is traveling. One way to reduce the amount of numerical dispersion is to increase Δt. As $c\Delta t$ approaches Δ, the size of each spatial cell, the numerical dispersion becomes negligible. However, as noted in (2.48)–(2.50), $c\Delta t$ has an upper limit less than Δ, so a trade off must be made so that the amount of error due to numerical dispersion is small while numerical stability is maintained.

2.1.5 Modeling of Perfect Electric Conductors

The interface between any two media is generally chosen to occur at integer nodes (i, j, k) so that electric fields are tangential and magnetic fields are normal to the surface. The case of a perfect electric conductor, or PEC, in free space region is particularly simple to model. The boundary condition for a PEC requires that all tangential electric fields are zero.

$$\hat{n} \times \vec{E} = 0 \tag{2.51}$$

In the case where the PEC does not align along the cells, a staircase approximation is used, and tangential electric fields of the approximate stair-step surface are set to zero. Since the cells are on the order of one tenth the smallest wavelength, this approximation generally does not introduce a significant error. In some cases, such as a PEC sphere, where a traveling wave is present, a staircase model can introduce significant errors. The errors may be minimized by either reducing the size of the cells or using a modified cell shape for the cells along the object being modeled [26].

2.1.6 Computational Domain

The computational domain is that region of space which is discretized with the FD-TD lattice, and is modeled by the FD-TD method. Figure 2-4 illustrates the two dimensional computational domain for the BOR FD-TD algorithm. It is divided into to three regions: the total field region, the scattered field region, and the absorbing boundary condition region. The distinction between total and scattered fields will be made clear in Section 2.3. While the general electromagnetic scattering problem is unbounded, the computational domain must be truncated by an appropriate boundary condition in order for the problem to be well-posed. The boundary condition along the bottom edge of the computational domain is accounted for by using the

on-axis equations developed in Section 2.1.3, however the boundary conditions for the other three edges remains unspecified. One possible boundary condition for the other edges is to simply set the fields along the edges to zero. However, this approach leads to the outside of the domain functioning as a perfect conductor, which causes reflections at the boundary. Provided the domain is large enough, an arbitrary geometry can modeled inside the domain for durations where the wave is does not reach the boundary. In practice, this requires that the computational domain be very large to ensure no reflections at the boundary, and requires much more computation time and memory. An alternate approach involves the use of an absorbing boundary condition, designed to minimize reflections at the computational domain edges, and effectively simulate an unbounded region beyond.

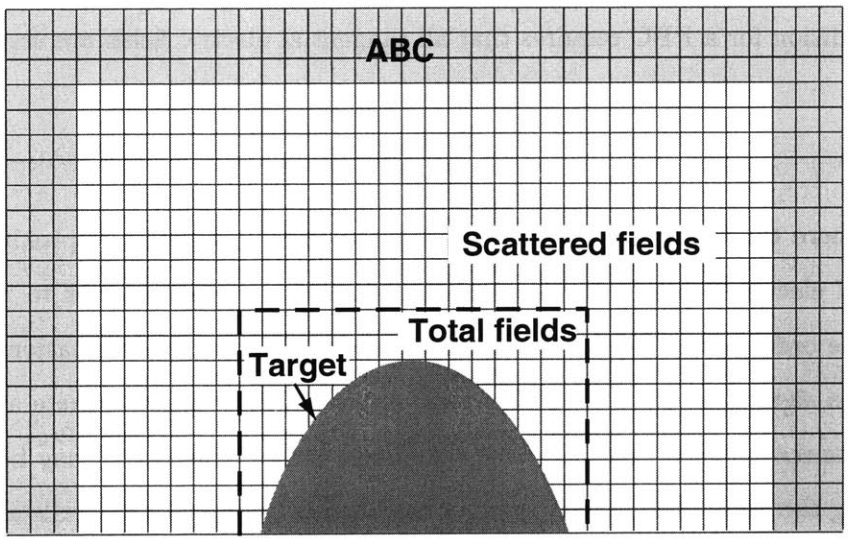

Figure 2-4: BOR FD-TD computational domain

2.2 Absorbing Boundary Conditions

2.2.1 2nd Order Boundary Condition

One approach to the absorbing boundary condition (ABC) problem is the second order boundary condition formulated by Engquist and Majda [14],

$$\left[\frac{\partial^2}{\partial n \partial \tau} + \frac{\partial^2}{\partial \tau^2} - \frac{1}{2}\left(\frac{\partial^2}{\partial T_1^2} + \frac{\partial^2}{\partial T_2^2}\right)\right] w = 0 \tag{2.52}$$

where w is a field quantity which is tangential to the absorbing boundary, $\hat{n}$ is the normal direction, $\hat{T_1}$, $\hat{T_2}$ are the tangential directions, and τ is time normalized with respect to the speed of light. The second-order absorbing boundary condition works very well for waves which are incident normally or nearly normally to the edges of the computational domain, and does not work as well for waves which are incident at grazing angles.

2.2.2 Berenger's Perfectly Matched Layer ABC

An alternate ABC, the perfectly matched layer (PML), capable of effectively absorbing waves which are incident at any angle was introduced by Berenger in 1994 [3]. Berenger's PML technique is based on the idea of using a layer of lossy material to absorb outgoing radiation from the computation domain. Ideally, the lossy layer should be designed such that a planar interface between the lossy layer and free space is reflectionless for all frequencies, polarizations, and angles of incidence. Loss in the PML region can be achieved by introducing the electric conductivity and magnetic loss terms in Maxwell's equations. For a media with electric conductivity σ, and magnetic conductivity σ^*, the impedance of the medium equals the impedance of free space when

$$\frac{\sigma}{\sigma^*} = \frac{\epsilon_0}{\mu_0}. \tag{2.53}$$

A wave traveling normally across a boundary between such a medium and free space will enter the absorbing region without reflection. However, waves which are not normally incident cannot transverse the boundary without reflections. In order to obtain a reflectionless interface for waves of arbitrary incident angles, additional degrees of freedom are needed. In Berenger's PML technique, the necessary additional degrees of freedom are obtained by splitting field components into two subcomponents (e.g. $H_x = H_{xy} + H_{xz}$), each derived from a single spatial derivative term of the curl expression in Maxwell's equations. For example, in the two dimensional TE case, the field components in the PML medium are governed by the following four equations:

$$\epsilon_0 \frac{\partial E_x}{\partial t} + \sigma_y E_x = \frac{\partial (H_{zx} + H_{zy})}{\partial y} \tag{2.54}$$

$$\epsilon_0 \frac{\partial E_y}{\partial t} + \sigma_x E_y = -\frac{\partial (H_{zx} + H_{zy})}{\partial x} \tag{2.55}$$

$$\mu_0 \frac{\partial H_{zx}}{\partial t} + \sigma_x^* H_{zx} = -\frac{\partial E_y}{\partial x} \tag{2.56}$$

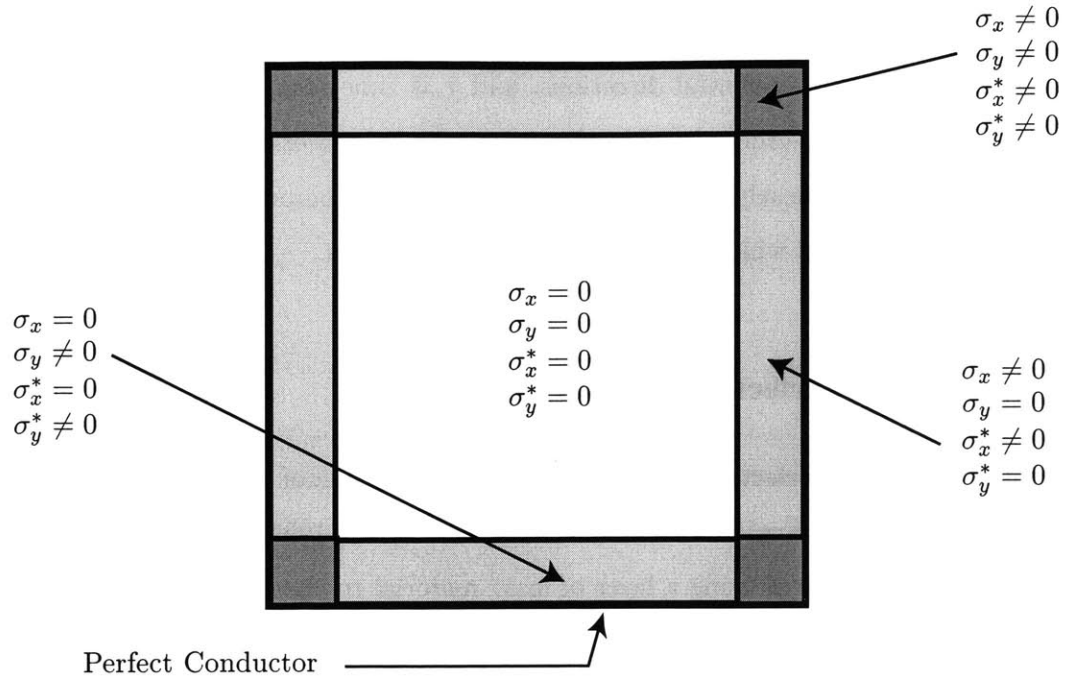

Figure 2-5: Selection of conductivities for 2D PML regions

$$\mu_0 \frac{\partial H_{zy}}{\partial t} + \sigma_y^* H_{zy} = \frac{\partial E_x}{\partial y} \tag{2.57}$$

where the parameters σ_x, σ_x^*, σ_y, and σ_y^* are the electric and magnetic conductivities of the medium. Note, if $\sigma_x = \sigma_x^* = \sigma_y = \sigma_y^* = 0$, then the above equations reduce to Maxwell's equations of free space. If $\sigma_x = \sigma_x^*$ and $\sigma_y = \sigma_y^*$, the above equations reduce to Maxwell's equations for an absorbing medium. In addition, if $\sigma_y = \sigma_y^* = 0$ and σ_x, σ_x^* nonzero, the PML medium can absorb a plane wave (E_y, H_{zx}) propagating along x, but it does not absorb a wave (E_x, H_{zy}) along y and vice versa for the waves if $\sigma_x = \sigma_x^* = 0$ and σ_y, σ_y^* are nonzero [3].

As Berenger has shown, the interface reflection between two PML media whose conductivities satisfy the impedance condition (2.53) is zero when at an interface normal to x, the two regions have equal σ_y and σ_y^*, or when at an interface normal to y, the two regions have equal σ_x and σ_x^*. Since free space can be considered a PML medium in which $\sigma_x = \sigma_x^* = \sigma_y = \sigma_y^* = 0$ the above condition can be used to construct the PML regions surrounding the computational domain. Figure 2-5 gives the parameters of the surrounding PML layers for a two- dimensional mesh. For instance, on both the left and right sides of the computational domain, the absorbing layers are a PML medium with $\sigma_y = \sigma_y^* = 0$, and nonzero σ_x, σ_x^* whose conductivities are related by the impedance condition (2.53).

Berenger originally developed the PML concept for truncating two dimensional Cartesian coordinate grids, however Katz, Thiele, and Taflove [27] later extended it for the truncation of three dimensional Cartesian coordinate grids. To extend the range of applicability, the PML concept was also extended to cylindrical coordinate grids [24] and nonorthogonal FD-TD grids [42, 45]. However, as discussed in [54], approximate impedance matching conditions were used, since the perfect matching conditions were derived based on the assumption that the metric coefficients are independent of the spatial coordinates.

2.2.3 Generalized PML ABCs with Stretched Coordinates

An alternate PML formulation approach based on a coordinate stretching viewpoint was proposed by Chew and Weedon in [9]. Their approach involves the development of a modified set of Maxwell's equations via a complex coordinate transform. The additional degrees-of-freedom introduced by the complex coordinate stretching allow for the specification of a lossy material layer such that the interface between free space regions and PML regions is reflectionless for all frequencies, polarizations, and angles of incidence. Maxwell's equations in a stretched coordinate system are given by ($e^{-i\omega t}$ convention) [9]

$$\nabla_\sigma \times \vec{E} = i\omega\mu\vec{H} \tag{2.58}$$

$$\nabla_\sigma \times \vec{H} = -i\omega\epsilon\vec{E} \tag{2.59}$$

$$\nabla_\sigma \cdot \epsilon\vec{E} = 0 \tag{2.60}$$

$$\nabla_\sigma \cdot \mu\vec{H} = 0 \tag{2.61}$$

where

$$\nabla_\sigma = \hat{x}\frac{1}{s_x}\frac{\partial}{\partial x} + \hat{y}\frac{1}{s_y}\frac{\partial}{\partial y} + \hat{z}\frac{1}{s_z}\frac{\partial}{\partial z}. \tag{2.62}$$

In the above, $s_i, i = x, y, z$ are complex coordinate stretching variables. With the change of variables,

$$\zeta \rightarrow \tilde{\zeta} = \int_0^\zeta s_\zeta(\zeta')d\zeta' \tag{2.63}$$

where $\zeta = x, y, z$, it is possible to show [8] that Maxwell's equations inside the PML medium can be recast into the same form of the original Maxwell's equations but on a complex variable

spatial domain. Under the change of variables, the ∇_σ operator becomes,

$$\nabla_\sigma \rightarrow \tilde{\nabla} = \hat{x}\frac{\partial}{\partial \tilde{x}} + \hat{y}\frac{\partial}{\partial \tilde{y}} + \hat{z}\frac{\partial}{\partial \tilde{z}} \tag{2.64}$$

since

$$\frac{\partial}{\partial \tilde{x}} = \frac{1}{s_x}\frac{\partial}{\partial x}, \quad \frac{\partial}{\partial \tilde{y}} = \frac{1}{s_y}\frac{\partial}{\partial y}, \quad \frac{\partial}{\partial \tilde{z}} = \frac{1}{s_z}\frac{\partial}{\partial z}. \tag{2.65}$$

It then follows that the Maxwell's equations inside the PML medium (2.58)–(2.61) become,

$$\tilde{\nabla} \times \vec{E} = i\omega\mu\vec{H} \tag{2.66}$$

$$\tilde{\nabla} \times \vec{H} = -i\omega\epsilon\vec{E} \tag{2.67}$$

$$\tilde{\nabla} \cdot \epsilon\vec{E} = 0 \tag{2.68}$$

$$\tilde{\nabla} \cdot \mu\vec{H} = 0. \tag{2.69}$$

In free space, $s_\zeta = 1$, and the transformed Maxwell's equations are the original Maxwell's equations, but if for example,

$$s_\zeta(\zeta') = 1 + \frac{i\sigma_\zeta(\zeta')}{\omega\epsilon} \tag{2.70}$$

the medium is a lossy PML region and the fields inside the PML are not Maxwellian since they obey the modified Maxwell's equations rather than the original Maxwell's equations. However, the interface between the PML region and non-PML regions is reflectionless if the s_ζ's satisfy conditions similar to Berenger's conditions on the σ_i's. Moreover, this change of variables formulation can be generalized to other coordinate systems to provide PML's on these systems [8]. In [54], PML formulations for a cylindrical coordinate system and a spherical coordinate system are developed. In order to absorbing outward traveling waves in both the z and ρ directions, the following mappings are used,

$$\tilde{z} = \int_0^z s_z(z')dz' = \int_0^z 1 + \frac{i\sigma_z(z')}{\omega\epsilon}dz' = z + \frac{i\Delta_z(z)}{\omega\epsilon} \tag{2.71}$$

$$\tilde{\rho} = \int_0^\rho s_\rho(\rho')d\rho' = \int_0^\rho 1 + \frac{i\sigma_\rho(\rho')}{\omega\epsilon}d\rho' = \rho + \frac{i\Delta_\rho(\rho)}{\omega\epsilon}. \tag{2.72}$$

In this case, the del operator in cylindrical coordinates becomes,

$$\tilde{\nabla} = \hat{\rho}\frac{1}{\tilde{\rho}}\frac{\partial}{\partial\tilde{\rho}}\tilde{\rho} + \hat{\phi}\frac{1}{\tilde{\rho}}\frac{\partial}{\partial\phi} + \hat{z}\frac{\partial}{\partial\tilde{z}}. \tag{2.73}$$

The BOR-FDTD PML formulation then proceeds by substituting the Fourier expansions (2.9)–(2.10) of the electric and magnetic fields into the modified Maxwell's equations (2.66)–(2.67) using (2.73). As a result, the following modal equations are obtained for the fields inside the PML region.

$$\pm\frac{m}{\tilde{\rho}}\hat{\phi}\times\vec{e}_{v,u}+\tilde{\nabla}\times\vec{e}_{u,v} = i\omega\mu\vec{h}_{u,v} \tag{2.74}$$

$$\pm\frac{m}{\tilde{\rho}}\hat{\phi}\times\vec{h}_{v,u}+\tilde{\nabla}\times\vec{h}_{u,v} = -i\omega\epsilon\vec{e}_{u,v} \tag{2.75}$$

Expanding the cross products and curls yields a set of twelve scalar equations of the same form as (2.13)–(2.24) but over a complex variable spatial domain. As before, the twelve equations decouple into two independent sets of six equations. The scalar equations corresponding to the first set for the fields inside the PML region are,

$$-i\omega\epsilon e_\rho = \frac{m}{\tilde{\rho}}h_z-\frac{\partial}{\partial\tilde{z}}h_\phi \tag{2.76}$$

$$-i\omega\epsilon e_\phi = \frac{\partial}{\partial\tilde{z}}h_\rho-\frac{\partial}{\partial\tilde{\rho}}h_z \tag{2.77}$$

$$-i\omega\epsilon e_z = -\frac{m}{\tilde{\rho}}h_\rho+\frac{1}{\tilde{\rho}}\frac{\partial}{\partial\tilde{\rho}}(\tilde{\rho}h_\phi) \tag{2.78}$$

$$-i\omega\mu h_\rho = \frac{m}{\tilde{\rho}}e_z+\frac{\partial}{\partial\tilde{z}}e_\phi \tag{2.79}$$

$$-i\omega\mu h_\phi = -\frac{\partial}{\partial\tilde{z}}e_\rho+\frac{\partial}{\partial\tilde{\rho}}e_z \tag{2.80}$$

$$-i\omega\mu h_z = -\frac{m}{\tilde{\rho}}e_\rho-\frac{1}{\tilde{\rho}}\frac{\partial}{\partial\tilde{\rho}}(\tilde{\rho}e_\phi). \tag{2.81}$$

In order to facilitate the conversion of the above equations into the time domain in a form suitable for time-stepping, the fields are split. For example, the ρ component of the electric field is split as $e_\rho = e_{\rho z} + e_{\rho\phi}$ where $e_{\rho z}$ and $e_{\rho\phi}$ are defined by

$$-i\omega\epsilon s_\phi e_{\rho\phi} = \frac{m}{\rho}h_z \tag{2.82}$$

$$-i\omega\epsilon s_z e_{\rho z} = \frac{\partial}{\partial z}h_\phi \tag{2.83}$$

and

$$s_\phi(\rho)=\frac{\tilde{\rho}}{\rho}=1+\frac{i\Delta_\rho(\rho)}{\rho\omega\epsilon}. \tag{2.84}$$

Similarly, the e_ϕ component is split as $e_\phi = e_{\phi z} + e_{\phi\rho}$ where $e_{\phi z}$ and $e_{\phi\rho}$ are defined from,

$$-i\omega\epsilon s_z e_{\phi z} = \frac{\partial}{\partial z} h_\rho \tag{2.85}$$

$$-i\omega\epsilon s_\rho e_{\phi\rho} = -\frac{\partial}{\partial \rho} h_z. \tag{2.86}$$

In order to split the e_z component in a manner suitable for time-stepping, it is necessary to first expand the derivative with respect to ρ.

$$-i\omega\epsilon e_z = \frac{\partial}{\partial \tilde{\rho}} h_\phi + \frac{1}{\tilde{\rho}} h_\phi + \frac{m}{\tilde{\rho}} h_\rho \tag{2.87}$$

The e_z can then be split as follows.

$$-i\omega\epsilon s_\rho e_{z\rho} = \frac{\partial}{\partial \rho} h_\phi \tag{2.88}$$

$$-i\omega\epsilon s_\phi e_{z\phi} = \frac{1}{\rho} h_\phi + \frac{m}{\rho} h_\rho \tag{2.89}$$

The h field terms are split in a similar fashion. Next, the frequency domain equations are converted to the time domain to yield a set twelve equations governing the fields inside the PML medium. Using the definitions, $\sigma_\phi = \Delta_\rho/\rho$, and $\sigma_i^* = \sigma_i \mu/\epsilon$ for $i = \rho, \phi, z$, the PML equations can be cast in a form similar to the PML equations given in [24] except that σ_ϕ is not independent of σ_ρ.

$$\epsilon\frac{\partial}{\partial t} e_{\rho z} + \sigma_z e_{\rho z} = -\frac{\partial}{\partial z}\left(h_{\phi z} + h_{\phi\rho}\right) \tag{2.90}$$

$$\epsilon\frac{\partial}{\partial t} e_{\rho\phi} + \sigma_\phi e_{\rho\phi} = \frac{m}{\rho}\left(h_{z\rho} + h_{z\phi}\right) \tag{2.91}$$

$$\epsilon\frac{\partial}{\partial t} e_{\phi z} + \sigma_z e_{\phi z} = \frac{\partial}{\partial z}\left(h_{\rho z} + h_{\rho\phi}\right) \tag{2.92}$$

$$\epsilon\frac{\partial}{\partial t} e_{\phi\rho} + \sigma_\rho e_{\phi\rho} = -\frac{\partial}{\partial \rho}\left(h_{z\rho} + h_{z\phi}\right) \tag{2.93}$$

$$\epsilon\frac{\partial}{\partial t} e_{z\rho} + \sigma_\rho e_{z\rho} = \frac{\partial}{\partial \rho}\left(h_{\phi z} + h_{\phi\rho}\right) \tag{2.94}$$

$$\epsilon\frac{\partial}{\partial t} e_{z\phi} + \sigma_\phi e_{z\phi} = -\frac{m}{\rho}\left(h_{\rho z} + h_{\rho\phi}\right) + \frac{1}{\rho}\left(h_{\phi z} + h_{\phi\rho}\right) \tag{2.95}$$

$$\mu\frac{\partial}{\partial t} h_{\rho z} + \sigma_z^* h_{\rho z} = \frac{\partial}{\partial z}\left(e_{\phi z} + e_{\phi\rho}\right) \tag{2.96}$$

$$\mu\frac{\partial}{\partial t} h_{\rho\phi} + \sigma_\phi^* h_{\rho\phi} = \frac{m}{\rho}\left(e_{z\rho} + e_{z\phi}\right) \tag{2.97}$$

$$\mu\frac{\partial}{\partial t}h_{\phi z} + \sigma_z^* h_{\phi z} = -\frac{\partial}{\partial z}(e_{\rho z} + e_{\rho\phi}) \tag{2.98}$$

$$\mu\frac{\partial}{\partial t}h_{\phi\rho} + \sigma_\rho^* h_{\phi\rho} = \frac{\partial}{\partial \rho}(e_{z\rho} + e_{z\phi}) \tag{2.99}$$

$$\mu\frac{\partial}{\partial t}h_{z\rho} + \sigma_\rho^* h_{z\rho} = -\frac{\partial}{\partial \rho}(e_{\phi z} + e_{\phi\rho}) \tag{2.100}$$

$$\mu\frac{\partial}{\partial t}h_{z\phi} + \sigma_\phi^* h_{z\phi} = -\frac{m}{\rho}(e_{\rho z} + e_{\rho\phi}) - \frac{1}{\rho}(e_{\phi z} + e_{\phi\rho}) \tag{2.101}$$

In order to achieve a reflectionless interface between free space and the PML medium, all the PML parameters must remain the same except for the complex stretching variable component normal to the interface [8]. The same condition holds for the interface between two PML regions to be reflectionless. For the BOR FD-TD PML, this implies that $s_z = 1$ or equivalently that $\sigma_z = 0$ at an interface with a ρ normal, and that $s_\rho = 1$, $\sigma_\rho = 0$ at an interface with a z normal. Figure 2-6 illustrates the selection of conductivities for the BOR PML regions.

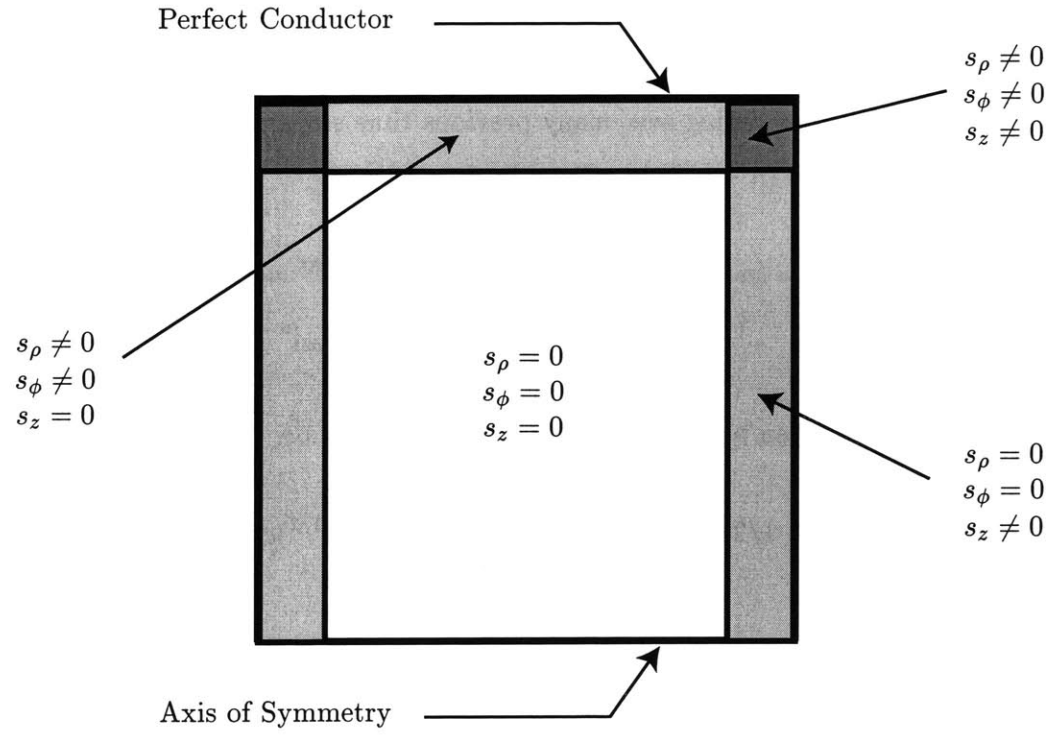

Figure 2-6: Selection of stretching variables for BOR PML regions

2.2.4 Discretization of PML Equations

As discussed previously, the PML technique works by providing a reflectionless interface to a medium that can absorbing outgoing waves. Since the PML medium is backed by a perfect conductor, the wave will be reflected back towards the computational domain. The magnitude of the reflected field will be determined by the amount of attenuation the PML medium provides, which is a function of the conductivities and thickness of the medium. Thus, to achieve the high loss in the reflected wave, it is desirable to use large conductivity values, however, the standard central difference approximation can not be used to accurately represent the rapidly decaying fields [50]. Instead, exponential time-stepping [22] is used to discretize equations (2.90)-(2.101). In this approach the equations are treated as ordinary differential equations which are solved explicitly. For example, consider the PML equation for the $e_{\rho z}$ term (2.90), whose total solution consists of a a homogeneous solution and a particular solution. The homogeneous solution is,

$$e_{\rho z}^{homog}(t) = Ce^{-(\sigma_z/\epsilon)t} \tag{2.102}$$

where C some constant. The constant C can be found by arguing that the homogeneous solution results from excitations combining over many previous time steps. Since the $e_{\rho z}$ is known at the previous time step, $t = (n-1/2)\Delta t$, C can be found by,

$$\begin{aligned} e_{\rho z}^{homog}(t = (n-1/2)\Delta t) &= Ce^{-(\sigma_z/\epsilon)(n-1/2)\Delta t} = e_{\rho z}|^{n-1/2} \\ \Rightarrow C &= e^{(\sigma_z/\epsilon)(n-1/2)\Delta t} e_{\rho z}|^{n-1/2}. \end{aligned} \tag{2.103}$$

The value of $e_{\rho z}^{homog}$ can then be found at the next time step to be,

$$\begin{aligned} e_{\rho z}^{homog}(t = (n+1/2)\Delta t) &= e^{(\sigma_z/\epsilon)(n-1/2)\Delta t}(e_{\rho z}|^{n-1/2})e^{-(\sigma_z/\epsilon)(n+1/2)\Delta t} \\ &= e_{\rho z}|^{n-1/2}e^{-(\sigma_z/\epsilon)\Delta t}. \end{aligned} \tag{2.104}$$

The particular solution is given by,

$$e_{\rho z}^{part}(t') = -\frac{1}{\sigma_z}\frac{\partial(h_{\phi z} + h_{\phi\rho})}{\partial z} + Ke^{-(\sigma_z/\epsilon)t'}. \tag{2.105}$$

Since the homogeneous solution accounts for contributions from all previous time steps, the particular solution can be seen to arise from the h_ϕ field at the current step. It then follows

that at the beginning of the time step $t' = 0$,

$$\begin{aligned} e_{\rho z}^{part}(t'=0) = 0 &= -\frac{1}{\sigma_z}\frac{\partial}{\partial z}(h_{\phi z} + h_{\phi\rho}) + K \\ \Rightarrow K &= +\frac{1}{\sigma_z}\frac{\partial}{\partial z}(h_{\phi z} + h_{\phi\rho}). \end{aligned} \tag{2.106}$$

Evaluating the particular solution at the end of the time step, $t' = \Delta t$.

$$e_{\rho z}^{part}(t'=\Delta t) = \frac{e^{-(\sigma_z/\epsilon)\Delta t} - 1}{\sigma_z}\frac{\partial}{\partial z}(h_{\phi z} + h_{\phi\rho}) \tag{2.107}$$

The time-stepping equation for the $e_{\rho z}$ field can then be obtained by combining the homogeneous and particular solutions and discretizing the $\frac{\partial}{\partial z}$ term,

$$\begin{aligned} e_{\rho z}|_{i+1/2,k+1/2}^{n+1/2} = e^{-\sigma_z \Delta t/\epsilon} e_{\rho z}|_{i+1/2,k+1/2}^{n-1/2} + \frac{e^{-\sigma_z \Delta t/\epsilon} - 1}{\sigma_z \Delta z}\Big(h_{\phi z}|_{i+1/2,k+1}^{n} \\ + h_{\phi\rho}|_{i+1/2,k+1}^{n} - h_{\phi z}|_{i+1/2,k}^{n} - h_{\phi\rho}|_{i+1/2,k+1}^{n}\Big). \end{aligned} \tag{2.108}$$

Equations (2.91)-(2.101) can be discretized in a similar fashion. However, since σ_ϕ goes as $1/\rho$ its value will be too small for exponential time-stepping to be used. In numerical experiments, the use of exponential time-stepping for terms involving σ_ϕ in the BOR FD-TD PML implementation led to numerical instabilities. Thus, standard central difference approximations were used to discretize the PML equations containing σ_ϕ terms. For instance, equation (2.91) is discretized as follows,

$$\begin{aligned} \epsilon\frac{e_{\rho\phi}|_{i+1/2,k+1/2}^{n+1/2} - e_{\rho\phi}|_{i+1/2,k+1/2}^{n-1/2}}{\Delta t} + \sigma_\phi \frac{e_{\rho\phi}|_{i+1/2,k+1/2}^{n+1/2} + e_{\rho\phi}|_{i+1/2,k+1/2}^{n-1/2}}{2} \\ = \frac{m}{(i+1/2)\Delta\rho}\left(h_{z\rho}|_{i+1/2,k+1/2}^{n} + h_{z\phi}|_{i+1/2,k+1/2}^{n}\right). \end{aligned} \tag{2.109}$$

Rearranging terms,

$$\begin{aligned} e_{\rho\phi}|_{i+1/2,k+1/2}^{n+1/2} = \left(\frac{\epsilon/\Delta t - \sigma_\phi/2}{\epsilon/\Delta t + \sigma_\phi/2}\right) e_{\rho\phi}|_{i+1/2,k+1/2}^{n-1/2} - \left(\frac{1}{\epsilon/\Delta t + \sigma_\phi/2}\right)\left(\frac{m}{(i+1/2)\Delta\rho}\right) \\ \left(h_{z\rho}|_{i+1/2,k+1/2}^{n} + h_{z\phi}|_{i+1/2,k+1/2}^{n}\right) \end{aligned} \tag{2.110}$$

The full set of PML equations can be obtained by discretizing the remaining equations in a similar fashion.

$$e_{\phi z}|_{i,k+1/2}^{n+1/2} = e^{-\sigma_z \Delta t/\epsilon} e_{\phi z}|_{i,k+1/2}^{n-1/2} - \frac{e^{-\sigma_z \Delta t/\epsilon} - 1}{\sigma_z \Delta z} \left(h_{\rho z}|_{i,k+1}^{n} + h_{\rho\phi}|_{i,k+1}^{n} - h_{\rho z}|_{i,k}^{n} - h_{\rho\phi}|_{i,k}^{n} \right) \tag{2.111}$$

$$e_{\phi\rho}|_{i,k+1/2}^{n+1/2} = e^{-\sigma_\rho \Delta t/\epsilon} e_{\phi z}|_{i,k+1/2}^{n-1/2} + \frac{e^{-\sigma_\rho \Delta t/\epsilon} - 1}{\sigma_\rho \Delta \rho} \left(h_{z\rho}|_{i+1/2,k+1/2}^{n} + h_{z\phi}|_{i+1/2,k+1/2}^{n} - h_{z\rho}|_{i-1/2,k+1/2}^{n} - h_{z\phi}|_{i-1/2,k+1/2}^{n} \right) \tag{2.112}$$

$$e_{z\rho}|_{i,k}^{n+1/2} = e^{-\sigma_\rho \Delta t/\epsilon} e_{z\rho}|_{i,k}^{n-1/2} - \frac{e^{-\sigma_\rho \Delta t/\epsilon} - 1}{\sigma_\rho \Delta \rho} \left(h_{\phi z}|_{i+1/2,k}^{n} + h_{\phi\rho}|_{i+1/2,k}^{n} - h_{\phi z}|_{i-1/2,k}^{n} - h_{\phi\rho}|_{i-1/2,k}^{n} \right) \tag{2.113}$$

$$e_{z\phi}|_{i,k}^{n+1/2} = \left(\frac{\epsilon/\Delta t - \sigma_\phi/2}{\epsilon/\Delta t + \sigma_\phi/2} \right) e_{\rho\phi}|_{i,k}^{n-1/2} - \left(\frac{1}{\epsilon/\Delta t + \sigma_\phi/2} \right) \left(\frac{1}{i\Delta\rho} \right) \left[m \left(h_{\rho z}|_{i,k}^{n} + h_{\rho\phi}|_{i,k}^{n} \right) - \frac{1}{2} \left(h_{\phi z}|_{i+1/2,k}^{n} + h_{\phi\rho}|_{i+1/2,k}^{n} + h_{\phi z}|_{i-1/2,k}^{n} + h_{\phi\rho}|_{i-1/2,k}^{n} \right) \right] \tag{2.114}$$

$$h_{\rho z}|_{i,k}^{n+1} = e^{-\sigma_z^* \Delta t/\mu} h_{\rho z}|_{i,k}^{n} - \frac{e^{-\sigma_z^* \Delta t/\mu} - 1}{\sigma_z^* \Delta z} \left(e_{\phi z}|_{i,k+1/2}^{n+1/2} + e_{\phi\rho}|_{i,k+1/2}^{n+1/2} - e_{\phi z}|_{i,k-1/2}^{n+1/2} - e_{\phi\rho}|_{i,k-1/2}^{n+1/2} \right) \tag{2.115}$$

$$h_{\rho\phi}|_{i,k}^{n+1} = \left(\frac{\epsilon/\Delta t - \sigma_\phi^*/2}{\epsilon/\Delta t + \sigma_\phi^*/2} \right) h_{\rho\phi}|_{i,k}^{n} + \left(\frac{1}{\epsilon/\Delta t + \sigma_\phi^*/2} \right) \left(\frac{m}{i\Delta\rho} \right) \left(e_{z\rho}|_{i,k}^{n+1/2} + e_{z\phi}|_{i,k}^{n+1/2} \right) \tag{2.116}$$

$$h_{\phi z}|_{i+1/2,k}^{n+1} = e^{-\sigma_z^* \Delta t/\mu} h_{\phi z}|_{i+1/2,k}^{n} + \frac{e^{-\sigma_z^* \Delta t/\mu} - 1}{\sigma_z^* \Delta z} \left(e_{\rho z}|_{i+1/2,k+1/2}^{n+1/2} + e_{\rho\phi}|_{i+1/2,k+1/2}^{n+1/2} - e_{\rho z}|_{i+1/2,k-1/2}^{n+1/2} - e_{\rho\phi}|_{i+1/2,k-1/2}^{n+1/2} \right) \tag{2.117}$$

$$h_{\phi\rho}|_{i+1/2,k}^{n+1} = e^{-\sigma_\rho^* \Delta t/\mu} h_{\phi z}|_{i,k+1/2}^{n} - \frac{e^{-\sigma_\rho^* \Delta t/\mu} - 1}{\sigma_\rho^* \Delta \rho} \left(e_{z\rho}|_{i+1,k}^{n+1/2} + e_{z\phi}|_{i+1,k}^{n+1/2} - e_{z\rho}|_{i,k}^{n+1/2} - e_{z\phi}|_{i,k}^{n+1/2} \right) \tag{2.118}$$

$$h_{z\rho}|_{i+1/2,k+1/2}^{n+1} = e^{-\sigma_\rho^* \Delta t/\mu} h_{z\rho}|_{i+1/2,k+1/2}^{n} + \frac{e^{-\sigma_\rho^* \Delta t/\mu} - 1}{\sigma_\rho^* \Delta\rho} \left(e_{\phi z}|_{i+1,k+1/2}^{n+1/2} + e_{\phi\rho}|_{i+1,k+1/2}^{n+1/2} - e_{\phi z}|_{i,k+1/2}^{n+1/2} - e_{\phi\rho}|_{i,k+1/2}^{n+1/2} \right) \tag{2.119}$$

$$h_{z\phi}|_{i+1/2,k+1/2}^{n+1} = \left(\frac{\epsilon/\Delta t - \sigma_\phi^*/2}{\epsilon/\Delta t + \sigma_\phi^*/2} \right) h_{\rho\phi}|_{i+1/2,k+1/2}^{n} - \left(\frac{1}{\epsilon/\Delta t + \sigma_\phi^*/2} \right) \left(\frac{1}{(i+1/2)\Delta\rho} \right) \left[m \left(e_{\rho z}|_{i+1/2,k+1/2}^{n+1/2} + e_{\rho\phi}|_{i+1/2,k+1/2}^{n+1/2} \right) + \frac{1}{2} \left(e_{\phi\rho}|_{i+1,k+1/2}^{n+1/2} + e_{\phi z}|_{i+1,k+1/2}^{n+1/2} + e_{\phi\rho}|_{i,k+1/2}^{n+1/2} + e_{\phi z}|_{i,k+1/2}^{n+1/2} \right) \right] \tag{2.120}$$

In the limit of a vanishingly small grid size, the loss factor can be chosen to be arbitrarily large and an arbitrarily thin PML layer can be used. However, in implementing the PML technique with the FD-TD technique, the discretized nature of the electric and magnetic fields must be considered. Thus, in order to reduce the amount of spurious reflections due to discretization, it is desirable to chose the PML region to be 8-15 cells thick, and to gradually increase the conductivity from zero to some maximum. One such conductivity profile proposed by Berenger is a polynomial curve,

$$\sigma_\zeta(\zeta) = \sigma_{\max} \left[\frac{\zeta}{\delta} \right]^n \tag{2.121}$$

where δ is the total thickness of the PML region. In practice, the choice of a quadratic profile, $n = 2$, has been found to work well. Note that, the exact position of the electric and magnetic fields on the grid should be used when computing σ.

2.3 Source Implementation

In beginning the FD-TD computation all the fields inside the computational domain are initialized to zero. Quantities are then added to simulate an excitation. For example, current sources may be introduced by adding a current density term, J, to the discretized Maxwell's equations, where the current source is discretized in the manner similar to that described above.

For RCS calculation, a plane wave excitation is typically required. This excitation is often implemented by dividing the computational domain into scattered field and total field regions, as shown in Figure 2-4. The incident field is included in calculated fields only inside of the total

field region, and is used as the excitation source. The scattered field is defined as

$$E_{\text{scat}} = E_{\text{total}} - E_{\text{inc}} \tag{2.122}$$

where E_{inc} is the incident field and E_{total} is the total field. The FD-TD equations for the cells at the interface of the total field and scattered field regions must account for the difference in the definition of the calculated fields which occurs at this boundary. For example, when computing a field in the total field region, if a field quantity in scattered field region is required, the incident field must first be added to it to produce a total field quantity.

The incident field is calculated using an analytic expression for the plane wave source. Since the FD-TD method is formulated in the time domain, a Gaussian pulse excitation is used so that multiple frequencies can be analyzed at once. The Gaussian pulse is often modulated near a center frequency so that the incident wave's power is concentrated at frequencies of interest, avoiding numerical errors due to numerical quantization which might occur if other frequency components were significantly larger. Field quantities can then be Fourier transformed to extract fields of a particular frequency

For the body of revolution geometry, the general form of the incident electric field, and corresponding magnetic field can be written in terms of horizontal and vertical polarization components,

$$\vec{E}_i = \left(E_h\hat{h} + E_v\hat{v}\right) P\left(t - \frac{\hat{k}_i \cdot \hat{r}}{c}\right) \tag{2.123}$$

$$\vec{H}_i = \frac{1}{\eta}\hat{k}_i \times \vec{E} = \frac{1}{\eta}\left(-E_h\hat{v} + E_v\hat{h}\right) P\left(t - \frac{\hat{k}_i \cdot \hat{r}}{c}\right) \tag{2.124}$$

$$\hat{r} = x\hat{x} + y\hat{y} + z\hat{z} \tag{2.125}$$

$$\hat{k}_i = -\hat{x}\sin\theta_i - \hat{z}\cos\theta_i \tag{2.126}$$

$$\hat{k}_i \cdot \hat{r} = -x\sin\theta_i - z\cos\theta_i = -\rho\cos\phi\sin\theta_i - z\cos\theta_i \tag{2.127}$$

$$\hat{h} = \hat{x}\cos\theta_i - \hat{z}\sin\theta_i = \hat{\rho}\cos\theta_i\cos\phi - \hat{\phi}\cos\theta_i\sin\phi - \hat{z}\sin\theta_i \tag{2.128}$$

$$\hat{v} = \hat{y} = \hat{\phi}\cos\phi + \hat{\rho}\sin\phi. \tag{2.129}$$

The function P is a modulated Gaussian pulse defined as,

$$P(\tau) = e^{-\tau^2/2\sigma}\sin(2\pi f\tau) \tag{2.130}$$

where the parameter σ defines the pulse width and f is the modulation frequency. Since the BOR formulation represents the ϕ dependence with Fourier modes, the incident fields must be decomposed into these Fourier components. For example, the $e^{\rho}_{m,u}$ component of the incident field is determined by,

$$e^{\rho}_{0,u} = \frac{1}{2\pi}\int_0^{2\pi}\left(E_h\cos\theta_i\cos\phi + E_v\sin\phi\right)P\left(t-\frac{\hat{k}_i\cdot\hat{r}}{c}\right)d\phi \tag{2.131}$$

$$e^{\rho}_{m,u} = \frac{1}{\pi}\int_0^{2\pi}\left(E_h\cos\theta_i\cos\phi + E_v\sin\phi\right)P\left(t-\frac{\hat{k}_i\cdot\hat{r}}{c}\right)\cos m\phi d\phi \tag{2.132}$$

In practice, these integrals are computed numerically using a Gaussian quadrature technique.

If the incident wave is propagating along the axis of symmetry in the $\pm\hat{z}$ direction the electric field has the special form,

$$\vec{E}_i = \left[\mp E_h(\hat{\rho}\cos\phi + \hat{\phi}\sin\phi) + E_v(\hat{\rho}\sin\phi + \hat{\phi}\cos\phi)\right].P\left(t \mp z/c\right) \tag{2.133}$$

For this case only the fields associated with the $m = 1$ mode are nonzero, since the argument to the function P does not have ϕ dependence. In general, however, for a wave incident off-axis, higher-order modes are present and the contribution of the incident field to each needs to be determined. Since P is an even function of ϕ, it can be expanded into a cosine Fourier series.

$$P\left(t+\frac{\rho\cos\phi\sin\theta_i + z\cos\theta_i}{c}\right) = a_0 + a_1\cos\phi + a_2\cos 2\phi + a_3\cos 3\phi + \cdots \tag{2.134}$$

In computing the Fourier components as in (2.131) and (2.132), six different types of ϕ integrals are encountered.

$$I_1 = \int_0^{2\pi} P\left(t+\frac{\rho\cos\phi\sin\theta_i + z\cos\theta_i}{c}\right)\cos m\phi\sin\phi d\phi \tag{2.135}$$

$$I_2 = \int_0^{2\pi} P\left(t+\frac{\rho\cos\phi\sin\theta_i + z\cos\theta_i}{c}\right)\cos m\phi\cos\phi d\phi \tag{2.136}$$

$$I_3 = \int_0^{2\pi} P\left(t+\frac{\rho\cos\phi\sin\theta_i + z\cos\theta_i}{c}\right)\cos m\phi d\phi \tag{2.137}$$

$$I_4 = \int_0^{2\pi} P\left(t+\frac{\rho\cos\phi\sin\theta_i + z\cos\theta_i}{c}\right)\sin m\phi\sin\phi d\phi \tag{2.138}$$

$$I_5 = \int_0^{2\pi} P\left(t+\frac{\rho\cos\phi\sin\theta_i + z\cos\theta_i}{c}\right)\sin m\phi\cos\phi d\phi \tag{2.139}$$

$$I_6 = \int_0^{2\pi} P\left(t+\frac{\rho\cos\phi\sin\theta_i + z\cos\theta_i}{c}\right)\sin m\phi d\phi \tag{2.140}$$

By orthogonality, the integrals I_1, I_5, and I_6 are identically zero for all modes. The remaining types of integrals, I_2, I_3, and I_4 are nonzero and contribute to Fourier components of the incident field. Thus, for an incident wave that only has either a horizontal or vertical polarization component, only six of the twelve Fourier field components will be nonzero. For a horizontally polarized incident wave, the nonzero Fourier field components correspond to the fields contained in equations (2.13)-(2.18). Since the other six Fourier field components are zero for a horizontally polarized incident wave, the second set of equations, (2.19)-(2.24) is not needed. Similarly, for a vertically polarized incident wave only the second set of equations is needed since the field components in the first set are zero for all modes.

2.4 Near to Far Field Transformation

As evidenced by equation (1.1), calculation of the RCS requires knowledge of scattered fields in the far field. Using the near fields calculated by the BOR FD-TD method, the far fields can be obtained by performing a near to far field transformation, formulated using Huygens' principle. This principle determines the electric and magnetic fields outside a region containing excitation sources in terms of the tangential electric and magnetic fields on a surface, S', which encloses the sources. The mathematical formulation of Huygens' principle for free space in three-dimensions, assuming an $e^{-i\omega t}$ time dependence, has the following forms [32],

$$\vec{E}(\vec{r}) = \oint_{S'} dS' \left\{ i\omega\mu\overline{\overline{G}}(\vec{r},\vec{r}') \cdot \hat{n} \times \vec{H}(\vec{r}') + \nabla \times \overline{\overline{G}}(\vec{r},\vec{r}') \cdot \hat{n} \times \vec{E}(\vec{r}') \right\} \tag{2.141}$$

$$\vec{H}(\vec{r}) = \oint_{S'} dS' \left\{ -i\omega\epsilon\overline{\overline{G}}(\vec{r},\vec{r}') \cdot \hat{n} \times \vec{E}(\vec{r}') + \nabla \times \overline{\overline{G}}(\vec{r},\vec{r}') \cdot \hat{n} \times \vec{H}(\vec{r}') \right\} \tag{2.142}$$

where $\overline{\overline{G}}(\vec{r},\vec{r}')$ is the dyadic Green's function given by

$$\overline{\overline{G}}(\vec{r},\vec{r}') = \left[\overline{\overline{I}} + \frac{1}{k^2}\nabla\nabla \right] \frac{e^{ik|\vec{r}-\vec{r}'|}}{4\pi|\vec{r}-\vec{r}'|} \tag{2.143}$$

and $\hat{n}$ is the outward normal to the surface. In the far field, ∇ can be approximated as $ik\hat{r}$ [39], and $[\overline{\overline{I}} - \nabla\nabla]$ becomes $[\hat{\theta}\hat{\theta} + \hat{\phi}\hat{\phi}]$. The electric field in the far field can then be written as

$$\vec{E}(\vec{r}) = \oint_{S'} dS' \left\{ i\omega\mu[\hat{\theta}\hat{\theta} + \hat{\phi}\hat{\phi}] \cdot \hat{n} \times \vec{H}(\vec{r}') + ik[\hat{\phi}\theta - \hat{\theta}\phi] \cdot \hat{n} \times \vec{E}(\vec{r}') \right\} \frac{e^{ik|\vec{r}-\vec{r}'|}}{4\pi|\vec{r}-\vec{r}'|}. \tag{2.144}$$

Furthermore, in the far field, the magnitude of $|\vec{r} - \vec{r}'|$ can be approximated by $|\vec{r}|$, while the phase term $e^{ik|\vec{r}-\vec{r}'|}$ can be represented by a linear phase approximation resulting in

$$\frac{e^{ik|\vec{r}-\vec{r}'|}}{4\pi|\vec{r} - \vec{r}'|} = \frac{e^{ikr}e^{-ik\hat{r}\cdot\vec{r}'}}{4\pi r}. \tag{2.145}$$

In the cylindrical coordinate system used in the BOR FD-TD approach, Huygens' principle is most easily formulated by choosing S' as a cylinder, and storing the fields on this cylindrical boundary. Since not every field in the grid will lie exactly on the cylinder, an interpolated value is calculated by averaging the nearest available field components. In addition, since a frequency domain Huygens' principle formulation is used, a temporal Fourier transform of the field components at each spatial point included in the integration must be computed.

2.5 Results

This section compares the RCS predictions of the BOR FD-TD method to those of exact and MoM techniques. The two geometries considered here are a cylinder and a biconical shaped target.

2.5.1 Bistatic RCS of a Circular Cylinder

The first object modeled is a cylinder whose geometry can easily be represented on the FD-TD lattice. In the following, the bistatic HH RCS at 1 GHz is calculated for a cylinder illuminated at its end-cap, as shown in Figure 2-7.

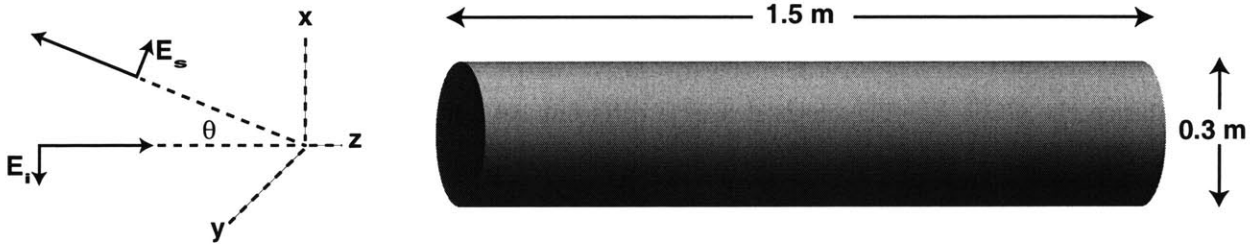

Figure 2-7: Geometry for Cylinder, $\theta_i = 0°$

Since the incident electric field propagates along the z axis, only one Fourier mode is computed. As evidenced in Figure 2-8, the HH bistatic RCS predictions of the BOR FD-TD method

compare well with the BOR MoM predictions. In practice, it has been found that smaller step size, on the order of $\lambda/40$, are needed to lessen the effects of numerical dispersion for incident directions near the axis are needed to lessen the effects of numerical dispersion.

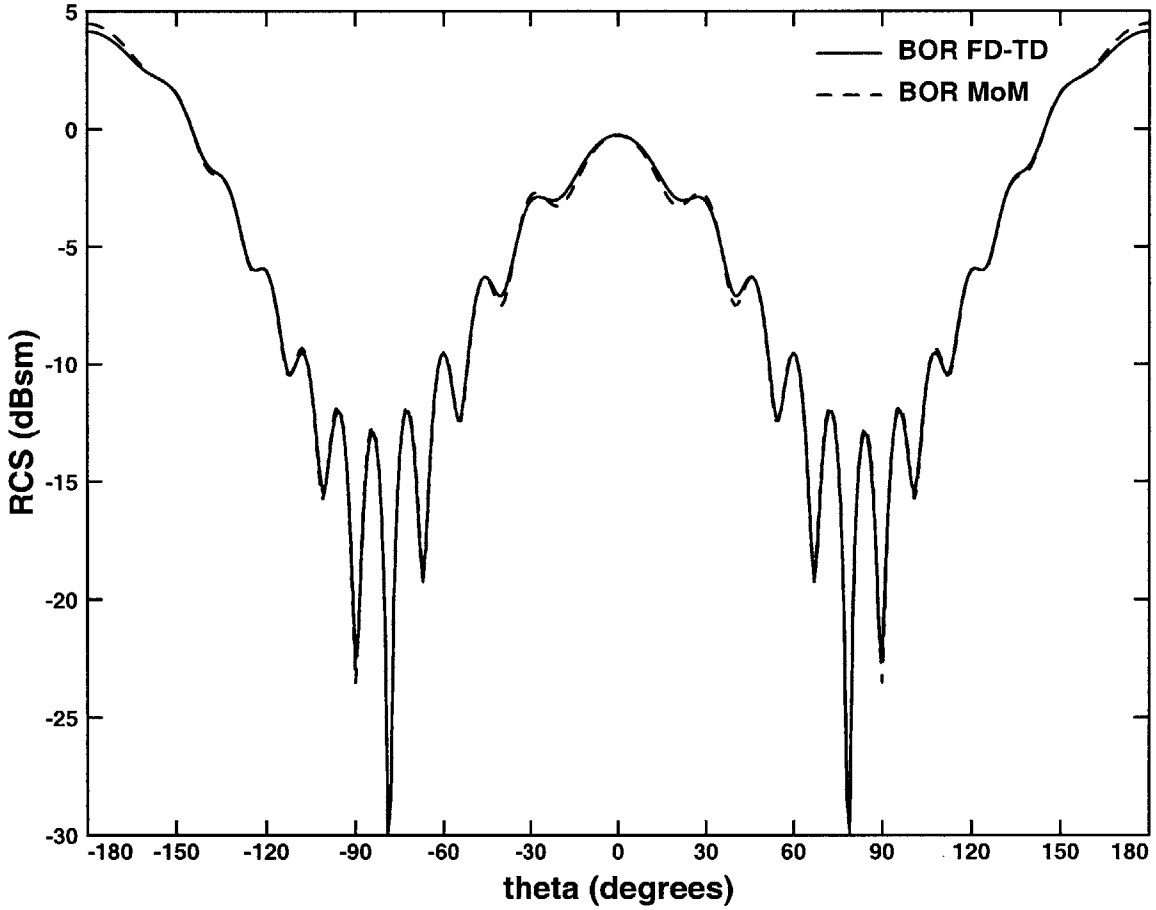

Figure 2-8: Bistatic RCS at 1 GHz of a cylinder illuminated at normal incidence. Shown is the HH polarization for a cut in θ with $\phi = 0°$.

Since the BOR FD-TD method uses a different set of equations for the prediction of the HH and VV radar cross sections, it is necessary to test both cases in order to validate the code. As shown in Figure 2-9, the BOR FD-TD prediction for the VV bistatic RCS is also in good agreement with the BOR MoM prediction providing further validation of the BOR FD-TD code.

In the two cases considered above, the cylinder is illuminated normal to its end-cap, so that only one Fourier mode is required. In the general case of off-axis incidence, the contributions of multiple modes must be considered. For example, if the cylinder is illuminated at 45°, as

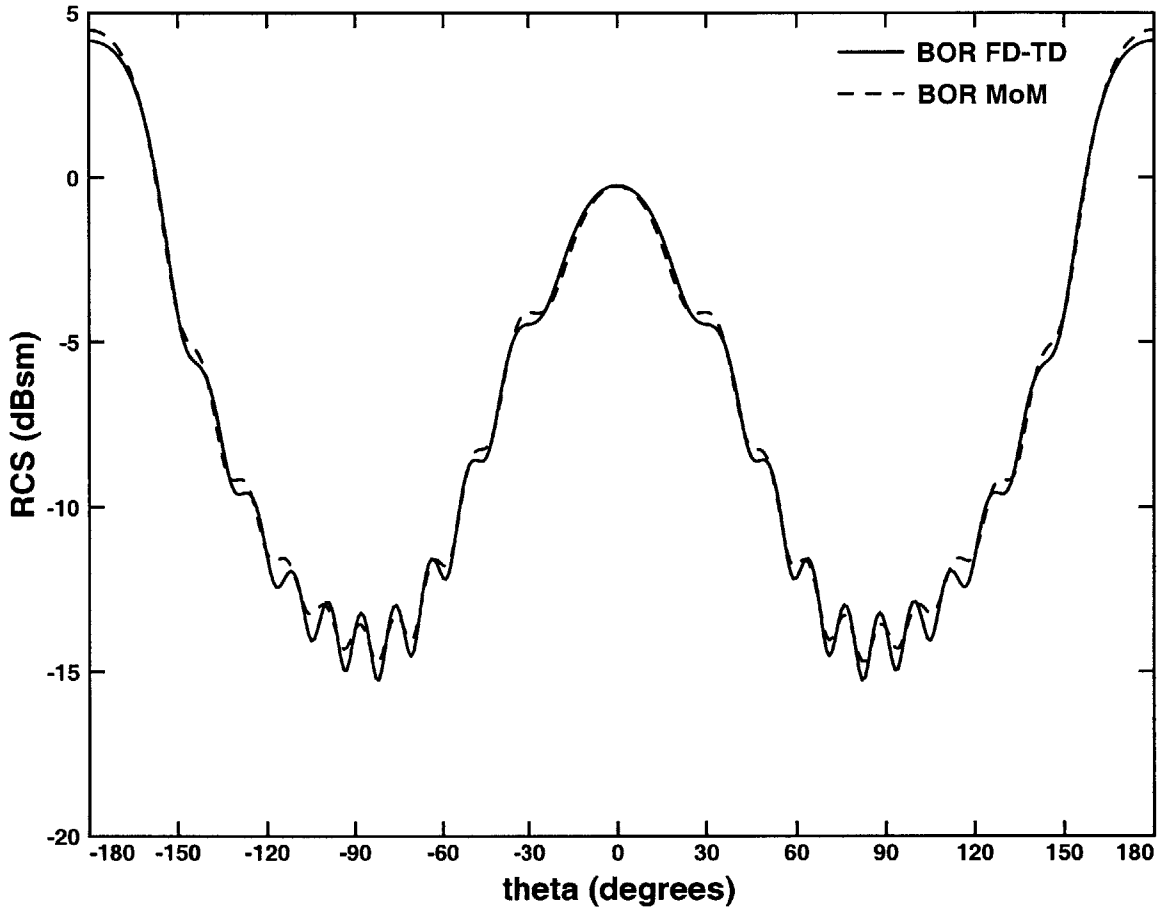

Figure 2-9: Bistatic RCS at 1 GHz of a cylinder illuminated at normal incidence. Shown is the VV polarization for a cut in θ with $\phi = 0°$.

shown in Figure 2-10, a two dimensional problem must be solved for each of the contributing modes.

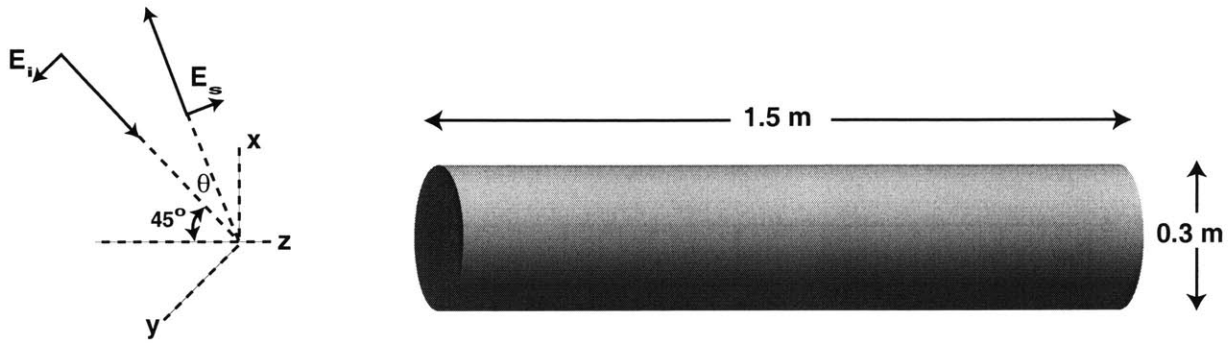

Figure 2-10: Geometry for Cylinder, $\theta_i = 45°$

Following the rule described in Section 2.1.1, the contribution of modes $m = 0$ through $m = 5$ are used to compute the bistatic RCS of the cylinder which is illuminated at 45°. As shown in Figure 2-11, the BOR FD-TD and BOR MoM predictions compare well validating the ability of the code to combine contributions from multiple modes.

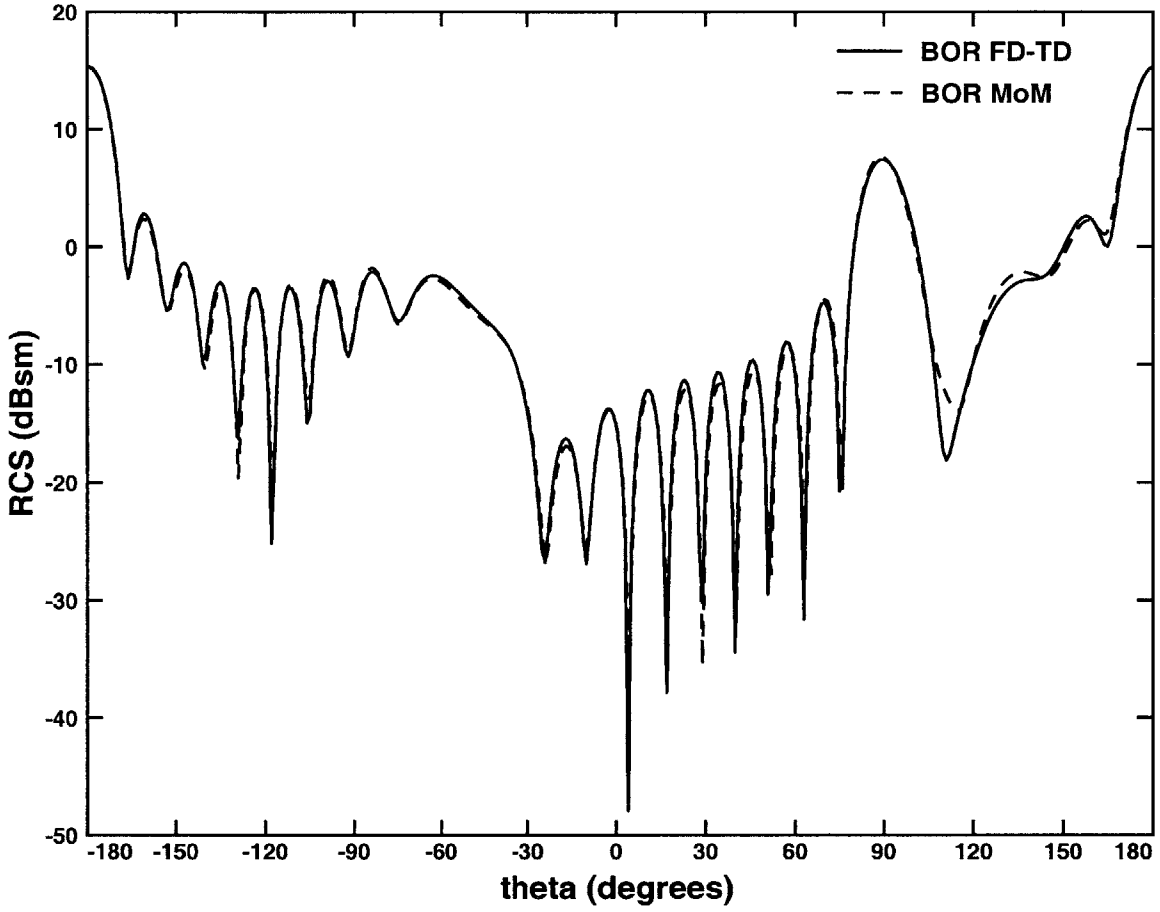

Figure 2-11: Bistatic RCS at 1 GHz of a cylinder illuminated at $\theta_i = 45°$. Shown is the HH polarization for a cut in θ with $\phi = 0°$.

2.5.2 RCS of Biconical Object

In order to validate the BOR FD-TD on a more realistic target, a biconical object similar to the geometry of a re-entry vehicle is modeled. Since the structure of the target does not align with the BOR FD-TD lattice, it must be approximated using a staircase representation. In the following, the biconical object is illuminated normal to its broadside, $\theta_i nc = 79°$, with a horizontally polarized incident wave, as shown in Figure 2-12.

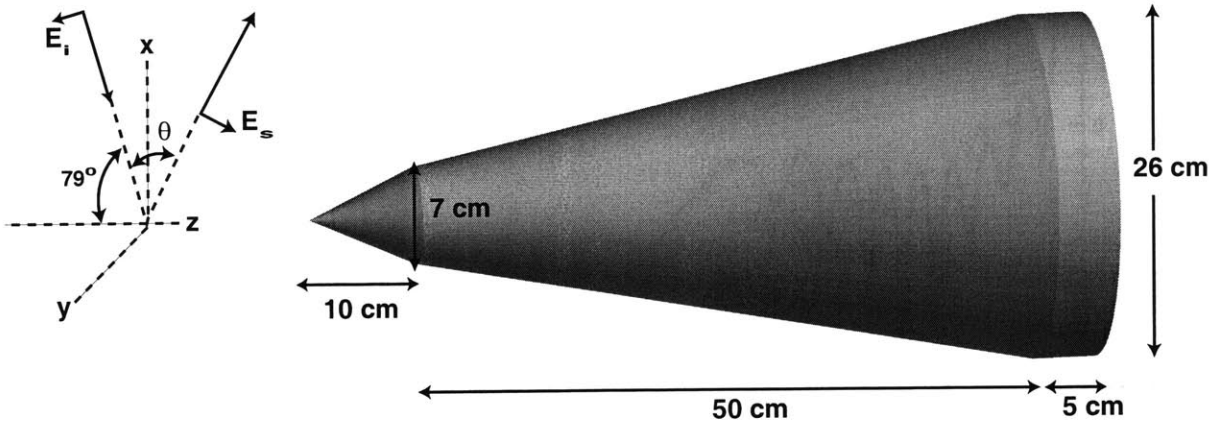

Figure 2-12: Geometry for Biconical Object, $\theta_i = 79°$

As shown in Figure 2-13, both the MoM and BOR FD-TD results are in good agreement showing the strong backscatter return at $\theta = 0°$. The small disagreements between the BOR MoM and BOR FD-TD predictions are likely due to discretization errors since each method represents the actual target geometry differently.

As discussed previously, one of the advantages of the FD-TD method is that predictions over an extended bandwidth can be obtained from a single simulation. Figure 2-14 compares BOR FD-TD and BOR MoM results of the backscatter RCS versus frequency, for a wave incident on the broadside of the biconical object at $\theta_i = 79°$. Again, the BOR FD-TD and BOR MoM predictions are in good agreement, and as expected the backscatter RCS increases with increasing frequency as the broadside of the biconical object becomes larger in terms of wavelength.

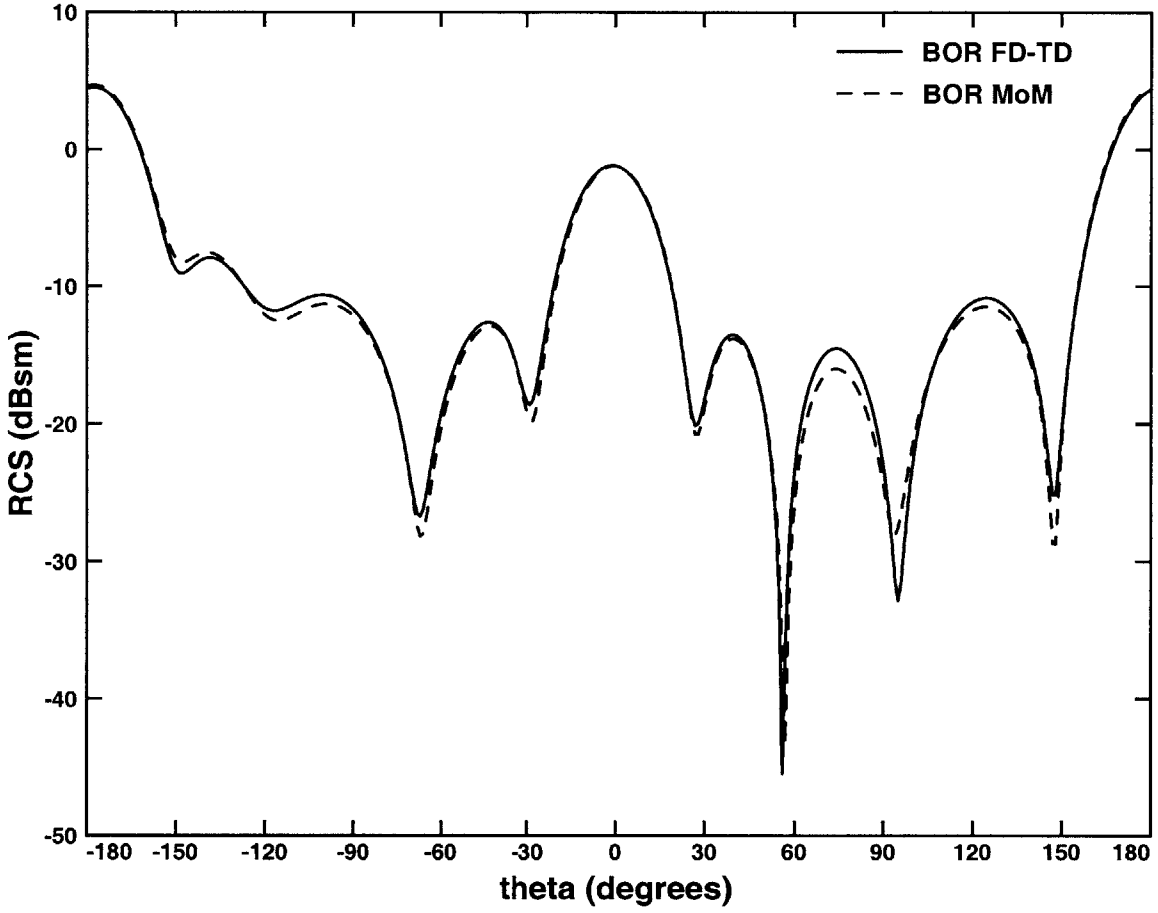

Figure 2-13: Bistatic RCS at 1 GHz of a biconical object illuminated at $\theta_i = 79°$. Shown is the HH polarization for a cut in θ with $\phi = 0°$.

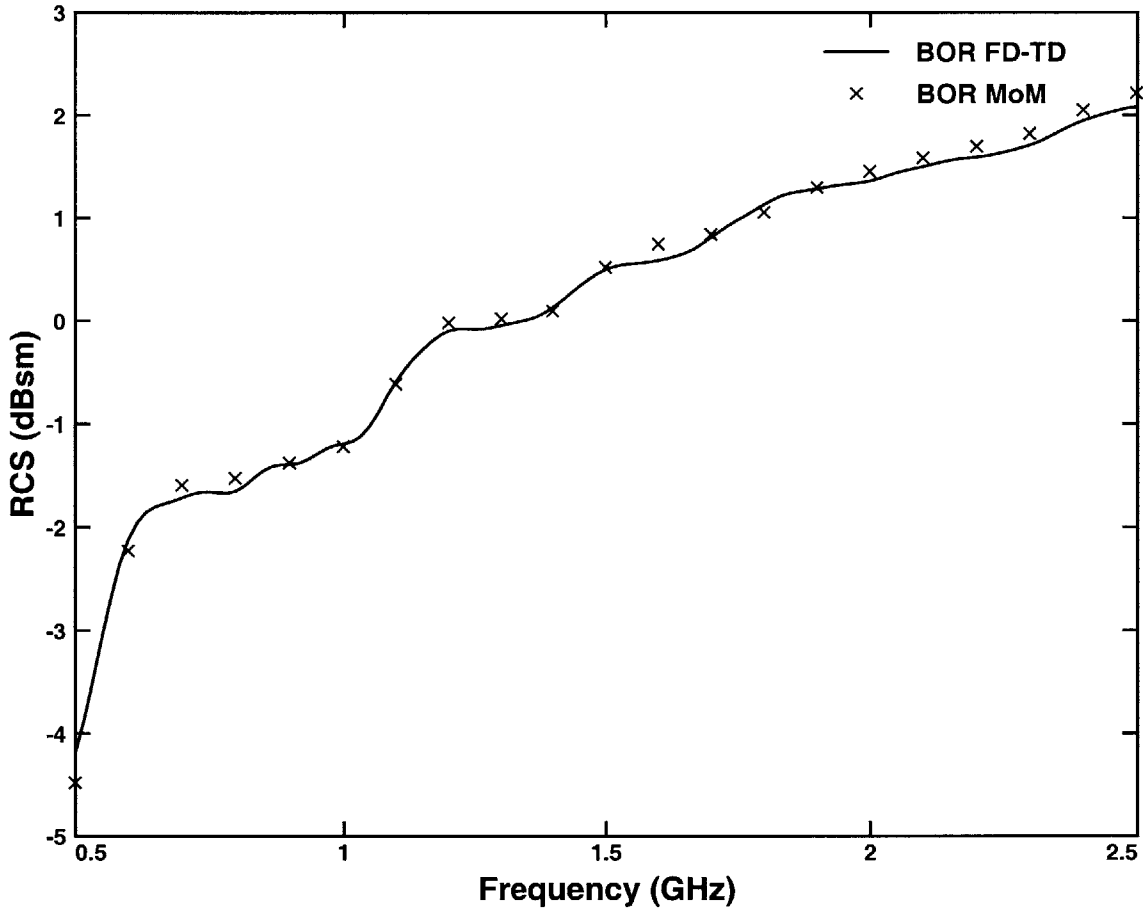

Figure 2-14: Backscatter RCS of a biconical object illuminated at $\theta = 79°$, $\phi = 0°$ for a cut in frequency.

2.6 Summary

In this chapter, a finite difference-time domain algorithm was presented for the modeling of objects with body of revolution symmetry. The BOR FD-TD formulation exploits the rotational symmetry of the problem by expressing the azimuthal (ϕ) dependence of the fields in a Fourier series. Maxwell's equations are rewritten in terms of these ϕ independent field components yielding the modal form of Maxwell's equations. Since the azimuthal variation is accounted for analytically, there is no ϕ gridding, which results in an algorithm that is two-dimensional in terms of computer memory usage. Maxwell's modal equations are discretized using a central difference approximation, and are placed on a two-dimensional computational grid. On this grid, each field is calculated from previous values of the neighboring fields and previous values of itself. Perfect electric conductors are modeled by setting the tangential electric fields to zero. Plane wave sources are implemented by creating a region of scattered fields and a region of total fields, and adding in the incident wave as it crosses into the total field region. A PML absorbing boundary condition is used to truncate the computational domain and absorb scattered energy incident on the boundary of the domain.

The BOR FD-TD method was applied to the prediction of the RCS of various BOR targets. Each of the perfectly conducting targets is represented in the computational domain by using a staircase model. In order to determine the RCS, the target is illuminated with a plane wave, and Fourier transformed electric and magnetic fields on a Huygens' surface are stored. Huygens' principle is applied to find the far field scattered fields, from which the RCS is determined. For both targets modeled, the cylinder and biconical shaped object, the RCS predictions of the BOR FD-TD method were found to be in good agreement with those of the BOR MoM method.

Chapter 3

The Conformal BOR FD-TD

In the previous chapter, the BOR FD-TD method was developed for modeling objects with axial symmetry. The modeling of surfaces which do not lie along grid lines was accomplished by creating a "staircase" approximation of the object aligned with the grid. Accuracy of this approximation depends on the size of grid cells used, with higher accuracy possible with smaller cell sizes, at the expense of an increase in the number of unknowns. More accuracy, however, may be obtained without increasing the computational overhead, by using a conformal gridding FD-TD approach [25, 26]. The conformal gridding FD-TD algorithm works by deforming the grid cells along the boundary of the object being modeled to fit the surface of that object. Contour integrals are evaluated to determine alternate finite-difference equations valid for the new deformed cells. In Section 3.1, the modified finite-difference equations will be developed, and the results of using the conformal gridding technique will be compared to the results obtained with the staircase method.

3.1 The Conformal BOR FD-TD Algorithm

While the normal BOR FD-TD difference equations can be derived from either the differential or integral form of Maxwell's equations, the modified difference equations for the conformal method are most easily derived from the integral form of Maxwell's equations,

$$\oint_C \vec{E} \cdot d\vec{l} = -\mu \frac{\partial}{\partial t} \iint_S \vec{H} \cdot d\vec{S} \tag{3.1}$$

$$\oint_C \vec{H} \cdot d\vec{l} = \epsilon \frac{\partial}{\partial t} \iint_S \vec{E} \cdot d\vec{S} \tag{3.2}$$

where the contour C encloses the surface patch S.

3.1.1 The h_ρ Surface-Conformal Patch Integral

The conformal BOR FD-TD difference equation for the h_ρ can be derived by carrying out the contour integral shown in Figure 3-1. The integration of Faraday's Law around the patch results

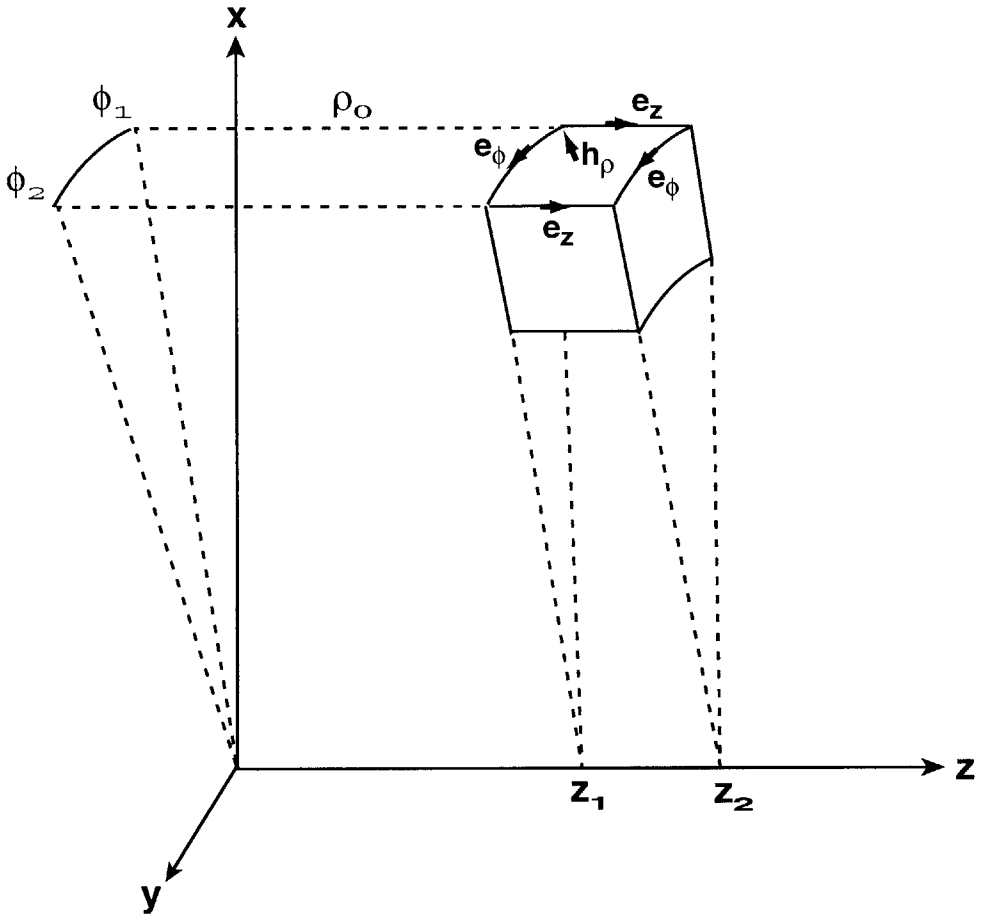

Figure 3-1: Faraday's Law contour for h_ρ

in the following equation,

$$\begin{aligned}
\mu\frac{\partial}{\partial t}\int_{\phi_1}^{\phi_2}\int_{z_1}^{z_2}[h_{\rho,u}(\rho_0,zz,t)\cos m\phi + h_{\rho,v}(\rho_0,zz,t)\sin m\phi]\,\rho_0\;dz\;d\phi \\
= \int_{z_1}^{z_2}[e_{z,u}(\rho_0,zz,t)\cos m\phi_2 + e_{z,v}(\rho_0,zz,t)\sin m\phi_2]\,dz \\
+ \int_{\phi_2}^{\phi_1}[e_{\phi,u}(\rho_0,z_2,t)\cos m\phi + e_{\phi,v}(\rho_0,z_2,t)\sin m\phi]\,\rho_0\;d\phi
\end{aligned}$$

$$+\int_{z_2}^{z_1}\left[e_{z,u}(\rho_0,zz,t)\cos m\phi_1+e_{z,v}(\rho_0,zz,t)\sin m\phi_1\right]dz$$
$$+\int_{\phi_1}^{\phi_2}\left[e_{\phi,u}(\rho_0,z_1,t)\cos m\phi+e_{\phi,v}(\rho_0,z_1,t)\sin m\phi\right]\rho_0\;d\phi \tag{3.3}$$

where $\rho_0 = i\Delta\rho$, $zz = k\Delta z$, and for a normal cell, $z_1 = (k-1/2)\Delta z$ and $z_2 = (k+1/2)\Delta z$. However, near a PEC is advantageous to choose a surface, S, that conforms to the PEC. Since the object is a body of revolution, the only modification to the cell shown in Figure 3-1 will be the position of z_1 and z_2. If the PEC object intersects a cell such that the position of the h_ρ field is inside the object, then the cell to the immediate left or right is extended to conform to the shape of the object. If the PEC object intersects the cell such that the h_ρ field is not inside the object then that cell is modified to conform to the shape of the object. In either case, $z_2 - z_1 = l_0\Delta z$ where l_0 is the length of the new cell in the z direction. Performing the integration in (3.3) yields,

$$\begin{aligned}
&-\mu\frac{\rho_0(z_2-z_1)}{m}\frac{\partial}{\partial t}\left[h_{\rho,u}(\rho_0,zz,t)\left(\sin m\phi_2-\sin m\phi_1\right)-h_{\rho,v}(\rho_0,zz,t)\left(\cos m\phi_2-\cos m\phi_1\right)\right]\\
&\quad=(z_2-z_1)\left[e_{z,u}(\rho_0,zz,t)\cos m\phi_2+e_{z,v}(\rho_0,zz,t)\sin m\phi_2\right]\\
&\quad+\frac{\rho_0}{m}\left[e_{\phi,u}(\rho_0,z_2,t)\left(\sin m\phi_1-\sin m\phi_2\right)-e_{\phi,v}(\rho,z_2,t)\left(\cos m\phi_1-\cos m\phi_2\right)\right]\\
&\quad+(z_1-z_2)\left[e_{z,u}(\rho_0,zz,t)\cos m\phi_1+e_{z,v}(\rho_0,zz,t)\sin m\phi_1\right]\\
&\quad+\frac{\rho_0}{m}\left[e_{\phi,u}(\rho_0,z_1,t)\left(\sin m\phi_2-\sin m\phi_1\right)-e_{\phi,v}(\rho,z_1,t)\left(\cos m\phi_2-\cos m\phi_1\right)\right].
\end{aligned}\tag{3.4}$$

Next, the sine and cosine terms can be separated to yield four equations where two of the four equations are redundant and can be discarded.

$$\begin{aligned}
&\left[-\mu\frac{\rho_0(z_2-z_1)}{m}\frac{\partial}{\partial t}h_{\rho,u}(\rho_0,zz,t)\right]\sin m\phi_2\\
&\quad=\left[(z_2-z_1)e_{z,v}(\rho_0,zz,t)+\frac{\rho_0}{m}e_{\phi,u}(\rho_0,z_1,t)-\frac{\rho_0}{m}e_{\phi,u}(\rho_0,z_2,t)\right]\sin m\phi_2
\end{aligned}\tag{3.5}$$

$$\begin{aligned}
&\left[\mu\frac{\rho_0(z_2-z_1)}{m}\frac{\partial}{\partial t}h_{\rho,u}(\rho_0,zz,t)\right]\sin m\phi_1\\
&\quad=\left[(z_1-z_2)e_{z,v}(\rho_0,zz,t)-\frac{\rho_0}{m}e_{\phi,u}(\rho_0,z_1,t)+\frac{\rho_0}{m}e_{\phi,u}(\rho_0,z_2,t)\right]\sin m\phi_1
\end{aligned}\tag{3.6}$$

$$\left[\mu\frac{\rho_0(z_2-z_1)}{m}\frac{\partial}{\partial t}h_{\rho,v}(\rho_0,zz,t)\right]\cos m\phi_2$$

$$= \left[(z_2 - z_1)e_{z,u}(\rho_0, zz, t) - \frac{\rho_0}{m}e_{\phi,v}(\rho_0, z_1, t) + \frac{\rho_0}{m}e_{\phi,v}(\rho_0, z_2, t)\right]\cos m\phi_2 \tag{3.7}$$

$$\left[-\mu\frac{\rho_0(z_2 - z_1)}{m}\frac{\partial}{\partial t}h_{\rho,v}(\rho_0, zz, t)\right]\cos m\phi_1$$
$$= \left[(z_1 - z_2)e_{z,u}(\rho_0, zz, t) + \frac{\rho_0}{m}e_{\phi,v}(\rho_0, z_1, t) - \frac{\rho_0}{m}e_{\phi,v}(\rho_0, z_2, t)\right]\cos m\phi_1 \tag{3.8}$$

Discretizing the time derivative using a central difference approximation and using the same notation as in Chapter 2, yields the following difference equations.

$$h_{\rho,v}|_{i,k}^{n+1} = h_{\rho,v}|_{i,k}^{n} + \frac{\Delta t}{\mu l_0 \Delta z}\left(e_{\phi,v}|_{i,k+1/2}^{n+1/2} - e_{\phi,v}|_{i,k-1/2}^{n+1/2}\right) + \frac{m\Delta t}{\mu i \Delta\rho}e_{z,u}|_{i,k}^{n+1/2} \tag{3.9}$$

$$h_{\rho,u}|_{i,k}^{n+1} = h_{\rho,u}|_{i,k}^{n} + \frac{\Delta t}{\mu l_0 \Delta z}\left(e_{\phi,u}|_{i,k+1/2}^{n+1/2} - e_{\phi,u}|_{i,k-1/2}^{n+1/2}\right) - \frac{m\Delta t}{\mu i \Delta\rho}e_{z,v}|_{i,k}^{n+1/2} \tag{3.10}$$

Note that if $l_0 = 1$, the two difference equations become the normal BOR FD-TD equations. If the PEC intersects the right-hand side of the cell, then the electric field tangential to the object will be the field $e_\phi|_{i,k+1/2}^{n+1/2}$ and it can be set to zero. Similarly, if the PEC intersects the left-hand side of the cell, then the electric field tangential to the object will be the field $e_\phi|_{i,k-1/2}^{n+1/2}$ and it can be set to zero.

3.1.2 The h_ϕ Surface-Conformal Patch Integral

Since the surface, S containing the h_ϕ field lies in the ρz plane, it can be intersected several different ways. If the PEC object intersects the cell such that the h_ϕ field is inside the PEC object, then one of the surrounding cells is extended to conform to the shape of the object. To ensure a reasonably sized conformal cell, the cell to the immediate left or right is extended to the right or left if the slope of the intersecting line is greater than one, and the cell from above is extended downwards if the slope of the intersecting line is less than one. Since the object is assumed to be a closed convex body of revolution, the cell from the bottom is never extended upwards. One possible intersection is represented by the contour integral shown in Figure 3-2. In this case, the PEC object intersects the bottom portion of the surface forming a trapezoidal shaped cell. The integration of Faraday's Law around the patch can be carried out by noting that the electric field tangential to the surface of the PEC is zero so that it does not contribute to the contour integral, $\oint_c \vec{E} \cdot d\vec{l}$.

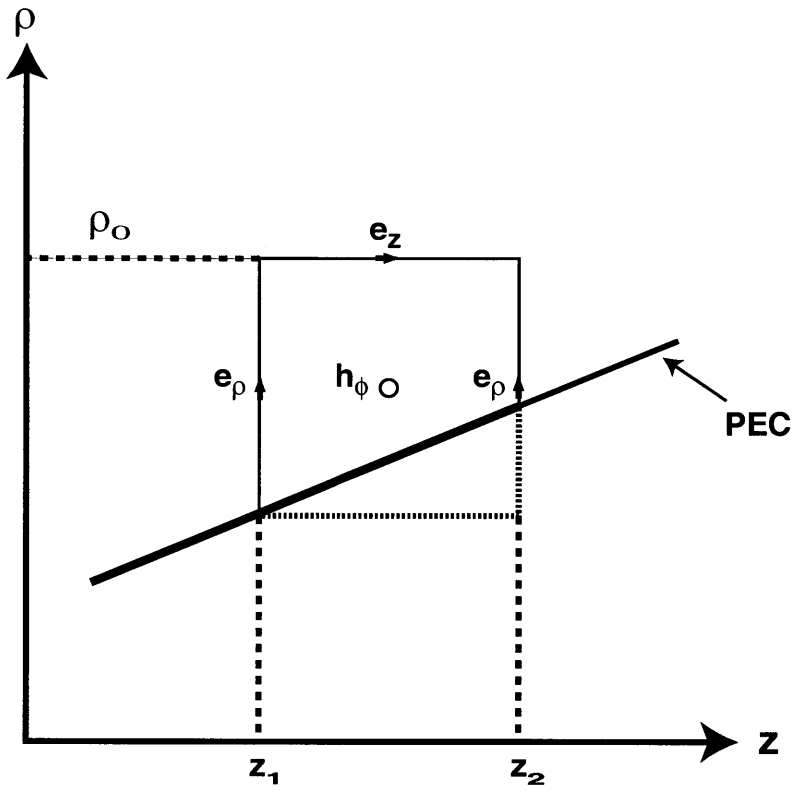

Figure 3-2: Faraday's Law contour for h_ϕ

$$
\begin{aligned}
-\mu\frac{\partial}{\partial t}\int\int_S &[h_{\phi,u}(\rho\rho, zz, t)\cos m\phi + h_{\phi,v}\sin m\phi] \\
&= \int_{z_2}^{z_1} [e_{z,u}(\rho_0, zz, t)\cos m\phi + e_{z,v}(\rho_0, zz, t)\sin m\phi]\, dz \\
&+ \int_{\rho_0}^{\rho_0 - l_1} [e_{\rho,u}(\rho\rho, z_1, t)\cos m\phi + e_{\rho,v}(\rho\rho, z_1, t)\sin m\phi]\, d\rho \\
&+ \int_{\rho_0 - l_2}^{\rho_0} [e_{\rho,u}(\rho\rho, z_2, t)\cos m\phi + e_{\rho,v}(\rho\rho, z_2, t)\sin m\phi]\, d\rho
\end{aligned} \tag{3.11}
$$

where l_1 and l_2 are the lengths of the sides containing the e_ρ field as shown in Figure 3-2, $\rho_0 = (i+1)\Delta\rho$, $\rho\rho = \rho_0 - \Delta\rho/2$, and $zz = z_1 + \Delta z/2$. Carrying out the integration in (3.11) yields,

$$
\begin{aligned}
-\mu A\frac{\partial}{\partial t} &[h_{\phi,u}(\rho\rho, zz, t)\cos m\phi + h_{\phi,v}(\rho\rho, zz, t)\sin m\phi] \\
&= -\Delta z\,[e_{z,u}(\rho_0, zz, t)\cos m\phi + e_{z,v}(\rho_0, zz, t)\sin m\phi] \\
&+ l_2\,[e_{\rho,u}(\rho\rho, z_2, t)\cos m\phi + e_{\rho,v}(\rho\rho, z_2, t)\sin m\phi] \\
&- l_1\,[e_{\rho,u}(\rho\rho, z_1, t)\cos m\phi + e_{\rho,v}(\rho\rho, z_1, t)\sin m\phi]
\end{aligned} \tag{3.12}
$$

where A is the area enclosed by the patch integral. Next, the sine and cosine terms can be separated yielding two independent equations.

$$
\mu A\frac{\partial}{\partial t}h_{\phi,u}(\rho\rho, zz, t) = l_1 e_{\rho,u}(\rho\rho, z_1, t) - l_2 e_{\rho,u}(\rho\rho, z_2, t) + \Delta z e_{z,u}(\rho_0, zz, t) \tag{3.13}
$$

$$
\mu A\frac{\partial}{\partial t}h_{\phi,v}(\rho\rho, zz, t) = l_1 e_{\rho,v}(\rho\rho, z_1, t) - l_2 e_{\rho,v}(\rho\rho, z_2, t) + \Delta z e_{z,v}(\rho_0, zz, t) \tag{3.14}
$$

Discretizing the time derivative as a central difference approximation, and using the same notation as in Chapter 2, the following difference equations result.

$$
\begin{aligned}
h_{\phi,u}|^{n+1}_{i+1/2,k} &= h_{\phi,u}|^{n}_{i+1/2,k} + \frac{\Delta t\Delta z}{\mu A}e_{z,u}|^{n+1/2}_{i+1,k} + \frac{\Delta t}{\mu A}\left(l_1 e_{\rho,u}|^{n+1/2}_{i+1/2,k-1/2}\right. \\
&\qquad \left. - l_2 e_{\rho,u}|^{n+1/2}_{i+1/2,k+1/2}\right)
\end{aligned} \tag{3.15}
$$

$$
\begin{aligned}
h_{\phi,v}|^{n+1}_{i+1/2,k} &= h_{\phi,v}|^{n}_{i+1/2,k} + \frac{\Delta t\Delta z}{\mu A}e_{z,v}|^{n+1/2}_{i+1,k} + \frac{\Delta t}{\mu A}\left(l_1 e_{\rho,v}|^{n+1/2}_{i+1/2,k-1/2}\right. \\
&\qquad \left. - l_2 e_{\rho,v}|^{n+1/2}_{i+1/2,k+1/2}\right)
\end{aligned} \tag{3.16}
$$

Similar conformal difference equations for the h_ϕ field can be derived if the PEC object intersects the cell in a different way by weighting the nonzero electric fields by the lengths of the conformal

cell and dividing by the area of the deformed patch surface.

3.1.3 The h_z Surface-Conformal Patch Integral

The conformal BOR FD-TD difference equation for the h_z can be derived by carrying out the contour integral shown in Figure 3-3. The integration of Faraday's Law around the patch results

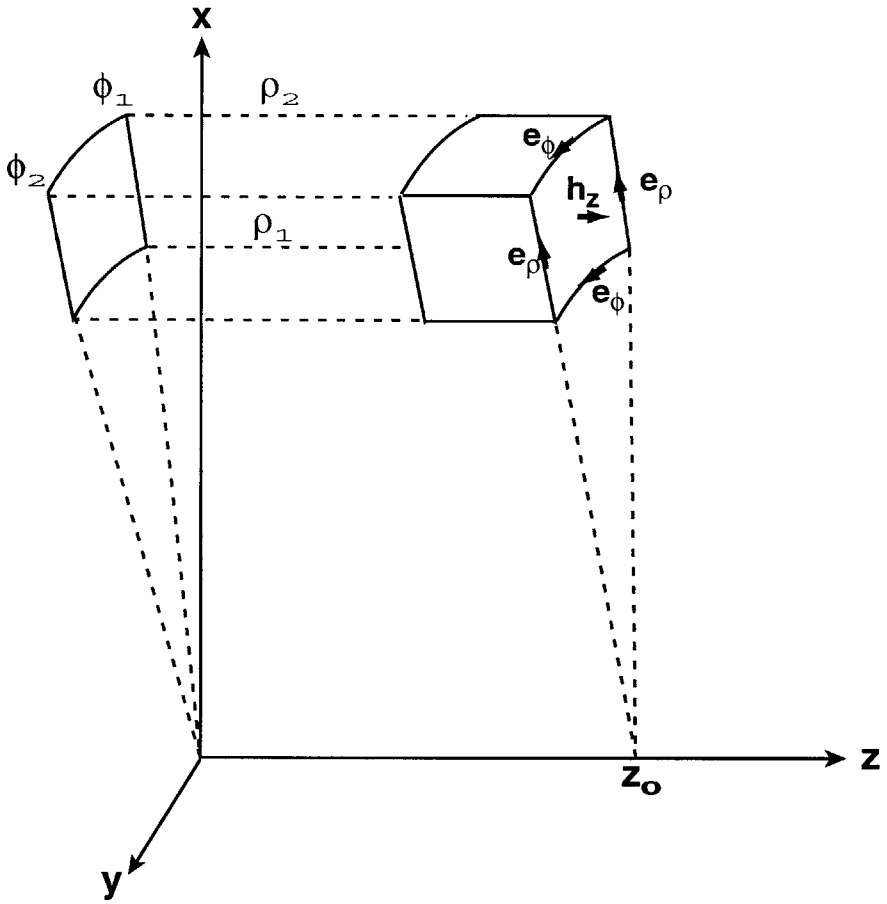

Figure 3-3: Faraday's Law contour for h_z

in the following equation,

$$
\begin{aligned}
-\mu\frac{\partial}{\partial t}\int_{\phi_1}^{\phi_2}\int_{\rho_1}^{\rho_2}[h_{z,u}(\rho\rho,z_0,t)\cos m\phi + h_{z,v}(\rho\rho,z_0,t)\sin m\phi]\,\rho\;d\phi\;d\rho \\
= \int_{\rho_1}^{\rho_2}[e_{\rho,u}(\rho\rho,z_0,t)\cos m\phi_1 + e_{\rho,v}(\rho\rho,z_0,t)\sin m\phi_1]\,d\rho \\
+ \int_{\phi_1}^{\phi_2}[e_{\phi,u}(\rho_2,z_0,t)\cos m\phi + e_{\phi,v}(\rho_2,z_0,t)\sin m\phi]\,\rho_2\;d\phi
\end{aligned}
$$

$$+\int_{\rho_2}^{\rho_1}\left[e_{\rho,u}(\rho\rho,z_0,t)\cos m\phi_2+e_{\rho,v}(\rho\rho,z_0,t)\sin m\phi_2\right]d\rho$$
$$+\int_{\phi_2}^{\phi_1}\left[e_{\phi,u}(\rho_1,z_0,t)\cos m\phi+e_{\phi,v}(\rho_1,z_0,t)\sin m\phi\right]\rho_1\ d\phi \tag{3.17}$$

where $z_0=(k+1/2)\Delta z$ and $\rho\rho=(i+1/2)\Delta\rho$. Again, the surface, S is chosen so that it conforms to the shape of the PEC. Assuming a closed convex body of revolution PEC, the object can only intersect the cell on the bottom portion. Hence, let $\rho_2=(i+1)\Delta\rho$ and $\rho_1=\rho_2-l_0\Delta\rho$ where l_0 determines the point of intersection. As before, if the PEC object intersects the cell such that the position of the h_z field is inside the object, then the cell immediately above is extended downwards to conform to the shape of the object. If, however, the PEC object intersects the cell such that the position of the h_z field is not inside the object, then that cell is deformed to match the shape of object. Carrying out the integrals in (3.17) yields,

$$\begin{aligned}
&-\mu\frac{\rho_2^2-\rho_1^2}{2m}\frac{\partial}{\partial t}\left[h_{z,u}(\rho\rho,z_0,t)(\sin m\phi_2-\sin m\phi_1)-h_{z,v}(\rho\rho,z_0,t)(\cos m\phi_2-\cos m\phi_1)\right]\\
&\quad=(\rho_2-\rho_1)\left[e_{\rho,u}(\rho\rho,z_0,t)\cos m\phi_1+e_{\rho,v}(\rho\rho,z_0,t)\sin m\phi_1\right]\\
&\quad+\frac{\rho_2}{m}\left[e_{\phi,u}(\rho_2,z_0,t)(\sin m\phi_2-\sin m\phi_1)-e_{\phi,v}(\rho_2,z_0,t)(\cos m\phi_2-\cos m\phi_1)\right]\\
&\quad+(\rho_1-\rho_2)\left[e_{\rho,u}(\rho\rho,z_0,t)\cos m\phi_2+e_{\rho,v}(\rho\rho,z_0,t)\sin m\phi_2\right]\\
&\quad+\frac{\rho_1}{m}\left[e_{\phi,u}(\rho_1,z_0,t)(\sin m\phi_1-\sin m\phi_2)-e_{\phi,v}(\rho_1,z_0,t)(\cos m\phi_1-\cos m\phi_2)\right].
\end{aligned}\tag{3.18}$$

Next, the sine and cosine terms can be separated to yield four equations where two of the four equations are redundant and can be discarded.

$$\begin{aligned}
&\left[-\mu\frac{\rho_2^2-\rho_1^2}{2m}\frac{\partial}{\partial t}h_{z,u}(\rho\rho,z_0,t)\right]\sin m\phi_2\\
&\quad=\left[\frac{\rho_2}{m}e_{\phi,u}(\rho_2,z_0,t)-\frac{\rho_1}{m}e_{\phi,u}(\rho_1,z_0,t)-(\rho_2-\rho_1)e_{\rho,v}(\rho\rho,z_0,t)\right]\sin m\phi_2
\end{aligned}\tag{3.19}$$

$$\begin{aligned}
&\left[\mu\frac{\rho_2^2-\rho_1^2}{2m}\frac{\partial}{\partial t}h_{z,u}(\rho\rho,z_0,t)\right]\sin m\phi_1\\
&\quad=\left[-\frac{\rho_2}{m}e_{\phi,u}(\rho_2,z_0,t)+\frac{\rho_1}{m}e_{\phi,u}(\rho_1,z_0,t)+(\rho_2-\rho_1)e_{\rho,v}(\rho\rho,z_0,t)\right]\sin m\phi_1
\end{aligned}\tag{3.20}$$

$$\left[\mu\frac{\rho_2^2-\rho_1^2}{2m}\frac{\partial}{\partial t}h_{z,v}(\rho\rho,z_0,t)\right]\cos m\phi_2$$

$$= \left[-\frac{\rho_2}{m}e_{\phi,v}(\rho_2, z_0, t) + \frac{\rho_1}{m}e_{\phi,v}(\rho_1, z_0, t) - (\rho_2 - \rho_1)e_{\rho,u}(\rho\rho, z_0, t)\right]\cos m\phi_2 \quad (3.21)$$

$$\left[-\mu\frac{\rho_2^2 - \rho_1^2}{2m}\frac{\partial}{\partial t}h_{z,v}(\rho\rho, z_0, t)\right]\cos m\phi_1$$
$$= \left[\frac{\rho_2}{m}e_{\phi,v}(\rho_2, z_0, t) - \frac{\rho_1}{m}e_{\phi,v}(\rho_1, z_0, t) + (\rho_2 - \rho_1)e_{\rho,u}(\rho\rho, z_0, t)\right]\cos m\phi_1 \quad (3.22)$$

Since the PEC intersects the bottom portion of the cell, the electric field tangential to the object will be the $e_\phi(\rho_1, z_0, t)$ and it can be set to zero. In addition, eliminating the sine and cosine factors yields the following two independent equations.

$$\frac{\partial}{\partial t}h_{z,u}(\rho\rho, z_0, t) = \frac{2m}{\mu(\rho_2 + \rho_1)}e_{\rho,v}(\rho\rho, z_0, t) - \frac{2\rho_2}{\mu(\rho_2^2 - \rho_1^2)}e_{\phi,u}(\rho_2, z_0, t) \quad (3.23)$$

$$\frac{\partial}{\partial t}h_{z,v}(\rho\rho, z_0, t) = -\frac{2m}{\mu(\rho_2 + \rho_1)}e_{\rho,u}(\rho\rho, z_0, t) - \frac{2\rho_2}{\mu(\rho_2^2 - \rho_1^2)}e_{\phi,v}(\rho_2, z_0, t) \quad (3.24)$$

Discretizing the time derivative using a central difference approximation, substituting in the values for ρ_1 and ρ_2, and using the same notation as in Chapter 2, yields the following conformal BOR FD-TD difference equations.

$$h_{z,u}|^{n+1}_{i+1/2,k+1/2} = h_{z,u}|^{n}_{i+1/2,k+1/2} + \frac{m\Delta t}{\mu(i + 1 - l_0/2)\Delta\rho}e_{\rho,u}|^{n+1/2}_{i+1/2,k+1/2} - \frac{(i+1)\Delta t}{\mu\left[(i+1)l_0 - l_0^2/2\right]}e_{\phi,u}|^{n+1/2}_{i+1,k+1/2} \quad (3.25)$$

$$h_{z,v}|^{n+1}_{i+1/2,k+1/2} = h_{z,v}|^{n}_{i+1/2,k+1/2} - \frac{m\Delta t}{\mu(i + 1 - l_0/2)\Delta\rho}e_{\rho,v}|^{n+1/2}_{i+1/2,k+1/2} - \frac{(i+1)\Delta t}{\mu\left[(i+1)l_0 - l_0^2/2\right]}e_{\phi,v}|^{n+1/2}_{i+1,k+1/2} \quad (3.26)$$

Because tangential magnetic fields are not necessarily zero on a PEC object, conformal BOR FD-TD difference equations for the e_ρ, e_ϕ, and e_z fields cannot be derived. Instead, any electric field whose contour surface intersects the PEC object is simply not calculated since it depends on a h field that will be inside the PEC object. Since the surrounding existing h fields depend on the values of these e fields, a nearest neighbor approximation of the e field must be used. For instance, Figure 3-4 illustrates a case where an e_ρ field must be borrowed in order to calculate an h_ϕ field. In this case, the difference equation for the h_ϕ field will be,

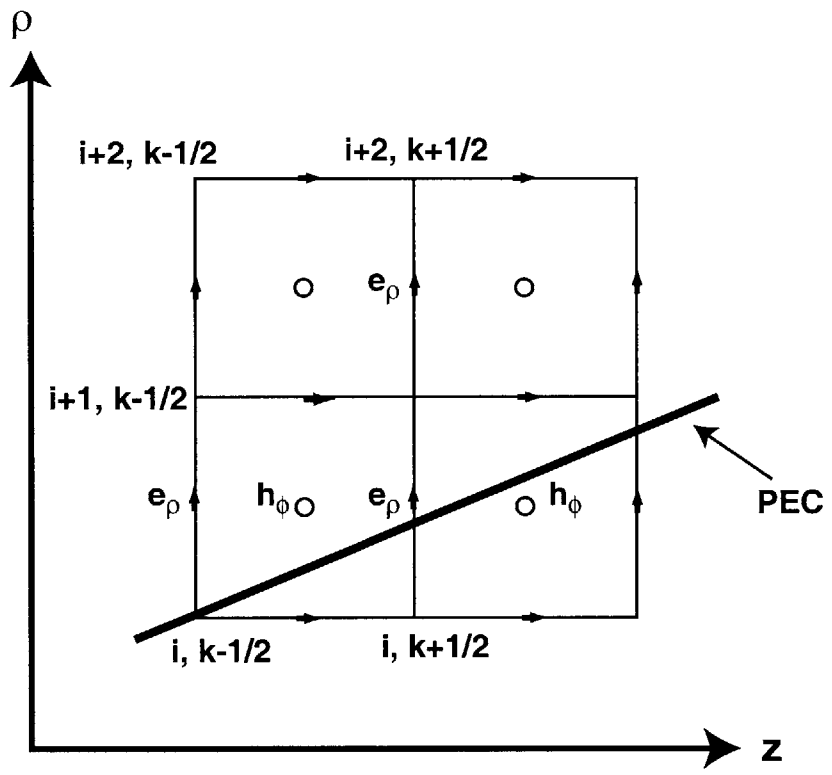

Figure 3-4: Here the field $h_\phi|_{i+1/2,k+1}$ does not exist, so that the field $e_\rho|_{i+1/2,k+1/2}$ cannot be computed. Hence, the field $h_\phi|_{i+1/2,k}$ is computed by "borrowing" the value of the neighboring field $e_\rho|_{i+3/2,k+1/2}$.

$$h_{\phi,u}|_{i+1/2,k}^{n+1} = h_{\phi,u}|_{i+1/2,k}^{n} + \frac{\Delta t \Delta z}{\mu A} e_{z,u}|_{i+1,k}^{n+1/2} + \frac{\Delta t}{\mu A}\left(l_1 e_{\rho,u}|_{i+1/2,k-1/2}^{n+1/2} - l_2 e_{\rho,u}|_{i+3/2,k+1/2}^{n+1/2}\right) \tag{3.27}$$

which is the same as equation (3.15) except that value of the field $e_\rho|_{i+3/2,k+1/2}$ is used in place of the field $e_\rho|_{i+1/2,k+1/2}$.

3.2 Comparison of Staircase and Conformal Predictions

In the following two sections, a sphere, and the biconical object modeled in the previous chapter are modeled using the conformal grid approach. The predictions using the conformal approach are compared to the predictions obtained when representing the target using a staircase approximation.

3.2.1 Monostatic RCS of Sphere

As discussed previously, due to the ragged structure of the staircase approximation, creeping waves that can be induced on smooth surfaces, such as a sphere, are not modeled accurtely. In the following, the effectiveness of the conformal approach for reducing this is error is explored. In order to measure the effectiveness of the conformal gridding approach, it is useful to consider the effect of decreasing the step size. In the following, the backscatter RCS of a sphere versus frequency is computed with the BOR FD-TD method at two different step sizes, and is compared to the exact Mie series solution. In order to reduce the effects of numerical dispersion, the RCS is calculated for a wave incident at $\theta = 45°$, as shown in Figure 3-5. The first BOR FD-TD

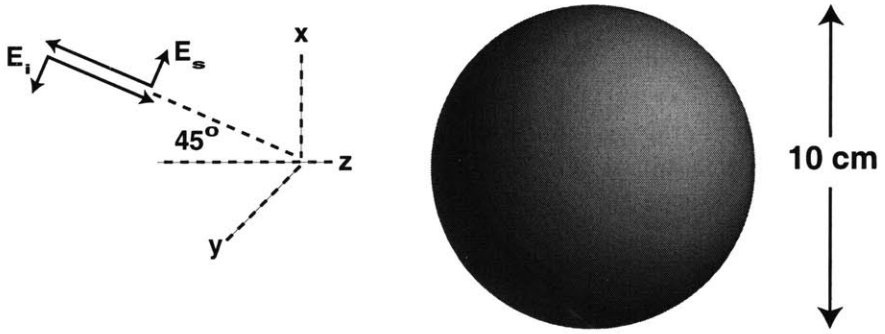

Figure 3-5: Geometry of a Sphere, $\theta_i = 45°$

result, shown in Figure 3-6 was obtained using a step size of $\lambda_0/10$ where λ_0 corresponds to the wavelength at 6 GHz. In the Rayleigh region, which extends through 1 GHz, this staircase model is able to accurately model the backscatter RCS of the sphere since the effect of the creeping wave is neglible. However, as the frequency increases into the resonance region beyond 3 GHz, the results using this staircase model begin to exhibit errors as the creeping wave term is incorrectly modeled. In order to model the creeping wave term to accurately predict the RCS through 6 GHz, it was necessary to reduce the step size to $\lambda_0/20$.

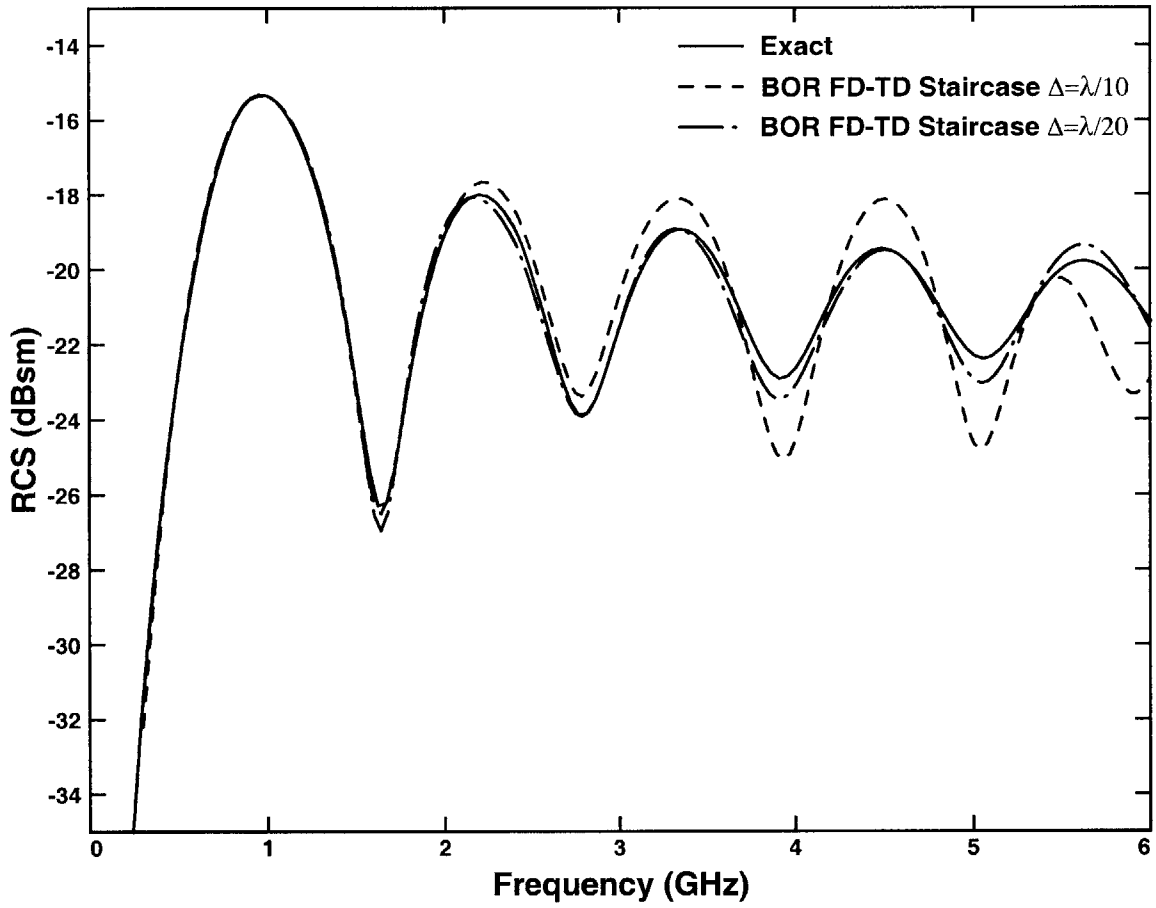

Figure 3-6: Comparison of staircase modeling at different step sizes for the backscatter RCS of a sphere illuminated at $\theta = 45°$, $\phi = 0°$ for a cut in frequency.

While the smaller step size increased the accuracy of the solution, it also increased the computational requirement both in terms of memory and computer time since smaller step sizes require smaller time steps. In order to increase the accuracy without increasing the computational requirements , the sphere is modeled using the conformal approach discussed

in this chapter. As shown in Figure 3-7, the conformal grid model for the sphere is able to accurately model the backscatter RCS of the sphere through 6 GHz with a step size of $\lambda_0/10$. However, although it was not necessary to reduce the step size to obtain increased accuracy, the time step was reduced to avoid numerical instabilities associated with conformal approaches. It was found necessary to reduce the time step to approximately 85% of the time step used when modeling the sphere with a staircase model. Despite the decreased time step, the accuracy gained by using the conformal approach makes the method advantageous over the staircase approach.

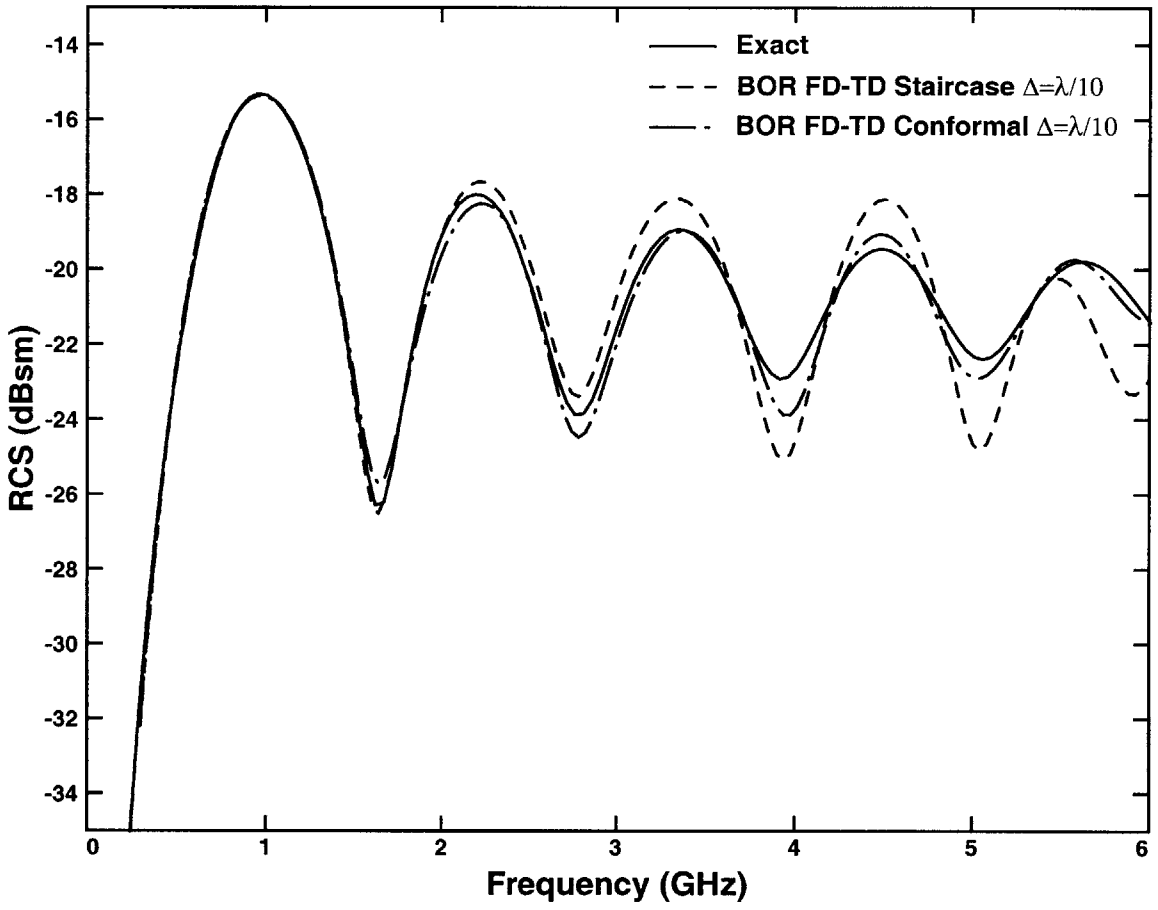

Figure 3-7: Comparison of conformal and staircase modeling for the backscatter RCS of a sphere illuminated at $\theta = 45°$, $\phi = 0°$ for a cut in frequency.

3.2.2 Bistatic RCS of Biconical Object

In this section, the biconical object is modeled using the conformal approach in order to determine the bistatic RCS at 1 GHz for an incident direction normal to the nosecone of the target. The predictions are compared with those obtained by modeling the target with a staircase model and with BOR MoM predictions. The dimensions of the target and scattering geometry are shown below in Figure 3-8. Due to the size of the nosecone, a very fine mesh was needed to

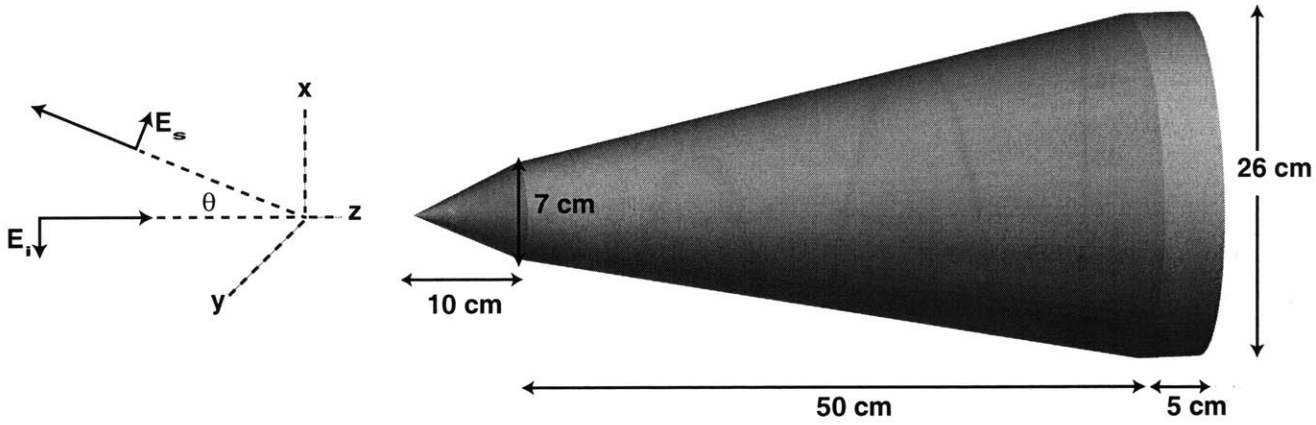

Figure 3-8: Geometry for Biconical Object, $\theta_i = 0°$

accurately represent the shape of the target. Figure 3-9 plots the bistatic RCS of the biconical shape computed by four different methods. The solid curve represents the results of using the BOR FD-TD method with a staircase model of the target at a step size of $\lambda/80$ while the dashed curved represents the BOR MoM predictions. As evidenced in the plot, the two results are in good agreement for all bistatic angles except those near backscatter. In order to more accurately model the nosecone, the BOR FD-TD conformal approach was used with step sizes of $\lambda/60$ and $\lambda/80$. In both cases, the bistatic RCS for angles near backscatter match well with the BOR MoM predictions implying that the staircase predictions were in error. The reason for this error is that the staircase model could not accurately represent the tip of the nosecone, and instead approximated it by a tiny flat surface, which explains why it predicted a higher backscatter RCS than the other three methods.

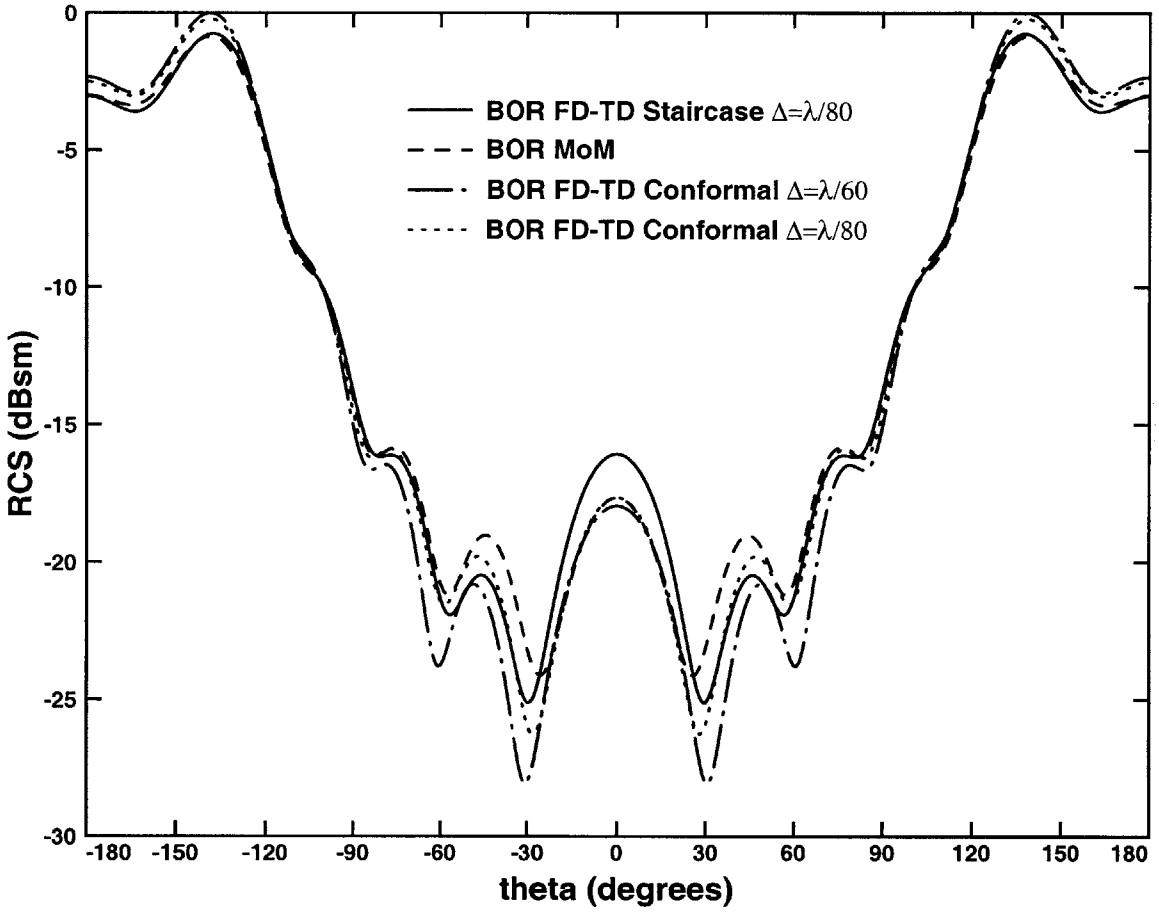

Figure 3-9: Conformal BOR FD-TD predictions at two different step sizes are compared to Staircase BOR FD-TD and BOR MoM prediction. The plot is of the Bistatic HH RCS of a biconical object illuminated at normal incidence for a cut in θ at $\phi = 0°$.

3.3 Summary

In this chapter, a conformal grid approach to the BOR FD-TD method was developed. Rather than representing the target by a staircase model, cells along the surface of the target were modified to conform with the target's shape. Along the surface of the target, h_ρ contours were extended or contracted in the $\hat{z}$ direction to match the surface of the target. Similarly, h_z contours were extended or contracted in the $\hat{\rho}$ direction, and h_ϕ contours were extended or contracted in both the $\hat{z}$ and $\hat{\rho}$ directions to conform with the surface of the target. Electric fields whose grid cell cuts through the target surface were not calculated, and magnetic fields which needed these values instead used the nearest neighbor electric field.

The method was applied to the modeling of a sphere and a biconical object. In the case of the sphere, the conformal method was shown to give predictions similar to staircase predictions with a step size two times smaller. In the case of the biconical object, it was shown that the conformal method improved the accuracy in predicting backscattering from the nosecone by nearly 3 dBsm. In cases where the primary scattering component is due to specular reflections, the conformal method did not greatly affect the accuracy of the predictions, since this scattering was already predicted accurately.

Chapter 4

RCS Prediction Using the Monostatic-Bistatic Equivalence Principle and the FD-TD/Geometrical Optics Hybrid Method

One advantage to the FD-TD method is that with appropriate excitation, the RCS over a band of frequencies can be calculated from a single simulation. However, one disadvantage for the FD-TD method is that only one aspect angle is obtained from each simulation, and calculation of RCS over a range of angles requires multiple simulations. In this chapter, the monostatic-bistatic equivalence principle is used to reduce the overall computational burden by reducing the number of angles at which calculations must be performed. A single FD-TD BOR simulation is used to calculate the monostatic signature for one incident angle, as well as bistatic signatures for adjacent observation directions. The bistatic equivalence theorem is then used to approximate monostatic signatures for other angles near the incident direction of the actual FD-TD BOR simulation.

A second disadvantage to the FD-TD method is that it is computationally expensive for electrically large targets. In the special case of body of revolution objects, the BOR FD-

TD reduces the required computation by expanding the ϕ dependence in Fourier modes. For electrically small targets, and incident angles near the z–axis, the number of modes required to represent the ϕ variation is small. However, for broadside incidence of electrically large targets more modes must be included, which increases the computation time. In order to reduce the computational expense, high-frequency techniques can be applied to efficiently model electrically large targets. Still, there remain cases where high-frequency techniques fail to achieve the desired accuracy, yet numerical techniques are impractical. For example, consider an electrically large structure with a smaller structure attached to it. Clearly, numerical techniques are impractical due to the overall target's size; moreover, high-frequency techniques cannot accurately model the small attached scatterer.

One approach, is to use a hybrid method that combines an exact technique such as the FD-TD method and a high-frequency technique such as Geometrical Optics. The hybrid method works by identifying individual scattering centers such as surface gaps, protrusions, or slope discontinuities, and deriving integral expressions for the scattering of each. In contrast to the BOR FD-TD technique, the FD-TD/GO hybrid method accounts for the ϕ variation analytically by evaluating these integral expressions by the method of stationary phase, in which the contribution is assumed to arise from a stationary phase point in the plane of incidence. A two-dimensional scattering problem is created by a local tangent plane approximation through the stationary phase point, and this is solved via a two dimensional FD-TD approach. The scattering from the large body on which the small protrusions are located is then calculated using Geometrical Optics. The scattering from each is coherently added to find the overall scattered fields and resulting radar cross section. The special case considered in this chapter will be the derivation of a hybrid formulation for the determination of the RCS from large body of revolutions with small BOR protrusions.

4.1 The Monostatic-Bistatic Equivalence Theorem

As discussed above, the FD-TD method is limited in that only one aspect angle is obtained from each simulation, and hence calculation of monostatic RCS over a range of incidence angles requires multiple simulations. One possible approach for reducing the computational burden imposed by this limitation is to use the monostatic-bistatic equivalence theorem. Although the equivalence theorem is often applied to approximate bistatic RCS results from monostatic RCS,

the principle will be used here to obtain monostatic RCS results from bistatic RCS calculations from the BOR FD-TD method.

The monostatic-bistatic equivalence is based on the fact that as the bistatic angle approaches zero, the bistatic RCS can be approximated by the monostatic RCS at the bisector of the bistatic angle. For many scatterers, the error in this approximation is small for bistatic angles up to several degrees, while for others, the error is larger. The magnitude of the error depends on the properties of the individual scattering centers located on the object.

The derivation of the monostatic-bistatic equivalence theorem, as presented by Kell in [28] begins from the definition of the RCS of a target,

$$\sigma = 4\pi R_o^2 \lim_{R_o \to \infty} \frac{\left|\vec{H}_R\right|^2}{\left|\vec{H}_0\right|^2} \tag{4.1}$$

where $\vec{H}_0$ is the incident magnetic field vector, and $\vec{H}_R$ is the scattered magnetic field. The RCS can be obtained by using a far-field transformation of the tangential electric and magnetic fields on the surface of the target. The radiation field can be obtained from the Stratton-Chu formulation,

$$\begin{aligned} \vec{H}_R e^{i\psi} &= -\frac{1}{4\pi} \iint \left[(n \times \vec{H}_s) \times \nabla \frac{e^{ik_0 r_o}}{r_o} + (n \cdot \vec{H}_s) \nabla \frac{e^{ik_0 r_o}}{r_o} \right. \\ &\qquad \left. -i\omega\epsilon_0 (n \times \vec{E}_s) \frac{e^{ik_0 r_o}}{r_o} \right] da \end{aligned} \tag{4.2}$$

where $\vec{H}_R$ is the re-radiated magnetic scattered field, ψ is the phase of the re-radiated fields relative to a chosen reference, and $\vec{E}_s$ and $\vec{H}_s$ are the fields at the target's surface.

Given the exact value of the fields on the target's surface, the RCS could of course be found directly, however, given knowledge of either only the bistatic or monostatic RCS, this is not possible. Instead, equation (4.2) is approximated using the method of stationary phase by factoring out phase delay terms.

The coordinate system used in deriving the monostatic-bistatic equivalence is shown in Figure 4-1. In the coordinate system, the values of R_i and R_o are distances from the origin to the transmitting and observation points, and the values of r_i and r_o are the distance from a differential area, da, on the target's surface to the transmitting and observation points. Assuming that the lengths R_o and R_i are much larger than the target's dimensions, the sum, $r_i + r_o$, can

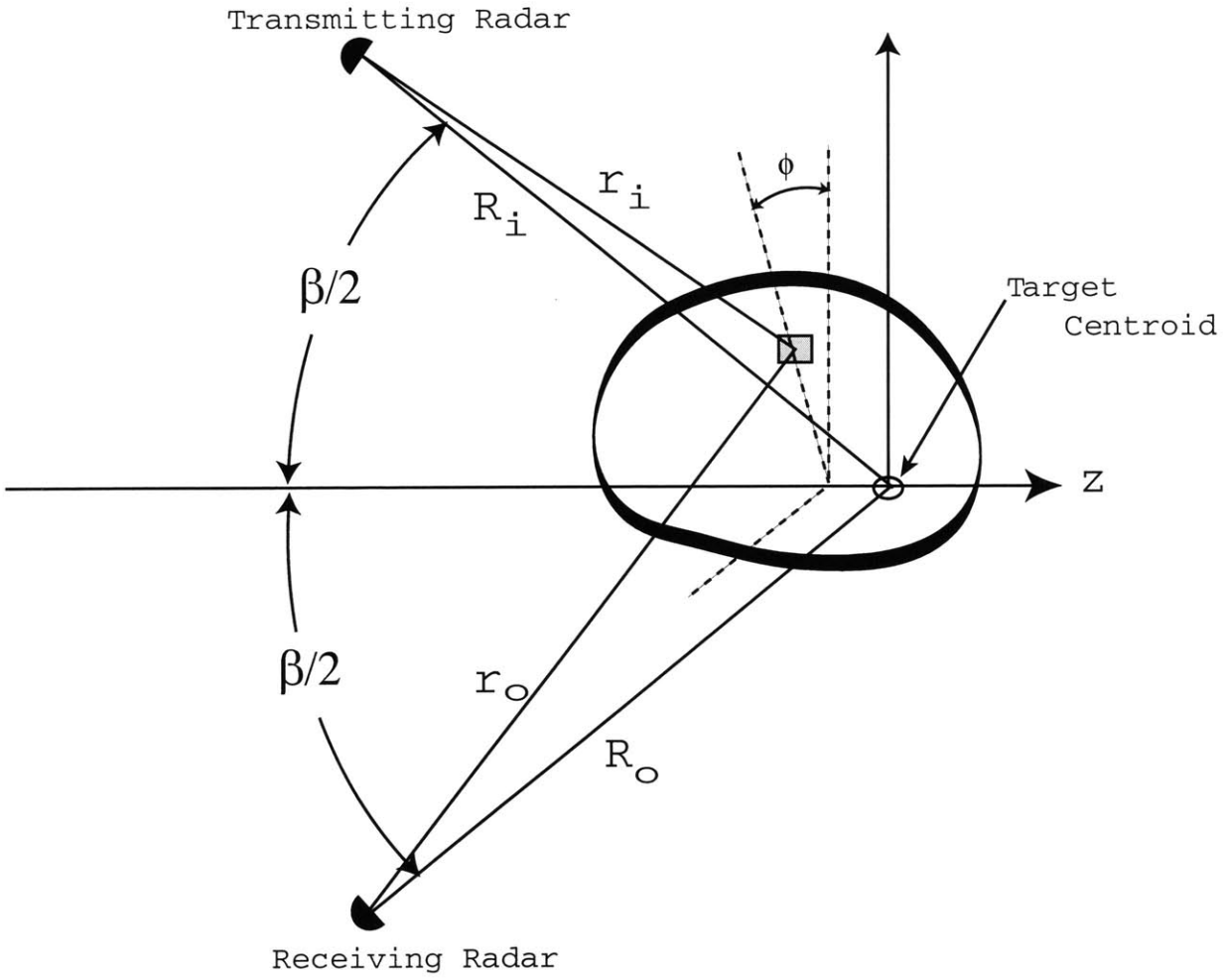

Figure 4-1: Coordinate system for Monostatic-Bistatic Equivalence Theorem

be approximated as independent of the angle ϕ as,

$$r_i + r_o \approx 2z\cos(\beta/2) + (R_i + R_o). \tag{4.3}$$

Applying the method of stationary phase to (4.2), the contributions from discrete scattering centers or saddle points are combined so that the total RCS of the target can be written as the sum of each of these individual scatterers. The RCS in terms of discrete scattering centers is given by,

$$\sigma = \left| \sum_{m=1}^{M} \sqrt{\sigma_m}\, e^{i\phi_m} \right|^2 \tag{4.4}$$

where σ_m is the RCS of the mth discrete scatterer on the target and ϕ_m is the associated phase factor relative to the phase of the first discrete scatterer. The above formulation can be used to describe either the monostatic or bistatic RCS. In the following, it is assumed that the σ_m terms in (4.4) represent the bistatic RCS of the target. Under the conditions for which the RCS can be written as the sum of individual scatterers, the phase factor, ϕ_m can be modified so that σ in (4.4) represents monostatic rather than bistatic RCS. Using the relations given in (4.3), the modified phase factor ϕ_m can be written as

$$\phi_m = 2k_0 z_m \cos(\beta/2) + \xi_m \tag{4.5}$$

where $z_m(\alpha)$ is the distance between the mth and first phase center, projected on the bisector axis, and ξ_m is the residual phase contributions of the mth center. The monostatic RCS in terms of the discrete scattering centers can then be written as,

$$\sigma = \left| \sum_{m=1}^{M} \sqrt{\sigma_m}\, e^{i2k_0 z_m \cos(\beta/2)+\xi_m} \right|^2 . \tag{4.6}$$

Assuming that values of z_m and ξ_m do not vary much over the range of bistatic angles considered, the only difference in the phase factors between (4.4) and (4.6) will be the $\cos(\beta/2)$ factor. For very small bistatic angles, the cosine factor can be approximated as constant, so that the bistatic RCS is equal to the monostatic cross section measured on the bisector. However, for larger bistatic angles the summation in (4.6) must be computed by identifying individual scattering centers, or alternatively, the factors $\cos(\beta/2)$ and k_0 can be grouped together to

represent a new frequency allowing the following statement of monostatic-bistatic equivalence,

$$\sigma_{MS}(f\cos(\beta/2), \theta = \alpha - \beta/2) = \sigma_{BS}(f, \theta_i = \alpha, \theta_b = \beta) \tag{4.7}$$

where α is the incident aspect angle and β is the bistatic angle. Briefly, the monostatic-bistatic equivalence theorem states that the bistatic cross section of aspect angle α and bistatic angle β is equal to the monostatic cross section measured on the bisector at a frequency lower by the factor $\cos(\beta/2)$.

Since the FD-TD method can calculate bistatic RCS for multiple frequencies in one simulation, the monostatic-bistatic equivalence theorem can be used estimate monostatic RCS for other aspect angles near the incident direction of the actual FD-TD simulation. The range of aspect angle at which the monostatic RCS can be accurately estimated depends on the scattering characteristics of the target. Since the equivalence theorem assumes that the RCS can be written as the squared sum of fields from discrete scattering centers, the method is expected to yield more accurate estimates at higher frequencies when the interaction between scattering centers is less significant.

4.2 Monostatic-Bistatic Equivalence Results

In order to test the usefulness of the monostatic-bistatic equivalence, the principle was applied to estimation of the monostatic RCS of two objects. The first object, a cylinder, was chosen to test the ability of the principle to estimate monostatic RCS that is dominated by simple scattering phenomena such as specular reflection. The second target, the biconical shape modeled in previous chapters, is used to test the ability of the principle to estimate monostatic RCS that results from scattering by a more complex object.

4.2.1 Monostatic RCS of Cylinder

Since the monostatic-bistatic equivalence principle is based in part on physical optics assumptions, the monostatic RCS predicted by this principle for an electrically large cylinder, such as the one shown in Figure 4-2, should be a good approximation to the true monostatic RCS. Due to the symmetry of the geometry, the monostatic RCS of the cylinder is only computed from $\theta = 0°$ to $\theta = 90°$. Hence, as shown in Figure 4-3, the monostatic estimates were obtained

with the BOR FD-TD method using incident angles of $\theta_i = 0°$ and $\theta_i = 90°$. The bistatic RCS results for endcap illumination were used to estimate the monostatic RCS, shown in Figure 4-3(a), from $\theta = 0°$ to $\theta = 45°$. Similarly, the bistatic RCS results for broadside illumination were used to estimate the monostatic RCS, shown in Figure 4-3(b), for angles between $\theta = 45°$ and $\theta = 90°$. As expected, the combined results of the two BOR FD-TD runs shown in Figure 4-3(c) are in good agreement with the MoM predictions.

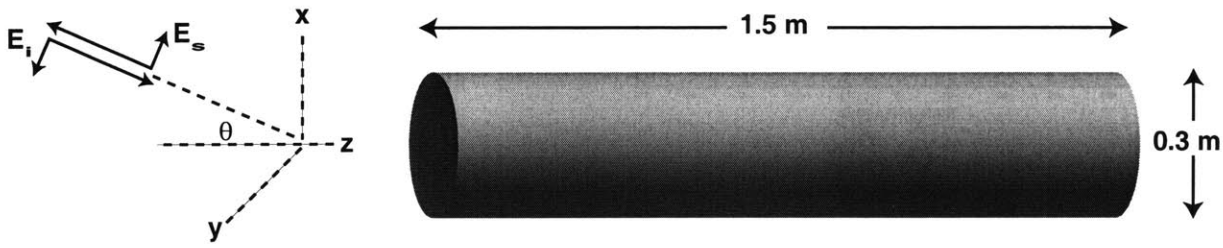

Figure 4-2: Geometry for Cylinder

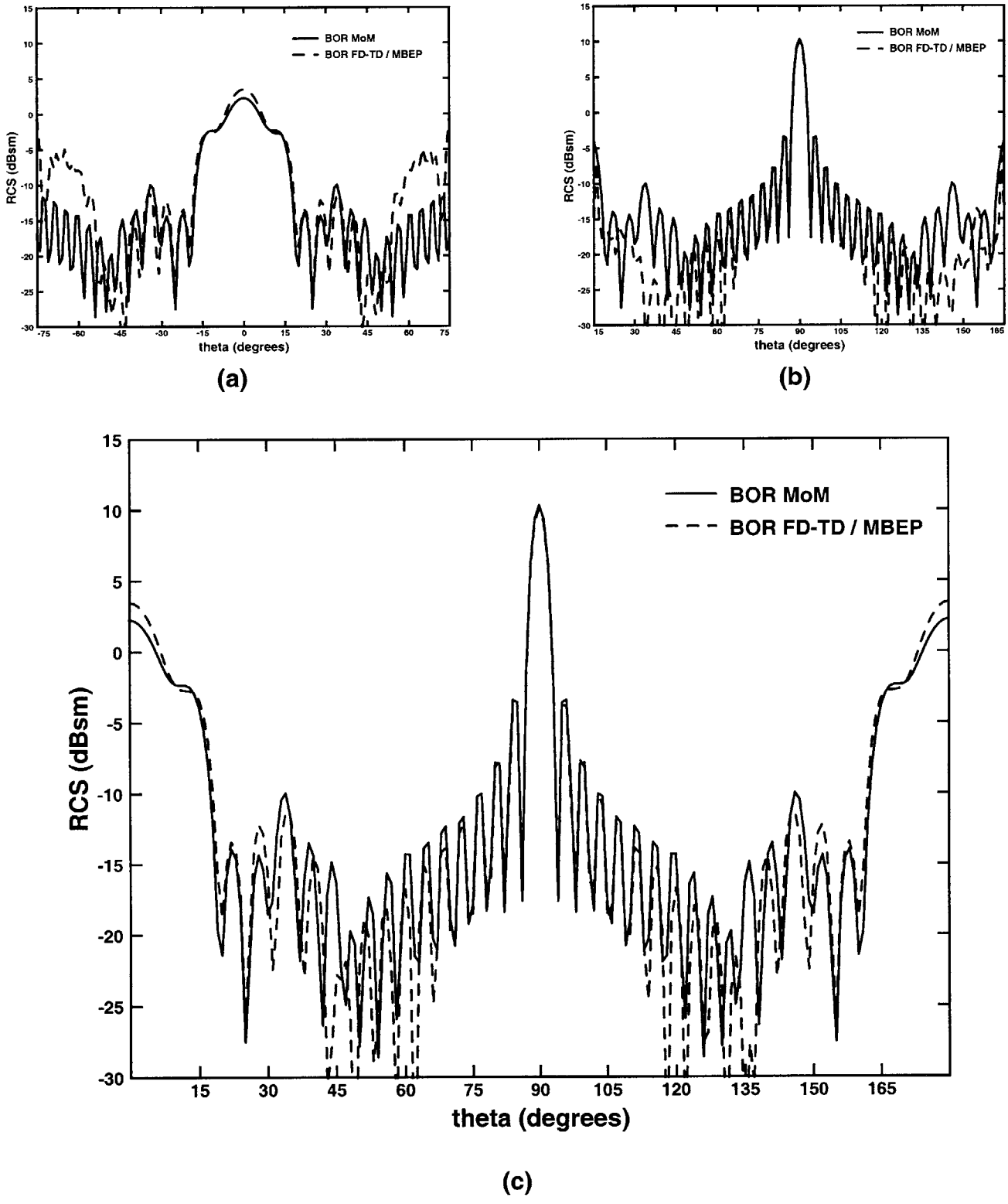

Figure 4-3: Estimated monostatic RCS at 1.5 GHz of a cylinder is compared to MoM predictions. Shown is HH polarization for a cut in θ with $\phi = 0°$. Results estimated by using the BOR FD-TD method with (a) $\theta_i = 0°$, and (b) $\theta_i = 90°$. Combined results shown in (c).

4.2.2 Monostatic RCS of Biconical Object

Unlike the cylinder modeled in the previous section, accurate monostatic RCS estimates for the biconical object, shown in Figure 4-4, were much more difficult to obtain requiring the use of bistatic results obtained from several incident angles. In addition, since the biconical object does not possess the symmetry of a cylinder, the monostatic RCS is computed for angles between $\theta = 0°$ and $\theta = 180°$.

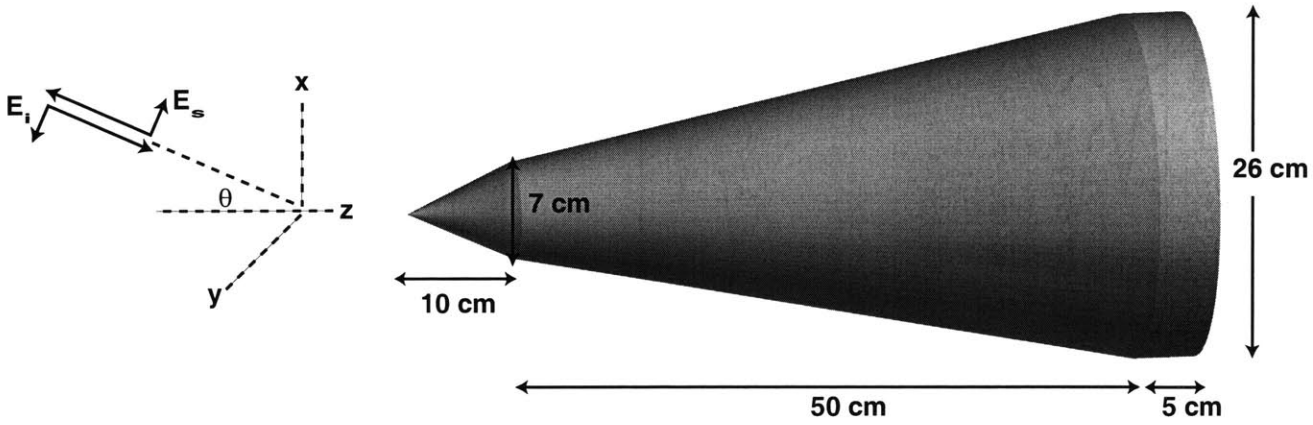

Figure 4-4: Geometry for Biconical Object

Initially, the monostatic RCS was estimated by using the BOR FD-TD method with incidence angles of $\theta = 0°$, $\theta = 90°$, and $180°$. As shown in plots (a)-(c) of Figure 4-5, accurate estimates were obtained for aspect angles near $\theta = 180°$ and $\theta = 79°$ where the specular reflection term dominates. At other aspect angles, especially those near $\theta = 0°$, the monostatic-bistatic equivalence estimates were not very accurate, indicating the need to incorporate bistatic RCS for other incident angles.

To obtain the additional bistatic RCS data, the BOR FD-TD method was run for several other incident angles. The results of these runs are shown in Figure 4-5(d) and Figures 4-6(a)-(d). As the aspect angles approached the nose of the target, it was found that the monostatic estimates became less accurate. This is likely due to the fact that the size of the target is electrically small. By sampling the bistatic RCS for more incident angles, however, the errors introduced by the equivalence principle can be reduced. Thus, as expected, while the overall estimated monostatic RCS shown in Figure 4-7 matches well with the BOR MoM predictions, it becomes less accurate near the nose of the target. To reduce the errors near $\theta = 0°$, further

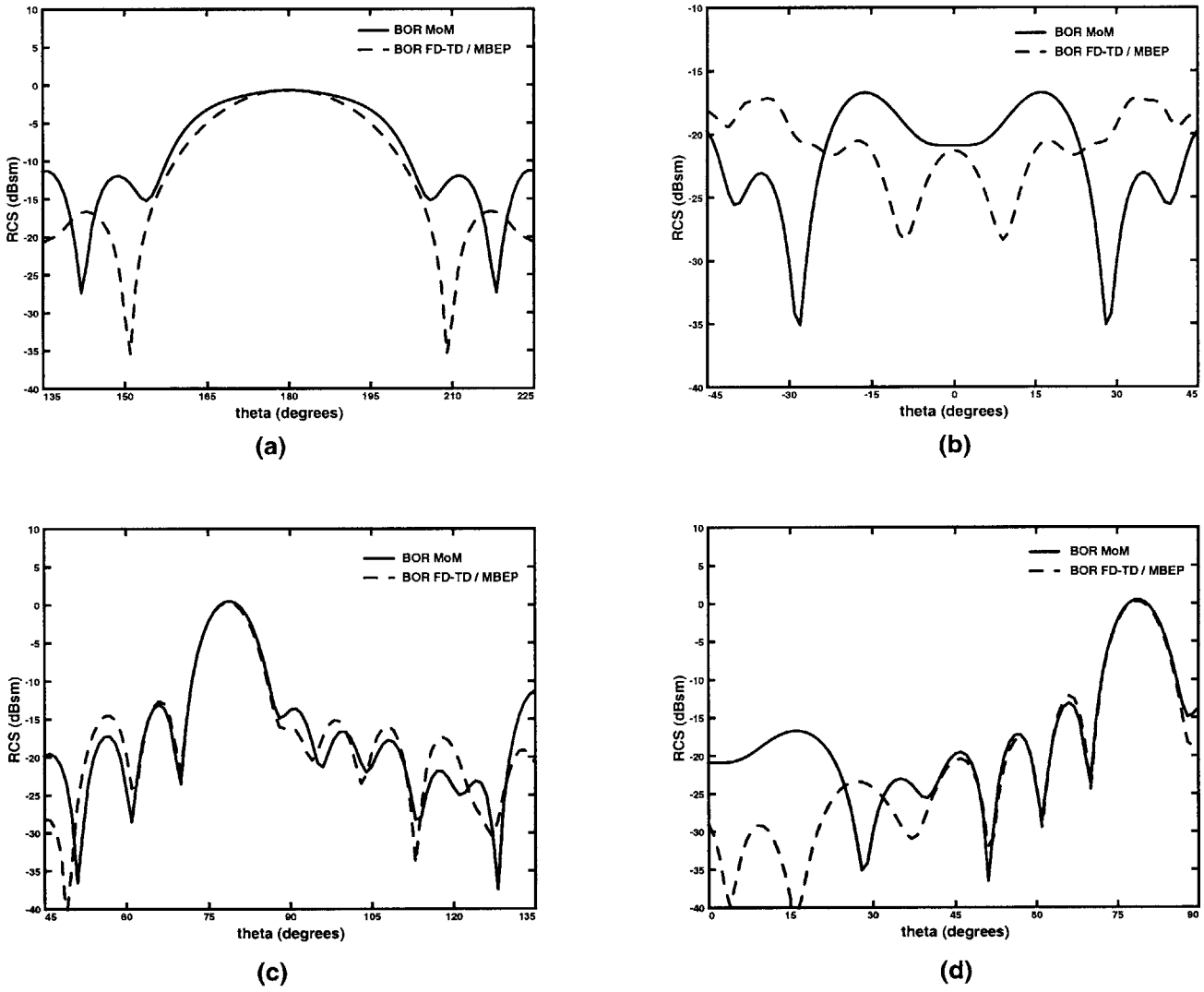

Figure 4-5: Estimated monostatic RCS at 1.5 GHz of biconical object is compared to MoM predictions. Shown is HH polarization for a cut in θ with $\phi = 0°$. Results estimated by using the BOR FD-TD method with (a) $\theta_i = 180°$, (b) $\theta_i = 0°$, (c) $\theta_i = 90°$, and (d) $\theta_i = 45°$.

BOR FD-TD runs for additional incident angles are needed.

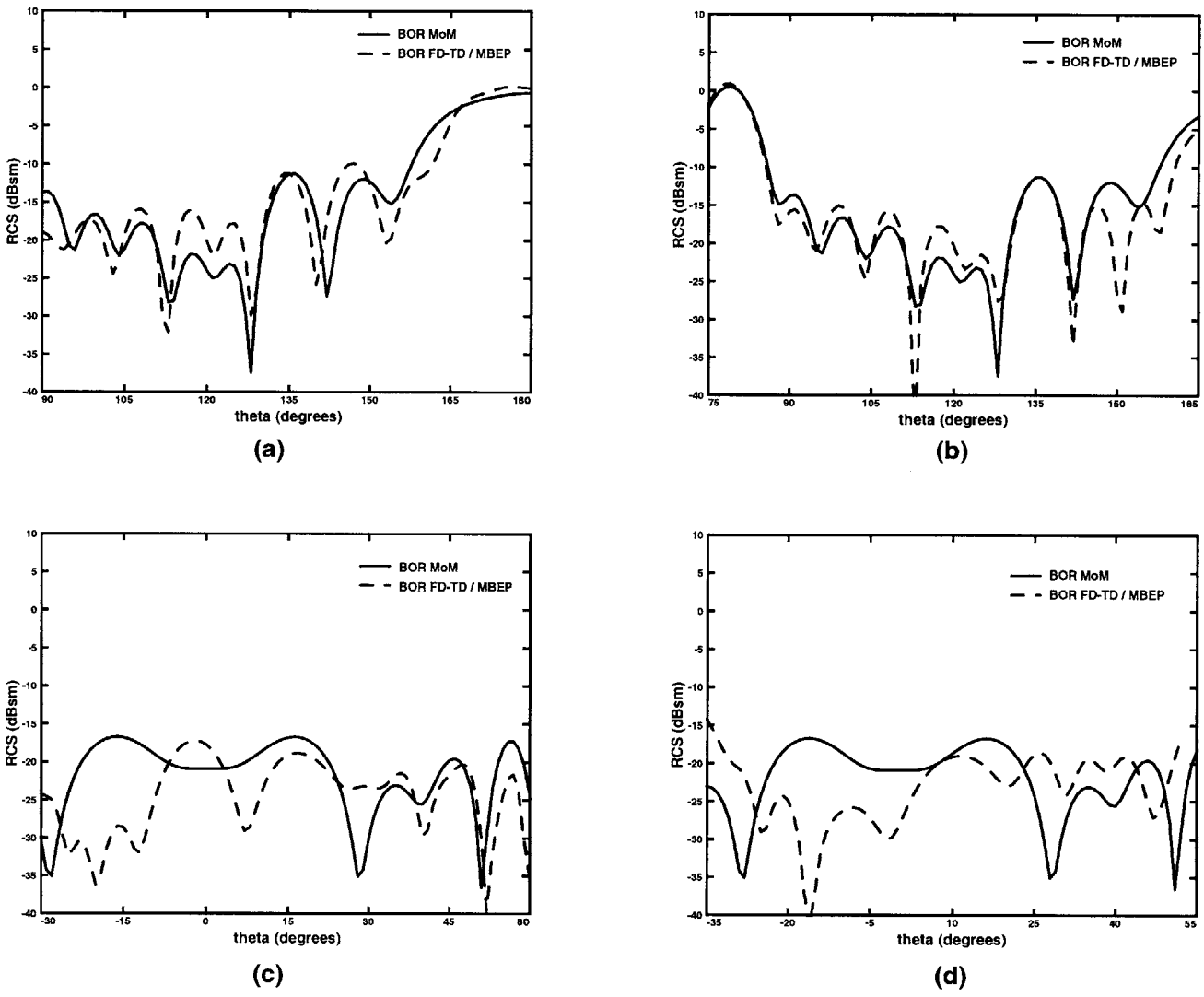

Figure 4-6: Estimated monostatic RCS at 1.5 GHz of biconical object is compared to MoM predictions. Shown is HH polarization for a cut in θ with $\phi = 0°$. Results estimated by using the BOR FD-TD method with (a) $\theta_i = 135°$, (b) $\theta_i = 120°$, (c) $\theta_i = 15°$, and (d) $\theta_i = 7.5°$.

Also shown in Figure 4-7 is the monostatic signature computed by the high frequency PO/PTD method. It is clear from the plot, that the PO/PTD method accurately predicts the monostatic RCS near the regions where the specular reflection term dominates. However, as expected, for other aspect angles, the PO/PTD's predictions do not match the BOR MoM predictions. This is most likely due to the small electrical size of target, for which the high-frequency assumptions of the PO/PTD method begin to break down. Because the ob-

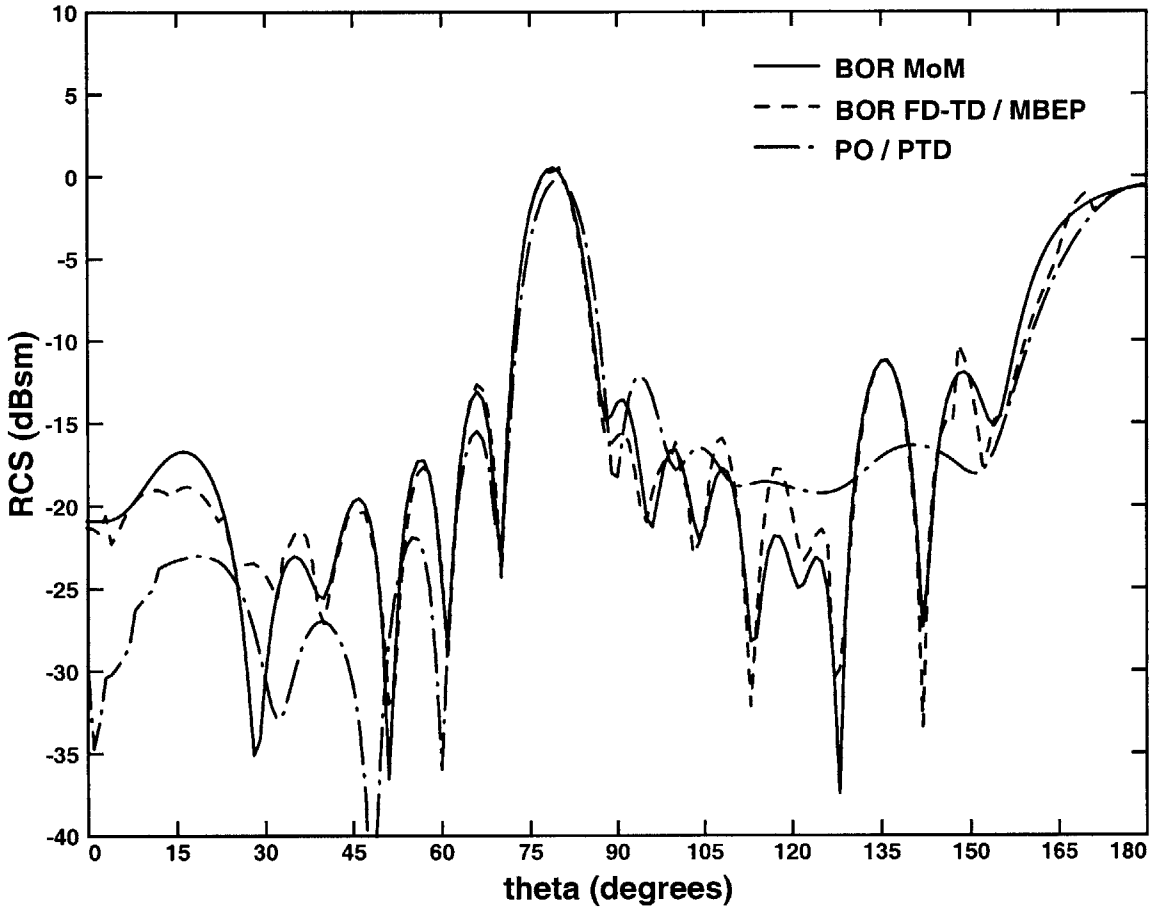

Figure 4-7: Combined result of BOR FD-TD runs using the equivalence principle with incident angles at $\theta_i = 0°, 7.5°, 15°, 45°, 90°, 120°, 135°$, and $180°$. The monostatic RCS at 1.5 GHz of the biconical object is compared to MoM predictions and PO/PTD predictions. Shown is HH polarization for a cut in θ with $\phi = 0°$.

ject modeled is electrically small the PO/PTD method was not able to correctly predict the monostatic signature for all aspect angles. On the other hand, the equivalence principle used in conjunction with an exact technique such as the BOR FD-TD can be used to obtain a good estimate of a monostatic signature.

4.3 The FD-TD/GO Hybrid Method

4.3.1 Integral Expression for Scattering from the Small Protrusion

Since the radar cross section is defined in terms of the scattered far-fields, the derivation of the FD-TD/GO Hybrid method begins with the formulation of an integral expression for the scattered far-fields of the entire target. From the resulting integral expression, the contribution of the small protrusion can be extracted and treated separately from the scattering due to the large body of revolution. For simplicity, the large body of revolution will be assumed to be a large conducting cylinder while the small protrusion can take on an arbitrary shape. Figure 4-8 illustrates the geometry of the scattering problem. The scattered fields $\vec{E}_s(\bar{r})$ and $\vec{H}_s(\bar{r})$ can be

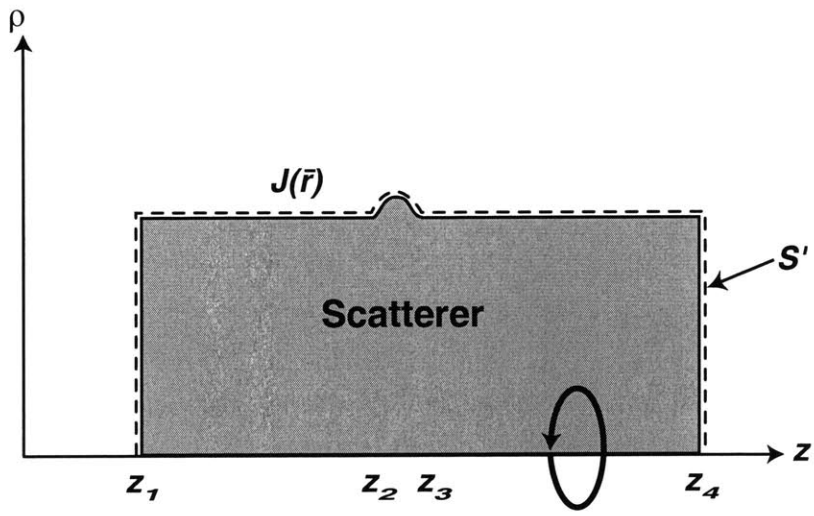

Figure 4-8: Original Huygens' Surface S': The scattered fields $\vec{E}_s(\bar{r})$ and $\vec{H}_s(\bar{r})$ are determined by the radiation of the induced electric current, $\vec{J}(\bar{r})$, that flows along the surface of the object.

calculated in terms of the induced current by using Huygens' principle,

$$\vec{E}_s(\bar{r}) = \iint_{S'} dS' \left\{ i\omega\mu_0 \overline{\overline{G}}(\bar{r},\bar{r}') \cdot \vec{J}(\bar{r}') \right\} \tag{4.8}$$

$$\vec{H}_s(\bar{r}) = \iint_{S'} dS' \left\{ \nabla \times \overline{\overline{G}}(\bar{r},\bar{r}') \cdot \vec{J}(\bar{r}') \right\} \tag{4.9}$$

where S' is surface of the object, $\vec{J}(\bar{r}') = \hat{n} \times \vec{H}_s(\bar{r}')$, and $\overline{\overline{G}}$ is the freespace Green's function. In order to simplify the surface of integration, an equivalent problem is created by using the surface equivalence principle. The equivalent problem is formed by replacing the actual sources on the

original surface of integration with equivalent sources located on a new surface of integration. The new surface, S'', shown in Figure 4-9, is chosen to be the same as the surface S' except near protrusion, where the surface is extended around the protrusion so that it completely encloses the protrusion but does not coincide with it. As before, the scattered fields are determined by

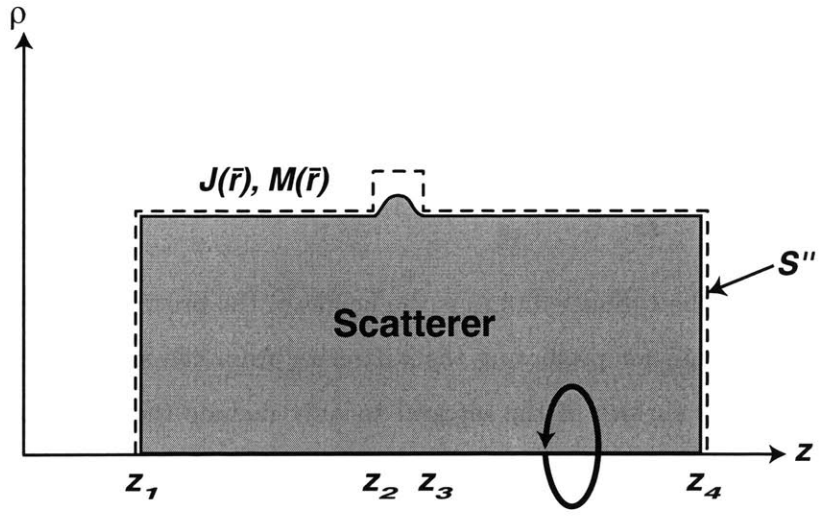

Figure 4-9: Equivalent Huygens' Surface S'': The new surface S'' extends around the protrusion so that the scattered fields $\vec{E}_s(\bar{r})$ and $\vec{H}_s(\bar{r})$ are determined by the radiation of the equivalent electric and magnetic currents, $\vec{J}(\bar{r})$ and $\vec{M}(\bar{r})$, that lie along the surface S''.

the radiation of the induced currents, however, because the new surface, S'', does not coincide with the target, the magnetic current source term in Huygens' principle must be included,

$$\vec{E}_s(\bar{r}) = \iint_{S''} \left\{ i\omega\mu \overline{\overline{G}}(\bar{r},\bar{r}') \cdot \vec{J}(\bar{r}') - \nabla \times \overline{\overline{G}}(\bar{r},\bar{r}') \cdot \vec{M}(\bar{r}') \right\} \tag{4.10}$$

$$\vec{H}_s(\bar{r}) = \iint_{S''} dS' \left\{ i\omega\epsilon_0 \overline{\overline{G}}(\bar{r},\bar{r}') \cdot \vec{M}(\bar{r}') + \nabla \times \overline{\overline{G}}(\bar{r},\bar{r}') \cdot \vec{J}(\bar{r}') \right\} \tag{4.11}$$

where $\vec{M}(\bar{r}') = -\hat{n} \times \vec{E}_s(\bar{r}')$. Under the far-field approximation (See Appendix A), the integral expression for the electric field can be rewritten as,

$$\vec{E}(\bar{r}) = \frac{e^{ikr}}{4\pi r} \iint_{S''} dS'' e^{-ik\hat{r}\cdot\bar{r}'} \left\{ i\omega\mu \left[\hat{\theta}\hat{\theta} + \hat{\phi}\hat{\phi} \right] \cdot \vec{J}(\bar{r}') - ik \left[\hat{\phi}\hat{\theta} - \hat{\theta}\hat{\phi} \right] \cdot \vec{M}(\bar{r}') \right\} \tag{4.12}$$

$$\hat{\theta} = -\hat{z}\sin\theta + \hat{x}\cos\theta\cos\phi + \hat{y}\cos\theta\sin\phi \tag{4.13}$$

$$\hat{\phi} = \hat{y}\cos\phi - \hat{x}\sin\phi \tag{4.14}$$

$$\hat{r} = \hat{x}\sin\theta\cos\phi + \hat{y}\sin\theta\sin\phi + \hat{z}\cos\theta. \tag{4.15}$$

Using the coordinate system shown in Figure 4-9, the surface of integration, S'' can be split into the integrals shown below.

$$\begin{aligned}\iint_{S''} dS'' &= \int_0^{\rho_0} d\rho' \int_0^{2\pi} \rho' d\phi' + \int_{z_1}^{z_2} dz' \int_0^{2\pi} \rho_0 d\phi' + \int_{\rho_0}^{\rho_0+\xi_0} d\rho' \int_0^{2\pi} \rho' d\phi' \\ &+ \int_{z_2}^{z_3} dz' \int_0^{2\pi} (\rho_0+\xi_0) d\phi' + \int_{\rho_0}^{\rho_0+\xi_0} d\rho' \int_0^{2\pi} \rho' d\phi' + \int_{z_3}^{z_4} dz' \int_0^{2\pi} \rho_0 d\phi' \\ &+ \int_0^{\rho_0} d\rho' \int_0^{2\pi} \rho' d\phi' \end{aligned} \tag{4.16}$$

where ρ_0 is the radius of the cylinder and ξ_0 is the height of the protrusion. Since the eventual goal is to develop a method for predicting the scattering from the small protrusion alone, it is desirable to reduce the surface of the integral to only include the region surrounding the protrusion. This can be accomplished by first arguing that since the cylinder is electrically large compared to the protrusion, there will be very little interaction from the end-caps of the cylinder and the protrusion. Hence, the integrals $\int_0^{\rho_0}$ can be neglected in the determination of the scattering from the protrusion alone. The effect of removing these two integrals is to place the points z_1 and z_2 at infinity, so that the Huygens' surface extends infinitely in both directions along the z–axis. Although, there is little interaction between the end-cap and the protrusion, there will be a significant interaction between the broadside of the cylinder and the protrusion. Thus, the contribution of the integrals, $\int_{z_1}^{z_2}$ and $\int_{z_3}^{z_4}$, can not be neglected. Furthermore, because the size of the cylinder is large compared to that of the protrusion, the value of $\rho' d\phi'$ along the surface enclosing the protrusion can be approximated as $\rho_0 d\phi'$, so that the surface integral becomes,

$$\begin{aligned}\iint_{S''} dS'' &= \left\{ \int_{z_1=-\infty}^{z_2} dz' + \int_{\rho_0}^{\rho_0+\xi_0} d\rho' + \int_{z_2}^{z_3} dz' + \int_{\rho_0}^{\rho_0+\xi_0} d\rho' + \int_{z_3}^{z_4=\infty} dz' \right\} \int_0^{2\pi} \rho_0 d\phi' \\ &= \int_{C'} dl' \int_0^{2\pi} \rho_0 d\phi' \end{aligned} \tag{4.17}$$

where the contour path C' is along the surface S'' defined by the above integrals. Although, the value of ρ' is approximated as constant in the determination of the differential area, $\rho' d\phi' \approx \rho_0 d\phi'$, the exact value of ρ' must be used in the exponential term that appears in the Huygens' integral expression.

Next, because of the axial symmetry present, the illuminating plane wave can be assumed, without loss generality, to be incident at $\phi_{inc} = 0$. For reasons that will becomes clear later, the scattering of interest will occur in the plane of incidence, so the observation angle ϕ can be assumed to be zero. The unit vectors thus become,

$$\hat{\theta} = \hat{x}\cos\theta - \hat{z}\sin\theta \equiv \hat{\alpha} \tag{4.18}$$

$$\hat{\phi} = \hat{y} \tag{4.19}$$

$$\hat{r} = \hat{x}\sin\theta + \hat{z}\cos\theta. \tag{4.20}$$

In addition, since the integration is over a cylindrical surface, the $\overline{r}'$ vector can be expressed as,

$$\overline{r}' = \rho_0\hat{\rho}' + z'\hat{z} \tag{4.21}$$

$$\hat{\rho}' = \hat{x}\cos\phi' + \hat{y}\sin\phi' \tag{4.22}$$

$$\hat{r}\cdot\overline{r}' = \hat{r}\cdot(r_0\hat{\rho}' + z'\hat{z}) = r_0\sin\theta\cos\phi' + z'\cos\theta. \tag{4.23}$$

The expression for the far-field electric field thus becomes,

$$\vec{E}(\overline{r}) = \frac{e^{ikr}}{4\pi r}\int_{C'} dl' \int_0^{2\pi} \rho_0 d\phi' e^{-ik(\rho_0+\xi')\sin\theta\cos\phi'} e^{-ikz'\cos\theta} \left\{ i\omega\mu\left[\hat{\alpha}\hat{\alpha} + \hat{y}\hat{y}\right]\cdot\vec{J}(\overline{r}') - ik\left[\hat{y}\hat{\alpha} - \hat{\alpha}\hat{y}\right]\cdot\vec{M}(\overline{r}')\right\} \tag{4.24}$$

where $\xi' = \rho' - \rho_0$.

In the limit of large $k\rho_0\sin\theta$, the ϕ' integral in (4.24) can be evaluated by the method of stationary phase. However, in order to apply the method, it is first necessary to factor out the illumination phase delay from the electric and magnetic current terms so that the phase of the integrand can be approximated as stationary. As shown in Figure 4-10, the illumination phase delay along a ϕ' loop at a constant z measured relative to the value of the phase at $\phi' = 0$ will depend on the angle of incidence and ϕ'. The illumination phase delay is given by,

$$\psi(\phi) = (1 - \cos\phi)k\rho_0\sin\theta_i \tag{4.25}$$

where θ_i is the angle of incidence. At $\theta_i = 90$, the incident wave travels along the x–axis so

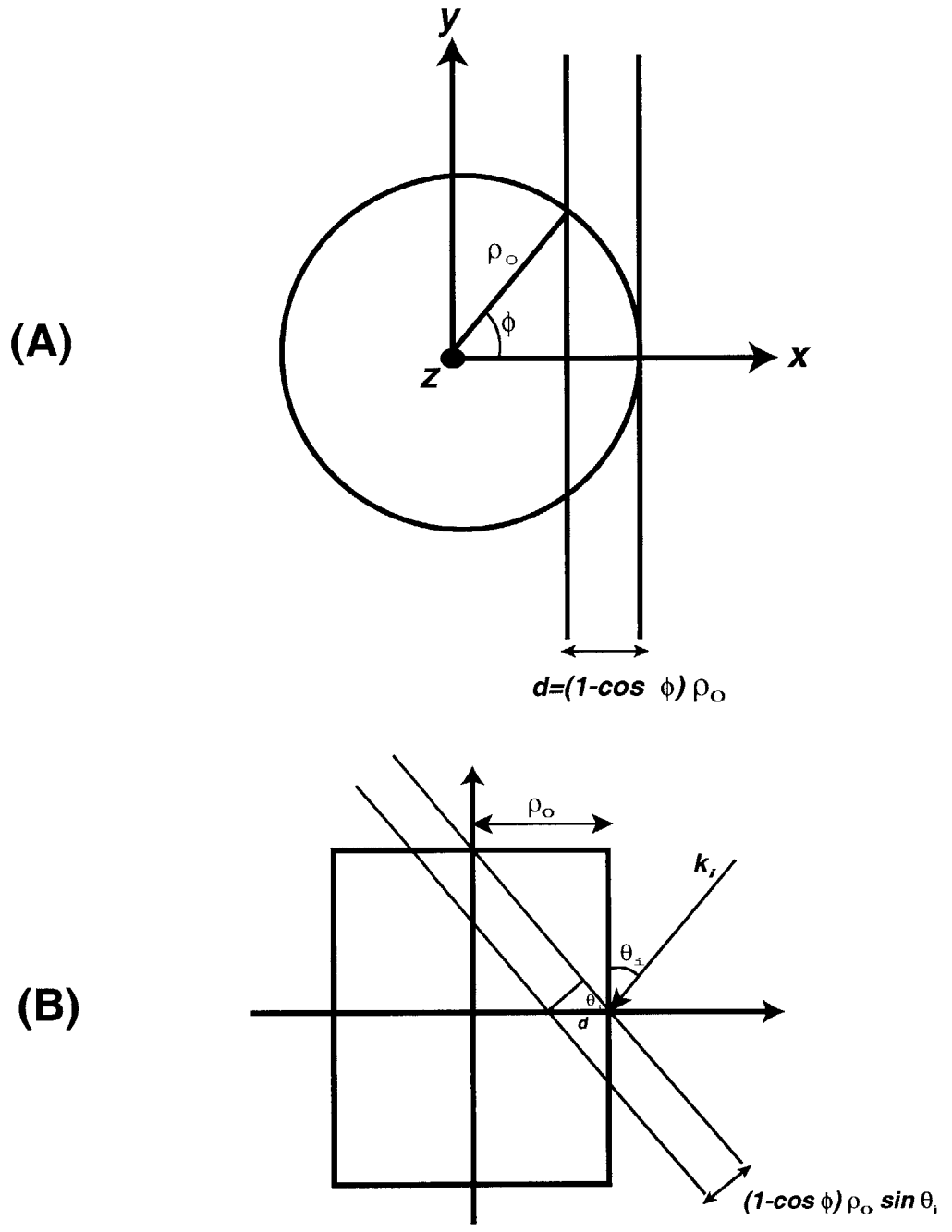

Figure 4-10: Determination of ϕ and θ_i dependent illumination phase delay factor. (A) The angle ϕ gives the distance along the x–axis of the relative phase from the reference phase front, and (B) with the angle θ_i the actual distance between the two phase fronts is determined.

that the wave must travel a distance equal to the diameter of the cylinder before illuminating every point along the ϕ' loop of interest. However, as the value of θ_i decreases, the illumination phase factor decreases until at $\theta_i = 0$, it becomes zero since the cylinder is illuminated at every point on the ϕ' loop at the same time. The electric and magnetic currents can now be written as,

$$\vec{J}(\bar{r}') = e^{i\psi(\phi)}\vec{J}'(\bar{r}') \tag{4.26}$$

$$\vec{M}(\bar{r}') = e^{i\psi(\phi)}\vec{M}'(\bar{r}') \tag{4.27}$$

where the phase of the primed functions, $\vec{J}$ and $\vec{M}$, is approximately constant with respect to ϕ'. The expression for the far-field electric field then becomes,

$$\vec{E}(\bar{r}) = \frac{e^{ikr}}{4\pi r}\int_{C'} dl' \int_0^{2\pi} \rho_0 d\phi' e^{-ik[\rho_0(\sin\theta+\sin\theta_i)+\xi'\sin\theta]\cos\phi'} e^{ik\rho_0\sin\theta_i} e^{-ikz'\cos\theta} \left\{ i\omega\mu\left[\hat{\alpha}\hat{\alpha}+\hat{y}\hat{y}\right]\cdot\vec{J}'(\bar{r}') - ik\left[\hat{y}\hat{\alpha}-\hat{\alpha}\hat{y}\right]\cdot\vec{M}'(\bar{r}')\right\}. \tag{4.28}$$

For large $k\rho_0(\sin\theta+\sin\theta_i)$, the method of stationary phase or the saddle-point method can be used to analytically evaluate the ϕ' integral. The saddle point method states that for large ν that the integral [32],

$$I(\nu) = \int_\Gamma d\alpha F(\alpha) e^{\nu f(\alpha)} \tag{4.29}$$

can be expressed as expansion about the saddle point α_0.

$$I(\nu) = F(\alpha_0)e^{\nu f(\alpha_0)}\sqrt{\frac{2\pi}{-\nu f''}}\left\{1+\frac{1}{2\nu f''}\left[\frac{f'''}{f''}\frac{F'}{F}+\frac{1}{4}\frac{f^{iv}}{f''}-\frac{5}{12}\frac{(f''')^2}{(f'')^2}-\frac{F''}{F}\right]+\cdots\right\} \tag{4.30}$$

The saddle point α_0 is determined from the point where the first derivative of the function $f(\alpha)$ is zero. Using this method, the saddle point is found to be at $\phi' = 0$, so that the expression for the electric field becomes,

$$\vec{E}(\bar{r}) = \frac{e^{ikr}}{4\pi r}\int_{C'} dl' \rho_0 e^{ik\rho_0\sin\theta_i} e^{-ikz'\cos\theta}\left\{ i\omega\mu\left[\hat{\alpha}\hat{\alpha}+\hat{y}\hat{y}\right]\cdot\vec{J}'(\bar{r}') - ik\left[\hat{y}\hat{\alpha}-\hat{\alpha}\hat{y}\right]\cdot\vec{M}'(\bar{r}')\right\} \sqrt{\frac{2\pi}{-ik\left[\rho_0(\sin\theta+\sin\theta_i)+\xi'\sin\theta\right]}} e^{-ik[\rho_0(\sin\theta+\sin\theta_i)+\xi'\sin\theta]}. \tag{4.31}$$

Note that the saddle point occurs at $\phi' = 0$, so that the dominating scattering term occurs in the plane of incidence, as expected. Equation (4.31) can be further simplified by noting that

the term in the denominator of the square root involving ξ' will be small compared to the term involving ρ_0 and can be neglected. In addition, since we are interested in the scattering of the small protrusion, we can define a local coordinate system near the protrusion in ζ' such that $z' = \zeta' + z_2$. The electric field expression thus becomes,

$$\vec{E}(\bar{r}) = \frac{e^{ikr+i\pi/4}e^{-ik(\rho_0 \sin\theta + z_2 \cos\theta)}}{r}\sqrt{\frac{\rho_0}{8\pi k(\sin\theta + \sin\theta_i)}}\int_{C'} dl' e^{-ik\zeta' \cos\theta} e^{-ik\xi' \sin\theta} \left\{ i\omega\mu \left[\hat{\alpha}\hat{\alpha} + \hat{y}\hat{y}\right] \cdot \vec{J}'(\bar{r}') - ik\left[\hat{y}\hat{\alpha} - \hat{\alpha}\hat{y}\right] \cdot \vec{M}'(\bar{r}') \right\}. \quad (4.32)$$

Since the integral expression for the far-field only involves fields at the waterline cut of $\phi' = 0$, the surface integral has become a two-dimensional contour path integral. In order to evaluate the contour integral, the values of the primed electric and magnetic current must be known at each point along the path C'. Since the illumination phase delay has been accounted for analytically, the primed electric and magnetic currents can be approximated by forming an equivalent 2D problem via the tangent plane approximation. Under the tangent plane approximation, the cylinder under the small protrusion is replaced by a infinite ground plane to yield the two-dimensional problem shown in Figure 4-11. The 2D problem is then solved using the 2D FD-TD method (see Appendix C).

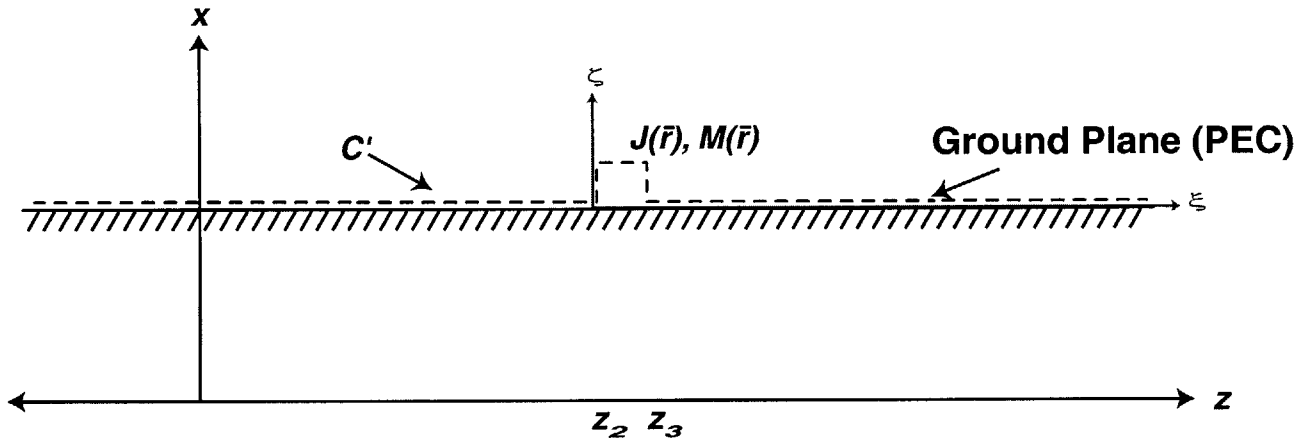

Figure 4-11: Approximate Equivalent 2D Problem for FD-TD/GO Hybrid Method

At this point the primed electric and magnetic currents are in terms of the scattered fields from both the cylinder and small protrusion. In order to predict the scattering from the

protrusion alone the reflected field from the cylinder is subtracted out.

$$E_{\text{scat}} = E_{\text{tot}} - E_{\text{inc}} - E_{\text{refl}} \tag{4.33}$$

The original electric and magnetic currents are then replaced by new equivalent currents, J_s' and M_s', which account only for the scattering due to the protrusion. Since the cylinder is being modeled as a ground plane near the protrusion, the reflected field can be found through an analytic solution. Also, since the path for the contour integral in (4.32) extends infinitely in both directions along the z–axis, it cannot be numerically evaluated. Image theory can be used to rewrite the integral in terms of a contour path that only extends around the small protrusion. Image theory states that the ground plane can be removed and replaced by image currents for each of the original currents. Along the ground plane, the tangential electric fields are zero and there is no magnetic current. The electric current along the ground plane, however, is not zero, but the image current is in the opposite direction and the two currents cancel. Hence, the only nonzero currents along the path C' will be where the path does not lie on the ground plane. If $J_{s,I}'$ and $M_{s,I}'$ are the image currents of J_s' and M_s', the electric field can be written as,

$$\begin{aligned}\vec{E}_s(\bar{r}) &= \frac{e^{ikr}e^{-ik(\rho_0\sin\theta+z_2\cos\theta)+i\pi/4}}{r}\sqrt{\frac{\rho_0}{8\pi k(\sin\theta+\sin\theta_i)}}\oint_{C''}dl'e^{-ik\zeta'\cos\theta}e^{-ik\xi'\sin\theta} \\ &\left\{i\omega\mu\left[\hat{\alpha}\hat{\alpha}+\hat{y}\hat{y}\right]\cdot\left[\vec{J}_s'(\bar{r}')+\vec{J}_{s,I}'(\bar{r}')\right]-ik\left[\hat{y}\hat{\alpha}-\hat{\alpha}\hat{y}\right]\cdot\left[\vec{M}_s'(\bar{r}')+\vec{M}_{s,I}'(\bar{r}')\right]\right\}\end{aligned} \tag{4.34}$$

where the closed loop path C'', shown in Figure 4-12, extends below the ξ–axis to the region where the image currents exist. Equation (4.34) is an expression for the far-field scattered electric field that arises from the protrusion alone and the interaction of the protrusion and the ground plane, but does not include the scattered field that arises from the ground plane alone. The radar cross section of the small protrusion alone is defined as

$$\sigma(\phi,\theta) = \lim_{r\to\infty} 4\pi r^2 \frac{|E_s(r,\phi,\theta)|^2}{|E_{inc}(r,\phi,\theta)|^2}. \tag{4.35}$$

In the limit as $r \to \infty$, the far-field expression for the scattered electric field can be used, so that the RCS is found to be

$$\sigma(\phi=0,\theta) = \frac{\rho_0}{2k(\sin\theta+\sin\theta_i)}\frac{|F(\theta)|^2}{|E_i|^2} \tag{4.36}$$

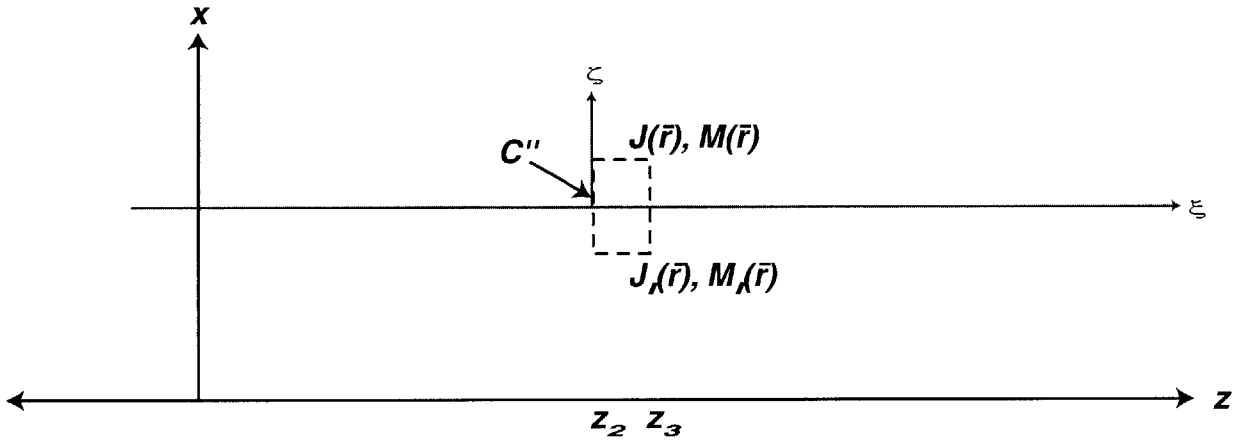

Figure 4-12: Equivalent problem with the ground plane removed. The contour integral is now a closed loop on the path C''.

where

$$F(\theta) = \oint_{C''} dl' e^{-ik\zeta' \cos\theta} e^{-ik\xi' \sin\theta} \left\{ i\omega\mu \left[\hat{\alpha}\hat{\alpha} + \hat{y}\hat{y}\right] \cdot \left[\vec{J}'(\bar{r}') + \vec{J}_I'(\bar{r}')\right] \right.$$
$$\left. -ik \left[\hat{y}\hat{\alpha} - \hat{\alpha}\hat{y}\right] \cdot \left[\vec{M}'(\bar{r}') + \vec{M}_I'(\bar{r}')\right] \right\}. \tag{4.37}$$

Since, the problem of modeling the small protrusion from the body of revolution has been reduced to a two dimensional problem, it is convenient to relate the three-dimensional RCS of the object on the BOR to the two-dimensional RCS of the protrusion's cross section. In the modeling to two-dimensional objects, the expression for the far-field electric field (See Appendix B) is given by,

$$\vec{E}(\bar{\rho}) = e^{ik\rho + i\pi/4} \sqrt{\frac{1}{8\pi k\rho}} \oint_{C'} e^{-ik(x' \sin\theta + z' \cos\theta)} \left\{ i\omega\mu \left[\hat{\alpha}\hat{\alpha} + \hat{y}\hat{y}\right] \cdot \vec{J}(\bar{\rho}') \right.$$
$$\left. - ik \left[\hat{\alpha}\hat{y} - \hat{y}\hat{\alpha}\right] \cdot \vec{M}(\bar{\rho}') \right\}. \tag{4.38}$$

Equation (4.38) is very similar to the expression for the electric field given in (4.32). One important difference is the phase factor $-ik(\rho_0 \sin\theta + z_2 \cos\theta)$ which accounts for the phase difference between the wave reflected from the small protrusion and the wave reflected from the large cylinder. Although this phase factor does not affect the RCS of the protrusion alone, it must be considered when coherently adding the RCS of the protrusion and the large cylinder.

If a ground plane is present, its effect can be accounted for, as before, using image theory, so that the expression now becomes,

$$\begin{aligned}\vec{E}(\overline{\rho}) &= e^{ik\rho}\sqrt{\frac{-i}{8\pi k\rho}}\oint_{C''} e^{-ik(x'\sin\theta+z'\cos\theta)}\left\{i\omega\mu\left[\hat{\alpha}\hat{\alpha}+\hat{y}\hat{y}\right]\cdot\left[\vec{J}(\overline{\rho}')+\vec{J}_I(\overline{\rho}')\right]\right. \\ &\quad \left. - ik\left[\hat{\alpha}\hat{y}-\hat{y}\hat{\alpha}\right]\cdot\left[\vec{M}(\overline{\rho}')+\vec{M}_I(\overline{\rho}')\right]\right\}. \end{aligned} \tag{4.39}$$

Using the definition of two-dimensional radar cross section,

$$\sigma(\theta) = \lim_{\rho\to\infty} 2\pi\rho\,\frac{|E_{scat}(\rho,\theta)|^2}{|E_{inc}(\rho,\theta)|^2} \tag{4.40}$$

the 2D RCS is found to be,

$$\sigma = \frac{1}{4k}\frac{|F(\theta)|^2}{|E_i|^2} \tag{4.41}$$

where

$$\begin{aligned}F(\theta) &= \oint_C' dl' e^{-ikx\sin\theta}e^{-ikz'\cos\theta}\left\{i\omega\mu[\hat{\alpha}\hat{\alpha}+\hat{y}\hat{y}]\cdot[J(\overline{r}')+J_I(\overline{r}')]\right. \\ &- \left. ik[\hat{y}\hat{\alpha}-\hat{\alpha}\hat{y}]\cdot[M(\overline{r}')+M_I(\overline{r}')]\right\}. \end{aligned} \tag{4.42}$$

Noting the similarities between (4.36) and (4.41), the following expression can be used to convert from two dimensional RCS to three dimensional RCS for the small protrusion on an electrically large conducting cylinder.

$$\sigma_{3D} = \frac{2r_0}{\sin\theta+\sin\theta_i}\sigma_{2D}. \tag{4.43}$$

Note, in this equation θ and θ_i must be large enough for the stationary phase approximation to be vald or equivalently that $k\rho_0(\sin\theta+\sin\theta_i)$ is large. The formula can be further simplified for backscatter RCS as,

$$\sigma_{3D} = \frac{r_0}{\sin\theta}\sigma_{2D} \tag{4.44}$$

so that at broadside incidence, $\theta = 90°$, the three-dimensional backscatter RCS of the protrusion alone is simply the product of the radius of the large cylinder and the two-dimensional

backscatter RCS of the protrusion's cross section.

4.3.2 Geometrical Optics Solution for RCS of Cylinder

In order to obtain the RCS of the entire target, cylinder and protrusion combined, the RCS of the cylinder is needed. Because the cylinder is assumed to be large compared to wavelength, high frequency techniques can be used to determine the RCS. Once the RCS of the cylinder and the small protrusion have been obtain, they can be combined to find the total RCS.

The geometrical optics solution for the bistatic radar cross section from an elliptic cylinder is given by [23],

$$\sigma(\theta_s, \theta_i, \phi_s, \phi_i) = \frac{a^2 b^2 \lambda \left| e^{ikDL} - 1 \right|^2}{\pi D^2 \left[(Aa)^2 + (Bb)^2 \right]^{3/2}} \left(G_1^2 + G_2^2 + G_3^2 \right) \tag{4.45}$$

where

$$\begin{aligned}
G_1 &= A\left(a_y \sin\theta_s \sin\phi_s + a_z \cos\theta_s\right) - B\left(a_x \sin\theta_s \sin\phi_s\right) \\
G_2 &= a_z \sin\theta_s \left(A \cos\phi_s + B \sin\phi_s\right) \\
G_3 &= B\left(a_x \sin\theta_s \cos\phi_s + a_z \cos\theta_s\right) - A\left(a_y \sin\theta_s \cos\phi_s\right) \\
A &= \sin\theta_i \cos\phi_i + \sin\theta_s \cos\phi_s \\
B &= \sin\theta_i \sin\phi_i + \sin\theta_s \sin\phi_s \\
D &= \cos\theta_i + \cos\theta_s \\
L &= \text{length of the elliptic cylinder} \\
a &= \text{semi-major axis} \\
b &= \text{semi-minor axis} \\
a_x, a_y, a_z &= \text{the } x, \text{ } y, \text{ and } z \text{ components of the polarization vector} \\
\lambda &= 2\pi/k = \text{the wavelength.}
\end{aligned}$$

For the special case of backscattering from a circular cylinder when $\phi = 0$, equation (4.45) becomes,

$$\sigma(\theta) = \frac{a \sin\theta \left| e^{ikDL} - 1 \right|^2}{4k \cos^2\theta} \tag{4.46}$$

where a is the radius of the cylinder. At angles that correspond to broadside incidence and

scattering the GO formula is very accurate, however, as the angle θ approaches 0, the solution becomes less accurate.

Due to the simplicity of the above formula, the GO solution for the backscattering from a cylinder is used in the following sections. If, however, more accuracy is desired, it is also possible to use the hybrid formulation with other high frequency techniques such as physical optics and the physical theory of diffraction. By using these methods, the monostatic signature of the cylinder can be more accurately obtained thereby increasing the accuracy of modeling the cylinder with the small protrusion.

4.4 Results of the FD-TD/GO Hybrid Method

As discussed above, the contribution of the protrusion to the RCS of the entire object can be approximated by solving a two-dimensional scattering problem. In this work, the 2D scattering problem is solved via the 2D FD-TD method (See Appendix C). The 2D FD-TD method is used to calculate the 2D RCS of the two dimensional cross section of the protrusion on an infinite ground plane. Once the 2D RCS of the protrusion alone is known, it can be combined with the geometrical optics solution of the cylinder using the methodology described in the previous section to obtain the total RCS. In addition, the relative phase of each scatterer must be included to account for the different spatial location of each of the scatterers. In the following two sections, the monostatic signature at 2 GHz of two different sized cylinders with the same small protrusion is computed using the FD-TD/GO Hybrid method.

The first cylinder modeled, shown in Figure 4-13, has a radius of $a = 25$ cm, so that at 2 GHz, $ka \approx 10$. Typically, geometrical optics solutions are valid for values of $ka > 20$ [23], so the results obtained are not expected to be very accurate.

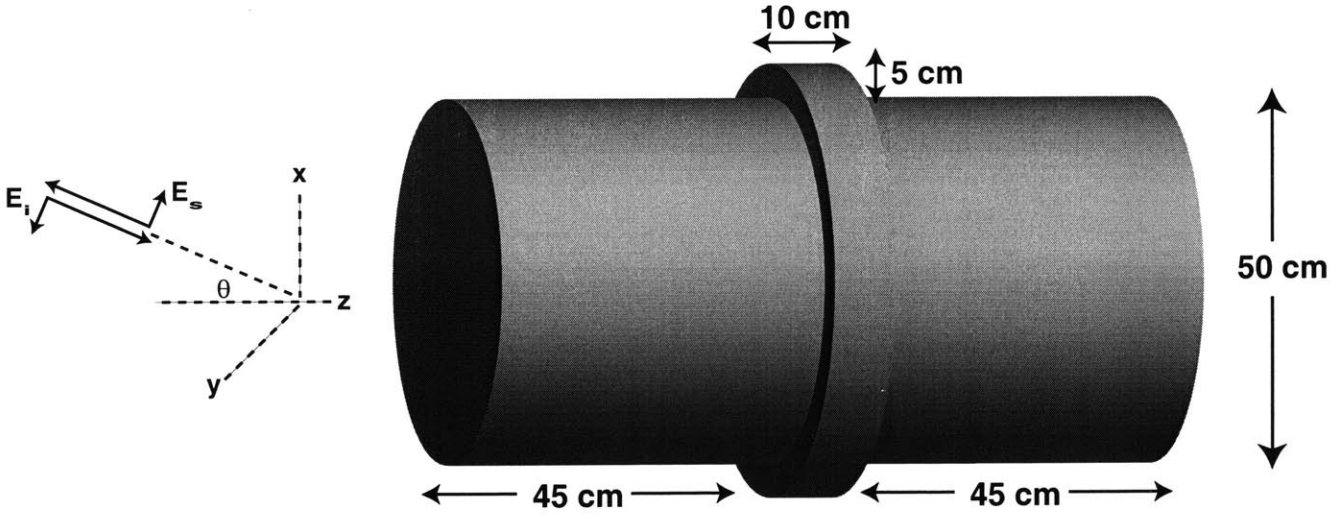

Figure 4-13: Geometry for cylinder with ring.

As shown in Figure 4-14, errors introduced by the geometrical optics assumptions are apparent. For example, at $\theta = 90°$, the RCS is overestimated by about 4 dBsm. In addition, although the peak amplitudes of the side lobes are captured to some degree, the widths of lobes predicted by the hybrid method are much smaller than those of the exact MoM solution.

As shown in the plots are the results obtained by the PO/PTD method, which are in good agreement with the BOR MoM predictions for angles near broadside incidence. As the aspect angle moves away from broadside incidence, however, the PO/PTD predictions becomes less accurate. This indicates that the effect of the protrusions is not significant for angles near broadside incidence, and for angles where the small protrusion does play a significant role, the PO/PTD does not accurately model its effect.

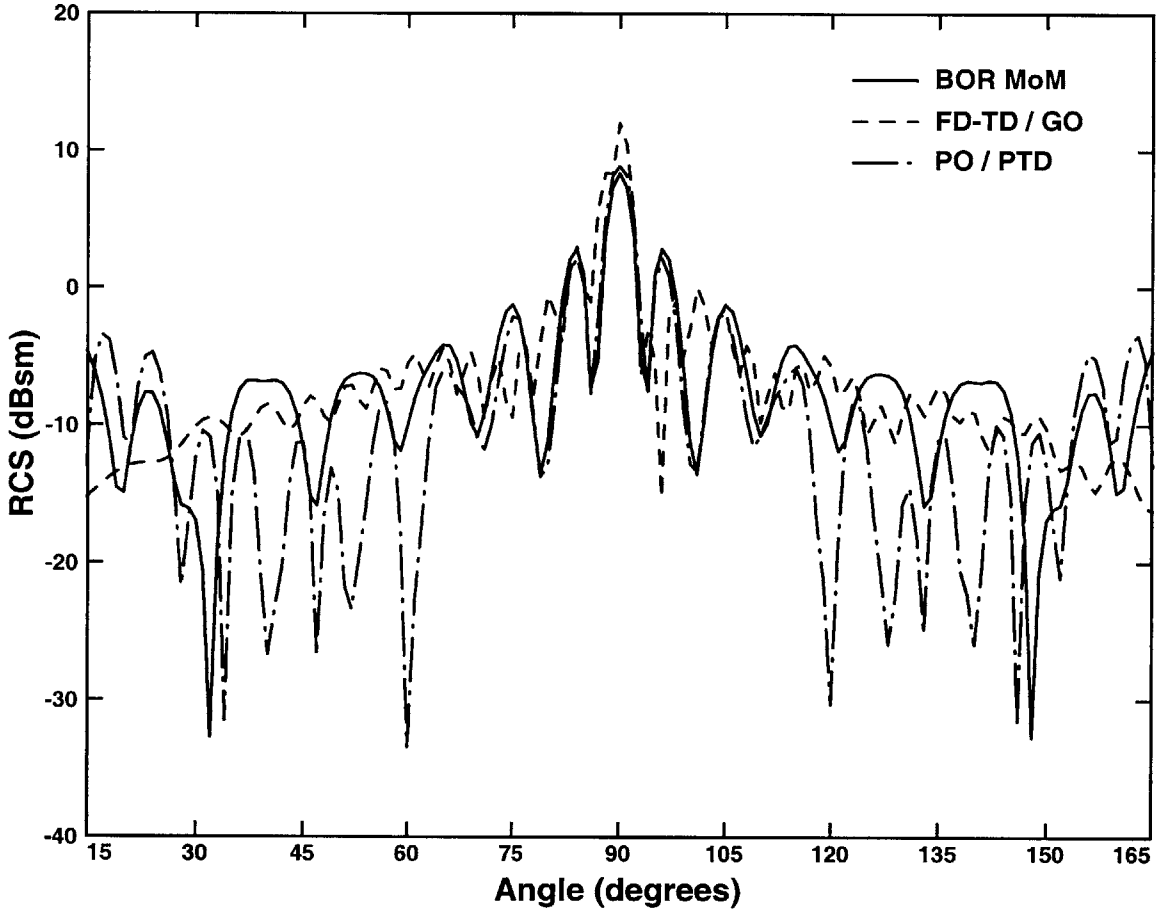

Figure 4-14: Hybrid FD-TD/GO monostatic RCS predictions of cylinder with ring is compared to MoM predictions. Shown is VV polarization at 2 GHz for a cut in θ with $\phi = 0°$.

Still, the hybrid method correctly models the amplitude, on average, of exact RCS predictions. However, as mentioned previously, it does not accurately model the widths of the side lobes. One possible reason is that the geometrical optics solution for the scattering due to the cylinder alone is not very accurate for a cylinder of this size. In the next section, a larger cylinder with the same protrusion as above, is modeled.

As the previous example demonstrated, in order to apply the hybrid method, the overall size of the target must be larger. In the next example considered, the cylinder, shown in Figure 4-15 is chosen to have a radius $a = 75$ cm. With this radius at 2 GHz, $ka \approx 30$, which implies

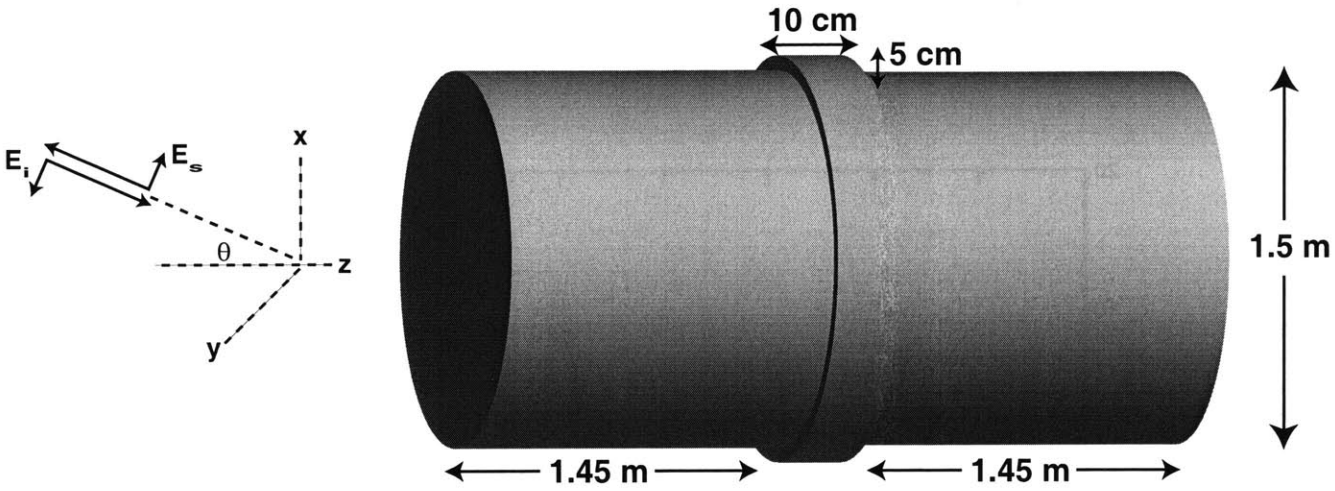

Figure 4-15: Geometry for larger cylinder with ring.

that the geometrical optics solution should be valid. Although, the size of the cylinder has increased, the effect of the protrusion should still be evident for non-broadside aspect angles. As evidenced by Figure 4-16, the hybrid technique yields accurate results for angles up to 45° from broadside incidence, approximately matching both the amplitude and width of side lobes. Also shown in the plot is the PO/PTD prediction for this geometry. As expected, at angles near broadside incidence the PO/PTD method accurately predicts the monostatic signature, however, as the aspect angle departs from broadside incident the PO/PTD predictions become less accurate. This is due to the fact the PO/PTD method cannot accurately model an object of the protrusion's size.

On the basis of the two previous examples, it is clear that the hybrid method is effective in capturing the effect of the small protrusion. One possible approach for improving the accuracy of the hybrid method is use a more accurate high-frequency model of the scattering from the cylinder alone.

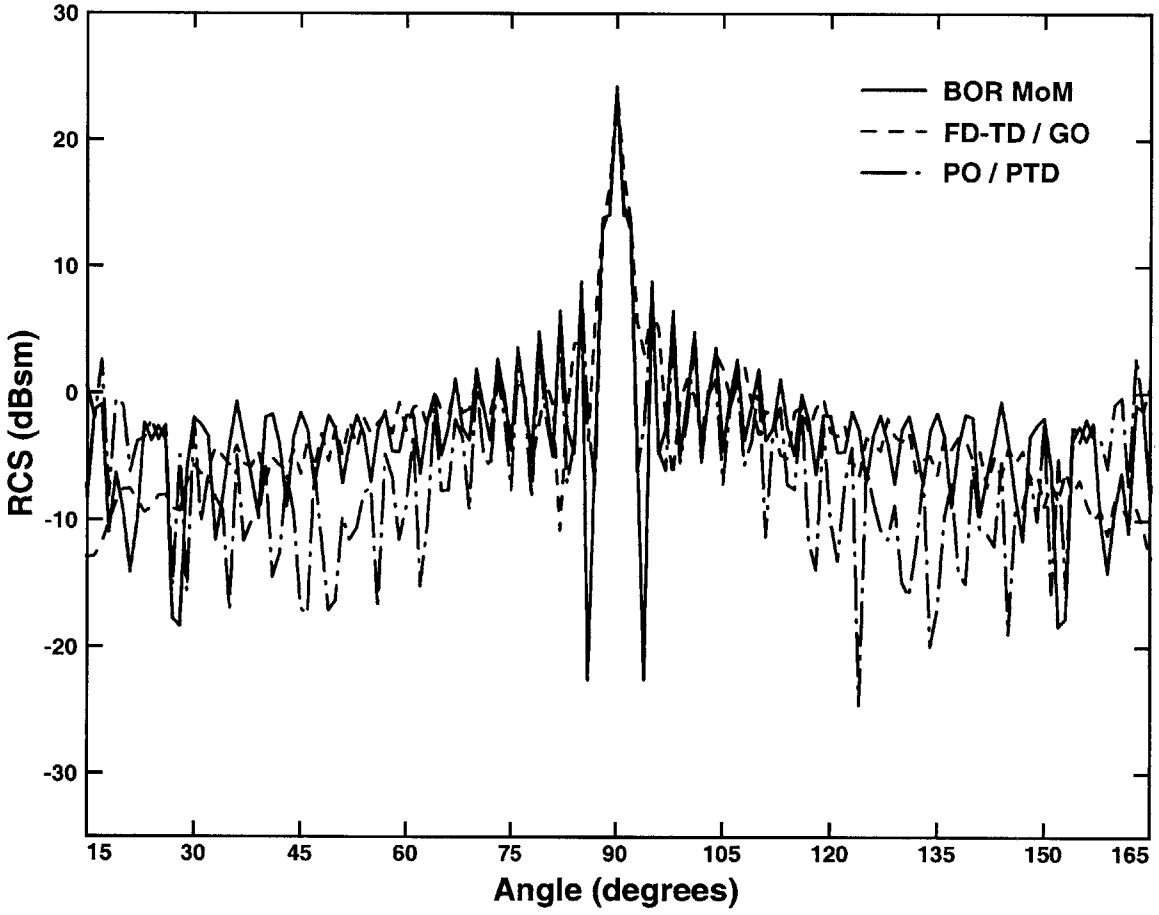

Figure 4-16: Hybrid FD-TD/GO monostatic RCS predictions of larger cylinder with ring is compared to MoM predictions. Shown is VV polarization at 2 GHz for a cut in θ with $\phi = 0°$.

4.5 Summary

In this chapter, two methods for reducing the computational burden associated with computing the RCS of large targets have been presented. The first method reduced the computational burden associated with computing the monostatic signature over a broadband of frequencies. In contrast, the second approach reduced computational requirements for BOR objects of large electrical radius by using a hybrid FD-TD and Geometrical Optics formulation.

In applying the first method, a single FD-TD BOR simulation was used to calculate the monostatic signature for one incident angle, as well as bistatic signatures for adjacent observation directions. The bistatic equivalence theorem was then used to approximate monostatic signatures for other angles near the incident direction of the actual FD-TD BOR simulation. The principle was applied to the monostatic RCS prediction a simple cylinder and a biconical shaped object. In the case of the cylinder, only two BOR FD-TD simulations were required to obtain accurate monostatic signature estimates. In the modeling of the biconical target, however, the bistatic signatures for several incident angles were required to accurately estimate the monostatic signature. Still, in comparison to the PO/PTD method, where predictions were only accurate for aspect angles near broadside and backend, the equivalence principle's estimates were more accurate overall.

The FD-TD/GO method was applied to determining the effect of a small BOR protrusion on a large cylinder. The scattering from the protrusion was modeled using the two-dimensional FD-TD method, while the scattering from the large cylinder was calculated using Geometrical Optics. As shown in the two targets modeled, the hybrid method was shown to have an accuracy advantage over the PO/PTD method since it was able to capture the effect of the small protrusion while the PO/PTD method was not able to do so. Moreover, because only the two-dimensional cross section of the small protrusion is modeled rigorously, the computational requirements are small compared to applying an exact technique to the full target, which would otherwise be necessary since, as shown, high-frequency techniques cannot be applied.

While the hybrid method was applied to the scattering from a large cylinder with a small protrusion, the method could in general be applied to other large body of revolution targets under the following two conditions. The first condition is that the interaction between the endcaps of the large BOR target and the small protrusion be small. The second condition is that the radius of the target near the protrusion must be large so that the saddle point method

and tangent plane approximation can be used. In order to determine the RCS of the overall target, however, an accurate model must be available for the large BOR target. If the target is a simple large shape, such as a cylinder or cone, PO/PTD predictions should be sufficient.

Chapter 5

RCS Prediction Using the BOR Parabolic Wave Equation Method

In the previous chapters, both exact and approximate techniques for predicting the radar cross section of body of revolution objects were described. Although the BOR FD-TD method can provide accurate RCS predictions, it is requires a large amount of memory and computation time; on the other hand, the GO/FD-TD technique requires less computation time and memory, but it can only be applied to limited geometries. Clearly, a more robust and accurate, yet computationally inexpensive technique, is needed to accurately model large body of revolution targets. One possible approach described in this chapter is the application of the paraxial approximation to the modeling of scattering from body of revolution objects. As with the BOR FD-TD technique, the fields are decomposed into a Fourier series in ϕ reducing the three dimensional problem to a sequence of independent two dimensional problems. In order to further simplify the computation, electric fields are assumed to be composed of a explicit fast phase factor and a slowly varying envelope function. The assumed form of the electric field is then substituted into the time-harmonic vector wave equation so as to obtain a new wave equation in terms of the slowly varying envelope functions. Because the new field variables are slowly varying, higher order derivatives with respect to range are neglected, reducing the vector wave equation to a set of coupled parabolic partial differential equations. These equations can then be solved using an efficient marching in space approach, so that the memory requirement for the method is one-dimensional.

5.1 Time-Harmonic Vector Wave Equation

The first step in applying the PWE technique involves writing the vector wave equation in a form appropriate for modal decomposition. The time-harmonic wave equation, derived from Maxwell's equations is (e^{-iwt} convention),

$$\nabla^2 \vec{E}(r) + k^2 \vec{E}(r) = 0 \tag{5.1}$$

where E is the time-harmonic electric field and k is the wave number. In the Cartesian coordinate system, the electric field is of the form,

$$\vec{E} = \hat{x}E_x(x,y,z) + \hat{y}E_y(x,y,z) + \hat{z}E_z(x,y,z). \tag{5.2}$$

Because the unit vectors, $\hat{x}$, $\hat{y}$, and $\hat{z}$ are independent of position, the vector wave equation can be separated into the following three scalar equations.

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} + k^2\right) E_x = 0 \tag{5.3}$$

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} + k^2\right) E_y = 0 \tag{5.4}$$

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} + k^2\right) E_z = 0 \tag{5.5}$$

In the modeling of body of revolution objects, the cylindrical coordinate system is used where the electric field is of the form,

$$\vec{E}(r) = \hat{\rho}E_\rho(\rho,\phi,z) + \hat{\phi}E_\phi(\rho,\phi,z) + \hat{z}E_z(\rho,\phi,z). \tag{5.6}$$

Unlike the Cartesian coordinate unit vectors, two of the cylindrical coordinate unit vectors, $\hat{\rho}$ and $\hat{\phi}$ are not independent of position so that,

$$\begin{aligned} \nabla^2(\hat{\rho}E_\rho) &\neq \hat{\rho}\nabla^2 E_\rho \\ \nabla^2(\hat{\phi}E_\phi) &\neq \hat{\phi}\nabla^2 E_\phi. \end{aligned} \tag{5.7}$$

Consequently, the vector wave equation can not be reduced to three independent scalar equations as in the Cartesian coordinate system. In order to reduce the vector wave equation to a set

of scalar equations in cylindrical coordinates, it is first necessary to rewrite the wave equation in its alternate form,

$$\nabla(\nabla \cdot \vec{E}) - \nabla \times \nabla \times \vec{E} = -k^2 \vec{E} \tag{5.8}$$

where the vector identity,

$$\nabla^2 \vec{E} = \nabla(\nabla \cdot \vec{E}) - \nabla \times \nabla \times \vec{E} \tag{5.9}$$

was used. The alternate form of the vector wave equation can now be expanded and reduced to the following three scalar partial differential equations,

$$\nabla^2 E_\rho + \left(-\frac{E_\rho}{\rho^2} - \frac{2}{\rho^2} \frac{\partial E_\phi}{\partial \phi} \right) = -k^2 E_\rho \tag{5.10}$$

$$\nabla^2 E_\phi + \left(-\frac{E_\phi}{\rho^2} + \frac{2}{\rho^2} \frac{\partial E_\rho}{\partial \phi} \right) = -k^2 E_\phi \tag{5.11}$$

$$\nabla^2 E_z = -k^2 E_z \tag{5.12}$$

where

$$\nabla^2 \psi = \frac{1}{\rho} \frac{\partial}{\partial \rho} \left(\rho \frac{\partial \psi}{\partial \rho} \right) + \frac{1}{\rho^2} \frac{\partial^2 \psi}{\partial \phi^2} + \frac{\partial^2 \psi}{\partial z^2}. \tag{5.13}$$

Unlike (5.3)–(5.5), which are independent of each other, equations (5.10) and (5.11) are coupled and must be solved simultaneously. However, equation (5.12) is not coupled to equations (5.10) and (5.11), and it can be solved independently.

In the following section, equations (5.10)-(5.12) will be simplified to a set of coupled parabolic partial differential equations by assuming propagation along the z axis. It is also possible to to assume propagation along a different direction, but because of the additional complications involved, the off-axis scattering formulation will not be discussed. In the Cartesian coordinate system, additional paraxial direction formulations can effectively be obtained by simply rotating the object being modeled and the direction of incident wave. However, a body of revolution object rotated in the cylindrical coordinate system will no longer be symmetric about the z–axis, which implies that the fields can not be decomposed into a Fourier series in ϕ. Thus, the parabolic version of Maxwell's equations must be reformulated for each paraxial direction in order to maintain the axial symmetry of the object being modeled.

5.2 PWE Formulation for On-Axis Scattering

5.2.1 Paraxial Approximation

Before describing the derivation of the paraxial approximation for the BOR case, it is useful to review the derivation in the Cartesian case. In the Cartesian coordinate system, the paraxial version of Maxwell's equations are derived by assuming the following form of the electric field [34, 36, 38, 59].

$$\vec{E}(x,y,z) = e^{ikz}\vec{\psi}(x,y,z) \tag{5.14}$$

where $\vec{\psi}$ is the slowly varying envelope function associated with the electric field, $\vec{E}$. The choice of the above definition of $\vec{\psi}$ defines the paraxial, or range, direction to be in the $\hat{z}$ direction. In this case, the envelope function $\vec{\psi}$ will be slowly varying in range for energy propagating close to the paraxial direction. Substituting the above definition into the vector wave equation yields three independent scalar equations. For example, the equation governing the $\hat{x}$ component of $\vec{\psi}$ is,

$$\frac{\partial^2\psi_x}{\partial x^2} + \frac{\partial^2\psi_x}{\partial y^2} + \frac{\partial^2\psi_x}{\partial z^2} + 2ik\frac{\partial\psi_x}{\partial z} = 0. \tag{5.15}$$

Similar equations exist that govern the $\hat{y}$ and $\hat{z}$ components of $\vec{\psi}$. The above equation can then be factored into two equations, one representing energy propagating in the forward paraxial direction, and the other representing the backward propagating energy,

$$\left[\frac{\partial}{\partial z} + ik(1-Q)\right]\left[\frac{\partial}{\partial z} + ik(1+Q)\right]\psi_x = 0 \tag{5.16}$$

where

$$Q = \sqrt{\frac{1}{k^2}\frac{\partial^2}{\partial x^2} + \frac{1}{k^2}\frac{\partial^2}{\partial y^2} + 1}. \tag{5.17}$$

The equation representing the forward scattering energy will be

$$\left[\frac{\partial}{\partial z} + ik(1-Q)\right]\psi_x = 0 \tag{5.18}$$

The forward scattering equation can be further simplified by approximating the square root in the Q operator as a two term Taylor series.

$$Q \approx 1 + \frac{1}{2k^2}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) \tag{5.19}$$

Under this approximation, the forward scattering equation reduces to the standard parabolic equation (SPE), which is a narrow-angle approximation very accurate for angles within 15° of the paraxial direction [59].

$$\frac{\partial \psi_x}{\partial z} = \frac{i}{2k}\left(\frac{\partial^2 \psi_x}{\partial x^2} + \frac{\partial^2 \psi_x}{\partial y^2}\right) \tag{5.20}$$

Note that the SPE can also be derived from (5.15) by making the paraxial approximation where the $\partial^2\psi/\partial z^2$ term is assumed to be very small and is neglected.

In the formulation of the BOR PWE method, the paraxial direction for the scattered electric field is taken to be along the axis of symmetry in the $\pm\hat{z}$ direction. In addition, in order to exploit the axial symmetry, the electric field is decomposed into a Fourier series in ϕ so that the form of the electric field is,

$$\vec{E} \;=\; e^{\pm ikz}\sum_{m=0}^{N} \vec{\psi}_{m,u}(\rho,z)\cos m\phi + \vec{\psi}_{m,v}(\rho,z)\sin m\phi. \tag{5.21}$$

Substituting (5.21) into the three scalar wave equations, (5.10)-(5.12), and utilizing orthogonality, yields the following set of modal equations.

$$\frac{\partial^2 \psi^{\rho}_{m,u}}{\partial \rho^2} + \frac{1}{\rho}\frac{\partial \psi^{\rho}_{m,u}}{\partial \rho} + \frac{\partial^2 \psi^{\rho}_{m,u}}{\partial z^2} \pm 2ik\frac{\partial \psi^{\rho}_{m,u}}{\partial z} - \left(\frac{m^2+1}{\rho^2}\right)\psi^{\rho}_{m,u} - \frac{2m}{\rho^2}\psi^{\phi}_{m,v} = 0 \tag{5.22}$$

$$\frac{\partial^2 \psi^{\phi}_{m,v}}{\partial \rho^2} + \frac{1}{\rho}\frac{\partial \psi^{\phi}_{m,v}}{\partial \rho} + \frac{\partial^2 \psi^{\phi}_{m,v}}{\partial z^2} \pm 2ik\frac{\partial \psi^{\phi}_{m,v}}{\partial z} - \left(\frac{m^2+1}{\rho^2}\right)\psi^{\phi}_{m,v} - \frac{2m}{\rho^2}\psi^{\rho}_{m,u} = 0 \tag{5.23}$$

$$\frac{\partial^2 \psi^{z}_{m,u}}{\partial \rho^2} + \frac{1}{\rho}\frac{\partial \psi^{z}_{m,u}}{\partial \rho} + \frac{\partial^2 \psi^{z}_{m,u}}{\partial z^2} \pm 2ik\frac{\partial \psi^{z}_{m,u}}{\partial z} - \frac{m^2}{\rho^2}\psi^{z}_{m,u} = 0 \tag{5.24}$$

$$\frac{\partial^2 \psi^{\rho}_{m,v}}{\partial \rho^2} + \frac{1}{\rho}\frac{\partial \psi^{\rho}_{m,v}}{\partial \rho} + \frac{\partial^2 \psi^{\rho}_{m,v}}{\partial z^2} \pm 2ik\frac{\partial \psi^{\rho}_{m,v}}{\partial z} - \left(\frac{m^2+1}{\rho^2}\right)\psi^{\rho}_{m,v} + \frac{2m}{\rho^2}\psi^{\phi}_{m,u} = 0 \tag{5.25}$$

$$\frac{\partial^2 \psi^{\phi}_{m,u}}{\partial \rho^2} + \frac{1}{\rho}\frac{\partial \psi^{\phi}_{m,u}}{\partial \rho} + \frac{\partial^2 \psi^{\phi}_{m,u}}{\partial z^2} \pm 2ik\frac{\partial \psi^{\phi}_{m,u}}{\partial z} - \left(\frac{m^2+1}{\rho^2}\right)\psi^{\phi}_{m,u} + \frac{2m}{\rho^2}\psi^{\rho}_{m,v} = 0 \tag{5.26}$$

$$\frac{\partial^2 \psi^{z}_{m,v}}{\partial \rho^2} + \frac{1}{\rho}\frac{\partial \psi^{z}_{m,v}}{\partial \rho} + \frac{\partial^2 \psi^{z}_{m,v}}{\partial z^2} \pm 2ik\frac{\partial \psi^{z}_{m,v}}{\partial z} - \frac{m^2}{\rho^2}\psi^{z}_{m,v} = 0 \tag{5.27}$$

As with the BOR FD-TD method, the modal equations separate into two decoupled sets of equations. The first set, equations (5.22)–(5.24), contain the fields excited by a horizontally polarized plane wave, and the second set, equations (5.25)–(5.27), contain the fields excited by a vertically polarized plane wave.

Because of the coupling between the ψ^ρ and ψ^ϕ fields, a factorization similar to that done in the derivation of the SPE is not possible, however, the paraxial approximation can still be used. Under the paraxial approximation, the $\partial^2/\partial z^2$ terms are neglected, which reduces the scalar wave equations, (5.22)-(5.27), to the following.

$$\frac{\partial^2 \psi^\rho_{m,u}}{\partial \rho^2} + \frac{1}{\rho}\frac{\partial \psi^\rho_{m,u}}{\partial \rho} \pm 2ik\frac{\partial \psi^\rho_{m,u}}{\partial z} - \left(\frac{m^2+1}{\rho^2}\right)\psi^\rho_{m,u} - \frac{2m}{\rho^2}\psi^\phi_{m,v} = 0 \tag{5.28}$$

$$\frac{\partial^2 \psi^\phi_{m,v}}{\partial \rho^2} + \frac{1}{\rho}\frac{\partial \psi^\phi_{m,v}}{\partial \rho} \pm 2ik\frac{\partial \psi^\phi_{m,v}}{\partial z} - \left(\frac{m^2+1}{\rho^2}\right)\psi^\phi_{m,v} - \frac{2m}{\rho^2}\psi^\rho_{m,u} = 0 \tag{5.29}$$

$$\frac{\partial^2 \psi^z_{m,u}}{\partial \rho^2} + \frac{1}{\rho}\frac{\partial \psi^z_{m,u}}{\partial \rho} \pm 2ik\frac{\partial \psi^z_{m,u}}{\partial z} - \frac{m^2}{\rho^2}\psi^z_{m,u} = 0 \tag{5.30}$$

$$\frac{\partial^2 \psi^\rho_{m,v}}{\partial \rho^2} + \frac{1}{\rho}\frac{\partial \psi^\rho_{m,v}}{\partial \rho} \pm 2ik\frac{\partial \psi^\rho_{m,v}}{\partial z} - \left(\frac{m^2+1}{\rho^2}\right)\psi^\rho_{m,v} + \frac{2m}{\rho^2}\psi^\phi_{m,u} = 0 \tag{5.31}$$

$$\frac{\partial^2 \psi^\phi_{m,u}}{\partial \rho^2} + \frac{1}{\rho}\frac{\partial \psi^\phi_{m,u}}{\partial \rho} \pm 2ik\frac{\partial \psi^\phi_{m,u}}{\partial z} - \left(\frac{m^2+1}{\rho^2}\right)\psi^\phi_{m,u} + \frac{2m}{\rho^2}\psi^\rho_{m,v} = 0 \tag{5.32}$$

$$\frac{\partial^2 \psi^z_{m,v}}{\partial \rho^2} + \frac{1}{\rho}\frac{\partial \psi^z_{m,v}}{\partial \rho} \pm 2ik\frac{\partial \psi^z_{m,v}}{\partial z} - \frac{m^2}{\rho^2}\psi^z_{m,v} = 0 \tag{5.33}$$

The above six equations represent the paraxial modal version of Maxwell's equations. It should be noted that solutions that satisfy the above equations do not exactly satisfy Maxwell's equations. With the paraxial approximation used, the calculated fields will be accurate within 15° of the paraxial direction. In addition, the paraxial approximation breaks down when energy scattered by the object undergoes large changes in direction. For example, the PWE method does not perform well in the modeling of non-convex objects and cavities [59] where multiple scattering interactions can occur. Another difficulty involves the modeling of objects small compared to a wavelength where creeping waves can travel all the way around the object. Creeping waves that travel around the object more than once can not be captured with the PWE method due to the one-way nature of the technique.

5.2.2 Boundary Conditions for a PEC

In order to model the scattering from a perfect electric conductor (PEC), the boundary condition given in (2.51) is used.

$$\hat{n} \times \vec{E} = 0 \tag{5.34}$$

Due to the linearity of Maxwell's equations, the total electric field can be split into a scattered field component and incident field component such that,

$$E_{\text{total}} = E_{\text{scat}} + E_{\text{inc}} \tag{5.35}$$

where both the E_{scat} and E_{inc} components must independently satisfy Maxwell's equations. In the PWE formulation, the parabolic wave equations are written in terms of scattered fields rather than total fields. As shown in Figure 5-1, this enables the independent specification of the paraxial and incident wave directions. If the parabolic wave equations were written in

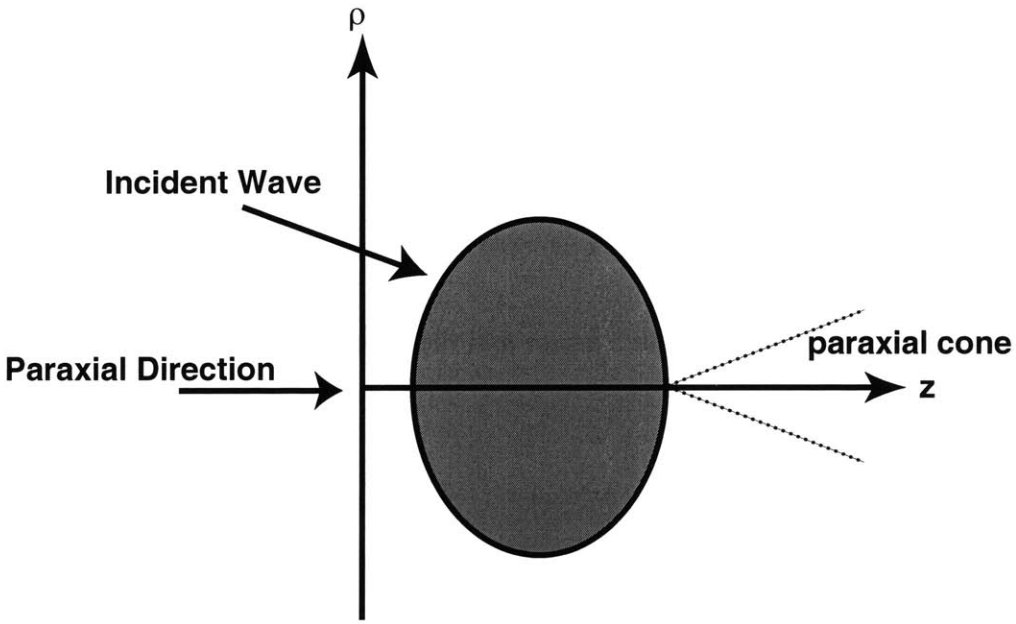

Figure 5-1: BOR PWE Paraxial Direction and Incidence Direction

terms of the total fields, a plane wave would need to be propagated towards the object being modeled. In order for an accurate representation of the wave to reach the object, the incident and paraxial direction would need to be in the same direction. The scattered field formulation removes this restriction through the use of the following boundary conditions.

Defining the ψ variables in (5.28)–(5.33) to be scattered fields, the boundary conditions for a PEC can now be written as,

$$\hat{n} \times \vec{E} = \hat{n} \times \left[\vec{E}^i + \vec{E}^s\right] = \hat{n} \times \left[\vec{E}^i + e^{\pm ikz}\vec{\Psi}\right] = 0 \tag{5.36}$$

where $\vec{E}^i$ is the incident field, $\vec{E}^s$ is the scattered field, and $\vec{\Psi}$ is the envelope function for the scattered field. Assuming that both the E^i and Ψ fields can be decomposed as,

$$\vec{E}^i = \sum_{m=0}^{N} \vec{e}^{\,i}_{m,u} \cos m\phi + \vec{e}^{\,i}_{m,v} \sin m\phi \tag{5.37}$$

$$\vec{\Psi} = \sum_{m=0}^{N} \vec{\psi}_{m,u} \cos m\phi + \vec{\psi}_{m,v} \sin m\phi \tag{5.38}$$

and utilizing orthogonality, independent boundary conditions for each of the Fourier mode components can be written.

$$\hat{n} \times \left[\vec{e}^{\,i}_{m_{u,v}} + e^{\pm ikz} \vec{\psi}_{m_{u,v}} \right] = 0 \tag{5.39}$$

Since the boundary conditions will be the same for both the sine and cosine Fourier components, the m, u, and v subscripts will be omitted. In the modeling of body of revolution objects, the normal to the object's surface can be written in general as,

$$\hat{n} = -\hat{z} \cos\alpha + \hat{\rho} \sin\alpha \tag{5.40}$$

where α is the angle between the unit $\hat{z}$ vector and the surface normal vector. Expanding (5.39) with (5.40),

$$\begin{aligned} (-\hat{z} \cos\alpha + \hat{\rho} \sin\alpha) \times \left(\hat{\rho} e^t_\rho + \hat{\phi} e^t_\phi + \hat{z} e^t_z \right) &= 0 \\ -\hat{\phi} \cos\alpha e^t_\rho + \hat{\rho} \cos\alpha e^t_\phi + \hat{z} \sin\alpha e^t_\phi - \hat{\phi} \sin\alpha e^t_z &= 0 \end{aligned} \tag{5.41}$$

where $\vec{e}^{\,t} = \vec{e}^{\,i} + e^{\pm ikz} \vec{\psi}$. Equation (5.41) can be separated into the following two scalar equations.

$$\begin{aligned} \cos\alpha \psi_\rho + \sin\alpha \psi_z &= -e^{\mp ikz} \left[\cos\alpha e^i_\rho + \sin\alpha e^i_z \right] \\ \psi_\phi &= -e^{\mp ikz} e^i_\phi \end{aligned} \tag{5.42}$$

From (5.42), it is clear that if the incident wave propagates in the direction of the paraxial direction, the exponential phase factor in the boundary condition becomes zero. As the angle between the incident wave and paraxial direction increase, the exponential phase factors increases to a maximum of $2ikz$ in the case of backscatter where the incident wave propagates in the exact opposite direction of the paraxial direction. In order to accurately represent this

phase variation, smaller step sizes in z are needed as the angle between the incident wave and paraxial direction increases. Consequently, backscatter calculations require more computation time, since the total number of range steps increases with the smaller step sizes.

The boundary condition in (5.42) forms a linear system with two equations in terms of three variables. Hence, an additional equation is needed in order to ensure a unique solution. This is provided by the divergence-free condition of Maxwell's equations,

$$\nabla \cdot \left[\vec{E}^i + \vec{E}^s\right] = \underbrace{\nabla \cdot \vec{E}^i}_{=0} + \nabla \cdot \vec{E}^s = \nabla \cdot \left[e^{\pm ikz}\vec{\Psi}\right] = 0 \tag{5.43}$$

where it is assumed the incident field component of the total electric field satisfies the divergence-free condition. Expanding (5.43) using (5.38),

$$\begin{aligned}\frac{1}{\rho}\frac{\partial}{\partial\rho}\left[\rho\psi^{\rho}_{m,u}\cos m\phi + \rho\psi^{\rho}_{m,v}\sin m\phi\right]e^{\pm ikz} + \frac{m}{\rho}\left[-\psi^{\phi}_{m,u}\sin m\phi + \psi^{\phi}_{m,v}\cos m\phi\right]e^{\pm ikz} \\ + \left[\frac{\partial\psi^{z}_{m,u}}{\partial z}\cos m\phi + \frac{\partial\psi^{z}_{m,v}}{\partial z}\sin m\phi \pm ik\left(\psi^{z}_{m,u}\cos m\phi + \psi^{z}_{m,v}\sin m\phi\right)\right]e^{\pm ikz} = 0\end{aligned} \tag{5.44}$$

where the cylindrical coordinates form of the divergence operator was used.

$$\nabla \cdot \vec{A} = \frac{1}{\rho}\frac{\partial}{\partial\rho}(\rho A_\rho) + \frac{1}{\rho}\frac{\partial A_\phi}{\partial\phi} + \frac{\partial A_z}{\partial z} \tag{5.45}$$

Next, by orthogonality, the sine and cosine terms for each mode can be separated, reducing (5.44) to the following modal equations.

$$\frac{1}{\rho}\frac{\partial}{\partial\rho}(\rho\psi^{\rho}_{m,u}) + \frac{m}{\rho}\psi^{\phi}_{m,v} + \frac{\partial\psi^{z}_{m,u}}{\partial z} \pm ik\psi^{z}_{m,u} = 0 \tag{5.46}$$

$$\frac{1}{\rho}\frac{\partial}{\partial\rho}(\rho\psi^{\rho}_{m,v}) - \frac{m}{\rho}\psi^{\phi}_{m,u} + \frac{\partial\psi^{z}_{m,v}}{\partial z} \pm ik\psi^{z}_{m,v} = 0 \tag{5.47}$$

In order to avoid estimation of range derivatives, the parabolic equations (5.30) and (5.33) are used to yield an expression involving only the fields at one range step. Equations (5.46) and (5.47) are rewritten as,

$$\frac{1}{\rho}\frac{\partial}{\partial\rho}(\rho\psi^{\rho}_{m,u}) + \frac{m}{\rho}\psi^{\phi}_{m,v} \pm ik\psi^{z}_{m,u} \pm \left(\frac{i}{2k}\right)\left[\frac{1}{\rho}\frac{\partial\psi^{z}_{m,u}}{\partial\rho} + \frac{\partial^2\psi^{z}_{m,u}}{\partial\rho^2} - \frac{m^2}{\rho^2}\psi^{z}_{m,u}\right] = 0 \tag{5.48}$$

$$\frac{1}{\rho}\frac{\partial}{\partial\rho}(\rho\psi^{\rho}_{m,v}) - \frac{m}{\rho}\psi^{\phi}_{m,u} \pm ik\psi^{z}_{m,v} \pm \left(\frac{i}{2k}\right)\left[\frac{1}{\rho}\frac{\partial\psi^{z}_{m,v}}{\partial\rho} + \frac{\partial^2\psi^{z}_{m,v}}{\partial\rho^2} - \frac{m^2}{\rho^2}\psi^{z}_{m,v}\right] = 0. \tag{5.49}$$

While the divergence-free condition ensures a unique solution to the boundary condition in (5.42), it also serves to enforce the divergence-free nature of the scattered fields. While a solution to the paraxial version of Maxwell's equations does not guarantee that the fields are divergence-free, it can be shown [59] that if the fields along the object's boundary are divergence-free, then the solution to the paraxial version of Maxwell's equations, (5.28)–(5.33), will be divergence-free everywhere.

5.3 Discretization of Parabolic Wave Equations

In order to solve for the scattered fields, the parabolic equations in (5.28)–(5.33) must be discretized. The following approximations to first and second derivatives are used in the discretization.

$$\frac{\partial f(\xi)}{\partial \xi} \approx \frac{f(\xi+\Delta\xi)-f(\xi)}{\Delta\xi} = \frac{f_{n+1}-f_n}{\Delta\xi} \tag{5.50}$$

$$\frac{\partial f(\xi)}{\partial \xi} \approx \frac{f(\xi+\Delta\xi)-f(\xi-\Delta\xi)}{2\Delta\xi} = \frac{f_{n+1}-f_{n-1}}{2\Delta\xi} \tag{5.51}$$

$$\frac{\partial^2 f(\xi)}{\partial \xi^2} \approx \frac{f(\xi+2\Delta\xi)-2f(\xi+\Delta\xi)+f(\xi)}{\Delta\xi} = \frac{f_{n+2}-2f_{n+1}+f_n}{\Delta\xi} \tag{5.52}$$

$$\frac{\partial^2 f(\xi)}{\partial \xi^2} \approx \frac{f(\xi+\Delta\xi)-2f(\xi)+f(\xi-\Delta\xi)}{\Delta\xi} = \frac{f_{n+1}-2f_n+f_{n-1}}{\Delta\xi} \tag{5.53}$$

Equations (5.50) and (5.52) represent first order accurate approximations of the first and second derivatives, whereas equations (5.51) and (5.53) are second order accurate approximations of the first and second derivatives. Although the central difference approximation is more accurate, it will not always be possible to use this form of discretization due to the placement of the field components on the computational grid. In the case of the range derivative, however, it is desirable to use the first order accurate representation so that a marching in space approach can be used to solve the parabolic wave equations.

5.3.1 Difference Equations for Freespace Fields

Without loss of generality, the first set of parabolic equations, (5.28)-(5.30), will be used in deriving the difference equations. The difference equations for the second set can be found by simply replacing m by $-m$ and interchanging the u and v subscripts in each of the six equations. Also, since only one set is being considered the u and v subscripts will be omitted

in the following sections. The following notation will be used for any function of space in the finite difference equations.

$$\psi_{n,l} = \psi(n\Delta z, l\Delta\rho) \tag{5.54}$$

As shown in Figure 5-2, the computational domain extends from $n = 0$ and $l = 0$ to $n = N$ and $l = L+1$. One general technique for solving parabolic equations is the backward difference

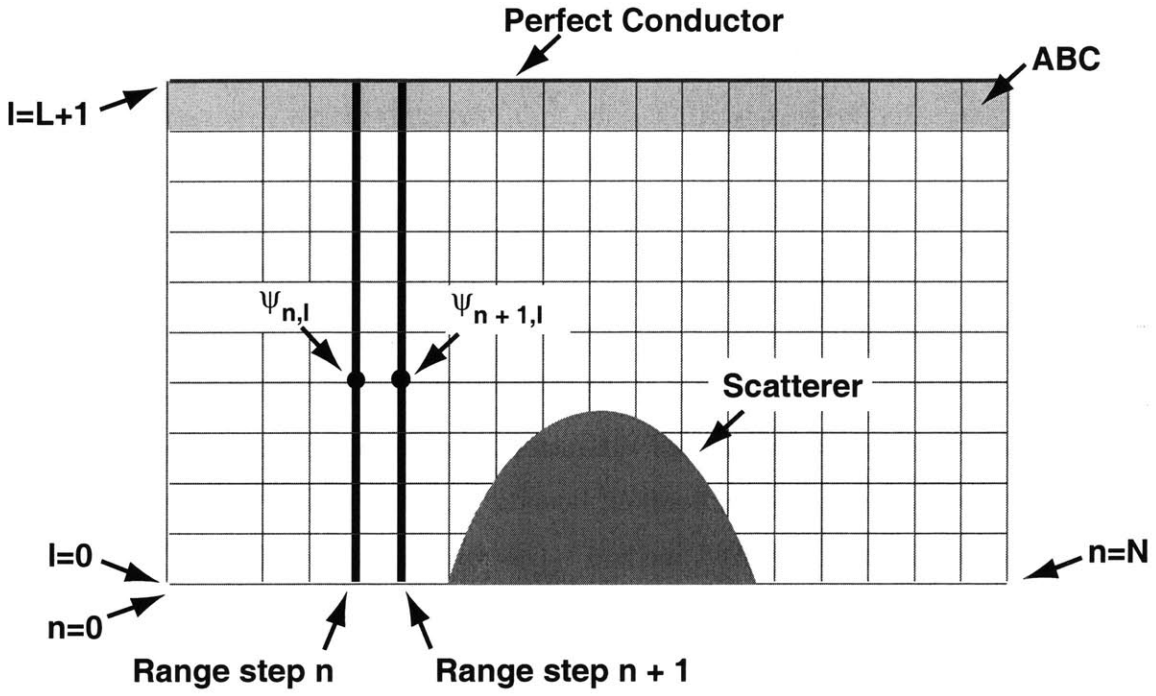

Figure 5-2: BOR PWE Computational Domain

method [12]. For example, consider a general parabolic equation of the form,

$$\frac{\partial\psi(\rho,z)}{\partial z} = a^2\frac{\partial^2\psi(\rho,z)}{\partial\rho^2} + S(\rho,z) \tag{5.55}$$

with the following boundary conditions,

$$\psi(0,\rho) = f(\rho) \tag{5.56}$$

$$\psi(z,0) = p(z) \tag{5.57}$$

$$\psi(z,(L+1)\Delta\rho) = q(z). \tag{5.58}$$

For $l = 1, L$, equation (5.55) can then be discretized into the following form,

$$\frac{\psi_{n+1,l} - \psi_{n,l}}{\Delta z} = a^2 \frac{\psi_{n+1,l+1} - 2\psi_{n+1,l} + \psi_{n+1,l-1}}{(\Delta\rho)^2} + S_{n+1,l} \tag{5.59}$$

With the given boundary conditions, a linear system with $L + 2$ unknowns can formed to solve for the fields at range step $n + 1$ using the value of the fields at range step n. In this way, given an initial condition at range step $n = 0$, all of the fields from range step $n = 0$ to $n = N$ can be solved for using a marching in space approach. Also, since the fields at range step $n + 1$ depend only on the fields at range step n, the method has a one-dimensional computer memory requirement.

Although, the backward difference method is an explicit finite-difference method, it does not have the same stability conditions required when using the FD-TD method. This is due to the fact that the Courant-Friedrichs-Lewy (CFL) stability condition applies only to hyperbolic and not parabolic partial differential equations. Thus, the criteria for choosing the $\Delta\rho$ and Δz step sizes depends only on the accuracy of the solution needed.

The parabolic equations for the BOR PWE method can be discretized in a similar fashion. In order to have a well posed problem, however, boundary conditions must be specified including an initial condition. Since the PWE method solves for the energy scattered in the paraxial direction and the plane wave source is implemented through the boundary conditions along the surface of the object, the initial field for the marching algorithm will be zero. For example, in the case of forward scattering in the $\hat{z}$ direction, the fields at the initial range step cannot not have any forward propagating components since the object has not yet been reached to introduce a scattered field through the boundary conditions. In this case, the initial condition is $\vec{\psi}(0, \rho) = 0$. The parabolic equations can then be solved by marching in the $+\hat{z}$ direction. Similarly, if the paraxial direction is in the $-\hat{z}$ direction, the initial condition will be $\vec{\psi}(N\Delta z, \rho) = 0$, and the parabolic equations are solved by marching in the $-\hat{z}$ direction.

In addition to the initial conditions discussed above, two additional boundary conditions at the upper and lower sides of the computational domain are needed. The boundary condition along the bottom edge of the computational domain is accounted for by using the on-axis equations developed in Section 5.3.2, whereas the domain is truncated along the upper side by using an absorbing boundary condition discussed in Section 5.4.

Using the backward difference methodology, the difference equations when the paraxial

direction is in the $+\hat{z}$ direction are given by,

$$\frac{\psi^{\rho}_{n+1,l}-\psi^{\rho}_{n,l}}{\Delta z} = \frac{i}{2k}\left[\frac{\psi^{\rho}_{n+1,l+1}-2\psi^{\rho}_{n+1,l}+\psi^{\rho}_{n+1,l-1}}{(\Delta\rho)^2}+\frac{\psi^{\rho}_{n+1,l+1}-\psi^{\rho}_{n+1,l-1}}{l(2\Delta\rho)^2} - \frac{m^2+1}{(l\Delta\rho)^2}\psi^{\rho}_{n+1,l}-\frac{2m}{(l\Delta\rho)^2}\psi^{\phi}_{n+1,l}\right] \tag{5.60}$$

$$\frac{\psi^{\phi}_{n+1,l}-\psi^{\phi}_{n,l}}{\Delta z} = \frac{i}{2k}\left[\frac{\psi^{\phi}_{n+1,l+1}-2\psi^{\phi}_{n+1,l}+\psi^{\phi}_{n+1,l-1}}{(\Delta\rho)^2}+\frac{\psi^{\phi}_{n+1,l+1}-\psi^{\phi}_{n+1,l-1}}{l(2\Delta\rho)^2} - \frac{m^2+1}{(l\Delta\rho)^2}\psi^{\phi}_{n+1,l}+\frac{2m}{(l\Delta\rho)^2}\psi^{\rho}_{n+1,l}\right] \tag{5.61}$$

$$\frac{\psi^{z}_{n+1,l}-\psi^{z}_{n,l}}{\Delta z} = \frac{i}{2k}\left[\frac{\psi^{z}_{n+1,l+1}-2\psi^{z}_{n+1,l}+\psi^{z}_{n+1,l-1}}{(\Delta\rho)^2}+\frac{\psi^{z}_{n+1,l+1}-\psi^{z}_{n+1,l-1}}{l(2\Delta\rho)^2} - \frac{m^2}{(l\Delta\rho)^2}\psi^{z}_{n+1,l}\right]. \tag{5.62}$$

Similarly, the difference equations when the paraxial direction is in the $-\hat{z}$ direction are given by,

$$\frac{\psi^{\rho}_{n+1,l}-\psi^{\rho}_{n,l}}{\Delta z} = -\frac{i}{2k}\left[\frac{\psi^{\rho}_{n,l+1}-2\psi^{\rho}_{n,l}+\psi^{\rho}_{n,l-1}}{(\Delta\rho)^2}+\frac{\psi^{\rho}_{n,l+1}-\psi^{\rho}_{n,l-1}}{l(2\Delta\rho)^2} - \frac{m^2+1}{(l\Delta\rho)^2}\psi^{\rho}_{n,l}-\frac{2m}{(l\Delta\rho)^2}\psi^{\phi}_{n,l}\right] \tag{5.63}$$

$$\frac{\psi^{\phi}_{n+1,l}-\psi^{\phi}_{n,l}}{\Delta z} = -\frac{i}{2k}\left[\frac{\psi^{\phi}_{n,l+1}-2\psi^{\phi}_{n+1,l}+\psi^{\phi}_{n,l-1}}{(\Delta\rho)^2}+\frac{\psi^{\phi}_{n,l+1}-\psi^{\phi}_{n,l-1}}{l(2\Delta\rho)^2} - \frac{m^2+1}{(l\Delta\rho)^2}\psi^{\phi}_{n,l}+\frac{2m}{(l\Delta\rho)^2}\psi^{\rho}_{n,l}\right] \tag{5.64}$$

$$\frac{\psi^{z}_{n+1,l}-\psi^{z}_{n,l}}{\Delta z} = -\frac{i}{2k}\left[\frac{\psi^{z}_{n,l+1}-2\psi^{z}_{n,l}+\psi^{z}_{n,l-1}}{(\Delta\rho)^2}+\frac{\psi^{z}_{n,l+1}-\psi^{z}_{n,l-1}}{l(2\Delta\rho)^2} - \frac{m^2}{(l\Delta\rho)^2}\psi^{z}_{n,l}\right]. \tag{5.65}$$

In each of the above equations, the first and second derivatives with respect to ρ are discretized using central difference approximations. Although the discretization of the first derivative used is second order accurate, the fields are spaced two steps sizes apart so that the error will be on the order of $4\Delta\rho^2$, whereas the truncation error of a first order discretization will be on the order of $\Delta\rho$. Hence, if $\Delta\rho > 0.25$, a first order discretization will be more accurate and should be used.

5.3.2 Difference Equations for On-Axis Freespace Fields

Along the z axis at any $z = z_0$, the $\hat{\rho}$ and $\hat{\phi}$ cylindrical coordinate components are not defined, so the discrete field components corresponding to $l = 0$ are placed at $\rho = \Delta\rho/2$ instead of at $\rho = 0$. In addition, since the computational domain must be truncated at the lower boundary, the difference equations presented in the previous section cannot be used. Instead, first order accurate discretizations of the first and second derivatives with respect to ρ must be used. The difference equations for the on axis cells when the paraxial direction is in the $+\hat{z}$ are given by,

$$\frac{\psi^{\rho}_{n+1,0} - \psi^{\rho}_{n,0}}{\Delta z} = \frac{i}{2k}\left[\frac{\psi^{\rho}_{n+1,2} - 2\psi^{\rho}_{n+1,1} + \psi^{\rho}_{n+1,0}}{(\Delta\rho)^2} + \frac{\psi^{\rho}_{n+1,1} - \psi^{\rho}_{n+1,0}}{0.5(\Delta\rho)^2} - \frac{m^2+1}{(0.5\Delta\rho)^2}\psi^{\rho}_{n+1,0} - \frac{2m}{(0.5\Delta\rho)^2}\psi^{\phi}_{n+1,0}\right] \tag{5.66}$$

$$\frac{\psi^{\phi}_{n+1,0} - \psi^{\phi}_{n,0}}{\Delta z} = \frac{i}{2k}\left[\frac{\psi^{\phi}_{n+1,2} - 2\psi^{\phi}_{n+1,1} + \psi^{\phi}_{n+1,0}}{(\Delta\rho)^2} + \frac{\psi^{\phi}_{n+1,1} - \psi^{\phi}_{n+1,0}}{0.5(\Delta\rho)^2} - \frac{m^2+1}{(0.5\Delta\rho)^2}\psi^{\phi}_{n+1,0} - \frac{2m}{(0.5\Delta\rho)^2}\psi^{\rho}_{n+1,0}\right] \tag{5.67}$$

$$\frac{\psi^{z}_{n+1,0} - \psi^{z}_{n,0}}{\Delta z} = \frac{i}{2k}\left[\frac{\psi^{z}_{n+1,2} - 2\psi^{z}_{n+1,1} + \psi^{z}_{n+1,0}}{(\Delta\rho)^2} + \frac{\psi^{z}_{n+1,1} - \psi^{z}_{n+1,0}}{0.5(\Delta\rho)^2} - \frac{m^2}{(0.5\Delta\rho)^2}\psi^{z}_{n+1,0}\right]. \tag{5.68}$$

Similar difference equations can be written for the case when the paraxial direction is in the $-\hat{z}$ direction.

5.3.3 Difference Equations for Boundary Conditions

The difference equations presented in the previous two sections apply to fields not along the object boundary. In order to generate the scattered field, the boundary conditions described in Section 5.2.2 must be enforced. Thus, for lattice points that lie on the object's surface, the freespace difference equations are replaced by the discretized boundary and divergence-free conditions. Since the boundary condition given in (5.42) does not contain any derivatives, its discrete version is simply,

$$\begin{aligned} \cos\alpha\ \psi^{\rho}_{n,l} + \sin\alpha\ \psi^{z}_{n,l} &= -e^{\mp ikz}\left[\cos\alpha\ e^{i}_{\rho}(l\Delta\rho, n\Delta z) + \sin\alpha\ e^{i}_{z}(l\Delta\rho, n\Delta z)\right] \\ \psi^{\phi}_{n,l} &= -e^{\mp ikz} e^{i}_{\phi}(l\Delta\rho, n\Delta z) \end{aligned} \tag{5.69}$$

where the values of the incident field, e^{i}_{ρ}, e^{i}_{ϕ}, and e^{i}_{z} are calculated analytically. Since the derivatives with respect to ρ at lattice point $(n+1, l)$ must be approximated using only points that lie along and outside of the object, a first order accurate discretization of the first and second derivatives is used. In this case when the paraxial direction is in the $+\hat{z}$ direction, the discrete version of the divergence-free condition corresponding to the first set of parabolic equations is,

$$\begin{aligned} &\frac{i}{2k}\left[\frac{\psi^{z}_{n+1,l+1} - \psi^{z}_{n+1,l}}{l(\Delta\rho)^2} + \frac{\psi^{z}_{n+1,l+2} - 2\psi^{z}_{n+1,l+1} + \psi^{z}_{n+1,l}}{(\Delta\rho)^2} - \frac{m^2}{(l\Delta\rho)^2}\psi^{z}_{n+1,l}\right] \\ &\quad + \frac{(l+1)\psi^{\rho}_{n+1,l+1} - (l)\psi^{\rho}_{n+1,l}}{l\Delta\rho} + \frac{m}{l\Delta\rho}\psi^{\phi}_{n+1,l} + ik\psi^{z}_{n+1,l} = 0. \end{aligned} \tag{5.70}$$

As usual, the divergence-free condition corresponding to the second set of parabolic equations can be found by replacing m with $-m$. Similarly, the discrete version of the divergence-free condition when the paraxial direction is in the $-\hat{z}$ direction is,

$$\begin{aligned} &-\frac{i}{2k}\left[\frac{\psi^{z}_{n,l+1} - \psi^{z}_{n,l}}{l(\Delta\rho)^2} + \frac{\psi^{z}_{n,l+2} - 2\psi^{z}_{n,l+1} + \psi^{z}_{n,l}}{(\Delta\rho)^2} - \frac{m^2}{(l\Delta\rho)^2}\psi^{z}_{n,l}\right] \\ &\quad + \frac{(l+1)\psi^{\rho}_{n,l+1} - (l)\psi^{\rho}_{n,l}}{l\Delta\rho} + \frac{m}{l\Delta\rho}\psi^{\phi}_{n,l} - ik\psi^{z}_{n,l} = 0. \end{aligned} \tag{5.71}$$

5.3.4 Matrix Formulation

The difference equations presented in the previous sections can be used to formulate a matrix equation in terms of the ψ variables. At each range step, there $L+2$ unknown ψ_{ρ} variables,

$L+2$ unknown ψ_ϕ variables, and $L+2$ unknown ψ_z variables for a total of $3L+6$ unknowns.

At each range step, the finite difference equations and boundary conditions presented in the previous three sections can be used to form a $(3L+6) \times (3L+6)$ system of linear equations that must be solved. In the case of forward scattering, the linear system formed can be written as the following block matrix equation,

$$\begin{pmatrix} A_{\rho\rho} & A_{\rho\phi} & A_{\rho z} \\ A_{\phi\rho} & A_{\phi\phi} & A_{\phi z} \\ A_{z\rho} & A_{z\phi} & A_{zz} \end{pmatrix} \begin{pmatrix} \psi^{\rho}_{n+1} \\ \psi^{\phi}_{n+1} \\ \psi^{z}_{n+1} \end{pmatrix} = \begin{pmatrix} \psi^{\rho}_{n} \\ \psi^{\phi}_{n} \\ \psi^{z}_{n} \end{pmatrix} \tag{5.72}$$

where the elements of the A matrices come from the difference equations given in the previous sections and

$$\psi^{i}_{n} = \begin{pmatrix} \psi^{i}_{n,0} \\ \vdots \\ \psi^{i}_{n,L+2} \end{pmatrix} \tag{5.73}$$

for $i = \rho, \phi, z$. If at range step $n+1$ there are no boundary conditions to enforce, then the $A_{\rho\rho}$, $A_{\phi\phi}$, and A_{zz} matrices will be tridiagonal matrices, the $A_{\rho\phi}$ and $A_{\phi\rho}$ matrices will be diagonal matrices, and the remaining $A_{\rho z}$, $A_{z\rho}$, $A_{\phi z}$, and $A_{z\phi}$ matrices will each be zero matrices. In this case, only the ψ_ρ and ψ_ϕ terms are coupled; the ψ_z terms can be solved for independently. At a boundary condition, however, all three fields are coupled. In this case, all of the block matrices will contain nonzero elements except for the $A_{\phi z}$. This is due to the fact that ψ_ϕ and ψ_z terms are only coupled in the divergence-free condition and not in the boundary condition given by equation (5.42). In both cases, the resulting matrix is sparse and can be efficiently solved by using an iterative technique, such as the biconjugate gradient method. In addition, the sparsity of the matrix can be exploited to reduce the memory requirement by storing only the nonzero elements of the matrix. In fact, the upper bound for the number of nonzero elements for a system with $3L+6$ unknowns is $11L - 2n + 14$, where n is the number of separate boundary conditions that must be enforced at the current range step. Thus, the memory requirement is linear, an improvement over the quadratic memory requirement of dense matrix methods.

Although, the choice of the step sizes does not affect the stability of the method, it does affect the condition number of matrix formed by the difference equations. In such cases, iterative

solution techniques such as the conjugate gradient may not converge to the correct solution. In particular, it has been observed in numerical experiments that the condition number of the matrix increases as the number of boundary conditions that need to be enforced increases. This is due to the fact that the matrix becomes less diagonally dominant with more boundary conditions since the matrices $A_{\rho\rho}$, $A_{\phi\phi}$, and A_{zz} are no longer fully tridiagonal.

5.4 PML Absorbing Boundary Condition

As with the BOR and 2D FD-TD techniques, the computational domain in the BOR PWE method needs to be truncated. Due to the nature of the technique in which the solution is marched along the z–axis, an absorbing boundary condition is only needed to truncate fields propagating in the $\hat{\rho}$ direction. Since the BOR PWE method is formulated in the frequency domain, the BOR PWE PML formulation is most easily derived from the complex coordinate stretching viewpoint described in Chapter 2. As in the BOR FD-TD technique, the following mapping is defined.

$$\tilde{\rho} = \int_0^{\rho} s_\rho(\rho')d\rho' \tag{5.74}$$

In order to have a lossy PML region, $s_\rho(\rho)$ is defined using a quadratic profile for a_ρ and σ_ρ as,

$$s_\rho(\rho) = a_\rho(\rho) + \frac{i\sigma_\rho(\rho)}{\epsilon\omega} = \begin{cases} 1 & \rho < \rho_o \\ 1 + \alpha\left(\frac{\rho-\rho_o}{\Gamma}\right)^2 + \beta\left(\frac{i}{\epsilon\omega}\right)\left(\frac{\rho-\rho_o}{\Gamma}\right)^2 & \rho > \rho_o \end{cases} \tag{5.75}$$

where the PML region begins at $\rho = \rho_o$, Γ is the thickness of the PML region, and α and β are parameters used to control the absorptive properties of the PML. As with the BOR FD-TD PML formulation, Maxwell's equations can be recast into the same form of the original Maxwell's equations but on a complex variable spatial domain. Under the change of variables the del operator becomes,

$$\nabla \rightarrow \tilde{\nabla} = \hat{\rho}\frac{\partial}{\partial\tilde{\rho}} + \hat{\phi}\frac{1}{\tilde{\rho}}\frac{\partial}{\partial\phi} + \hat{z}\frac{\partial}{\partial z}. \tag{5.76}$$

The derivation of the parabolic wave equations can then be carried as before on this new complex variable spatial domain. For example, the parabolic wave equation for the ψ^ρ field in the PML region is,

$$\frac{\partial^2\psi^\rho_{m,u}}{\partial\tilde{\rho}^2} + \frac{1}{\tilde{\rho}}\frac{\partial\psi^\rho_{m,u}}{\partial\tilde{\rho}} + 2ik\frac{\partial\psi^\rho_{m,u}}{\partial z} - \left(\frac{m^2+1}{\tilde{\rho}^2}\right)\psi^\rho_{m,u} - \frac{2m}{\tilde{\rho}^2}\psi^\phi_{m,v} = 0. \tag{5.77}$$

While equation (5.77) is of the same form of the parabolic wave equation for the ψ^ρ field derived in Section 5.2.1, it cannot be discretized in the same manner due to the complex spatial variable derivatives. Instead, these derivatives must be recast in terms of real spatial variables using the following relations,

$$\frac{1}{\tilde{\rho}}\frac{\partial}{\partial\tilde{\rho}} = \frac{1}{\tilde{\rho}}\frac{1}{s_\rho}\frac{\partial}{\partial\rho} \tag{5.78}$$

$$\frac{\partial^2}{\partial\tilde{\rho}^2} = \frac{\partial}{\partial\tilde{\rho}}\frac{\partial}{\partial\tilde{\rho}} = \frac{1}{s_\rho}\frac{\partial}{\partial\rho}\left[\left(\frac{1}{s_\rho}\right)\left(\frac{\partial}{\partial\rho}\right)\right] = \frac{1}{s_\rho}\left(q_\rho\frac{\partial}{\partial\rho} + \frac{1}{s_\rho}\frac{\partial^2}{\partial\rho^2}\right) \tag{5.79}$$

where

$$\tilde{\rho}(\rho) = \begin{cases} \rho & \rho < \rho_o \\ \rho + \frac{(\rho-\rho_o)^3}{3\Gamma^2}\left[\alpha + \beta\left(\frac{i}{\epsilon\omega}\right)\right] & \rho > \rho_o \end{cases} \tag{5.80}$$

and

$$q_\rho(\rho) = \frac{\partial}{\partial\rho}\left(\frac{1}{s_\rho}\right) = -\frac{\frac{2\alpha}{\Gamma^2}(\rho-\rho_o) + \frac{2i\beta}{\epsilon\omega\Gamma^2}(\rho-\rho_o)}{\left[1+\alpha\left(\frac{\rho-\rho_o}{\Gamma}\right)^2 + \frac{i\beta}{\epsilon\omega}\left(\frac{\rho-\rho_o}{\Gamma}\right)^2\right]^2}. \tag{5.81}$$

Equation (5.77) can now be written in terms of real spatial variable derivatives as,

$$\left(\frac{1}{s_\rho}\right)^2\frac{\partial^2\psi^\rho_{m,u}}{\partial\rho^2} + \left(\frac{q_\rho}{s_\rho} + \frac{1}{\tilde{\rho}s_\rho}\right)\frac{\partial\psi^\rho_{m,u}}{\partial\rho} + 2ik\frac{\partial\psi^\rho_{m,u}}{\partial z} - \left(\frac{m^2+1}{\tilde{\rho}^2}\right)\psi^\rho_{m,u} - \frac{2m}{\tilde{\rho}^2}\psi^\phi_{m,v} = 0 \tag{5.82}$$

Equation (5.82) can now discretized using central difference approximations. Unlike the BOR FD-TD PML, the fields do not need to be split since the PWE formulation is done in the frequency domain. Also, note that equation (5.82) reduces to the freespace parabolic wave equation when $\alpha = \beta = 0$. Since there is no additional cost in memory, it convenient to use scalar wave equations for a generalized PML media in place of the freespace difference equations derived in the previous sections. For the freespace region, the parameters α and β are simply set to zero. Discretizing (5.82) yields,

$$\begin{aligned}\frac{\psi^\rho_{n+1,l} - \psi^\rho_{n,l}}{\Delta z} = \frac{i}{2k}\Bigg[&\frac{1}{s^2_\rho(l)}\frac{\psi^\rho_{n+1,l+1} - 2\psi^\rho_{n+1,l} + \psi^\rho_{n+1,l-1}}{(\Delta\rho)^2} - \left(\frac{m^2+1}{\tilde{\rho}^2(l)}\right)\psi^\rho_{n+1,l} \\ &+ \left(\frac{q_\rho(l)}{s_\rho(l)} + \frac{1}{s^2_\rho(l)\tilde{\rho}(l)}\right)\frac{\psi^\rho_{n+1,l+1} - \psi^\rho_{n+1,l-1}}{2\Delta\rho} - \frac{2m}{\tilde{\rho}^2(l)}\psi^\phi_{n+1,l}\Bigg] = 0\end{aligned} \tag{5.83}$$

where

$$\tilde{\rho}(l) = l\Delta\rho + \frac{(l-l_o)^3}{3\gamma^2}\left[\alpha(l) + \beta(l)\left(\frac{i}{\epsilon\omega}\right)\right]\Delta\rho \tag{5.84}$$

$$q_\rho(l) = -\frac{2\alpha(l)\,(l-l_o) + \frac{2i\beta(l)}{\epsilon\omega}\,(l-l_o)}{\gamma^2\Delta\rho\left[1+\alpha(l)\left(\frac{l-l_o}{\gamma}\right)^2 + \frac{i\beta(l)}{\epsilon\omega}\left(\frac{l-l_o}{\gamma}\right)^2\right]^2} \tag{5.85}$$

and $\rho = l\Delta\rho$, $\rho_o = l_o\Delta\rho$, and $\Gamma = \gamma\Delta\rho$. The parameters α and β can be defined in terms of unit step functions,

$$\alpha(l) = \alpha_o \mathrm{u}(l-l_o) \tag{5.86}$$

$$\beta(l) = \beta_o \mathrm{u}(l-l_o) \tag{5.87}$$

so that they are zero in the freespace region and nonzero constants in the PML region. The values chosen for α_o and β_o depends on the thickness of the PML region. As with the fields in freespace, the PML region must be truncated, and this is done by setting the value of the fields beyond the PML to be zero.

5.5 Plane Wave Decomposition

As with the FD-TD methods, all the fields inside the computational domain are initially set to zero. For RCS and scattering simulations, a plane wave excitation is used. Unlike the FD-TD methods discussed in the previous chapters, the plane wave excitation for the BOR PWE method is implemented through a scattered field formulation. That is, no plane wave source is propagated from some initial source position, but rather the scattered fields arise from the enforcement of the boundary condition for tangential electric fields, $E_{scat} = -E_{inc}$, as discussed in Section 5.2.2. The incident field quantity is calculated using an analytical expression for the plane wave source. Similar to the plane wave source used with the BOR FD-TD method, the general form of the incident electric field can be written in terms of horizontal and vertical polarization components in time-harmonic form as,

$$\vec{E}_i = \left(E_h\hat{h} + E_v\hat{v}\right)e^{i\hat{k}_i\cdot\hat{r}} \tag{5.88}$$

where

$$\begin{aligned}
\hat{r} &= x\hat{x} + y\hat{y} + z\hat{z} \\
\hat{k_i} &= -\hat{x}\sin\theta_i - \hat{z}\cos\theta_i \\
\hat{k_i}\cdot\hat{r} &= -x\sin\theta_i - z\cos\theta_i = -\rho\cos\phi\sin\theta_i - z\cos\theta_i \\
\hat{h} &= \hat{x}\cos\theta_i - \hat{z}\sin\theta_i = \hat{\rho}\cos\theta_i\cos\phi - \hat{\phi}\cos\theta_i\sin\phi - \hat{z}\sin\theta_i \\
\hat{v} &= \hat{y} = \hat{\phi}\cos\phi + \hat{\rho}\sin\phi.
\end{aligned} \tag{5.89}$$

Expanding the incident electric field yields,

$$\begin{aligned}
\vec{E}^i = &\Big[E_h\left(\hat{\rho}\cos\theta_i\cos\phi - \hat{\phi}\cos\theta_i\sin\phi - \hat{z}\sin\theta_i\right) \\
&+ E_v\left(\hat{\phi}\cos\phi + \hat{\rho}\sin\phi\right)\Big] e^{-ik(\rho\cos\phi\sin\theta_i + z\cos\theta_i)}
\end{aligned} \tag{5.90}$$

such that

$$\begin{aligned}
E^i_\rho &= \left[E_h\cos\theta_i\cos\phi + E_v\sin\phi\right] e^{-ik(\rho\cos\phi\sin\theta_i + z\cos\theta_i)} \\
E^i_\phi &= \left[-E_h\cos\theta_i\sin\phi + E_v\cos\phi\right] e^{-ik(\rho\cos\phi\sin\theta_i + z\cos\theta_i)} \\
E^i_z &= \left[-E_h\sin\theta_i\right] e^{-ik(\rho\cos\phi\sin\theta_i + z\cos\theta_i)}.
\end{aligned} \tag{5.91}$$

Since the scattered fields have been expanded in a Fourier series in ϕ, the incident fields must be decomposed into their Fourier components. For example, the $e^\rho_{m_{u,v}}$ components of the incident electric field are calculated from,

$$\begin{aligned}
e^\rho_{0,u} &= \frac{E_h}{2\pi}\cos\theta_i e^{-ikz\cos\theta_i}\int_0^{2\pi}\cos\phi\; e^{-ik\rho\sin\theta_i\cos\phi}d\phi \\
&\quad + \frac{E_v}{2\pi}e^{-ikz\cos\theta_i}\int_0^{2\pi}\sin\phi\; e^{-ik\rho\sin\theta_i\cos\phi}d\phi \\
e^\rho_{m,u} &= \frac{E_h}{\pi}\cos\theta_i e^{-ikz\cos\theta_i}\int_0^{2\pi}\cos m\phi\cos\phi\; e^{-ik\rho\sin\theta_i\cos\phi}d\phi \\
&\quad + \frac{E_v}{\pi}e^{-ikz\cos\theta_i}\int_0^{2\pi}\cos m\phi\sin\phi\; e^{-ik\rho\sin\theta_i\cos\phi}d\phi.
\end{aligned} \tag{5.92}$$

$$\begin{aligned}
e^\rho_{0,v} &= 0 \\
e^\rho_{m,v} &= \frac{E_h}{\pi}\cos\theta_i e^{-ikz\cos\theta_i}\int_0^{2\pi}\sin m\phi\cos\phi\; e^{-ik\rho\sin\theta_i\cos\phi}d\phi
\end{aligned}$$

$$+\frac{E_v}{\pi}e^{-ikz\cos\theta_i}\int_0^{2\pi}\sin m\phi\sin\phi\; e^{-ik\rho\sin\theta_i\cos\phi}d\phi \tag{5.93}$$

The remaining two components of the electric field can be expanded in a similar fashion. In order to evaluate the resulting integrals the following relationships are needed,

$$\begin{aligned}\int_0^{2\pi}\cos m\phi\; e^{-ik\rho\sin\theta_i\cos\phi}d\phi &= 2\pi e^{im\frac{3\pi}{2}}J_m(k\rho\sin\theta_i)\\ \int_0^{2\pi}\sin m\phi\; e^{-ik\rho\sin\theta_i\cos\phi}d\phi &= 0\end{aligned} \tag{5.94}$$

$$\begin{aligned}\cos m\phi\cos\phi &= \frac{1}{2}\cos\left[(m-1)\phi\right]+\frac{1}{2}\cos\left[(m+1)\phi\right]\\ \sin m\phi\sin\phi &= \frac{1}{2}\cos\left[(m-1)\phi\right]-\frac{1}{2}\cos\left(m+1\right)\phi]\\ \cos m\phi\sin\phi &= \frac{1}{2}\sin\left[(m+1)\phi\right]-\frac{1}{2}\sin\left[(m-1)\phi\right]\\ \sin m\phi\cos\phi &= \frac{1}{2}\sin\left[(m-1)\phi\right]+\frac{1}{2}\sin\left[(m+1)\phi\right]\end{aligned} \tag{5.95}$$

where J_m is the mth order Bessel function. Using the identities given in (5.94) and (5.95) the complete set of Fourier components for the incident plane wave can be written in closed form as,

$$\begin{aligned}e^{\rho}_{0,u} &= E_h\cos\theta_i\; e^{-ikz\cos\theta_i}e^{i\frac{3\pi}{2}}J_1(k\rho\sin\theta_i)\\ e^{\rho}_{m,u} &= E_h\cos\theta_i\; e^{-ikz\cos\theta_i}\left[e^{i(m+1)\frac{3\pi}{2}}J_{m+1}(k\rho\sin\theta_i)\right.\\ &\qquad \left.+e^{i(m-1)\frac{3\pi}{2}}J_{m-1}(k\rho\sin\theta_i)\right]\\ e^{\rho}_{0,v} &= 0\\ e^{\rho}_{m,v} &= E_v e^{-ikz\cos\theta_i}\left[e^{i(m-1)\frac{3\pi}{2}}J_{m-1}(k\rho\sin\theta_i)\right.\\ &\qquad \left.-e^{i(m+1)\frac{3\pi}{2}}J_{m+1}(k\rho\sin\theta_i)\right]\end{aligned} \tag{5.96}$$

$$\begin{aligned}e^{\phi}_{0,u} &= E_v e^{-ikz\cos\theta_i}e^{i\frac{3\pi}{2}}J_1(k\rho\sin\theta_i)\\ e^{\phi}_{m,u} &= E_v e^{-ikz\cos\theta_i}\left[e^{i(m+1)\frac{3\pi}{2}}J_{m+1}(k\rho\sin\theta_i)\right.\\ &\qquad \left.+e^{i(m-1)\frac{3\pi}{2}}J_{m-1}(k\rho\sin\theta_i)\right]\\ e^{\phi}_{0,v} &= 0\end{aligned}$$

$$\begin{aligned} e^{\phi}_{m,v} &= -E_h \cos\theta_i \; e^{-ikz\cos\theta_i} \left[e^{i(m-1)\frac{3\pi}{2}} J_{m-1}(k\rho\sin\theta_i) \right. \\ &\quad \left. - e^{i(m+1)\frac{3\pi}{2}} J_{m+1}(k\rho\sin\theta_i) \right] \end{aligned} \tag{5.97}$$

$$\begin{aligned} e^{z}_{0,u} &= -E_h \sin\theta_i \; e^{-ikz\cos\theta_i} J_0(k\rho\sin\theta_i) \\ e^{z}_{m,u} &= -2E_h \sin\theta_i \; e^{-ikz\cos\theta_i} e^{im\frac{3\pi}{2}} J_m(k\rho\sin\theta_i) \\ e^{z}_{0,v} &= 0 \\ e^{z}_{m,v} &= 0. \end{aligned} \tag{5.98}$$

If the incident field is propagating in the $+\hat{z}$ direction ($\theta_i = 180°$), the incident electric field will have the form,

$$\begin{aligned} \vec{E}^i &= e^{ikz}\left[-E_h\hat{x} + E_v\hat{y}\right] \\ &= e^{ikz}\left[\hat{\rho}\left(-E_h\cos\phi + E_v\sin\phi\right) + \hat{\phi}\left(E_h\sin\phi + E_v\cos\phi\right)\right]. \end{aligned} \tag{5.99}$$

For this case, it is clear that only the fields associated with the $m = 1$ mode are nonzero. In addition, since the wave is traveling along the $\hat{z}$ direction, the incident electric field does not have a $\hat{z}$ component. Hence, the only nonzero incident field components are

$$\begin{aligned} e^{\rho}_{1,u} &= -E_h e^{ikz} \\ e^{\rho}_{1,v} &= E_v e^{ikz} \\ e^{\phi}_{1,u} &= E_v e^{ikz} \\ e^{\phi}_{1,v} &= E_h e^{ikz}. \end{aligned} \tag{5.100}$$

Since the incident field only excites the mode, $m = 1$, the scattered field can be completely represented by,

$$\vec{E}^s = e^{ikz}\left[\vec{\psi}_u\cos\phi + \vec{\psi}_v\sin\phi\right]. \tag{5.101}$$

In this case, if the paraxial direction is also in the $+\hat{z}$ direction, the boundary condition in

(5.42) can be simplified as,

$$\begin{aligned}
\cos\alpha\psi^{\rho}_{m=1,u} + \sin\alpha\psi^{z}_{m=1,u} &= \cos\alpha E_h \\
\psi^{\phi}_{m=1,v} &= -E_h \\
\cos\alpha\psi^{\rho}_{m=1,v} + \sin\alpha\psi^{z}_{m=1,v} &= -\cos\alpha E_v \\
\psi^{\phi}_{m=1,u} &= -E_v.
\end{aligned} \tag{5.102}$$

If, however, the paraxial direction is in the $-\hat{z}$ direction, the boundary condition in (5.42) would be,

$$\begin{aligned}
\cos\alpha\psi^{\rho}_{m=1,u} + \sin\alpha\psi^{z}_{m=1,u} &= \cos\alpha E_h e^{2ikz} \\
\psi^{\phi}_{m=1,v} &= -E_h e^{2ikz} \\
\cos\alpha\psi^{\rho}_{m=1,v} + \sin\alpha\psi^{z}_{m=1,v} &= -\cos\alpha E_v e^{2ikz} \\
\psi^{\phi}_{m=1,u} &= -E_v e^{2ikz}
\end{aligned} \tag{5.103}$$

As noted in Section 5.2.2, in order to accurately represent the exponential phase variation, the step size, Δz, must become smaller as the angle between the incident wave and paraxial direction increases with backscattering as the most stressing case, and forward scattering as the least stressing.

5.6 RCS Prediction

The calculation of the RCS from the scattered field data is very similar to that described in Chapter 2 for the BOR FD-TD method. Since the PWE method works in the frequency domain, Huygens' principle can be directly applied to the fields without the need to Fourier transform time-domain fields. In order to use Huygens' principle the electric field Fourier components must be recovered from the envelope functions by adding back the phase components.

$$\vec{e}_{m_{u,v}} = \vec{\psi}_{m_{u,v}}\, e^{\pm ikz} \tag{5.104}$$

In addition, Huygens' principle requires the knowledge of the magnetic, H fields, which can be obtained from the curl of the electric field.

$$\vec{H} = \frac{1}{i\omega\mu}\nabla \times \vec{E} \tag{5.105}$$

As with the electric field, the magnetic field is decomposed into a Fourier series in ϕ.

$$\vec{H} = \sum_{m=0}^{N} \left(\vec{h}_{m,u}\cos m\phi + \vec{h}_{m,v}\sin m\phi\right) \tag{5.106}$$

Substituting (5.106) into (5.105) yields,

$$h^{\rho}_{m,v} = \frac{1}{i\omega\mu}\left[-\frac{m}{\rho}e^{z}_{m,u} - \frac{\partial}{\partial z}e^{\phi}_{m,v}\right] \tag{5.107}$$

$$h^{\phi}_{m,u} = \frac{1}{i\omega\mu}\left[\frac{\partial}{\partial z}e^{\rho}_{m,u} - \frac{\partial}{\partial \rho}e^{z}_{m,u}\right] \tag{5.108}$$

$$h^{z}_{m,v} = \frac{1}{i\omega\mu}\left[\frac{m}{\rho}e^{\rho}_{m,u} + \frac{1}{\rho}\frac{\partial}{\partial\rho}(\rho e^{\phi}_{m,v})\right] \tag{5.109}$$

$$h^{\rho}_{m,u} = \frac{1}{i\omega\mu}\left[\frac{m}{\rho}e^{z}_{m,v} - \frac{\partial}{\partial z}e^{\phi}_{m,u}\right] \tag{5.110}$$

$$h^{\phi}_{m,v} = \frac{1}{i\omega\mu}\left[\frac{\partial}{\partial z}e^{\rho}_{m,v} - \frac{\partial}{\partial \rho}e^{z}_{m,v}\right] \tag{5.111}$$

$$h^{z}_{m,u} = \frac{1}{i\omega\mu}\left[-\frac{m}{\rho}e^{\rho}_{m,v} + \frac{1}{\rho}\frac{\partial}{\partial\rho}(\rho e^{\phi}_{m,u})\right]. \tag{5.112}$$

The first three equations, (5.107)–(5.109), describe the relationship between the electric and magnetic fields excited by a horizontally polarized incident field, whereas the second three equations, (5.110)–(5.112), describe the relationship for the fields excited by a vertically polarized incident field.

Instead of computing the magnetic fields directly from the electric fields, it is desirable to compute them in terms of the envelope functions, in order to avoid the need to estimate derivatives of exponential functions. Using (5.104), equations (5.107)–(5.109) can be written as,

$$h^{\rho}_{m,v} = \frac{e^{\pm ikz}}{i\omega\mu}\left[-\frac{m}{\rho}\psi^{z}_{m,u} - \frac{\partial}{\partial z}\psi^{\phi}_{m,v} \mp ik\psi^{\phi}_{m,v}\right] \tag{5.113}$$

$$h^{\phi}_{m,u} = \frac{e^{\pm ikz}}{i\omega\mu}\left[\frac{\partial}{\partial z}\psi^{\rho}_{m,u} \pm ik\psi^{\rho}_{m,u} - \frac{\partial}{\partial \rho}\psi^{z}_{m,u}\right] \tag{5.114}$$

$$h^{z}_{m,v} = \frac{e^{\pm ikz}}{i\omega\mu}\left[\frac{m}{\rho}\psi^{\rho}_{m,u} + \frac{1}{\rho}\frac{\partial}{\partial \rho}(\rho\psi^{\phi}_{m,v})\right]. \tag{5.115}$$

Similar equations exist for the second set of equations, (5.110)–(5.112). Discretizing (5.113)-(5.115) using the same notation as before yields,

$$h^{\rho}_{n,l} = \frac{e^{\pm ikn\Delta z}}{i\omega\mu}\left[-\frac{m}{l\Delta\rho}\psi^{z}_{n,l} - \frac{\psi^{\phi}_{n+1,l} - \psi^{\phi}_{n-1,l}}{2\Delta z} \mp ik\psi^{\phi}_{n,l}\right] \tag{5.116}$$

$$h^{\phi}_{n,l} = \frac{e^{\pm ikn\Delta z}}{i\omega\mu}\left[\frac{\psi^{\rho}_{n+1,l} - \psi^{\rho}_{n-1,l}}{2\Delta z} \pm ik\psi^{\rho}_{n,l} - \frac{\psi^{z}_{n,l+1} - \psi^{z}_{n,l-1}}{2\Delta\rho}\right] \tag{5.117}$$

$$h^{z}_{n,l} = \frac{e^{\pm ikn\Delta z}}{i\omega\mu}\left[\frac{m}{l\Delta\rho}\psi^{\rho}_{n,l} + \frac{(l+1)\psi^{\phi}_{n,l+1} - (l-1)\psi^{\phi}_{n,l-1}}{2l\Delta z}\right]. \tag{5.118}$$

Note that, in order to accurately calculate the magnetic fields, central difference approximations were used in the above discretizations. Since the magnetic fields depend on the surrounding electric field components, the electric fields must be stored on three adjacent surfaces even though only the fields from the middle surface will be used in the RCS calculation. Once the magnetic fields have been calculated, the RCS can be calculated by applying the far-field formulation described in Appendix A.

5.7 Results

In order to test the BOR PWE algorithm developed in the proceeding sections, the method was implemented for both forward and back scattering along the axis of symmetry. While the paraxial direction of the current implementation is limited to be in the $\pm\hat{z}$ directions, the angle of incidence is not restricted. In the following sections, the bistatic RCS of various targets is compared with BOR MoM predictions to determine the accuracy limitations of the method. In addition, the results are compared with Geometrical Optics (GO) and the Physical Theory of Diffraction (PTD) predictions to determine the accuracy advantage of the BOR PWE method.

5.7.1 Bistatic RCS of Small Cylinder

The first case considered is that of the cylinder, shown in Figure 5-3, which is illuminated by an incident wave traveling in the $+\hat{z}$ direction. In this case, the paraxial direction is chosen to be in the $\hat{z}$ direction so that forward scattering bistatic RCS results are obtained. In addition to

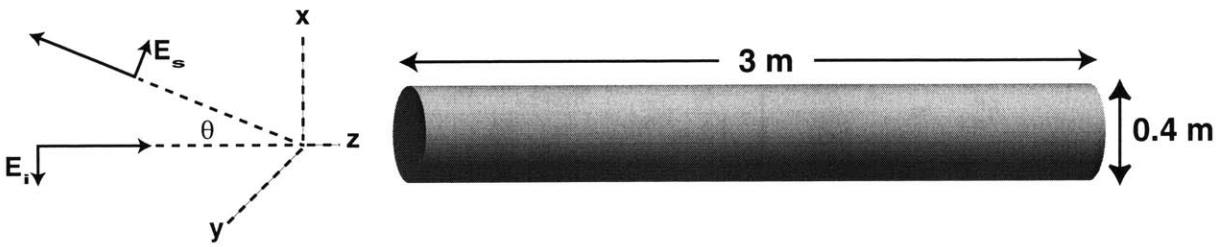

Figure 5-3: Geometry for Cylinder

testing the BOR PWE algorithm, this example serves to test the BOR PWE PML absorbing boundary condition. In contrast to time domain methods, where time gating can be used to test the effectiveness of an absorbing boundary condition, an ABC used in a frequency domain method, such as the BOR PWE method, can only be tested by comparing final results, such as RCS, with and without the ABC. Figure 5-4 compares BOR PWE predictions for three different boundary conditions with BOR MoM predictions. The first two BOR PWE curves shown indicate the results obtained by using a perfectly conducting boundary condition at 10 and 50 cells above the target. As expected, as the distance between the target and the edge of the computational domain is increased, the accuracy of the results increases as well. In contrast, the last BOR PWE curve was obtained by using a PML absorbing condition 5 cells thick that begins 5 cells above the target. As evidenced in the plot, higher accuracy is obtained with the PML in place, which validates the PML formulation developed for the BOR PWE method.

In the previous case, the paraxial direction was chosen to be in the same direction as the incident wave. However, often times, the quantity of interest is backscatter RCS. Figure 5-5 shows the bistatic HH RCS for the same cylinder considered above for bistatic angles near backscatter. As discussed previously, the step size, Δz, must be reduced to accurately represent the phase at the boundary condition. As shown in the plot, the results obtained with $\Delta z = \lambda/30$ improves over the results obtained with $\Delta z = \lambda/15$, however, smaller range step sizes did not improve the results obtained. This implies, as expected, that the BOR PWE method is limited

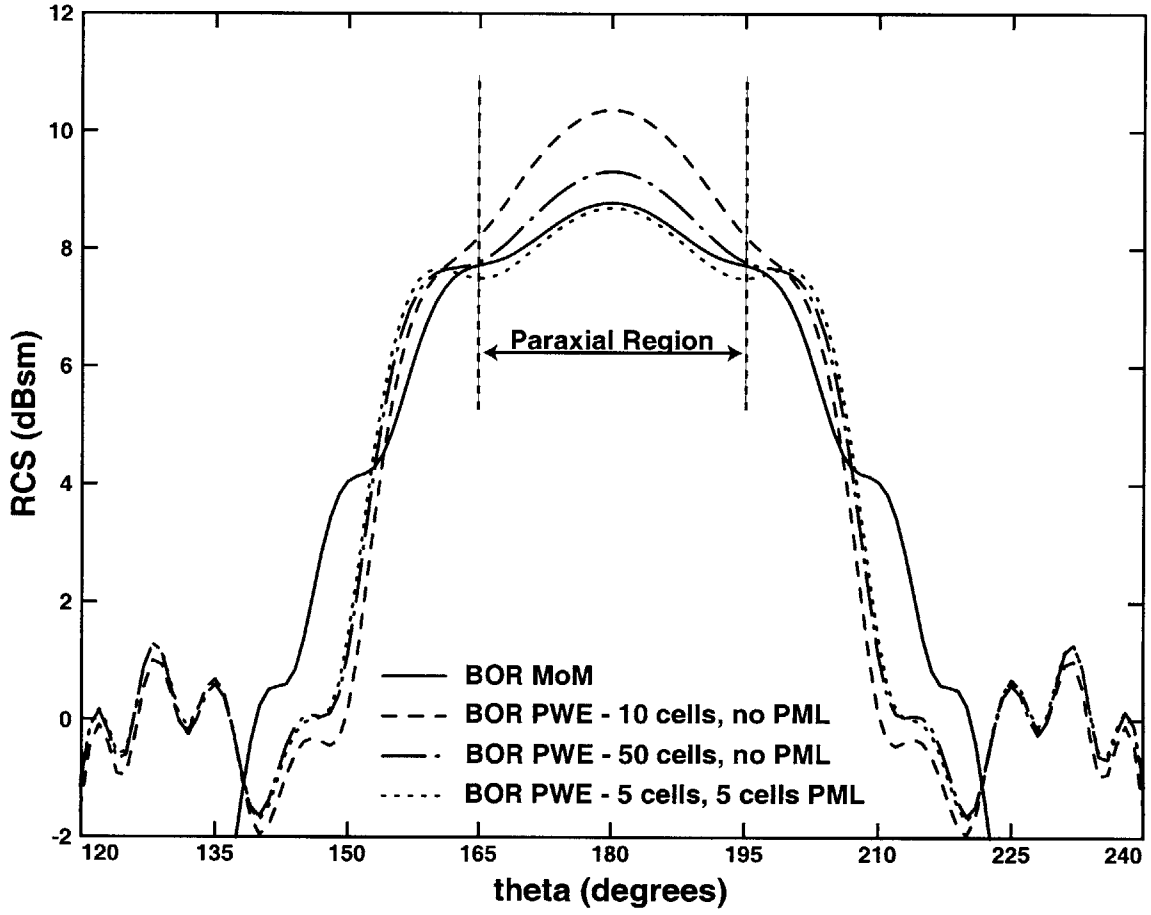

Figure 5-4: Bistatic RCS at 1 GHz of a cylinder illuminated at normal incidence obtained with paraxial direction in $+\hat{z}$ direction. Shown is the HH polarization for a cut in θ with $\phi = 0°$. Results obtained without using PML with 10 and 50 cell spacings are compared to the result obtained by using 5 cells of PML with a 5 cell spacing.

in capturing the effects of two way scattering phenomena that is present in a cylinder of this size. In the next section, the size of the cylinder is increased to determine the improvement, if any, of the BOR PWE method's predictions.

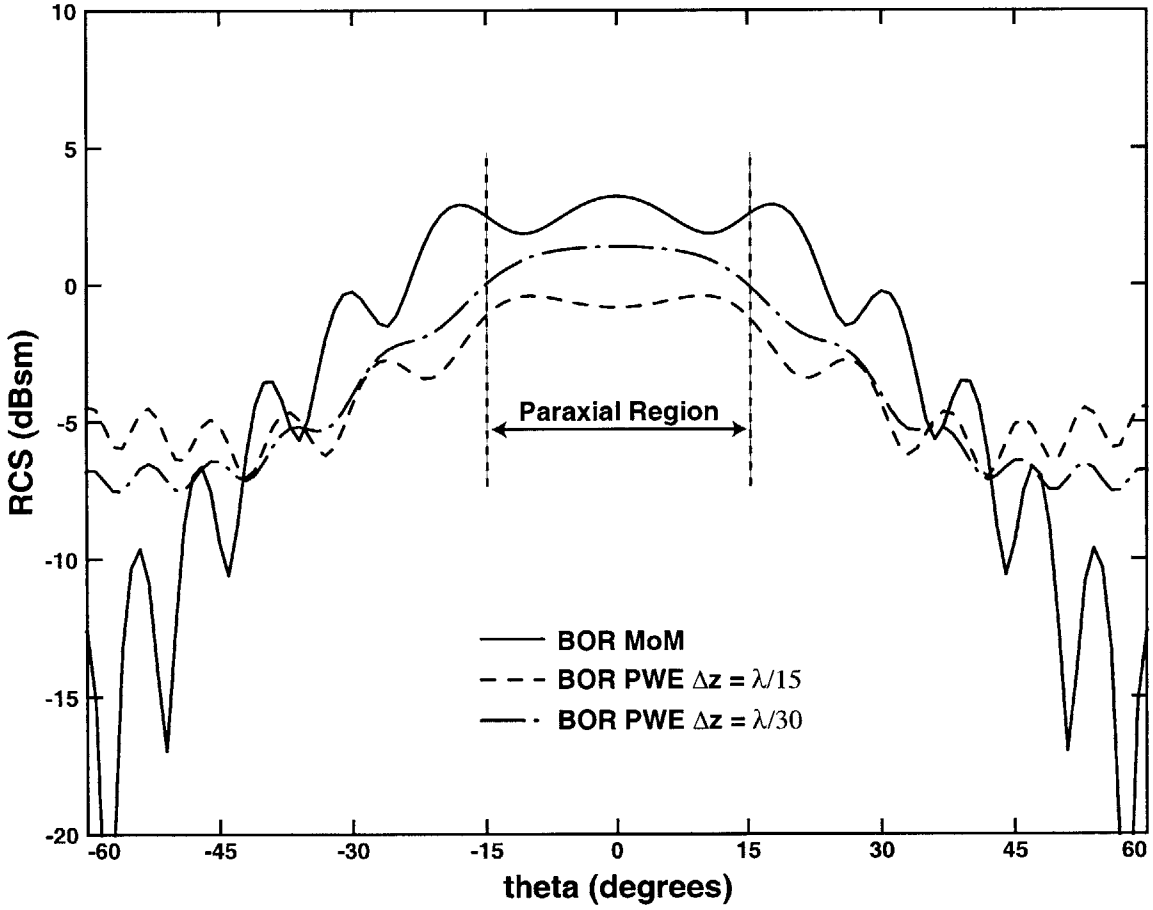

Figure 5-5: Bistatic RCS at 1 GHz of a cylinder illuminated at normal incidence obtained with paraxial direction in $-\hat{z}$ direction. Shown is the HH polarization for a cut in θ with $\phi = 0°$.

5.7.2 Bistatic RCS of Large Cylinder

In this section, the BOR PWE method is used to model the cylinder shown in Figure 5-6, which at 1 GHz has a length of 10 wavelengths and a radius of 2 wavelengths. Due to increased area of

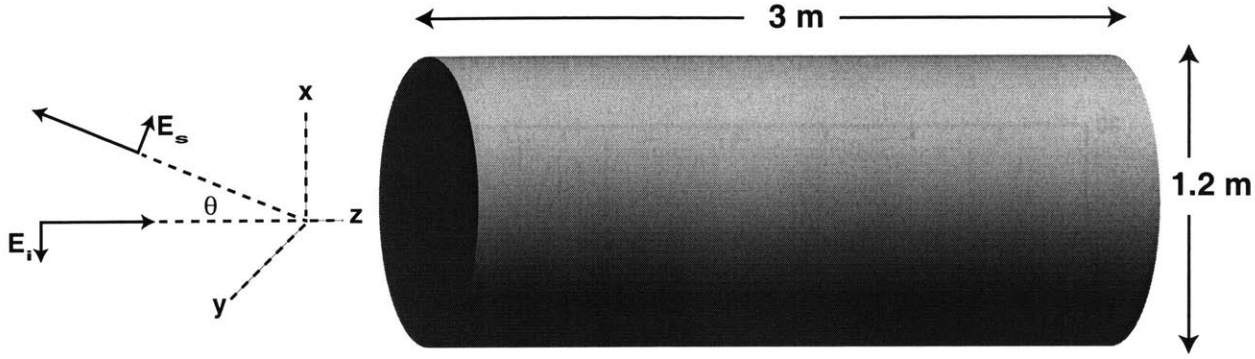

Figure 5-6: Geometry for larger Cylinder

the end-cap, it is expected that the BOR PWE method's predictions of forward and backward scattering will improve. As before, the first case considered is normal incidence at the endcap. Due to the size of the cylinder and the angle of incidence the scattering will be dominated by the reflection and shadowing effects of the endcaps. With the paraxial direction set in the $+\hat{z}$ direction the results shown in Figure 5-7 are obtained. As evidenced by the plot, the BOR PWE method's predictions, within the paraxial region, are in good agreement with the BOR MoM predictions. Because the scattering is dominated by the effects of the endcaps, it was possible to use a large range step size of $\lambda/2$, although the step size in ρ was kept at $\lambda/15$. Also shown in the plot is the bistatic RCS calculations computed by the GO/PTD method, which over estimates the forward scattering cross sections. In additional there is a small discontinuity for the bistatic angle $\theta = 180°$. The errors in the GO/PTD calculations are most likely due to the fact that the high frequency technique has difficultly precisely modeling the shadowing effects.

While the previous examples calculated the bistatic RCS for the HH polarizations, it is also useful to compare the results for the VV polarization. As before, the cylinder is illuminated normal to its endcap and the paraxial direction is set in the $+\hat{z}$ direction. As shown in Figure 5-8, the BOR PWE method's predictions for aspect angles within the paraxial region are in good agreement with the BOR MoM predictions. As before, the predictions are also compared

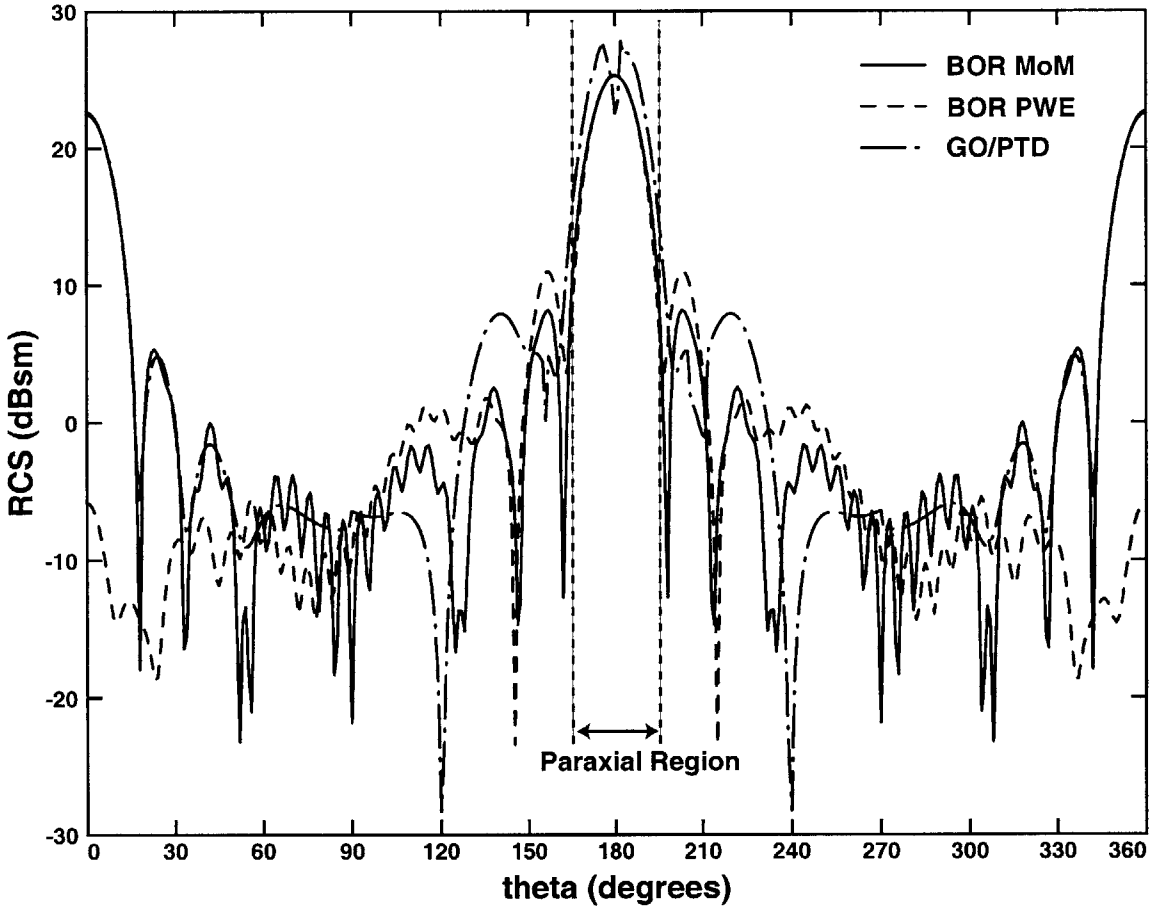

Figure 5-7: Bistatic RCS at 1 GHz of larger cylinder illuminated at normal incidence obtained with paraxial direction in $+\hat{z}$ direction. Shown is the HH polarization for a cut in θ with $\phi = 0°$.

with the results predicted by the GO/PTD method. Similar to the HH polarization case, the GO/PTD method over predicts the forward scattering RCS and exhibits a discontinuity. Still, while the BOR PWE method's forward scattering predictions are more accurate than the GO/PTD method, it remains to be shown that the BOR PWE can accurately predict the bistatic RCS for angles near backscatter.

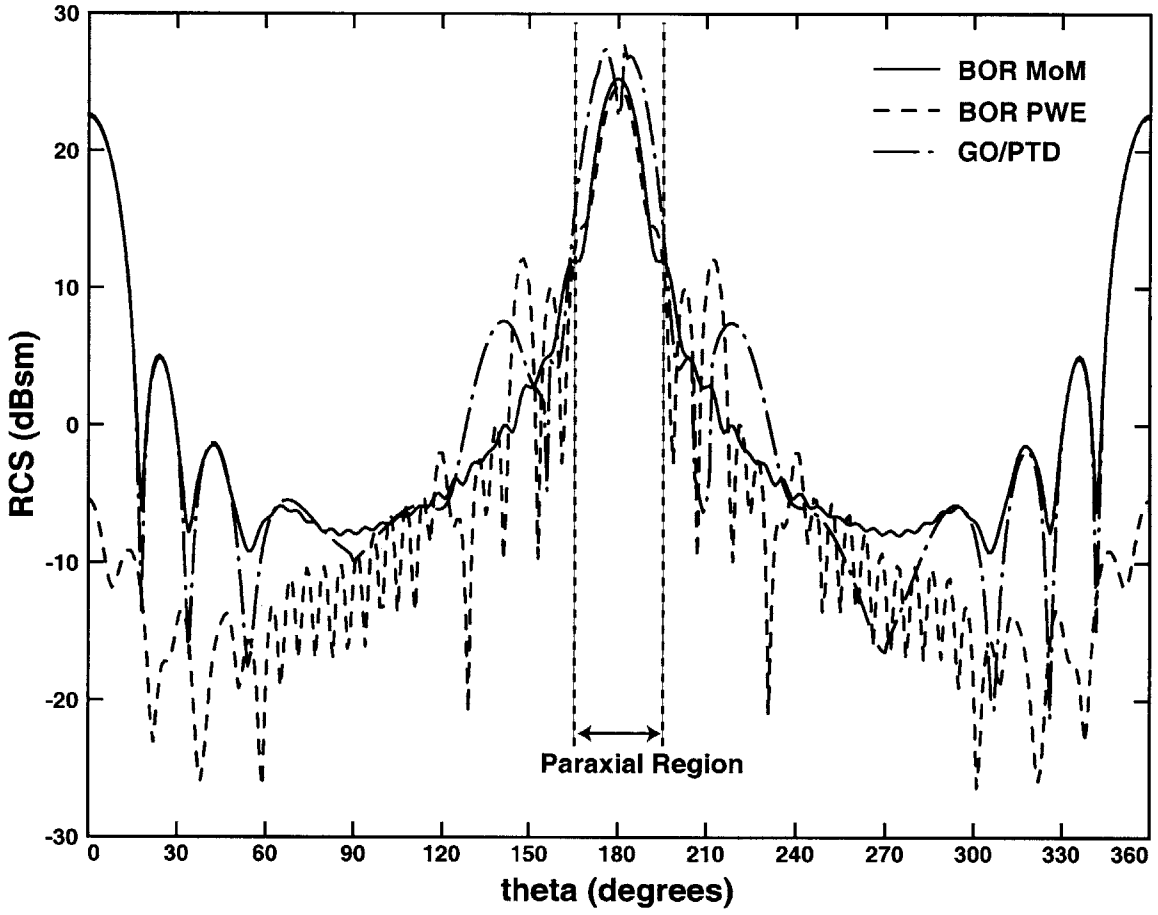

Figure 5-8: Bistatic RCS at 1 GHz of larger cylinder illuminated at normal incidence obtained with paraxial direction in $+\hat{z}$ direction. Shown is the VV polarization for a cut in θ with $\phi = 0°$.

Shown in the Figure 5-9 is the bistatic RCS, for normal incident at the endcap, calculated by using the BOR PWE method with the paraxial direction set in the $-\hat{z}$. Since the dominating scattering phenomena is the specular reflection off the endcap, the range step size does not need to be decreased. Within the paraxial region, the predictions from the BOR PWE method compare well with the exact predictions of the BOR MoM method. While the GO/PTD method was not able to accurately predict the forward scattering results, it produced accurate results

for bistatic angles near backscattering. This is due to the fact that the GO technique is most accurate in modeling specular reflections, which in this case dominates the total return.

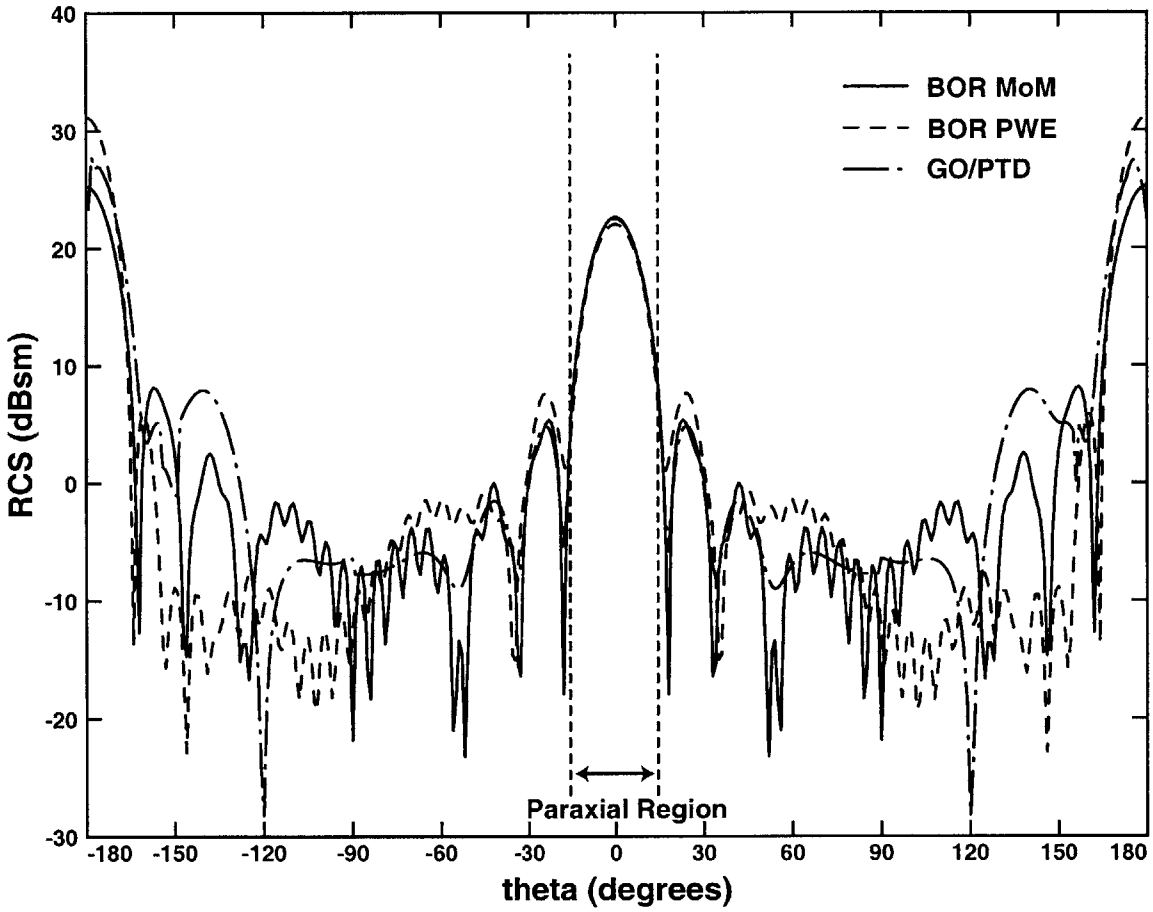

Figure 5-9: Bistatic RCS at 1 GHz of larger cylinder illuminated at normal incidence obtained with paraxial direction in $-\hat{z}$ direction. Shown is the HH polarization for a cut in θ with $\phi = 0°$.

While the BOR PWE formulation presented in this chapter is limited to setting the paraxial direction in the $\pm\hat{z}$ direction, the direction of the incident wave is not restricted. In the following, the incident direction is chosen to be 45° off axis, as shown in Figure 5-10. The bistatic RCS of the cylinder for this incident direction was computed with the BOR PWE method, and results compared to BOR MoM and GO/PTD predictions. Figure 5-11 shows the results obtained by setting the paraxial direction in the $+\hat{z}$ direction. Although the paraxial approximation is generally limited to a ±15° region, in this case, it is clear that the paraxial approximation remained valid for a wider range of angles. The BOR PWE method captured the specular reflection at $\theta_s = 90°$, shadowing effects at $\theta_s = 180°$, as well as several other angles.

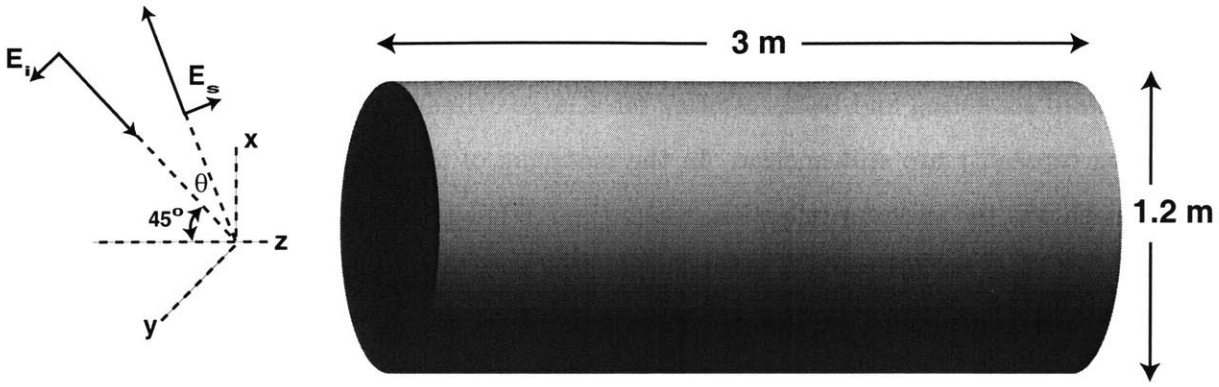

Figure 5-10: Geometry for cylinder, $\theta_i = 45°$.

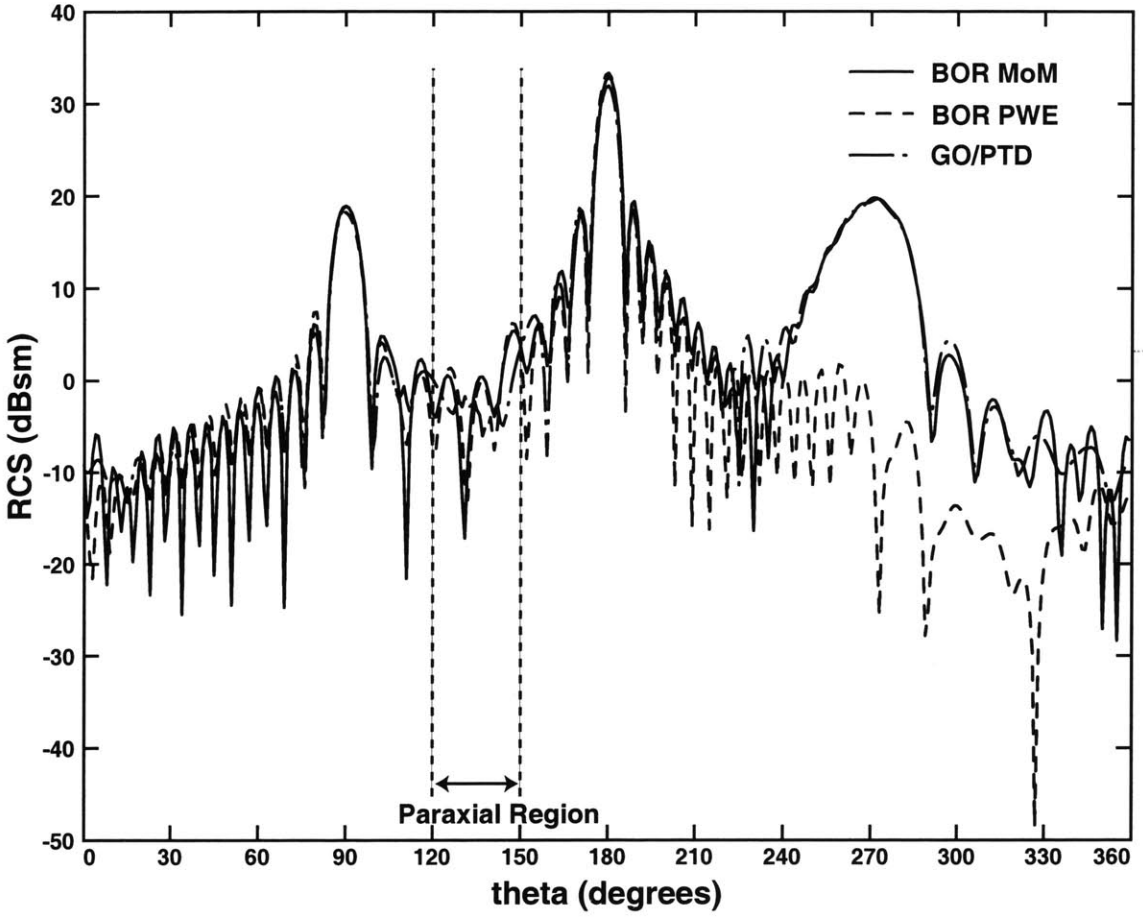

Figure 5-11: Bistatic RCS at 1 GHz of cylinder geometry illuminated at $\theta_i = 45°$ obtained with paraxial direction in $+\hat{z}$ direction. Shown is the HH polarization for a cut in θ with $\phi = 0°$.

5.7.3 Bistatic RCS of the Sphere-Cylinder and Biconical Targets

While the BOR PWE method was able to predict the bistatic RCS of a cylinder for angles near forward scattering and backscattering for a large cylinder, the limitations in terms of its modeling capability are still unclear. In the modeling of the cylinder, the dominating wave phenomena was the specular reflection, which the PWE was able to capture. However, in the modeling of more realistic targets other phenomena such as traveling waves or creeping waves exist that can contribute significantly to the RCS of the target. In this section, the sphere-cylinder and biconical targets, shown in Figure 5-12 and Figure 5-15, are modeled to better understand the capabilities and limitations of the BOR PWE method.

The first target modeled is the sphere-cylinder geometry, which is shown in Figure 5-12. Due

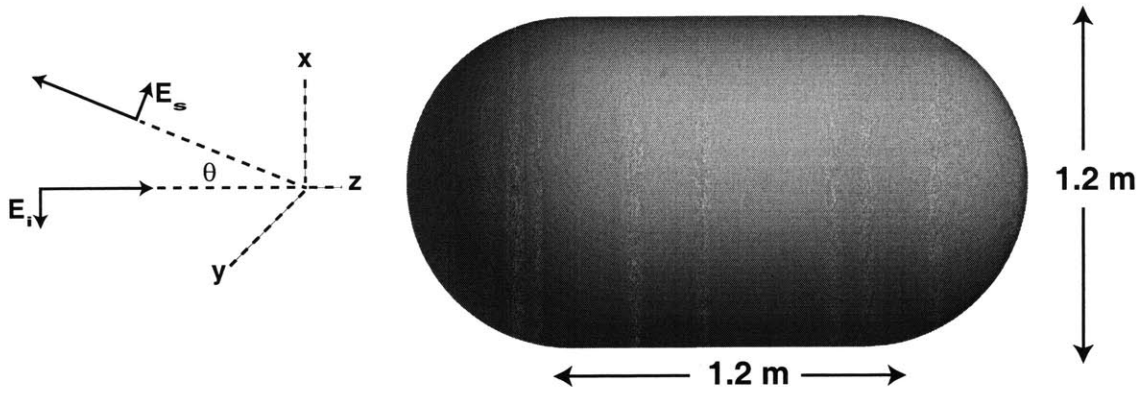

Figure 5-12: Sphere-Cylinder Geometry

to the large cross section area of the target, the bistatic RCS for angles near forward scattering will be similar to RCS of a cylinder. For this reason, it is expected that the BOR PWE method will be able to accurately predict bistatic RCS for angles near forward scattering. As evidenced in Figure 5-13, the BOR PWE predictions compare well with those of the BOR MoM method. In this case, most likely due to the smooth surface of the target, the GO/PTD method also produced accurate results without the discontinuity that was present in the previous examples.

While it is clear that the BOR PWE method can accurately predict the bistatic RCS for angles near forward scattering, the RCS prediction for bistatic angles near backscatter will be more stressing for the BOR PWE method since many types of wave phenomena will contribute to the overall RCS. Figure 5-14 shows the bistatic RCS predictions, for angles near backscatter, obtained with the BOR MoM and BOR PWE methods as well as the results from the GO/PTD

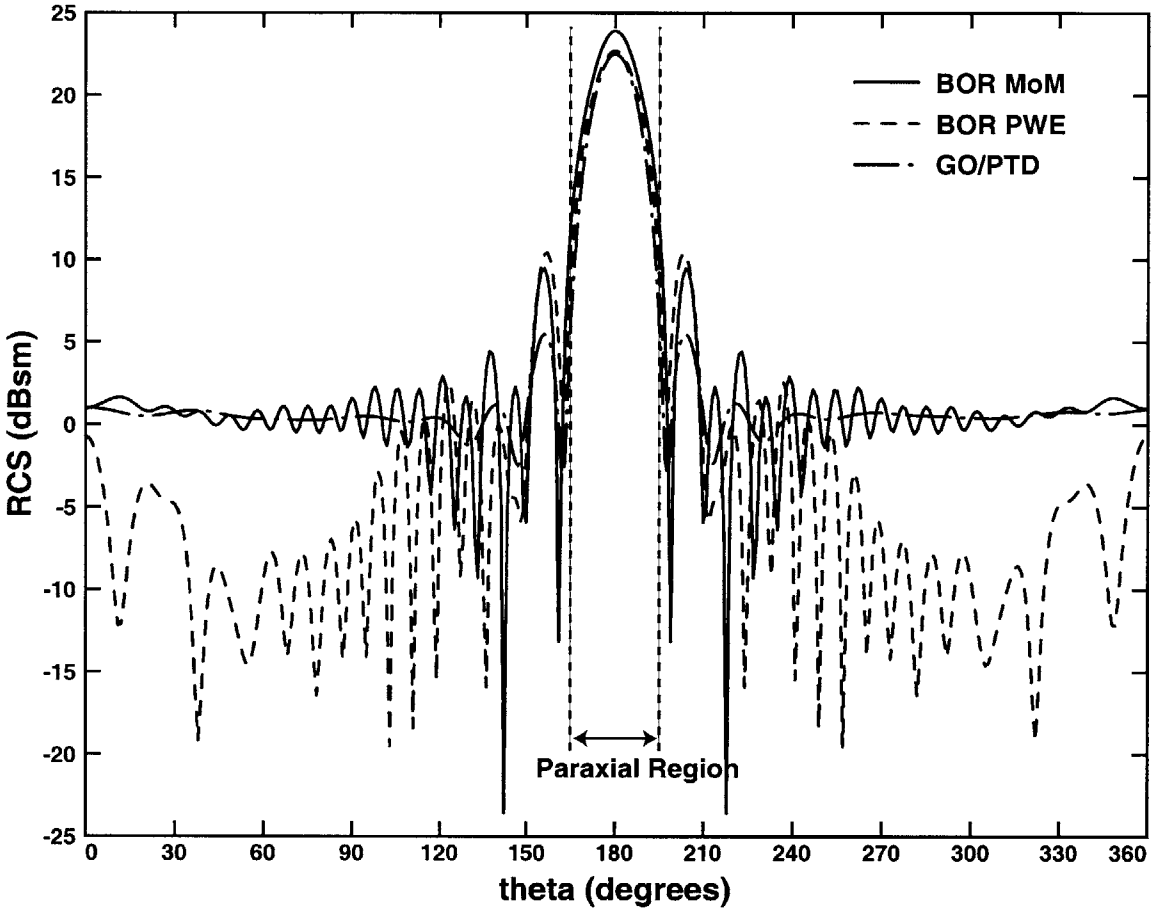

Figure 5-13: Bistatic RCS at 1 GHz of sphere-cylinder geometry illuminated at normal incidence obtained with paraxial direction in $+\hat{z}$ direction. Shown is the HH polarization for a cut in θ with $\phi = 0°$.

method. The two BOR PWE results were obtained by setting the paraxial direction in the

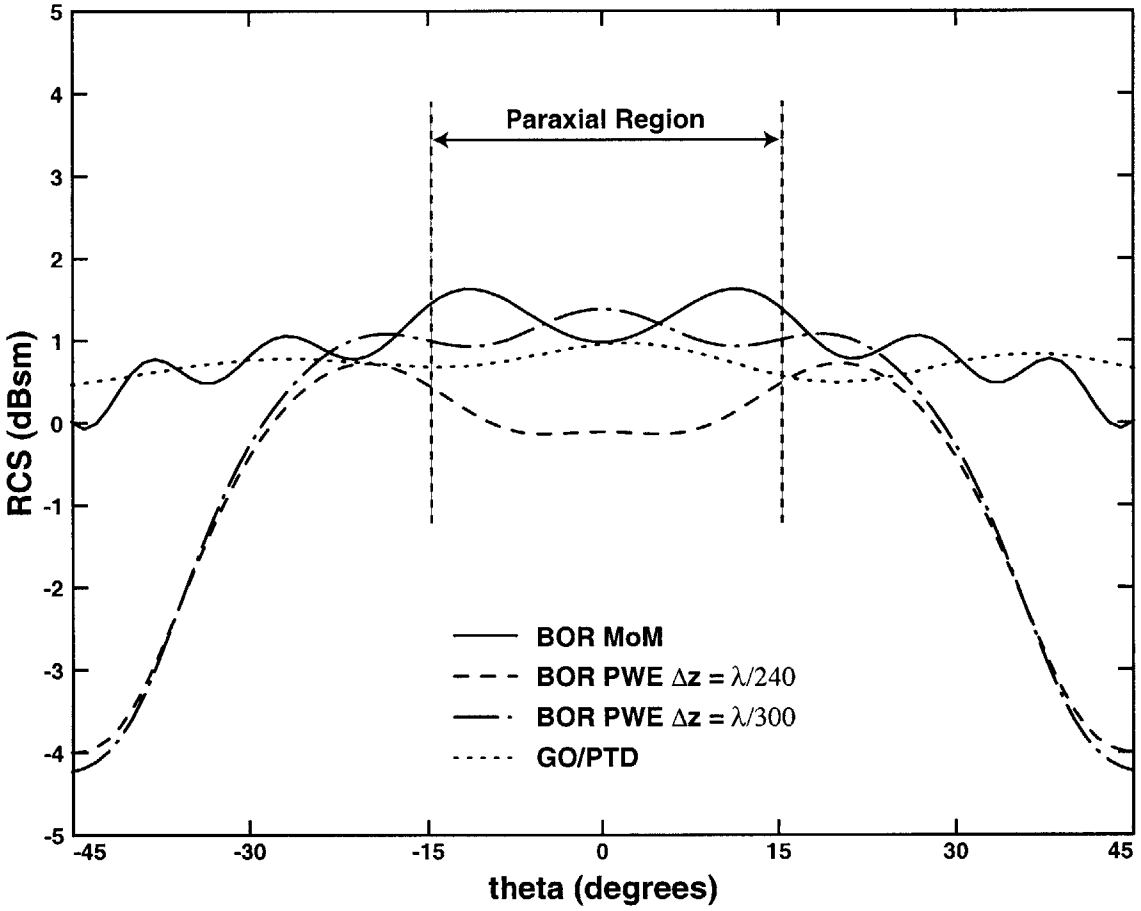

Figure 5-14: Bistatic RCS at 1 GHz of sphere-cylinder geometry illuminated at normal incidence obtained with paraxial direction in $-\hat{z}$ direction. Shown is the HH polarization for a cut in θ with $\phi = 0°$.

$-\hat{z}$ direction using range step sizes of $\lambda/240$ and $\lambda/300$. While step sizes this small are not required to represent the phase variation in the boundary condition, it was found that they were necessary to accurately model the curved surface of the target. Within the paraxial region, the BOR PWE and GO/PTD predictions are both within 1 dBsm of the BOR MoM predictions although the general shape of the curves do not match well with the BOR MoM curve, which indicates that neither method captured all of the wave phenomena present.

In the next example, the biconical object shown in Figure 5-15, is modeled using the PWE method. Since the PWE method can accurately represent the curvature of this surface, the potential for errors due to discretization will be reduced. Also, due to the target's size, which at

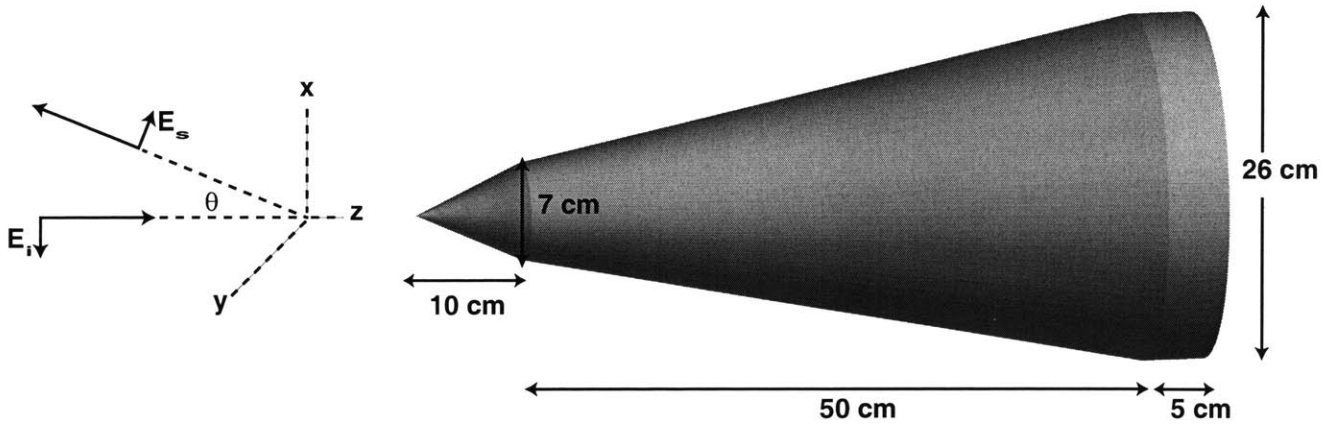

Figure 5-15: Biconical Object Geometry

1 GHz is 2 wavelengths in length and has a maximum diameter of 1 wavelength, it is expected that the GO/PTD results will not be very accurate. As before, the BOR PWE method was run with the paraxial direction set in the $\pm\hat{z}$ directions.

Figure 5-16 shows the bistatic RCS for the biconical object for a wave incident normal to the nosecone. As evidenced in the plot, the results obtained using the BOR PWE method within the paraxial region match the BOR MoM predictions quite well compared to the GO/PTD predictions.

In order to compute the bistatic RCS for angles near backscatter, the paraxial direction was chosen to be in the $-\hat{z}$ direction. As shown in Figure 5-17, the BOR PWE method was run with range step sizes of $\lambda/45$ and $\lambda/60$. For both range step choices, the predictions made by the BOR PWE method were relatively accurate compared to GO/PTD results with more accuracy being obtained with a smaller step size. Despite the 3 dBsm error for the backscatter RCS, the general shape of the BOR PWE curve matches the BOR MoM predictions quite well.

In contrast to the BOR PWE predictions, the GO/PTD backscatter error is larger than 10 dBsm, and the curve's shape does not match the general shape of the BOR MoM predictions.

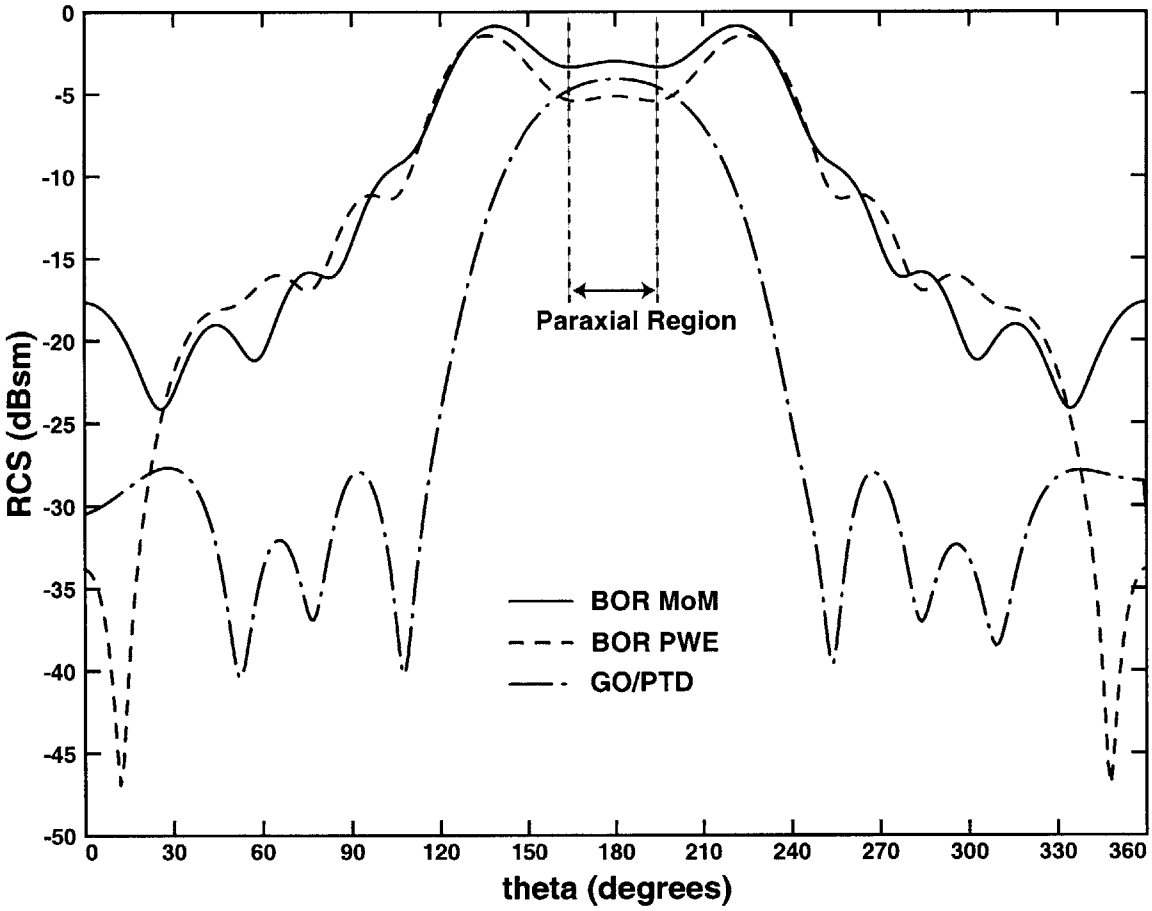

Figure 5-16: Bistatic RCS at 1 GHz of biconical target illuminated at normal incidence obtained with paraxial direction in $+\hat{z}$ direction. Shown is the HH polarization for a cut in θ with $\phi = 0°$.

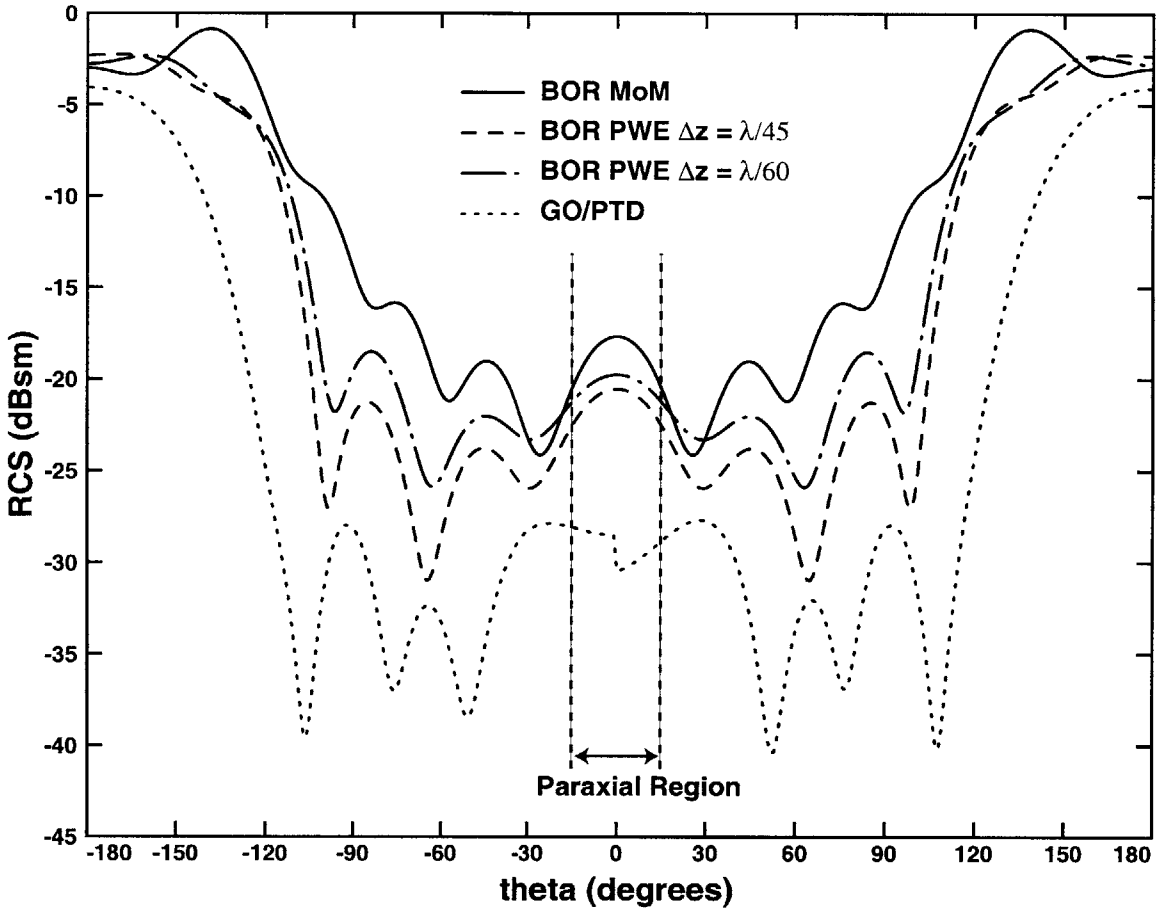

Figure 5-17: Bistatic RCS at 1 GHz of biconical target illuminated at normal incidence obtained with paraxial direction in $-\hat{z}$ direction. Shown is the HH polarization for a cut in θ with $\phi = 0°$.

5.8 Summary

In this chapter, the body of revolution parabolic wave equation method was described and applied to RCS prediction. The time-harmonic vector wave equation was rewritten in terms of slowing varying envelope functions, and the paraxial approximation was used to develop a system of parabolic partial differential equations. These equations were discretized using first and second order accurate difference approximations enabling the development of an efficient marching in space algorithm, which is order L in its computer memory requirement. Perfect electric conductors were modeled by forcing the total tangential electric fields along the surface of the target to be zero. Plane wave sources were modeled through the use of the boundary conditions along the surface of the target so that the paraxial and incident wave directions could be specified independently. The computational domain was truncated by the PML absorbing boundary condition formulated for the BOR PWE method through the use of the complex stretched coordinates viewpoint.

The method was applied to the RCS prediction of several body of revolution targets. The results obtained were compared for validation with BOR MoM predictions, and were compared with GO/PTD predictions to determine the accuracy advantage of the PWE method over high-frequency techniques. In general, the accuracy level of the BOR PWE's predictions for bistatic angles near forward scattering exceeded that of the GO/PTD results. For electrically large targets, the BOR PWE and GO/PTD bistatic RCS predictions were found to perform equally well in the bistatic RCS prediction for angles near backscatter. However, in the case of the biconical target, which is on the order of a wavelength in size, the predictions of BOR PWE were found to be much more accurate than the GO/PTD results indicating the ability of the PWE method to predict near resonant size objects that the GO/PTD method cannot.

Chapter 6

Conclusion

A number of radar cross section prediction techniques have been developed which exploit body of revolution (BOR) symmetry, however the use of finite-difference techniques with these geometries has not been throughly explored. This thesis has investigated several finite-difference approaches which vary both in the approximations they introduce as well as the computational resources they require. These techniques included a body of revolution (BOR) finite-difference time-domain (FD-TD) method with both staircase and conformal grids, a hybrid FD-TD geometrical optics method, and a body of revolution parabolic wave equation method. In addition, the use of the monostatic-bistatic equivalence principle was explored in approximating monostatic RCS at multiple angles from a single FD-TD simulation.

In evaluating the performance of any RCS prediction technique, the issues of accuracy and computational cost must be addressed. While an accurate technique is desirable, it must also be practical for the technique to be applied to the target of interest. The feasibility of a method for modeling a particular geometry relates to the associated computational costs of computer time and memory. For electrically small targets, exact techniques such as the Method of Moments and the FD-TD method can be used to exactly model the target. For the modeling of body of revolution targets, the BOR MoM and BOR FD-TD algorithms can be used to reduce the associated computational costs. In order to understand the limitations of these techniques, it is necessary to understand how the techniques scale in terms of computer time and computer memory as the electrical size of the target increases. Table 6.1 summarizes the computational requirements of several RCS prediction techniques including the three finite difference techniques explored in this thesis.

The methods listed in the first column of the table are Geometrical Optics (GO) and the Physical Theory of Diffraction (PTD). These techniques determine the RCS of the target through the methods of ray tracing and edge diffraction. However, they can only be applied to electrically large and simple targets such as a large cylinder. They do not accurately model the effect of small protrusions nor do they model traveling or creeping waves. Still, they have the smallest computational requirements that do not scale with the electrical size of the target.

On the other hand, the next three methods listed are exact techniques that numerically solve Maxwell's equations. Since the 3D MoM method solves for the currents induced on the surface of the target, the number of unknowns scales as L^2 where L represents the maximum dimension of the target. In implementing the MoM method, an $L^2 \times L^2$ linear system is formed that results in a memory requirement of order L^4 and a computational time of order L^6. Due to the large scaling factors of the method, in practice, only electrically small targets can be modeled with this method.

If, however, the target exhibits body of revolution symmetry, the amount of computation can be reduced. Two approaches that exploit BOR symmetry are the frequency domain Method of Moments (MoM) BOR algorithms, and the Finite-Difference Time-Domain (FD-TD) BOR implementations. As shown in Table 6.1, the computational requirements for both methods are very similar. In cases where the signature is desired over an extended bandwidth, FD-TD BOR techniques have the advantage of calculating the entire frequency extent simultaneously.

In the first part of this thesis, the BOR FD-TD algorithm was successfully implemented and used for RCS prediction of both canonical and realistic targets. Initially, targets were modeled using a staircase representation, and the results obtained were compared with BOR MoM and exact results for validation. Due to numerical dispersion, it was necessary to use a discretization on the order of $\lambda/40$ to model waves incident along the axis. For waves incident off-axis, a discretization on the order of $\lambda/10$–$\lambda/20$ was found to be sufficient.

To address the issue of accuracy, a conformal approach to the BOR FD-TD method was developed. It was shown, in the case of the sphere, that a staircase model can lead to errors in RCS predictions due the difficulty of correctly predicting phenomena as traveling and creeping waves. The conformal approach was able to capture these effects by accurately modeling the surface of the target. In the case of the sphere, the conformal BOR FD-TD approach was able to accurately model the creeping wave. In the case of the biconical target, it was shown that the conformal approach was able to model the scattering from the nosecone more accurately

	GO/PTD	3D MoM	BOR MoM	BOR FDTD	2D FDTD/GO	BOR PWE
Method	Ray tracing, edge diffraction.	Exact (Solve Integral Eqns)	Exact (Solve integral Eqns)	Exact (Solve PDES)	Specular point reflection + exact	Solve reduced scalar wave equations
Number of unknowns		L^2	$L \cdot L$	$L \cdot L^2$	L^2	$L \cdot L^2$
Memory requirement		L^4	L^2	L^2	L^2	L
Computer time		L^6 Matrix decomposition	$L \cdot L^3$ Matrix decomposition	$L \cdot L^3$ Time stepping	L^3 Time stepping	$L \cdot L^2$ Range stepping
Disadvantages/ limitations	High freq only. Neglects many phenomena. Small protrusions ignored.	Low freq. only Single freq.	Low freq. only Single freq.	Low freq. only Single incident angle.	High freq. Only accurate for certain aspect angles	Single freq. Only accurate for small range of angles.
Advantages	Easiest computationally	Exact	Exact Memory efficient	Exact Memory efficient	Fast computation. Includes many phenomena.	Very memory efficient

Table 6.1: Summary of RCS Prediction Techniques

than the staircase approach.

However, one disadvantage of the FD-TD method is that the FD-TD simulation must be repeated for each incident angle of interest, and as the target size becomes large, and wideband signatures required at many incident directions, additional reductions in computation are desirable. To reduce the computational burden associated with calculating monostatic signatures with the FD-TD method, two approaches were used.

The first approach reduced the overall computational burden by reducing the number of angles at which calculations must be performed. A single FD-TD BOR simulation was used to calculate the monostatic signature for one incident angle, as well as bistatic signatures for adjacent observation directions. The bistatic equivalence theorem was then used to approximate monostatic signatures for other angles near the incident direction of the actual FD-TD BOR simulation. The principle was applied to the monostatic RCS prediction of a simple cylinder and a biconical shaped object. In the modeling of the biconical target it was shown that the PO/PTD method's predictions were only accurate at aspect angles incident on the target's broadside and backend, while the estimates obtained by the BOR FD-TD and monostatic/bistatic equivalence method were relatively accurate for all aspect angles.

Thus, if wideband monostatic signatures over several aspect angles are desired, the monostatic-bistatic equivalence can be used with the BOR FD-TD method to estimate these signatures. However, for narrowband signatures, the BOR MoM technique has the advantage of being able to exactly calculate the monostatic signature over several aspect angles in one simulation.

In contrast, the second approach reduces computational requirements for BOR objects of large electrical radius by using a hybrid FD-TD and Geometrical Optics formulation. Individual scattering centers such as surface gaps, protrusions, or slope discontinuities are identified, and integral expressions derived for the scattering of each. These expressions are evaluated by the method of stationary phase, in which the contribution is assumed to arise from a stationary phase point in the plane of incidence. A two-dimensional scattering problem is created by a local tangent plane approximation through the stationary phase point, and this is solved via a two-dimensional FD-TD approach.

The specific case studied was the backscattering from a large cylinder with a small ring. For a large cylinder, the hybrid method was able to accurately predict the backscatter RCS for a large range of angles. In addition, the method was shown to have an accuracy advantage over the PO/PTD method which does not correctly model the small protrusion. Moreover, the

method is computational efficient compared to exact BOR MoM and BOR FD-TD methods, which would otherwise be required to accurately mode this mixed size. As shown in Table 6.1, the computer memory requirement scales at the same rate of the BOR methods, however, because only the small protrusion is being modeled, the computational requirements are much smaller than those of the full BOR methods.

Although the hybrid method is capable of accurately modeling large targets with small protrusions, it is limited to a specific type of geometry. To remove this limitation, a body of revolution (BOR) parabolic wave equation (PWE) method was developed. The time-harmonic vector wave equation was rewritten in terms of slowing varying envelope functions. As with the BOR FD-TD method, axial symmetry was exploited by expressing the azimuthal dependence of the fields in a Fourier series. The paraxial approximation was then used to develop a system of parabolic partial differential equations that were solved using a memory efficient marching in space approach.

The method was applied to the RCS prediction of several body of revolution targets. The results obtained were compared for validation with BOR MoM predictions, and were compared with GO/PTD predictions to determine the accuracy advantage of the PWE method over high-frequency techniques. In general the forward scattering predictions made by the BOR PWE method were more accurate than GO/PTD predictions. However, in most cases, the BOR PWE and GO/PTD performed equally well in the prediction of the bistatic RCS for angles near backscatter. Still, in the case of the biconical target, which is on the order of a wavelength in size, the predictions of BOR PWE were found to be much more accurate than the GO/PTD results indicating the ability of the PWE method to model smaller targets for which the GO/PTD approximations are not valid.

Because the BOR PWE formulation reduces the scattering problem to a sequence of two-dimensional problems, which are solved using the marching in space approach, the computer memory requirement grows as order L. In addition, as shown, the BOR PWE has the advantage over high-frequency techniques being able to correctly model resonant size object scattering. Still, due to the one-way nature of the technique, the PWE method does not perform well in the modeling of non-convex objects and cavities where multiple scattering interactions can occur. Another difficulty involves the modeling of objects small compared to a wavelength where creeping waves can travel all the way around the object. Creeping waves that travel around the object more than once can not be captured with the PWE method due to the

one-way nature of the technique.

Much work remains to be done in the use of finite difference method for modeling body of revolution targets. As mentioned in Chapter 2, the BOR FD-TD suffers from large numerical dispersion errors along the axis of symmetry. Future work may include the development of a technique to reduce the numerical dispersion by analytically accounting for the numerical phase velocity of waves traveling in the BOR FD-TD lattice. In addition, in order to model other body of revolution targets such as radomes, a BOR FD-TD method for modeling targets with dielectric surfaces could be implemented. Furthermore, in many cases, objects such as missiles, exhibit body of revolution structure except for localized 3D features such as fins. The BOR FD-TD technique could be combined with a general 3D FD-TD code to model the interaction between the BOR portion of the object and the non-BOR portion of the object.

As discussed in Chapter 4, the hybrid method can also be applied to large targets other than cylinders. Here, much work can be done to explore the range of validity of the hybrid method for modeling small BOR protrusions on other large BOR targets such as a cone. Moreover, work could be done to explore the use of combining high frequency techniques with general three dimensional exact techniques for modeling small 3D structures that exist on large bodies.

Finally, while the results obtained using the BOR PWE method are encouraging, much work remains to further develop the BOR PWE technique. For example, BOR PWE formulations for paraxial directions off-axis can be developed to extend its range of accuracy. Additionally, work can be done to explore the use of variable range step sizes to reduce the amount of computation. Also, as mentioned previously, the condition number of the resulting matrix equations are often very large for targets with large number of boundary conditions to enforce at one range step. Further work here could include the development of methods for reducing the condition number and exploring the use of other iterative techniques for solving the resulting matrix equations. Finally, due to the one-way nature of the technique, a method that uses a back and forth algorithm could be developed to improve the method's accuracy.

In general, each of techniques presented in thesis has been tested for several examples in addition to the ones presented here. However, further testing of each of the methods is required to better understand their capabilities and limitations.

Appendix A

Calculation of Far-Field Scattered Fields for BOR Geometries

A.1 General 3D Formulation

The scattered fields in the far-field region can be determined from the electric and magnetic fields on a surface S' by the following equations.

$$\vec{E}(\overline{r}) = \iint_{S'} dS' \left\{ i\omega\mu_0 \overline{\overline{G}}(\overline{r},\overline{r}') \cdot \left[\hat{n} \times \vec{H}(\overline{r}')\right] + \nabla \times \overline{\overline{G}}(\overline{r},\overline{r}') \cdot \left[\hat{n} \times \vec{E}(\overline{r}')\right] \right\} \quad \text{(A.1)}$$

$$\vec{H}(\overline{r}) = \iint_{S'} dS' \left\{ -i\omega\epsilon_0 \overline{\overline{G}}(\overline{r},\overline{r}') \cdot \left[\hat{n} \times \vec{E}(\overline{r}')\right] + \nabla \times \overline{\overline{G}}(\overline{r},\overline{r}') \cdot \left[\hat{n} \times \vec{H}(\overline{r}')\right] \right\} \quad \text{(A.2)}$$

$$\hat{r} = \hat{x}\sin\theta\cos\phi + \hat{y}\sin\theta\sin\phi + \hat{z}\cos\theta \quad \text{(A.3)}$$

$$\hat{\theta} = \hat{x}\cos\theta\cos\phi + \hat{y}\cos\theta\sin\phi - \hat{z}\sin\theta \quad \text{(A.4)}$$

$$\hat{\phi} = \hat{y}\cos\phi - \hat{x}\sin\phi \quad \text{(A.5)}$$

$$\overline{\overline{G}}(\overline{r},\overline{r}') = \left[\overline{\overline{I}} + \frac{1}{k^2}\nabla\nabla\right] \frac{e^{ik|\overline{r}-\overline{r}'|}}{4\pi|\overline{r}-\overline{r}'|} \quad \text{(A.6)}$$

In the far field, $ikr \gg 1$, ∇ can be approximated as $ik\hat{r}$,

$$\vec{E}(\overline{r}) = \iint_{S'} dS' \left\{ i\omega\mu_0 \frac{e^{ik|\overline{r}-\overline{r}'|}}{4\pi|\overline{r}-\overline{r}'|} \left[\overline{\overline{I}} - \hat{r}\hat{r}\right] \cdot \left[\hat{n} \times \vec{H}(\overline{r}')\right] + ik\hat{r} \times \frac{e^{ik|\overline{r}-\overline{r}'|}}{4\pi|\overline{r}-\overline{r}'|} \left[\overline{\overline{I}} - \hat{r}\hat{r}\right] \cdot \left[\hat{n} \times \vec{E}(\overline{r}')\right] \right\} \quad \text{(A.7)}$$

In addition, under the far-field approximation,

$$|\overline{r} - \overline{r}'| \approx r - \hat{r} \cdot \overline{r}' \tag{A.8}$$

$$\begin{aligned} \vec{E}(\overline{r}) &= \frac{e^{ikr}}{4\pi r} \iint_{S'} dS' e^{-ik\hat{r}\cdot\overline{r}'} \left\{ i\omega\mu_0 \left[\hat{\theta}\hat{\theta} + \hat{\phi}\hat{\phi}\right] \cdot \left[\hat{n} \times \vec{H}(\overline{r}')\right] \right. \\ &\quad \left. + ik \left[\hat{\phi}\hat{\theta} - \hat{\theta}\hat{\phi}\right] \cdot \left[\hat{n} \times \vec{E}(\overline{r}')\right] \right\} \end{aligned} \tag{A.9}$$

A.2 BOR Formulation

Split the integration over the cylinder into three regions as,

$$\iint_{S'} dS' = \underbrace{\int_{z_1}^{z_2} dz' \int_0^{2\pi} \rho_0 d\phi'}_{\rho=\rho_0} + \underbrace{\int_0^{\rho_0} d\rho' \int_0^{2\pi} \rho' d\phi'}_{z'=z_1} + \underbrace{\int_0^{\rho_0} d\rho' \int_0^{2\pi} \rho' d\phi'}_{z'=z_2} \tag{A.10}$$

Substituting in cylindrical coordinate form of the electric and magnetic fields,

$$\begin{aligned} \vec{E}(\overline{r}) &= \frac{e^{ikr}}{4\pi r} \int_{z_1}^{z_2} dz' \int_0^{2\pi} \rho_0 d\phi' e^{-ik\hat{r}\cdot(\rho_0\hat{\rho}' + z'\hat{z})} \\ &\quad \left\{ i\omega\mu \left[\hat{\theta}\hat{\theta} + \hat{\phi}\hat{\phi}\right] \cdot \left[\hat{\rho}' \times \left(H_\rho(\overline{r}')\hat{\rho}' + H_\phi(\overline{r}')\hat{\phi}' + H_z(\overline{r}')\hat{z}'\right)\right] \right. \\ &\quad \left. ik \left[\hat{\phi}\hat{\theta} - \hat{\theta}\hat{\phi}\right] \cdot \left[\hat{\rho}' \times \left(E_\rho(\overline{r}')\hat{\rho}' + E_\phi(\overline{r}')\hat{\phi}' + E_z(\overline{r}')\hat{z}'\right)\right] \right\} \\ &+ \frac{e^{ikr}}{4\pi r} \int_0^{\rho_0} d\rho' \int_0^{2\pi} \rho' d\phi' e^{-ik\hat{r}\cdot(r'\hat{\rho}' + z_2\hat{z})} \\ &\quad \left\{ i\omega\mu \left[\hat{\theta}\hat{\theta} + \hat{\phi}\hat{\phi}\right] \cdot \left[\hat{z}' \times \left(H_\rho(\rho', \phi', z_2)\hat{\rho}' + H_\phi(\rho', \phi', z_2)\hat{\phi}' + H_z(\rho', \phi', z_2)\hat{z}'\right)\right] \right. \\ &\quad \left. ik \left[\hat{\phi}\hat{\theta} - \hat{\theta}\hat{\phi}\right] \cdot \left[\hat{z}' \times \left(E_\rho(\rho', \phi', z_2)\hat{\rho}' + E_\phi(\rho', \phi', z_2)\hat{\phi}' + E_z(\rho', \phi', z_2)\hat{z}'\right)\right] \right\} \\ &- \frac{e^{ikr}}{4\pi r} \int_0^{\rho_0} d\rho' \int_0^{2\pi} \rho' d\phi' e^{-ik\hat{r}\cdot(r'\hat{\rho}' + z_1\hat{z})} \\ &\quad \left\{ i\omega\mu \left[\hat{\theta}\hat{\theta} + \hat{\phi}\hat{\phi}\right] \cdot \left[\hat{z}' \times \left(H_\rho(\rho', \phi', z_1)\hat{\rho}' + H_\phi(\rho', \phi', z_1)\hat{\phi}' + H_z(\rho', \phi', z_1)\hat{z}'\right)\right] \right. \\ &\quad \left. ik \left[\hat{\phi}\hat{\theta} - \hat{\theta}\hat{\phi}\right] \cdot \left[\hat{z}' \times \left(E_\rho(\rho', \phi', z_1)\hat{\rho}' + E_\phi(\rho', \phi', z_1)\hat{\phi}' + E_z(\rho', \phi', z_1)\hat{z}'\right)\right] \right\} \end{aligned} \tag{A.11}$$

where

$$\overline{r}' = \hat{x}x' + \hat{y}y' + \hat{z}z' = \hat{\rho}'\rho' + \hat{z}'z' \tag{A.12}$$

$$\hat{\rho}' = \hat{x}\cos\phi' + \hat{y}\sin\phi' \tag{A.13}$$

$$\hat{z}' = \hat{z} \tag{A.14}$$

Expanding and simplifying,

$$\begin{aligned}
\vec{E}(\theta,\phi) &= ik\frac{e^{ikr}}{4\pi r}\int_{z_1}^{z_2} dz' \int_0^{2\pi} \rho_0 d\phi' e^{-ik\rho_0 \sin\theta\cos(\phi'-\phi)} e^{-ikz\cos\theta} \\
&\quad \Big\{\hat{\theta}\,[-\sin\theta\ \eta H_\phi(\rho_0,\phi',z') + \cos\theta\sin(\phi'-\phi)\ \eta H_z(\rho_0,\phi',z') \\
&\qquad + \cos(\phi'-\phi)E_z(\rho_0,\phi',z')] \\
&\quad +\hat{\phi}\,[-\cos(\phi'-\phi)\ \eta H_z(\rho_0,\phi',z') - \sin\theta E_\phi(\rho_0,\phi',z')] \\
&\qquad + \cos\theta\sin(\phi'-\phi)E_z(\rho_0,\phi',z')]\Big\} \\
&+ ik\frac{e^{ikr}}{4\pi r}\int_0^{\rho_0} d\rho' \int_0^{2\pi} \rho' d\phi' e^{-ik\rho'\sin\theta\cos(\phi'-\phi)} e^{-ikz_2\cos\theta} \\
&\quad \Big\{\hat{\theta}\,[-\cos\theta\sin(\phi'-\phi')\ \eta H_\rho(\rho',\phi',z_2) - \cos\theta\cos(\phi'-\phi)\ \eta H_\phi(\rho',\phi',z_2) \\
&\qquad - \cos(\phi'-\phi)E_\rho(\rho',\phi',z_2) + \sin(\phi'-\phi)E_\phi(\rho',\phi',z_2)] \\
&\quad +\hat{\phi}\,[-\cos(\phi'-\phi)\ \eta H_\rho(\rho',\phi',z_2) - \sin(\phi'-\phi)\ \eta H_\phi(\rho',\phi',z_2) \\
&\qquad - \cos\theta\sin(\phi'-\phi)E_\rho(\rho',\phi',z_2) - \cos\theta\cos(\phi'-\phi)E_\phi(\rho',\phi',z_2)]\Big\} \\
&- ik\frac{e^{ikr}}{4\pi r}\int_0^{\rho_0} d\rho' \int_0^{2\pi} \rho' d\phi' e^{-ik\rho'\sin\theta\cos(\phi'-\phi)} e^{-ikz_1\cos\theta} \\
&\quad \Big\{\hat{\theta}\,[-\cos\theta\sin(\phi'-\phi')\ \eta H_\rho(\rho',\phi',z_1) - \cos\theta\cos(\phi'-\phi)\ \eta H_\phi(\rho',\phi',z_1) \\
&\qquad - \cos(\phi'-\phi)E_\rho(\rho',\phi',z_1) + \sin(\phi'-\phi)E_\phi(\rho',\phi',z_1)] \\
&\quad +\hat{\phi}\,[-\cos(\phi'-\phi)\ \eta H_\rho(\rho',\phi',z_1) - \sin(\phi'-\phi)\ \eta H_\phi(\rho',\phi',z_1) \\
&\qquad - \cos\theta\sin(\phi'-\phi)E_\rho(\rho',\phi',z_1) - \cos\theta\cos(\phi'-\phi)E_\phi(\rho',\phi',z_1)]\Big\}
\end{aligned} \tag{A.15}$$

Expressing the electric and magnetic fields as a Fourier series,

$$A(\rho',\phi',z') = \sum_{m=0}^{M} A_{m,u}(\rho',z')\cos(m\phi') + A_{m,v}(\rho',z')\sin(m\phi') \tag{A.16}$$

where A stands for the electric and magnetic field components. Next rewrite it as,

$$A(\rho',\phi',z') = \sum_{m=0}^{M} A_{m,u}(\rho',z')\cos\left[m(\phi'-\phi)+m\phi\right] +$$

$$+ \sum_{m=0}^{m} A_{m,v}(\rho', z') \sin\left[m(\phi' - \phi) + m\phi'\right] \tag{A.17}$$

and define,

$$A'_{m,u}(\rho', z') = A_{m,u}(\rho', z') \cos m\phi + A_{m,v}(\rho', z') \sin m\phi \tag{A.18}$$

$$A'_{m,v}(\rho', z') = A_{m,v}(\rho', z') \cos m\phi - A_{m,u}(\rho', z') \sin m\phi \tag{A.19}$$

so that,

$$A(\rho', \phi', z') = \sum_{m=0}^{M} A'_{m,u}(\rho', z') \cos\left[m(\phi' - \phi)\right] + A'_{m,v}(\rho', z') \sin\left[m(\phi' - \phi)\right]. \tag{A.20}$$

Next, (A.15) is rewritten using (A.20) with the following change of variables,

$$\phi'_{\text{new}} = \phi'_{\text{old}} - \phi \tag{A.21}$$

Because the integrand is periodic, the limits of integration can remain unchanged, and (A.15) becomes,

$$\begin{aligned}
\vec{E}(\theta, \phi) = \; & ik \frac{e^{ikr}}{4\pi r} \sum_{m=0}^{M} \int_{z_1}^{z_2} dz' \int_0^{2\pi} \rho_0 d\phi' e^{-ik\rho_0 \sin\theta \cos\phi'} e^{-ikz\cos\theta} \\
& \Big\{ \hat{\theta} \cos m\phi' \Big[-\sin\theta \; \eta H'_{\phi_{m,u}}(\rho_0, z') + \cos\theta \sin\phi' \; \eta H'_{z_{m,u}}(\rho_0, z') \\
& \qquad + \cos\phi' E'_{z_{m,u}}(\rho_0, z') \Big] \\
& + \hat{\theta} \sin m\phi' \Big[-\sin\theta \; \eta H'_{\phi_{m,v}}(\rho_0, z') + \cos\theta \sin\phi' \; \eta H'_{z_{m,v}}(\rho_0, z') \\
& \qquad + \cos\phi' E'_{z_{m,v}}(\rho_0, z') \Big] \\
& + \hat{\phi} \cos m\phi' \Big[-\cos\phi' \; \eta H'_{z_{m,u}}(\rho_0, z') - \sin\theta \; E'_{\phi_{m,u}}(\rho_0, z') \\
& \qquad + \cos\theta \sin\phi' E'_{z_{m,u}}(\rho_0, z') \Big] \\
& + \hat{\phi} \sin m\phi' \Big[-\cos\phi' \; \eta H'_{z_{m,v}}(\rho_0, z') - \sin\theta \; E'_{\phi_{m,v}}(\rho_0, z') \\
& \qquad + \cos\theta \sin\phi' E'_{z_{m,v}}(\rho_0, z') \Big] \Big\} \\
+ \; & ik \frac{e^{ikr}}{4\pi r} \sum_{m=0}^{M} \int_0^{\rho_0} d\rho' \int_0^{2\pi} \rho' d\phi' e^{-ik\rho' \sin\theta \cos\phi'} e^{-ikz_2 \cos\theta} \\
& \Big\{ \hat{\theta} \cos m\phi' \Big[-\cos\theta \sin\phi' \; \eta H_{\rho_{m,u}}(\rho', z_2) - \cos\theta \cos\phi' \; \eta H'_{\phi_{m,u}}(\rho', z_2)
\end{aligned}$$

$$
\begin{aligned}
&\quad - \cos\phi' E'_{\rho_{m,u}}(\rho', z_2) + \sin\phi' E'_{\phi_{m,u}}(\rho', z_2)\Big] \\
&+ \hat{\theta}\sin m\phi' \Big[-\cos\theta\sin\phi'\ \eta H_{\rho_{m,v}}(\rho', z_2) - \cos\theta\cos\phi'\ \eta H'_{\phi_{m,v}}(\rho', z_2) \\
&\quad - \cos\phi' E'_{\rho_{m,v}}(\rho', z_2) + \sin\phi' E'_{\phi_{m,v}}(\rho', z_2)\Big] \\
&+ \hat{\phi}\cos m\phi' \Big[\cos\phi'\ \eta H_{\rho_{m,u}}(\rho', z_2) - \sin\phi'\ \eta H'_{\phi_{m,u}}(\rho', z_2) \\
&\quad - \cos\theta\sin\phi' E'_{\rho_{m,u}}(\rho', z_2) - \cos\theta\sin\phi' E'_{\phi_{m,u}}(\rho', z_2)\Big] \\
&+ \hat{\phi}\sin m\phi' \Big[\cos\phi'\ \eta H_{\rho_{m,v}}(\rho', z_2) - \sin\phi'\ \eta H'_{\phi_{m,v}}(\rho', z_2) \\
&\quad - \cos\theta\sin\phi' E'_{\rho_{m,v}}(\rho', z_2) - \cos\theta\sin\phi' E'_{\phi_{m,v}}(\rho', z_2)\Big]\Big\} \\
&- ik\frac{e^{ikr}}{4\pi r}\sum_{m=0}^{M}\int_0^{\rho_0} d\rho' \int_0^{2\pi} \rho' d\phi' e^{-ik\rho'\sin\theta\cos\phi'} e^{-ikz_1\cos\theta} \\
&\Big\{\hat{\theta}\cos m\phi' \Big[-\cos\theta\sin\phi'\ \eta H_{\rho_{m,u}}(\rho', z_1) - \cos\theta\cos\phi'\ \eta H'_{\phi_{m,u}}(\rho', z_1) \\
&\quad - \cos\phi' E'_{\rho_{m,u}}(\rho', z_1) + \sin\phi' E'_{\phi_{m,u}}(\rho', z_1)\Big] \\
&+ \hat{\theta}\sin m\phi' \Big[-\cos\theta\sin\phi'\ \eta H_{\rho_{m,v}}(\rho', z_1) - \cos\theta\cos\phi'\ \eta H'_{\phi_{m,v}}(\rho', z_1) \\
&\quad - \cos\phi' E'_{\rho_{m,v}}(\rho', z_1) + \sin\phi' E'_{\phi_{m,v}}(\rho', z_1)\Big] \\
&+ \hat{\phi}\cos m\phi' \Big[\cos\phi'\ \eta H_{\rho_{m,u}}(\rho', z_1) - \sin\phi'\ \eta H'_{\phi_{m,u}}(\rho', z_1) \\
&\quad - \cos\theta\sin\phi' E'_{\rho_{m,u}}(\rho', z_1) - \cos\theta\sin\phi' E'_{\phi_{m,u}}(\rho', z_1)\Big] \\
&+ \hat{\phi}\sin m\phi' \Big[\cos\phi'\ \eta H_{\rho_{m,v}}(\rho', z_1) - \sin\phi'\ \eta H'_{\phi_{m,v}}(\rho', z_1) \\
&\quad - \cos\theta\sin\phi' E'_{\rho_{m,v}}(\rho', z_1) - \cos\theta\sin\phi' E'_{\phi_{m,v}}(\rho', z_1)\Big]\Big\}
\end{aligned}
\tag{A.22}
$$

The ϕ integrals are then analytically evaluated using Bessel function relations yielding,

$$
\begin{aligned}
\vec{E}(\theta,\phi) \;=\; & ik\frac{e^{ikr}}{2\pi r}\sum_{m=0}^{M}\int_{z_1}^{z_2} dz' \rho_0 e^{-ikz\cos\theta} \\
& \Big\{\hat{\theta}\Big[-\sin\theta\ \eta H'_{\phi_{m,u}}(\rho_0, z')F_1(\rho_0) + E'_{z_{m,u}}(\rho_0, z')F_3(\rho_0) \\
& \quad + \cos\theta\ \eta H'_{z_{m,v}}(\rho_0, z')F_5(\rho_0)\Big] \\
& + \hat{\phi}\Big[-\eta H'_{z_{m,u}}(\rho_0, z')F_3(\rho_0) - \sin\theta E'_{\phi_{m,u}}(\rho_0, z')F_1(\rho_0) \\
& \quad + \cos\theta E'_{z_{m,v}}(\rho_0, z')F_5(\rho_0)\Big]\Big\} \\
& + ik\frac{e^{ikr}}{2\pi r}\sum_{m=0}^{M}\int_0^{\rho_0} d\rho' \rho' e^{-ikz_2\cos\theta}
\end{aligned}
$$

$$
\begin{aligned}
&\Big\{\hat{\theta}\Big[\Big(-\cos\theta\ \eta H'_{\phi_{m,u}}(\rho',z_2) - E'_{\rho_{m,u}}(\rho',z_2)\Big)\,F_3(\rho') \\
&\qquad + \Big(-\cos\theta\ \eta H'_{\rho_{m,v}}(\rho',z_2) + E'_{\phi_{m,v}}(\rho',z_2)\Big)\,F_5(\rho')\Big] \\
&+ \hat{\phi}\Big[\Big(\eta H'_{\rho_{m,u}}(\rho',z_2) - \cos\theta E'_{\phi_{m,u}}(\rho',z_2)\Big)\,F_3(\rho') \\
&\qquad + \Big(-\eta H'_{\phi_{m,v}}(\rho',z_2) - \cos\theta E_{\rho_{m,v}}(\rho',z_2)\Big)\,F_5(\rho')\Big]\Big\}
\end{aligned}
$$

$$
\begin{aligned}
- \quad & ik\frac{e^{ikr}}{2\pi r}\sum_{m=0}^{M}\int_0^{\rho_0} d\rho'\rho' e^{-ikz_1\cos\theta} \\
&\Big\{\hat{\theta}\Big[\Big(-\cos\theta\ \eta H'_{\phi_{m,u}}(\rho',z_1) - E'_{\rho_{m,u}}(\rho',z_1)\Big)\,F_3(\rho') \\
&\qquad + \Big(-\cos\theta\ \eta H'_{\rho_{m,v}}(\rho',z_1) + E'_{\phi_{m,v}}(\rho',z_1)\Big)\,F_5(\rho')\Big] \\
&+ \hat{\phi}\Big[\Big(\eta H'_{\rho_{m,u}}(\rho',z_1) - \cos\theta E'_{\phi_{m,u}}(\rho',z_1)\Big)\,F_3(\rho') \\
&\qquad + \Big(-\eta H'_{\phi_{m,v}}(\rho',z_1) - \cos\theta E_{\rho_{m,v}}(\rho',z_1)\Big)\,F_5(\rho')\Big]\Big\}
\end{aligned}
\tag{A.23}
$$

where

$$
F_1(\rho) = 2\pi e^{im\frac{3\pi}{2}} J_m(k\rho\sin\theta) \tag{A.24}
$$

$$
F_3(\rho) = 2\pi e^{i(m+1)\frac{3\pi}{2}} J_{m+1}(k\rho\sin\theta) + \frac{2\pi im}{k\rho\sin\theta} e^{im\frac{3\pi}{2}} J_m(k\rho\sin\theta) \tag{A.25}
$$

$$
F_5(\rho) = \frac{2\pi im}{k\rho\sin\theta} e^{im\frac{3\pi}{2}} J_m(k\rho\sin\theta). \tag{A.26}
$$

Appendix B

Calculation of Far-Field Scattered Fields for 2D Geometries

B.1 General Two-Dimensional Formulation

For two-dimensional Huygens' surfaces which extend to infinity in the $\hat{y}$ direction, equation (A.7) can be simplified using the following identity.

$$\frac{1}{i\pi}\int_{-\infty}^{\infty} dy' \frac{e^{ik|\overline{r}-\overline{r}'|}}{|\overline{r}-\overline{r}'|} = H_0^{(1)}(k|\overline{\rho}-\overline{\rho}'|) \tag{B.1}$$

$$\overline{\rho} = \hat{x}x + \hat{z}z = \hat{\rho}\rho \tag{B.2}$$

$$\overline{\rho}' = \hat{x}x' + \hat{z}z' = \hat{\rho}\rho' \tag{B.3}$$

$$\hat{\rho} = \hat{z}\cos\alpha + \hat{x}\sin\alpha \tag{B.4}$$

$$\hat{\alpha} = -\hat{z}\sin\alpha + \hat{x}\cos\alpha \tag{B.5}$$

$$\begin{aligned}\vec{E}(\overline{\rho}) = \oint_{C'} dC' \left\{ i\omega\mu_0 \left[\hat{\alpha}\hat{\alpha} + \hat{y}\hat{y}\right] \cdot \left[\hat{n}\times\vec{H}(\overline{\rho}')\right] + ik\left[-\hat{y}\hat{\alpha} + \hat{\alpha}\hat{y}\right] \cdot \left[\hat{n}\times\vec{E}(\overline{\rho}')\right] \right\} \\ \cdot \frac{i}{4} H_0^{(1)}(k|\overline{\rho}-\overline{\rho}'|)\end{aligned} \tag{B.6}$$

Under the far-field approximation,

$$|\overline{\rho} - \overline{\rho}'| = \rho - \hat{\rho} \cdot \overline{\rho}' = \rho - z' \cos\alpha - x' \sin\alpha \tag{B.7}$$

the asymptotic form of the Hankel function can be used.

$$H_0^{(1)}(|\overline{\rho} - \overline{\rho}'|) = \sqrt{\frac{2}{\pi\rho}} e^{i(\rho - z'\cos\alpha - x'\sin\alpha - \pi/4)} \tag{B.8}$$

$$\begin{aligned} \vec{E}(\alpha) \;=\; & e^{ik\rho}\sqrt{\frac{k}{-i8\pi\rho}} \oint_{C'} e^{-ik(x'\sin\alpha + z'\cos\alpha)} \Big\{ [\hat{\alpha}\hat{\alpha} + \hat{y}\hat{y}] \cdot \Big[\hat{n} \times \eta\vec{H}(\overline{\rho}')\Big] \\ & + [-\hat{y}\hat{\alpha} + \hat{\alpha}\hat{y}] \cdot \Big[\hat{n} \times \vec{E}(\overline{\rho}')\Big] \Big\} \end{aligned} \tag{B.9}$$

B.2 2D TE Formulation

In two dimensional, the nonzero field components for the TE mode are the E_y, H_x, and H_z fields. If the Huygens' surface is a box which has a center at the origin, and lower-left hand and upper-right hand corners at $(-x_0, -z_0)$ and (x_0, z_0), then the far-field electric field can be computed as follows,

$$\begin{aligned}\vec{E}(\alpha) \;=\; \hat{y}\, e^{ik\rho}\sqrt{\frac{k}{-i8\pi\rho}}\Big\{ & e^{+ikx_0\sin\alpha}\int_{-z_0}^{z_0} dz'\,[\eta H_z(-x_0,z') - \sin\alpha E_y(-x_0,z')]\, e^{-ikz'\cos\alpha} \\ & + e^{-ikz_0\cos\alpha}\int_{-x_0}^{x_0} dx'\,[\eta H_x(x',z_0) + \cos\alpha E_y(x',z_0)]\, e^{-ikx'\sin\alpha} \\ & + e^{-ikx_0\sin\alpha}\int_{-z_0}^{z_0} dz'\,[-\eta H_z(x_0,z') + \sin\alpha E_y(x_0,z')]\, e^{-ikz'\cos\alpha} \\ & + e^{+ikz_0\cos\alpha}\int_{-x_0}^{x_0} dx'\,[-\eta H_x(x',-z_0) - \cos\alpha E_y(x',-z_0)]\, e^{-ikx'\sin\alpha}\Big\}\end{aligned} \tag{B.10}$$

If an infinite ground plane exists at the yz plane, image theory can be used to account for the presence of the perfectly conducting half space. An equivalent problem is created by removing the ground plane, and adding in the image fields. Image fields are created such that the boundary condition, $\hat{x} \times \vec{E} = 0$ is satisifed. In this case, the electric and magnetic fields of the equivalent source will be,

$$E'_y(x,z) \;=\; \begin{cases} +E_y(x,z) & x \geq 0 \\ -E_y(-x,z) & x < 0 \end{cases} \tag{B.11}$$

$$H'_x(x,z) \;=\; \begin{cases} +H_x(x,z) & x \geq 0 \\ -H_x(-x,z) & x < 0 \end{cases} \tag{B.12}$$

$$H'_z(x,z) \;=\; \begin{cases} +H_z(x,z) & x \geq 0 \\ +H_z(-x,z) & x < 0 \end{cases} \tag{B.13}$$

The far-field electric field can the be calculated using equation (B.10) with the electric and magnetic fields replaced by (B.12)–(B.13).

B.3 2D TM Formulation

The formulation for the two dimensional TM mode case is very similar. In this case, the nonzero field components for the TM mode are the H_y, E_x, and E_z fields. If the Huygens' surface is a box which has a center at the origin, and lower-left hand and upper-right hand corners at $(-x_0, -z_0)$ and (x_0, z_0), then the far-field electric field can be computed as follows,

$$\begin{aligned}\vec{E}(\alpha) \;=\; \hat{\alpha}\, e^{ik\rho}\sqrt{\frac{k}{-i8\pi\rho}}\Bigg\{ & e^{+ikx_0\sin\alpha}\int_{-z_0}^{z_0} dz'\,\left[H_z(-x_0,z') + \sin\alpha\;\eta H_y(-x_0,z')\right] e^{-ikz'\cos\alpha} \\ & + e^{-ikz_0\cos\alpha}\int_{-x_0}^{x_0} dx'\,\left[E_x(x',z_0) - \cos\alpha\;\eta H_y(x',z_0)\right] e^{-ikx'\sin\alpha} \\ & + e^{-ikx_0\sin\alpha}\int_{-z_0}^{z_0} dz'\,\left[-E_z(x_0,z') - \sin\alpha\;\eta H_y(x_0,z')\right] e^{-ikz'\cos\alpha} \\ & + e^{+ikz_0\cos\alpha}\int_{-x_0}^{x_0} dx'\,\left[-E_x(x',-z_0) + \cos\alpha\;\eta H_y(x',-z_0)\right] e^{-ikx'\sin\alpha}\Bigg\}\end{aligned} \tag{B.14}$$

Similary, if an infinite ground plane exists at the yz plane, image theory can be used to account for the presence of the perfectly conducting half space. The the electric and magnetic fields of the equivalent source will be,

$$H'_y(x,z) \;=\; \begin{cases} +H_y(x,z) & x \geq 0 \\ +H_y(-x,z) & x < 0 \end{cases} \tag{B.15}$$

$$E'_x(x,z) \;=\; \begin{cases} +E_x(x,z) & x \geq 0 \\ +E_x(-x,z) & x < 0 \end{cases} \tag{B.16}$$

$$E'_z(x,z) \;=\; \begin{cases} +E_z(x,z) & x \geq 0 \\ -E_z(-x,z) & x < 0 \end{cases} \tag{B.17}$$

The far-field electric field can the be calculated using equation (B.14) with the electric and magnetic fields replaced by (B.16)–(B.17).

Appendix C

The Two-Dimensional FD-TD Method

The formulation of the 2D FD-TD method is similar to that of the BOR FD-TD formulation, and will not be carried out in detail here as there are many references such as [50] that contain the formulation. As with the BOR FD-TD technique, the 2D FD-TD numerically solves Maxwell's equations, which are presented here in their differential form,

$$\epsilon_0 \frac{\partial}{\partial t}\vec{E} = \nabla \times \vec{H} \tag{C.1}$$

$$\mu_0 \frac{\partial}{\partial t}\vec{H} = -\nabla \times \vec{E}. \tag{C.2}$$

For a two dimensional problem, which is uniform in the $\hat{y}$ direction, Maxwell's equations can be decomposed into two independent sets of scalar equations. The equations for the TM mode are,

$$\epsilon_0 \frac{\partial E_x}{\partial t} = -\frac{\partial H_y}{\partial z} \tag{C.3}$$

$$\epsilon_0 \frac{\partial E_z}{\partial t} = \frac{\partial H_y}{\partial x} \tag{C.4}$$

$$\mu_0 \frac{\partial H_y}{\partial t} = \frac{\partial E_z}{\partial x} - \frac{\partial E_x}{\partial z}. \tag{C.5}$$

The equations for the TE mode are,

$$\mu_0 \frac{\partial H_x}{\partial t} = \frac{\partial E_y}{\partial z} \tag{C.6}$$

$$\mu_0 \frac{\partial H_z}{\partial t} = -\frac{\partial E_y}{\partial x} \tag{C.7}$$

$$\epsilon_0 \frac{\partial E_y}{\partial t} = \frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x}. \tag{C.8}$$

Difference equations can then be obtained by discretizing the time and space derivatives using central difference approximations. For reference the difference equations for free space [50] as well as the difference equations for the PML medium [3] are listed in the following sections.

C.1 TM Mode

C.1.1 TM Mode FD-TD Difference Equations

$$H_x|^{n+1/2}_{i+1/2,j} = H_x|^{n-1/2}_{i+1/2,j} - \frac{\Delta t}{\mu \Delta y}\left(E_z|^{n}_{i+1/2,j+1/2} - E_z|^{n}_{i+1/2,j-1/2}\right) \tag{C.9}$$

$$H_y|^{n+1/2}_{i,j+1/2} = H_y|^{n-1/2}_{i,j+1/2} + \frac{\Delta t}{\mu \Delta x}\left(E_z|^{n}_{i+1/2,j+1/2} - E_z|^{n}_{i-1/2,j+1/2}\right) \tag{C.10}$$

$$\begin{aligned} E_z|^{n+1}_{i+1/2,j+1/2} &= E_z|^{n}_{i+1/2,j+1/2} + \frac{\Delta t}{\epsilon \Delta x}\left(H_y|^{n+1/2}_{i+1,j+1/2} - H_y|^{n+1/2}_{i,j+1/2}\right) \\ &- \frac{\Delta t}{\epsilon \Delta y}\left(H_x|^{n+1/2}_{i+1/2,j+1} - H_x|^{n+1/2}_{i+1/2,j}\right) \end{aligned} \tag{C.11}$$

C.1.2 TM Mode PML Difference Equations

$$\begin{aligned} H_x|^{n+1/2}_{i+1/2,j} &= e^{-(\sigma_y^*/\mu)\Delta t} H_x|^{n-1/2}_{i+1/2,j} - \frac{1 - e^{-(\sigma_y^*/\mu)\Delta t}}{\sigma_y^* \Delta y} \times \\ &\left(E_{zx}|^{n}_{i+1/2,j+1/2} - E_{zx}|^{n}_{i+1/2,j-1/2}\right. \\ &+ \left. E_{zy}|^{n}_{i+1/2,j+1/2} - E_{zy}|^{n}_{i+1/2,j-1/2}\right) \end{aligned} \tag{C.12}$$

$$\begin{aligned} H_y|^{n+1/2}_{i,j+1/2} &= e^{-(\sigma_x^*/\mu)\Delta t} H_y|^{n-1/2}_{i,j+1/2} + \frac{1 - e^{-(\sigma_x^*/\mu)\Delta t}}{\sigma_x^* \Delta x} \times \\ &\left(E_{zx}|^{n}_{i+1/2,j+1/2} - E_{zx}|^{n}_{i-1/2,j+1/2}\right. \\ &+ \left. E_{zy}|^{n}_{i+1/2,j+1/2} - E_{zy}|^{n}_{i-1/2,j+1/2}\right) \end{aligned} \tag{C.13}$$

$$\begin{aligned} E_{zx}|^{n+1}_{i+1/2,j+1/2} &= e^{-(\sigma_x/\epsilon)\Delta t} E_{zx}|^{n}_{i+1/2,j+1/2} + \frac{1 - e^{-(\sigma_x/\epsilon)\Delta t}}{\sigma_x \Delta x} \times \\ &\left(H_y|^{n+1/2}_{i+1,j+1/2} - H_y|^{n+1/2}_{i,j+1/2}\right) \end{aligned} \tag{C.14}$$

$$\begin{aligned} E_{zy}|^{n+1}_{i+1/2,j+1/2} &= e^{-(\sigma_y/\epsilon)\Delta t} E_{zy}|^{n}_{i+1/2,j+1/2} - \frac{1 - e^{-(\sigma_y/\epsilon)\Delta t}}{\sigma_y \Delta y} \times \\ &\left(H_x|^{n+1/2}_{i+1/2,j+1} - H_x|^{n+1/2}_{i+1/2,j}\right) \end{aligned} \tag{C.15}$$

C.2 TE Mode

C.2.1 TE Mode FD-TD Difference Equations

$$E_x|^{n+1/2}_{i+1/2,j} = E_x|^{n-1/2}_{i+1/2,j} + \frac{\Delta t}{\epsilon \Delta y}\left(H_z|^{n}_{i+1/2,j+1/2} - H_z|^{n}_{i+1/2,j-1/2}\right) \tag{C.16}$$

$$E_y|^{n+1/2}_{i,j+1/2} = E_y|^{n-1/2}_{i,j+1/2} - \frac{\Delta t}{\epsilon \Delta x}\left(H_z|^{n}_{i+1/2,j+1/2} - H_z|^{n}_{i-1/2,j+1/2}\right) \tag{C.17}$$

$$\begin{aligned} H_z|^{n+1}_{i+1/2,j+1/2} &= H_z|^{n}_{i+1/2,j+1/2} - \frac{\Delta t}{\mu \Delta x}\left(E_y|^{n+1/2}_{i+1,j+1/2} - E_y|^{n+1/2}_{i,j+1/2}\right) \\ &+ \frac{\Delta t}{\mu \Delta y}\left(E_x|^{n+1/2}_{i+1/2,j+1} - E_x|^{n+1/2}_{i+1/2,j}\right) \end{aligned} \tag{C.18}$$

C.2.2 TM Mode PML Difference Equations

$$\begin{aligned} E_x|^{n+1/2}_{i+1/2,j} &= e^{-(\sigma_y/\epsilon)\Delta t} E_x|^{n-1/2}_{i+1/2,j} + \frac{1 - e^{-(\sigma_y/\epsilon)\Delta t}}{\sigma_y \Delta y} \times \\ &\left(H_{zx}|^{n}_{i+1/2,j+1/2} - H_{zx}|^{n}_{i+1/2,j-1/2}\right. \\ &+ \left. H_{zy}|^{n}_{i+1/2,j+1/2} - H_{zy}|^{n}_{i+1/2,j-1/2}\right) \end{aligned} \tag{C.19}$$

$$\begin{aligned} E_y|^{n+1/2}_{i,j+1/2} &= e^{-(\sigma_x/\epsilon)\Delta t} E_y|^{n-1/2}_{i,j+1/2} - \frac{1 - e^{-(\sigma_x/\epsilon)\Delta t}}{\sigma_x \Delta x} \times \\ &\left(H_{zx}|^{n}_{i+1/2,j+1/2} - H_{zx}|^{n}_{i-1/2,j+1/2}\right. \\ &+ \left. H_{zy}|^{n}_{i+1/2,j+1/2} - H_{zy}|^{n}_{i-1/2,j+1/2}\right) \end{aligned} \tag{C.20}$$

$$\begin{aligned} H_{zx}|^{n+1}_{i+1/2,j+1/2} &= e^{-(\sigma^*_x/\mu)\Delta t} H_{zx}|^{n}_{i+1/2,j+1/2} - \frac{1 - e^{-(\sigma^*_x/\mu)\Delta t}}{\sigma^*_x \Delta x} \times \\ &\left(E_y|^{n+1/2}_{i+1,j+1/2} - E_y|^{n+1/2}_{i,j+1/2}\right) \end{aligned} \tag{C.21}$$

$$\begin{aligned} H_{zy}|^{n+1}_{i+1/2,j+1/2} &= e^{-(\sigma^*_y/\mu)\Delta t} H_{zy}|^{n}_{i+1/2,j+1/2} + \frac{1 - e^{-(\sigma^*_y/\mu)\Delta t}}{\sigma^*_y \Delta y} \times \\ &\left(E_x|^{n+1/2}_{i+1/2,j+1} - E_x|^{n+1/2}_{i+1/2,j}\right) \end{aligned} \tag{C.22}$$

Appendix D

Source Code

D.1 BOR FD-TD Program

The BOR FD-TD program calculates monostatic or bistatic radar cross sections of a PEC body of revolution with arbitrary cross section. Bistatic signatures are calculated exactly, while monostatic signatures are estimated using the monostatic bistatic equivalence principle. Both monostatic and bistatic signatures can be calculated over an extended bandwidth. The user can specify whether the object should be represented using a staircase approximation or a conformal grid representation.

The following, **bor_fdtd.f**, contains the subroutines for reading in the input parameters from the user, as well as the core FD-TD equations used for time stepping.

```
************************************************************************
* BOR-FDTD CODE:                                                       *
*  This programs calculates the scattering pattern of a                *
*        incident plane wave on a body of revolution.  The user        *
*        may input the two dimensional shape of the BOR                *
*                                                                      *
************************************************************************

      program bor_fdtd

      implicit none
      include 'common.f'
      integer menu_choice

      dbase = 'data'

 10   write(6,*)
      write(6,*) 'BOR FDTD Options'
      write(6,*) '1 = FDTD,WRITE FREQ,RCS'
      write(6,*) '2 = FDTD,RCS'
      write(6,*) '3 = READ FREQ,RCS'
      write(6,*) '4 = FDTD'
      write(6, '(''*Enter option: '', $)')
      read(5,*) menu_choice
      if (menu_choice.lt.1.OR.menu_choice.gt.4) goto 10

      if (menu_choice.eq.1) then
         call get_rcs_out_ranges(.FALSE.)
         call get_primary_input
         call init_fields
         call init_freq
         call fdtd_loop(.TRUE.)
         call write_phasors
         call calc_rcs
      else if (menu_choice.eq.2) then
         call get_rcs_out_ranges(.FALSE.)
         call get_primary_input
         call init_fields
         call init_freq
         call fdtd_loop(.TRUE.)
         call calc_rcs
      else if (menu_choice.eq.3) then
         call read_phasors
         call get_rcs_out_ranges(.TRUE.)
         call calc_rcs
      else if (menu_choice.eq.4) then
         calc_bist = .TRUE.
         call get_primary_input
         call init_fields
         call fdtd_loop(.FALSE.)
      end if

      end

c***********************************************************************
c GET_PRIMARY_INPUT gets info from user about geomfile name, incident
c    wave, duration of simulation, out file names, etc
c***********************************************************************

      SUBROUTINE get_primary_input

      implicit none
      include 'common.f'

      integer conf_stair, totsteps, movie_test, mode_index, round,
     1     x1,x2,y1,y2, polarization
      real dt_out, width, TIME_TO_DELAY, cost, sint
      real theta_1,theta_2,theta_3,theta_4

C**** get Geometry and data filename
      write(6,'(''*Enter geometry file name: '',$)')
      read(5,*) fnamein

      write(6,'(''*Store for movie? (1=Y,2=N): '',$)')
      read(5,*) movie_test
      store_movie = movie_test.eq.1
      if (store_movie) then
         write(6,'(''*Movie header name: '',$)')
         read(5,*) mhname
         write(6,'(''*Movie file name: '',$)')
         read(5,*) mfname
         write(6,'(''*Number of time steps between each frame: '',$)')
         read(5,*) movie_step
         write(6,*) 'Field ids: er=1,ez=2,ephi=3,hr=4,hz=5,hphi=6'
         write(6,'(''*Enter id of field to store: '',$)')
         read(5,*) movie_num
         movie_type = 1
      end if

c**** Note: There is alternative staircasing routine which can handle
c**** any type of geometry, including nonconvex objects. It can also
c**** scale the staircase nodes as needed based on any delta; that
c**** is, given a set of data points it will perform a linear
c**** interpolation to determine all points in between before
c**** staircasing the resulting object. This is option 3.

 40   write(6,'(''*Enter model (1=staircase, 2=conformal): '',$)')
      read(5,*) conf_stair
      if (conf_stair.gt.3.OR.conf_stair.lt.1) goto 40
      use_conformal = (conf_stair.eq.2)
      use_stair2 = (conf_stair.eq.3)

      if (use_conformal) then
         call geometry
      else
         if (conf_stair.eq.3) then
            write(6,*) 'Warning: Using second staircase method...'
            call setup_staircase
         else
            call geometry
         end if
      end if
      call setup_scat

c**** Calculate sigma_max so that reflections are 40 dB down
      sigma_max = 70*3/eta/40./0.434294481903/(PMLDEPTH*dz)
c      write(6,*) 'sigma_max = ',sigma_max
c      write(6,*) 'Enter sigma max'
c      read(5,*) sigma_max

      if (calc_bist) then
         write(6,'(''*Enter incident angle theta in degrees: '',$)')
         read(5,*) inc_ang
      end if

 33   write(6,'(''*Select polarization (1=HORZ,2=VERT): '',$)')
      read(5,*) polarization
      if (polarization.eq.1) then
         Ehg = 1.0
         Evg = 0.0
      else if (polarization.eq.2) then
         Ehg = 0.0
         Evg = 1.0
      else
         goto 33
      end if

 32   write(6,'(''*Enter duration of simulation (ns): '',$)')
      read(5,*) sim_duration
      if (sim_duration.lt.0.5) then
         print *,'Simulation must last longer than 0.5 ns.'
         goto 32
      end if

C**** Modulation of Gaussian Pulse (1-on 0-off)
 50   write(6,'(''*Modulate incident wave? (1=Y,0=N): '',$)')
      read(5,*) modulate
      if (modulate.gt.1.OR.modulate.lt.0) goto 50
      if (modulate.eq.1) then
         write(6,'(''*Enter modulation frequency: '',$)')
         read(5,*) modfreq
      else
         modfreq = -1
      end if

C**** Convert incident angle to radians
      inc_ang=(inc_ang/180)*pi

      if (abs(inc_ang-pi).lt.tole.OR.abs(inc_ang).lt.tole) then
         mode_start = 1
         mode_end = 1
      else
         modes = int(obj_height*2*pi*high_freq/c+1)
         write(6,*) 'Estimated modes required: ',modes
         write(6,'(''*Enter start mode: '', $)')
         read(5,*) mode_start
         write(6,'(''*Enter end mode: '', $)')
         read(5,*) mode_end
      end if

      if (abs(Ehg-1).lt.tole) then
         eqset_start = 2
         eqset_end = 2
      else if (abs(Evg-1).lt.tole) then
         eqset_start = 1
         eqset_end =1
      else
```

```
         eqset_start = 1
         eqset_end = 2
      end if

C**** Standard Dev and Wave Delay Calculations
      if (modulate.eq.0) then
         sdev=5.0*dt_out(1)
      else
         sdev=(1.0/modfreq/4.0)
      end if
      width = sdev*sqrt(10.0)
c***  calculate time delay

 10   if (inc_ang.ge.(2*pi)) then
         inc_ang = inc_ang-2*pi
         goto 10
      end if

 20   if (inc_ang.lt.0) then
         inc_ang = inc_ang+2*pi
         goto 20
      end if
      cost = cos(inc_ang)
      sint = sin(inc_ang)

      print *, rcsz1, rcsz2, mheight
      x1 = rcsz1
      x2 = rcsz2
      y1 = 1
      y2 = mheight

c***** determine the time delay so that wave arrives at the target at
c***** around time step 100-150.

      TIME_TO_DELAY = 100*dt_out(mode_start)
      theta_1 = atan2(y2*1.0,(x2-x1)*1.0)
      theta_2 = atan2(y2*4.0,(x2-x1)*1.0)
      theta_3 = atan2(y2*4.0,(x1-x2)*1.0)
      theta_4 = atan2(y2*1.0,(x1-x2)*1.0)

      if (inc_ang.ge.0.AND.inc_ang.lt.theta_1) then
         gd = (x2*dz*cost+0*dz*sint)/c + TIME_TO_DELAY + 2*sdev
      elseif (inc_ang.ge.theta_1.AND.inc_ang.lt.theta_2) then
         gd = (x2*dz*cost+y2*dz*sint)/c + TIME_TO_DELAY + 2*sdev
      elseif (inc_ang.ge.theta_2.AND.inc_ang.lt.theta_3) then
         gd = ((x1+x2)/2*dz*cost+y2*dz*sint)/c + TIME_TO_DELAY + 2*sdev
      elseif (inc_ang.ge.theta_3.AND.inc_ang.lt.theta_4) then
         gd = (x1*dz*cost+y2*dz*sint)/c + TIME_TO_DELAY + 2*sdev
      else
         gd = (x1*dz*cost+0*dz*sint)/c + TIME_TO_DELAY + 2*sdev
      end if

      totsteps = 0
      do 80 mode_index = mode_start,mode_end
         dt = dt_out(mode_index)
         totsteps = totsteps + round(sim_duration*1e-9/dt)
 80   continue

      if (store_movie) call setup_movie(totsteps)

      RETURN
      END

c**************************************************************************
c GET_RCS_OUT_RANGES gets info from user about what angles and freqs
c    to calc the RCS for.
c**************************************************************************

      SUBROUTINE get_rcs_out_ranges(skip_fd)

      implicit none
      include 'common.f'

      integer nang, fi, fi2, mono_bi
      real mono_ang, ma, tempfreqlist(1:MAX_FREQS)
      logical skip_fd

      if (.NOT.skip_fd) then
         write(6,'(''*Enter lowest frequency of interest: '',$)')
         read(5,*) low_freq
         write(6,'(''*Enter highest frequency of interest: '',$)')
         read(5,*) high_freq

         if (abs(low_freq-high_freq).gt.tole) then

 10         write(6,'(''*Enter the number of frequencies: '',$)')
            read(5,*) num_freqs

            if (num_freqs.gt.MAX_FREQS) then
               write(6,*) 'Error. Number of freqs must be ',
     1              'less than ', MAX_FREQS, ' or raise ',
     2              'MAX_FREQS parmaeter'
               write(6,*)
               goto 10
            end if

            minf = 1
            maxf = num_freqs
            dfreq = (high_freq-low_freq)/(num_freqs-1.0)

            do 20 fi = minf, maxf
               freqlist(fi,1) = low_freq + dfreq*(fi-1.0)
c************* Define type as normal RCS freq
               freqlist(fi,2) = 0
               tempfreqlist(fi) = freqlist(fi,1)
 20         continue

            stepf = 1
         else
            freqlist(1,1) = low_freq
c********** Define type as normal RCS freq
            freqlist(1,2) = 0
            tempfreqlist(1) = freqlist(1,1)
            num_freqs = 1
            minf = 1
            maxf = 1
            stepf = 1
         end if
      end if

 100  write(6,*)
      write(6,*) '1. Calculate bistatic RCS vs angle for given freqs'
      write(6,*) '2. Estimate monostatic RCS vs angle for given freqs'
      write(6,'(''*Enter your choice: '',$)')
      read(5,*) mono_bi
      if (mono_bi.ne.1.AND.mono_bi.ne.2) then
         goto 100
      else
         calc_bist = mono_bi.eq.1
      end if

      if (calc_bist) then
         write(6,*) 'Bistatic RCS angles (in degrees)'
         write(6,'(''*Enter initial and final phi: '',$,$)')
         read(5,*) low_phi,high_phi

         if (abs(low_phi-high_phi).lt.tole) then
            dphi = high_phi-low_phi+1.0
         else
            write(6,'(''*Enter number of angles: '',$)')
            read(5,*) nang
            dphi = (high_phi-low_phi)/ real(nang-1.0)
         end if

         write(6,'(''*Enter initial and final theta: '',$,$)')
         read(5,*) low_theta,high_theta

         if (abs(low_theta-high_theta).lt.tole) then
            dtheta = high_theta-low_theta+1.0
         else
            write(6,'(''*Enter number of angles: '',$)')
            read(5,*) nang
            dtheta = (high_theta-low_theta)/ real(nang-1.0)
         end if
      else
         write(6,'(''*Enter incident angle theta in degrees: '',$)')
         read(5,*) inc_ang

         write(6,*) 'Monostatic RCS angles (in degrees)'
         write(6,'(''*Enter fixed phi angle: '',$)')
         read(5,*) low_phi
         high_phi = low_phi
         dphi = 1.0

         write(6,*) 'Note, monostatic angle range = inc_ang (+/-) ',
     1        'max_ang'
         write(6,'(''*Enter max angle: '',$,$)')
         read(5,*) mono_ang
         mono_ang = abs(mono_ang)
         low_theta = inc_ang-mono_ang
         high_theta = inc_ang+mono_ang

         if (abs(low_theta-high_theta).lt.tole) then
            dtheta = high_theta-low_theta+1.0
         else
            write(6,'(''*Enter number of angles (must be odd): '',$)')
            read(5,*) nang
            if (real(nang/2).eq.real(nang)/2.0) then
               write(6,*) 'Increasing nang to ', nang+1
               nang = nang+1
            end if
            dtheta = (high_theta-low_theta)/ real(nang-1.0)
         end if
         mono_nang = nang
```

```
c******* Determine freqs that need to be calculated.

c******* Total frequencies needed num_freqs*(nang+1)/2
        if (num_freqs*(nang+1)/2.gt.MAX_FREQS) then
           write(6,*) 'MAX_FREQS error'
           pause
        end if

        fi2 = 1
        do 30 fi=1,num_freqs
c********** Update freqlist components so that they are considered
c********** for use in monostatic calculations
           mono_freq_ind(fi) = fi2
           do 40 ma = 0,mono_ang,dtheta
              freqlist(fi2,1) = tempfreqlist(fi)*cos(ma/180*pi)
              freqlist(fi2,2) = 1
              fi2 = fi2+1
 40        continue
 30     continue
        minf = 1
        maxf = num_freqs*(nang+1)/2
      end if

c      do 50 fi = 1,num_freqs*(nang+1)/2
c         print *,fi, freqlist(fi,1),freqlist(fi,2)
c 50   continue
c      do 60 fi = 1,num_freqs
c         print *,mono_freq_ind(fi)
c 60   continue

      RETURN
      END


c*********************************************************************
c MEMORY_CHECK checks if enough memory has been allocated and reports
c   all errors stored in error buffer.
c*********************************************************************

      SUBROUTINE memory_check

      implicit none
      include 'common.f'

      integer i, id

      if ((2*mheight+rcsz2-rcsz1-1).gt.mxdp) then
         print *,'error not enough memory for RCS components'
         print *,'set the parameter mxdp higher than',
     1        2*mheight+rcsz2-rcsz1-1
         enough_memory = .FALSE.
      end if

      if (nm.gt.mode_start) then
         write(6,*)
         print *,'nm =',nm,' is greater than the starting mode'
         print *,'number', mode_start, '.  Adjust the nm parameter'
         enough_memory = .FALSE.
      end if

      if (mm.lt.mode_end) then
         write(6,*)
         print *,'mm =',mm,' is less than the ending mode'
         print *,'number', mode_end, '.  Adjust the mm parameter'
         enough_memory = .FALSE.
      end if

      if (errorcount.gt.0) then
         write(6,*) '*********************************************'
         write(6,*) 'Insufficient memory to begin simulation. The'
         write(6,*) 'following parameter(s) in the common.f file'
         write(6,*) 'need to be adjusted:'

         do 10 i=1,errorcount
            id = errors(i)
            write(6,*)
            if (id.eq.NODE_ERROR) then
               write(6,*) 'Set MAX_NODES to at least',total_nodes
            else if (id.eq.MAX_Z_ERROR) then
               write(6,*) 'Set MAX_Z_CELLS to at least',maxz
            else if (id.eq.MAX_R_ERROR) then
               write(6,*) 'Set MAX_R_CELLS to at least',maxr
            else if (id.eq.MAX_STAIR_ERROR) then
               write(6,*) 'Set MAX_STAIR_NODES to at least',
     1              stair_node_count
            else if (id.eq.MAX_RCS_ERROR) then
               write(6,*) 'Set MAX_RCS_NODES to at least',
     1              2*mheight+(rcsz1-rcsz2)
            end if

 10      continue
         write(6,*) '*********************************************'
         stop

      end if

      RETURN
      END

c*********************************************************************
c WRITE_OUT_ALL_PARMS outputs to a file all important parameters used
c    in running the simulation
c*********************************************************************

      SUBROUTINE write_out_all_parms

      implicit none
      include 'common.f'

      integer totsteps, mode_index, round
      real dt_out

      open(unit=9,file='bor.out',status='unknown',form='formatted')

 89   format('Scatter field end points (',I4,',',I4,'), (',I4,',',I4
     1     ')')
      write(9,89) xscat_sp,1,maxz-xscat_sp,int(obj_height/dz)+ytot_sp

C     *** WRITE TO fnameout ***
      write(9,11)
      write(9,17) (high_freq/1E9), (low_freq/1E9)
      write(9,11)
      if (modulate.eq.1) then
         write(9,31) modfreq/1.0E9
      else
         write(9,31) -1
      end if
      write(9,11)
      write(9,26) len,obj_height
      write(9,18) maxz,maxr,dz
      write(9,11)
      write(9,19) sigma_max
      write(9,21) movie_num
      write(9,22) movie_type
      write(9,23) mheight,rcsz1
      write(9,24) rcsz2, 2*mheight+rcsz2-rcsz1-1

      write(9,*)
      write(9,*) '          sdev =  ',sdev
      write(9,*) '       inc_ang =  ',inc_ang/pi*180,' (deg)'
      write(9,*) '            gd =  ',gd,' (sec)'
      write(9,*)

      write(9,*) ' Simulation Duration (ns) = ',sim_duration

      totsteps = 0
      do 80 mode_index = mode_start,mode_end
         dt = dt_out(mode_index)
         N = round(sim_duration*1e-9/dt)
         totsteps = totsteps + N
         write(9,36) mode_index,dt,N
 80   continue
 36   format(2X,'Mode = ',I2,2X'dt = ',E12.7,2X,'N time steps =',I6)
      write(9,*) ' Total Steps to run = ',totsteps

      write(9,11)
      write(9,34) eqset_start, eqset_end
 34   format(2X,'Running eqset ',I1,' through ',I1)
      write(9,*)
      if (abs(Ehg-1.0).lt.tole) then
         write(9,*) 'HH RCS calculated'
      else
         write(9,*) 'VV RCS calculated'
      end if

      if (calc_bist) then
         write(9,*) 'Bistatic RCS calculated'
      else
         write(9,*) 'Estimated Monostatic RCS calculated'
      end if

      write(9,11)
      if (.NOT.use_conformal) then
         write(9,*) ' Staircase approximation gridding used for ',
     1        fnamein,' geomfile'
      else
         write(9,*) ' Conformal gridding used for ', fnamein,
     1        ' geomfile'
      end if
      write(9,25)

      write(9,27) xtot_sp, ytot_sp
      write(9,28) xscat_sp, yscat_sp
      write(9,29) xhuy_sp, yhuy_sp
```

```
      write(9,30) xall_sp, yall_sp

 27   format('        xtot_sp = ',I8,'          ytot_sp = ',I8)
 28   format('       xscat_sp = ',I8,'         yscat_sp = ',I8)
 29   format('        xhuy_sp = ',I8,'          yhuy_sp = ',I8)
 30   format('        xall_sp = ',I8,'          yall_sp = ',I8)

      call plotb(ZB,RB,NP,51,41)

      close(unit=9)

C     *** FORMAT LINES ***

 09   format(I3)
 11   format('')
 17   format('High Freq (GHz) =',F6.2,3X,'Low Freq (GHz) = ',F6.2)
 31   format('Modulation Freq (GHz) (-1 = unmodulated)',F6.2)
 26   format(5X,'Length (m) = ',F4.2,4X,'Height (m) = ',F4.2)
 18   format(11X,'maxz = ',I5, 9X,'maxr = ',I4,9X,'dz = ',F8.7,' (m)')
 19   format('      sigma_max = ',F12.8)
 21   format('      movie_num = ',I12,'    (er=1, ez=2, ephi=3, hr=4, '
     1    ,'hz=5, hphi=6)')
 22   format('     movie_type = ',I12,'      (movie=1, wrtraw=2)')
 23   format('        mheight = ',I12,'         rcsz1 = ',I8)
 24   format('          rcsz2 = ',I12,'       NPInRCS = ',I8)

 25   format(/,'ADJUSTED DATA POINTS TO FIT FDTD GRID')

      RETURN
      END

c**********************************************************************
c DT_OUT returns the required dt for stability based on mode number
c    and dz. Function used so that all dts in the program are calculated
c    in the same way.
c**********************************************************************

      REAL FUNCTION dt_out(mode)

      implicit none
      include 'common.f'

      integer mode

c**** Taflove's stability criterion, note with conformal method, must
c**** use a smaller time step.
      dt_out=dz/((max(mode+1.0,1.45))*c)
      if (use_conformal) then
         dt_out = 0.80*dt_out
      else
         dt_out = 0.95*dt_out
      end if

c      dt_out = 0.90*dt_out

c***** Davidson stability creiterion that only works for low order modes.
c      dt_out = 0.90*(dz/c)*(((mode+1.0)**2.0 + 2.8)/4 + 1.0)**(-0.5)

      RETURN
      END

c**********************************************************************
c the initialize routine
c**********************************************************************

      SUBROUTINE init_fields

      implicit none
      include 'common.f'
      integer k,i

      do 10 k=1,maxz
         do 20 i=1,maxr
            er(k,i)=0.0
            ez(k,i)=0.0
            ephi(k,i)=0.0
            hr(k,i)=0.0
            hz(k,i)=0.0
            hphi(k,i)=0.0
 20      continue
 10   continue

      do 30 k=1,pmldepth+1
         do 40 i=0,pmldepth+maxr+1
            erphil(k,i)=0.0
            erzl(k,i)=0.0

            ezphil(k,i)=0.0
            ezrl(k,i)=0.0

            ephirl(k,i)=0.0
            ephizl(k,i)=0.0

            hrphil(k,i)=0.0
            hrzl(k,i)=0.0

            hphirl(k,i)=0.0
            hphizl(k,i)=0.0

            hzphil(k,i)=0.0
            hzrl(k,i)=0.0
 40      continue
 30   continue

      do 50 k=1,pmldepth+1
         do 60 i=0,pmldepth+maxr+1
            erphir(k,i)=0.0
            erzr(k,i)=0.0

            ezrr(k,i)=0.0
            ezphir(k,i)=0.0

            ephizr(k,i)=0.0
            ephirr(k,i)=0.0

            hrphir(k,i)=0.0
            hrzr(k,i)=0.0

            hphizr(k,i)=0.0
            hphirr(k,i)=0.0

            hzphir(k,i)=0.0
            hzrr(k,i)=0.0
 60      continue
 50   continue

      do 90 k=1,maxz
         do 100 i=1,pmldepth+1
            erzt(k,i)=0.0
            erphit(k,i)=0.0

            ezphit(k,i)=0.0
            ezrt(k,i)=0.0

            ephizt(k,i)=0.0
            ephirt(k,i)=0.0

            hrphit(k,i)=0.0
            hrzt(k,i)=0.0

            hphirt(k,i)=0.0
            hphizt(k,i)=0.0

            hzphit(k,i)=0.0
            hzrt(k,i)=0.0
 100     continue
 90   continue

      return
      end

c**********************************************************************c
c FDTD Loop: loops through all time steps updating electric and
c    magnetic fields and call boundary condition routines to enforce
c    PEC BCs
c**********************************************************************c

      SUBROUTINE fdtd_loop(store_freqs)

      implicit none
      include 'common.f'

      integer k,i,m,eqset,ms,round,movie_frame
      logical store_freqs
      real dt_out

      open(unit=18,file='effscat.dat',status='unknown',form='formatted')
      open(unit=19,file='hffscat.dat',status='unknown',form='formatted')

c      print *,'debug 0'
      call memory_check
      call write_out_all_parms

      movie_frame = 0
      time = 0

      print *, "Starting simulation..."
      do 5 m=mode_start,mode_end
         dt = dt_out(m)
         N = round(sim_duration*1e-9/dt)

         do 10 eqset=eqset_start,eqset_end
            ms=(-1)**(eqset+1)
```

```
            print *,"Mode=",m,"  Equation Set #",eqset
            do 20 time=1,N
               print *,time,' of ',N
               do 30 k=1,maxz
                  do 40 i=1,maxr
                     call free_space_E(k,i,m,ms,use_conformal)
 40               continue
 30            continue

               call pmlEeqn(m*ms,ms)

               if (use_conformal) then
                  call boundary_conditions(m,ms)
               else
                  if (use_stair2) then
                     call stair_boundary_conditions
                  else
                     call staircase_approx
                  end if
               end if

               do 50 k=1,maxz
                  do 60 i=1,maxr
                     call free_space_H(k,i,m,ms,use_conformal)
 60               continue
 50            continue
               call pmlHeqn(m*ms,ms)

c**************Store some fields for analysis
c               write(18,*) ez(236,75), er(236,75), ephi(236,75)
c               write(19,*) ez(236,35), er(236,35), ephi(236,35)

c               print *, ez(136,75), er(136,75), ephi(136,75)
c               print *, ez(136,35), er(136,35), ephi(136,35)

               if (store_movie) then
                  if (movie_frame.eq.movie_step) then
                     call movie(m,ms)
                     movie_frame = 0
                  else
                     movie_frame = movie_frame + 1
                  end if
               end if
               if (store_freqs) call update_dft(m, eqset)

 20         continue
            call init_fields
 10      continue

 5    continue

c**closing movie file
      close(unit=4)

c**closing eff and hff scat files.
      close(unit=18)
      close(unit=19)

      return
      end

c*********************************************************************c
c determines whether the grid cell (k,i) is a total or scattered
c field.
c*********************************************************************c

      logical function inside(k,i)

      implicit none
      include 'common.f'
      integer k,i,t

      t=scattot(k,i)

C     *** total fields are 2-9,14 ***

      inside=(t.ge.2.AND.t.le.9)
      inside=(inside.OR.t.eq.14)

      return
      end

c*********************************************************************c
c free_space_E contains the core update equations for calculating
c the free space E fields.
c*********************************************************************c

      SUBROUTINE free_space_E(k,i,m,ms,conformal)

      implicit none
      include 'common.f'
      integer k,i,m,st,ms
      real c1,c2,c3,c4,c5
      real gquad
      logical conformal

      st=scattot(k,i)

      if (i.ne.1) THEN
C **********Calculate E fields at time n+0.5

C ***************************Ez
         if ((ez_conform1(k,i).eq.0.AND.conformal).OR.(.not.
     1        conformal)) then
            c1=(i+0.5-1.0)*dt/(eps*(i+0.0-1.0)*dz)
            c2=(i-0.5-1.0)*dt/(eps*(i+0.0-1.0)*dz)
            c3=(m+0.0)*dt/(eps*(i+0.0-1.0)*dz)

            c4=hphi(k,i-1)

            if (st.eq.11) c4=c4-gquad(0.0,2*pi,ms*11,m,time*dt,(i-1)*dz,
     1           k*dz,inc_ang)

            ez(k,i)=ez(k,i)+(c1*hphi(k,i)-c2*c4+ms*c3*hr(k,i))/eta
         end if

C ***************************Ephi
         if ((ephi_conform1(k,i).eq.0.AND.conformal).OR.(.not.
     1        conformal)) then
            c1=dt/(eps*dz)
            c4=hz(k,i-1)

            if (k+1.gt.maxz) THEN
               c5=(hrzr(1,i)+hrphir(1,i))
            ELSE
               c5=hr(k+1,i)
            END IF

            if (st.eq.6.OR.st.eq.7.OR.st.eq.8) c5=c5+gquad(0.0,2*pi,
     1           ms*7,m,time*dt,i*dz,(k+1)*dz,inc_ang)
            if (st.eq.1) c5=c5-gquad(0.0,2*pi,ms*7,m,time*dt,i*dz,
     1           (k+1)*dz,inc_ang)
            if (st.eq.11) c4=c4-gquad(0.0,2*pi,ms*9,m,time*dt,(i-1)*
     1           dz,k*dz,inc_ang)

            ephi(k,i)=ephi(k,i)+(c1*(c4-hz(k,i)+c5-hr(k,i)))/eta
         end if
C ***************************Er
         if ((er_conform1(k,i).eq.0.AND.conformal).OR.(.not.
     1        conformal)) then
            if (k.eq.maxz) THEN
               c5=hphizr(1,i)+hphirr(1,i)
            ELSE
               c5=hphi(k+1,i)
            END IF

            if (st.eq.6.OR.st.eq.7.OR.st.eq.8) THEN
               c5=c5+gquad(0.0,2*pi,ms*11,m,time*dt,i*dz,(k+1)*dz,
     1              inc_ang)
            END IF

            if (st.eq.1) c5=c5-gquad(0.0,2*pi,ms*11,m,time*dt,i*dz,
     1           (k+1)*dz,inc_ang)

            c1=dt/(eps*dz)
            c2=(m*dt/eps)/((i+0.5-1.0)*dz)

            er(k,i)=er(k,i)+(c1*(hphi(k,i)-c5)-ms*c2*hz(k,i))/eta
         end if

       ELSE
C ********************************************************************** C
C ***************************On Axis Equations************************** C
C ********************************************************************** C

C ***************************Ez
         if ((ez_conform1(k,i).eq.0.AND.conformal).OR.(.not.
     1        conformal)) then
            c1=4*dt/(eps*dz)
            ez(k,i)=ez(k,i)+(c1*hphi(k,i))/eta
C*****************If the Fourier mode is not 0 ez(k,1) is zero.
            if (m.ne.0) ez(k,i)=0.0
         end if

C ***************************Ephi

         if ((ephi_conform1(k,i).eq.0.AND.conformal).OR.(.not.
     1        conformal)) then
            c1=2*dt/(eps*dz)
            c2=dt/(eps*dz)
            if (k.eq.maxz) THEN
               c5=hrzr(1,i)+hrphir(1,i)
            ELSE
```

```
            c5=hr(k+1,i)
          END IF

          if (st.eq.8) c5=c5+gquad(0.0,2*pi,ms*7,m,time*dt,i*dz,
     1          (k+1)*dz,inc_ang)
          if (st.eq.1) c5=c5-gquad(0.0,2*pi,ms*7,m,time*dt,i*dz,
     1          (k+1)*dz,inc_ang)

          ephi(k,i)=ephi(k,i)+(-c1*hz(k,i)+c2*(c5-hr(k,i)))/eta
c     ******If the fourier mode !=1 then ephi(k,1) and hr(k,i) = zero
          if (m.ne.1) THEN
            ephi(k,i)=0.0
          END IF
        end if

C ***************************Er
        if ((er_conform1(k,i).eq.0.AND.conformal).OR.(.not.
     1       conformal)) then
          if (k.eq.maxz) THEN
            c5=hphizr(1,i)+hphirr(1,i)
          ELSE
            c5=hphi(k+1,i)
          END IF

          if (st.eq.8) c5=c5+gquad(0.0,2*pi,ms*11,m,time*dt,i*dz,
     1          (k+1)*dz,inc_ang)

          if (st.eq.1) c5=c5-gquad(0.0,2*pi,ms*11,m,time*dt,i*dz,
     1          (k+1)*dz,inc_ang)

          c1=c1/2.0
          c2=(m*dt/eps)/((i+0.5-1.0)*dz)
          er(k,i)=er(k,i)+(c1*(hphi(k,i)-c5)-ms*c2*hz(k,i))/eta
        end if
      END IF

      return
      end

c**********************************************************************c
c free_space_H contains the core update equations for calculating
c the free space H fields.
c**********************************************************************c

      SUBROUTINE free_space_H(k,i,m,ms,conformal)

      implicit none
      include 'common.f'

      integer k,i,m,st,ms
      real c1,c2,c3,c4,c5,gquad
      logical conformal

      st=scattot(k,i)

      if (i.ne.1) THEN

C ***************************Hr

        if ((conform_hr1(k,i).eq.0.AND.conformal).OR.(.not.
     1       conformal)) then
          c1=dt/(mu*dz)
          c2=(m*dt)/(mu*(i+0.0-1.0)*dz)
          if (k.eq.1) THEN
            c5=ephizl(1,i)+ephirl(1,i)
          ELSE
            c5=ephi(k-1,i)
          END IF

          if (st.eq.2.OR.st.eq.3.or.st.eq.4) c5=c5+gquad(0.0,2*pi,
     1          ms*2,m,time*dt,i*dz,(k-1)*dz,inc_ang)
          if (st.eq.12) c5=c5-gquad(0.0,2*pi,ms*2,m,time*dt,i*dz,
     1          (k-1)*dz,inc_ang)

          hr(k,i)=hr(k,i)+eta*(c1*(ephi(k,i)-c5)-ms*c2*ez(k,i))
        end if

C ***************************Hphi
C ******* only calculate if not a boundary cell as defined by
C ******* the conform_grid1 array

        if ((conform_grid1(k,i).eq.0.AND.conformal).OR.(.not.
     1       conformal)) then
          c1=dt/(mu*dz)
          if (k.eq.1) THEN
            c5=erzl(1,i)+erphil(1,i)
          ELSE
            c5=er(k-1,i)
          END IF

          if (i.eq.maxr) THEN
            c4=ezrt(k,1)+ezphit(k,1)
          ELSE
            c4=ez(k,i+1)
          END IF

          if (st.eq.4.or.st.eq.5.or.st.eq.6) c4=c4+gquad(0.0,2*pi,
     1          ms*6,m,time*dt,(i+1)*dz,k*dz,inc_ang)
          if (st.eq.2.or.st.eq.3.or.st.eq.4) c5=c5+gquad(0.0,2*pi,
     1          ms*4,m,time*dt,i*dz,(k-1)*dz,inc_ang)
          if (st.eq.12) c5=c5-gquad(0.0,2*pi,ms*4,m,time*dt,i*dz,
     1          (k-1)*dz,inc_ang)

          hphi(k,i)=hphi(k,i)+eta*(c1*(c4-ez(k,i)+c5-er(k,i)))
        end if

C ***************************Hz

        if ((conform_hz1(k,i).eq.0.AND.conformal).OR.(.not.
     1       conformal)) then
          c1=((i+0.0-1.0)*dt/mu)/((i+0.5-1.0)*dz)
          c2=((i+1.0-1.0)*dt/mu)/((i+0.5-1.0)*dz)
          c3=(m*dt/mu)/((i+0.5-1.0)*dz)
          if (i.eq.maxr) THEN
            c4=ephirt(k,1)+ephizt(k,1)
          ELSE
            c4=ephi(k,i+1)
          END IF

          if (st.eq.4.or.st.eq.5.or.st.eq.6) c4=c4+gquad(0.0,2*pi,
     1          ms*2,m,time*dt,(i+1)*dz,k*dz,inc_ang)

          hz(k,i)=hz(k,i)+eta*(c1*ephi(k,i)-c2*c4+ms*c3*er(k,i))

        end if

      ELSE
C*********************************************************************** C
C ***************************On Axis Equations********************** C
C ***************************************************************** C

C***************************Hr
        if ((conform_hr1(k,i).eq.0.AND.conformal).OR.(.not.
     1       conformal)) then
c           if (k.eq.9) print *,k,i,ephi(k,i),ephi(k,i+1)
          c1=dt/(mu*dz)
          if (k.eq.1) THEN
            c5=ephizl(1,i)+ephirl(1,i)
          ELSE
            c5=ephi(k-1,i)
          END IF

          if (st.eq.2) c5=c5+gquad(0.0,2*pi,ms*2,m,time*dt,i*dz,
     1          (k-1)*dz,inc_ang)
          if (st.eq.12) c5=c5-gquad(0.0,2*pi,ms*2,m,time*dt,i*dz,
     1          (k-1)*dz,inc_ang)

          hr(k,i)=hr(k,i)+eta*(-ms*c1*ez(k,i+1)+c1*(ephi(k,i)-c5))

c******If the fourier mode !=1 then ephi(k,1) and hr(k,1) = zero
          if (m.ne.1) THEN
            hr(k,i)=0.0
          END IF
        end if

C ***************************Hphi

        if ((conform_grid1(k,i).eq.0.AND.conformal).OR.(.not.
     1       conformal)) then
          c1=dt/(mu*dz)
          if (k.eq.1) THEN
            c5=erzl(1,i)+erphil(1,i)
          ELSE
            c5=er(k-1,i)
          END IF
          if (i.eq.maxr) THEN
            c4=ezrt(k,1)+ezphit(k,1)
          ELSE
            c4=ez(k,i+1)
          END IF

          if (st.eq.4.or.st.eq.5.or.st.eq.6) c4=c4+gquad(0.0,2*pi,
     1          ms*6,m,time*dt,(i+1)*dz,k*dz,inc_ang)

          if (st.eq.2) c5=c5+gquad(0.0,2*pi,ms*4,m,time*dt,i*dz,
     1          (k-1)*dz,inc_ang)

          if (st.eq.12) c5=c5-gquad(0.0,2*pi,ms*4,m,time*dt,i*dz,
     1          (k-1)*dz,inc_ang)

          hphi(k,i)=hphi(k,i)+eta*(c1*(c4-ez(k,i)+c5-er(k,i)))
        end if

C ***************************Hz
```

```
         if ((conform_hzi(k,i).eq.0.AND.conformal).OR.(.not.
     1        conformal)) then
c           print *,k,i
            c1=((i+0.0-1.0)*dt/mu)/((i+0.5-1.0)*dz)
            c2=((i+1.0-1.0)*dt/mu)/((i+0.5-1.0)*dz)
            c3=(m*dt/mu)/((i+0.5-1.0)*dz)

            if (i.eq.maxr) THEN
               c4=ephirt(k,1)+ephizt(k,1)
            ELSE
               c4=ephi(k,i+1)
            END IF

            if (st.eq.4.or.st.eq.5.or.st.eq.6) c4=c4+gquad(0.0,2*
     1           pi,ms*2,m,time*dt,(i+1)*dz,k*dz,inc_ang)

            hz(k,i)=hz(k,i)+eta*(c1*ephi(k,i)-c2*c4+ms*c3*sr(k,i))

         end if
      END IF

      return
      end

C*********************************************************************C
C     Write out numerical values for each point in the bitmap         C
C*********************************************************************C

      SUBROUTINE matlab
      include 'common.f'
      integer i,k

      open(unit=81,file='matlab.dat',status='unknown',form='formatted')

      do 10 i=pmldepth,1,-1
         do 20 k=pmldepth,1,-1
            write(81,*) hphirl(k,i+maxr)+hphizl(k,i+maxr)
 20      continue
         do 30 k=1,maxz
            write(81,*) hphirt(k,i)+hphizt(k,i)
 30      continue
         do 40 k=1,pmldepth
            write(81,*) hphirr(k,i+maxr)+hphizr(k,i+maxr)
 40      continue
 10   continue

      do 50 i=maxr,1,-1
         do 60 k=pmldepth,1,-1
            write(81,*) hphirl(k,i)+hphizl(k,i)
 60      continue
         do 70 k=1,maxz
            write(81,*) hphi(k,i)
 70      continue
         do 80 k=1,pmldepth
            write(81,*) hphirr(k,i)+hphizr(k,i)
 80      continue
 50   continue

      return
      end
```

The following, **geom.f**, contains the subroutines for setting up the computational domain as well as determining the staircase or conformal grid representation of the object.

```
c*********************************************************************c
      integer function round(x)

      real x, dec

      dec = int(x)-x

      if (abs(dec).gt.0.5) then
         round = int(x)+1
      else
         round = int(x)
      end if

      return
      end

c*********************************************************************c

      logical function hphi_inside(r1,z1,r2,z2,k,i)

      implicit none
      include 'common.f'

      real r1,z1,r2,z2, slope, yk
      integer k,i
      logical inside_pec

C***  This subroutine determines if the hphi field at cell (k,i),
C***  but at physical location (k,i-0.5) is inside or outside
C***  the PEC.  It assumes the surface is closed and is convex.

C***  Check for vertical line

      if (abs(z1-z2).lt.(dz/2)) then

C******  Decide if target to left or right.

         if (r2.gt.r1) then

C******  Target on the right side of line

            if ((k+0.0).gt.z2.AND.(i-0.5).lt.r2) then
               hphi_inside = .true.
            else
               hphi_inside = .false.
            end if

         else

C******  Target on the left side of line

            if ((k+0.0).lt.z2.AND.(i-0.5).lt.r1) then
               hphi_inside = .true.
            else
               hphi_inside = .false.
            end if

         end if

      else

C******  Either horizontal, slanted right or left.

         slope = (r2-r1+0.0)/(z2-z1+0.0)
         yk = slope*(k+0.0)-z1*slope+r1

c         print *,r1,z1,slope,yk,k,i

         if (yk.gt.(i-0.5)) then
            hphi_inside = .true.
         else
            hphi_inside = .false.
         end if

      end if

c      if (k.eq.285) then
c         print *,k,i,z1,r1,z2,r2,hphi_inside
c      end if

      if (hphi_inside.ne.inside_pec(k,i,hphif)) print *,k,i,hphi_inside

      return
      end

c*********************************************************************c

      logical function OnBoundQ(z1,r1,z2,r2,k,i)

      implicit none
      include 'common.f'

      real z1,r1,z2,r2, rh, rl
      integer k,i

      rh = max(r2,r1)
      rl = min(r2,r1)

c      OnBoundQ = (i.lt.rh.AND.i.gt.rl)
c      OnBoundQ = OnBoundQ.OR.((i-1).lt.rh.AND.i.gt.rl)
c      if (abs(z2-z1).gt.tole) then
c         OnBoundQ = OnBOundQ.AND.k.le.z2.AND.k.ge.z1
c      else
c      end if

c      OnBoundQ = (i.le.rh+1E-7.AND.i.ge.rl-1E-7)
c      OnBoundQ = OnBoundQ.OR.((i-1).le.rh+1E-7.AND.i.ge.rl-1E-7)

      OnBoundQ = (i.lt.rh.AND.i.gt.rl)
      OnBoundQ = OnBoundQ.OR.((i-1).lt.rh.AND.i.gt.rl)
c      OnBoundQ = OnBoundQ.OR.abs(k+0.5-z2).lt.tole

      return
      end
```

```
c*****************************************************************c
c Determine the enter and exit points of the surface given      c
c the cell (k,i) location and the index of the data points      c
c surrounding it.  Note the physical loc is (k,i-0.5)           c
c*****************************************************************c

      subroutine enter_exit(k,i,index,ze,re,zx,rx,type,id)

      implicit none
      include 'common.f'
      integer k,i,index,type,entry_side,exit_side,id
      real z1,r1,z2,r2,slope,ze,re,zx,rx

C***  ze,re: entry point; zx,rx: exit point

C***  type describes the type of intersection
C***  type = 1 means intersects two perpendicular lines
C***  type = 2 means intersects two parallel lines.

C***  entry_side, exit_side describe entry and exit sides.
C***  e_s = 1 implies parallel to z-axis, perp to r-axis
C***  e_s = 2 implies parallel to r-axis, perp to z-axis
      integer par_z, par_r
      parameter(par_z=1,par_r=2)

      if (index.le.0.OR.index.gt.NP) then
         print *,'error 119, index out of range at ', id,k,i,index,id
         stop
      end if

      z1 = ZB(index)
      z2 = ZB(index+1)
      r1 = RB(index)
      r2 = RB(index+1)

      print *,index,z1,k,z2,r1,i,r2

      if (abs(z2-z1).gt.tole) then

         slope = (r2-r1+0.0)/(z2-z1+0.0)

         if (r1.lt.(i+0.0).AND.r1.gt.(i-1.0)) then
            ze = z1
            re = r1
            entry_side = par_r
         else
            if (slope.ge.(0.0)) then
               re = i-1.0
            else
               re = i+0.0
            end if
            ze = (re-r1)/slope + z1

            if (ze.gt.z2.OR.ze.lt.z1) then
               print *,'error 115 at ', id,k,i,z1,z2,r1,r2,ze,re,index
               stop
            end if
            entry_side = par_z
         end if

         if (r2.lt.(i+0.0).AND.r2.gt.(i-1.0)) then
            zx = z2
            rx = r2
            exit_side = par_r
         else
            if (slope.ge.(0.0)) then
               rx = i + 0.0
            else
               rx = i - 1.0
            end if
            zx = (rx-r1)/slope + z1

            if (zx.gt.z2.OR.zx.lt.z1) then
               print *,'error 116 at ', id,k,i,z1,z2,r1,r2,ze,re,zx,rx
               stop
            end if
            exit_side = par_z
         end if

c         print *,entry_side,exit_side
         if (entry_side.eq.exit_side) then
            type = parallel
         else
            type = perp
         end if

      else

C****** Vertical Line

         if (r2.gt.r1) then
            re = i-1.0
            ze = z1
            rx = i+0.0
            zx = z2
         else
            re = i+0.0
            ze = z1
            rx = i-1.0
            zx = z2
         end if

         type = parallel

      end if


      return
      end

c*****************************************************************c
c INSIDE_PEC(k,i) returns TRUE if the physical location of the c
c cell type is inside the pec. Uses global constants erf, ezf, c
c ephif, hrf, hzf, and hphif.                                   c
c*****************************************************************c

      logical function inside_pec(k,i,type)

      implicit none
      include 'common.f'

      integer k,i,index,above,below,type
      real tk, ti, min_z, max_z, min_r, max_r, slope
      real zi, ri, zip1, rip1

C***  in array cell k,i is FDTD grid location
C***  i-0.5, k+0.5 for er
C***  i-1.0, k+0.5 for ephi
C***  i-1.0, k     for ez
C***  i-1.0, k     for hr
C***  i-0.5, k     for hphi
C***  i-0.5, k+0.5 for hz

      if (type.eq.erf) then
         tk = k+0.5
         ti = i-0.5
      else if (type.eq.ephif) then
         ti = i-1.0
         tk = k+0.5
      else if (type.eq.ezf) then
         ti = i-1.0
         tk = k+0.0
      else if (type.eq.hrf) then
         ti = i-1.0
         tk = k+0.0
      else if (type.eq.hphif) then
         ti = i-0.5
         tk = k+0.0
      else if (type.eq.hzf) then
         ti = i-0.5
         tk = k+0.5
      else
         print *,'Unknown field type ',type
         stop
      end if

      min_z = maxz+1.0
      max_z = -1.0
      min_r = maxr+1.0
      max_r = -1.0
      do 10 index = 1,NP
         if (ZB(index).gt.max_z) max_z = ZB(index)
         if (ZB(index).lt.min_z) min_z = ZB(index)
         if (RB(index).gt.max_r) max_r = RB(index)
         if (RB(index).lt.min_r) min_r = RB(index)
 10   continue

      above = 0
      below = 0

      if (tk.gt.max_z.OR.tk.lt.min_z.OR.ti.gt.max_r.OR.ti.lt.min_r)
     1     then
         inside_pec = .FALSE.
      else
C***     count the number of intersections.
         do 20 index = 1,NP
            zi = ZB(index)
            ri = RB(index)
            if (index.eq.NP) then
               zip1 = ZB(1)
               rip1 = RB(1)
            else
               zip1 = ZB(index+1)
```

```
               rip1 = RB(index+1)
            end if
            if ( (tk.lt.zi.AND.tk.ge.zip1).OR.(tk.ge.zi.AND.tk.lt.
     1           zip1) ) then
               slope = (ri-rip1)/(zi-zip1)
               if ( (slope*(tk-zi)+ri).gt.ti ) then
                  above = above + 1
               else
                  below = below + 1
               end if
            end if
 20      continue

         if ( mod(above,2).eq.0.AND.mod(below,2).eq.0 ) then
            inside_pec = .FALSE.
         else
            inside_pec = .TRUE.
         end if
      end if

      return
      end

c*****************************************************************c
c ON_PEC(k,i) returns TRUE if the physical location of the      c
c cell type is on the surface of the pec. Uses global constants c
c erf, ezf, ephif, hrf, hzf, and hphif.                         c
c*****************************************************************c

      logical function on_pec(k,i,type)

      implicit none
      include 'common.f'

      integer k,i,index,type
      real tk, ti, min_z, max_z, min_r, max_r, slope
      real zi, ri, zip1, rip1
      logical conc

C***  in array cell k,i is FDTD grid location
C***  i-0.5, k+0.5 for er
C***  i-1.0, k+0.5 for ephi
C***  i-1.0, k     for ez
C***  i-1.0, k     for hr
C***  i-0.5, k     for hphi
C***  i-0.5, k+0.5 for hz

      if (type.eq.erf) then
         tk = k+0.5
         ti = i-0.5
      else if (type.eq.ephif) then
         ti = i-1.0
         tk = k+0.5
      else if (type.eq.ezf) then
         ti = i-1.0
         tk = k+0.0
      else if (type.eq.hrf) then
         ti = i-1.0
         tk = k+0.0
      else if (type.eq.hphif) then
         ti = i-0.5
         tk = k+0.0
      else if (type.eq.hzf) then
         ti = i-0.5
         tk = k+0.5
      else
         print *,'Unknown field type ',type
         stop
      end if

      min_z = maxz+1.0
      max_z = -1.0
      min_r = maxr+1.0
      max_r = -1.0
      do 10 index = 1,NP
         if (ZB(index).gt.max_z) max_z = ZB(index)
         if (ZB(index).lt.min_z) min_z = ZB(index)
         if (RB(index).gt.max_r) max_r = RB(index)
         if (RB(index).lt.min_r) min_r = RB(index)
 10   continue

      conc = .FALSE.
      if (tk.gt.max_z.OR.tk.lt.min_z.OR.ti.gt.max_r.OR.ti.lt.min_r)
     1     then
         on_pec = .FALSE.
         conc = .TRUE.
      else
C***     count the number of intersections.
         do 20 index = 1,NP
            zi = ZB(index)
            ri = RB(index)
            if (index.eq.NP) then
               zip1 = ZB(1)
               rip1 = RB(1)
            else
               zip1 = ZB(index+1)
               rip1 = RB(index+1)
            end if
            if (abs(zi-zip1).lt.tole) then
               if (abs(zi-tk).lt.tole) then
                  if (ti.le.max(ri,rip1).AND.ti.ge.min(ri,rip1)) then
                     on_pec = .TRUE.
                     conc = .TRUE.
                  end if
               end if
            else
               if (tk.ge.min(zi,zip1).AND.tk.le.max(zi,zip1)) then
                  slope = (ri-rip1)/(zi-zip1)
                  if (abs(slope*(tk-zi)+ri-ti).lt.tole) then
                     on_pec = .TRUE.
                     conc = .TRUE.
                  end if
               end if
            end if
 20      continue
      end if

      return
      end


c*****************************************************************c
c EZ_INSIDE(k,i) returns TRUE if the physical location of the   c
c ez_field at cell k,i is inside the scatter.                   c
c*****************************************************************c

      logical function ez_inside(k,i)

      implicit none
      include 'common.f'

      integer k,i
c      integer index1, findex
c      real pz, pr, z1, z2, r1, r2, cr
      logical inside_pec

c      pz = float(k)
c      pr = float(i)

C***  First find the correct indices of the surrounding data points.

c      findex = -999
c      do 10 index1 = 1,NP-1
c         z1 = ZB(index1)
c         z2 = ZB(index1+1)
c         if (pz.gt.z1.AND.pz.lt.z2) findex = index1
c 10   continue

c      if (findex.ne.-999) then
c         r1 = RB(findex)
c         r2 = RB(findex+1)
c         cr = (r1+r2)/2.0
c         if (cr.lt.pr) then
c            ez_inside = .FALSE.
c         else
c            ez_inside = .TRUE.
c         end if
c      else
c         ez_inside = .FALSE.
c      end if

      ez_inside = inside_pec(k,i,ezf)
      return
      end

c*****************************************************************c
c This subroutine is the geometry analyzer.                     c
c It calculates contour lengths and determines cell types,etc   c
c                                                               c
c The analzyer requires that the surface be closed, that        c
c is: r(1) = r(NP)                                              c
c                                                               c
c The BOR is formed by connecting the data points with straight c
c lines.                                                        c
c*****************************************************************c

      subroutine geometry

      implicit none
      include 'common.f'
      integer i, k, errorcode, round

      real toler
      parameter (toler = 9.99E-6)
```

```
      character filnam*1024, frmt*30
      integer ilen, spacing1

      real maxrb, zshift, rshift, slope

C***  On Boundary
C***  totOB is the total number of cells in array OnBdy that
C***  have part of the PEC cut through them.

      real OnBdy(1:4,MAXCP), z1,z2,r1,r2,z3
      integer totOB, index1, temp
      integer unknown, inside, outside, floor, accessNPi
      logical hphi_inside, OnBoundQ, ez_inside, inside_pec,on_pec
      parameter(accessNPi=4,unknown=0,inside=-1,outside=1)
      real minzp, maxzp, minrp, maxrp

C***  (k,i) is cell identifer with physical loc of hphi at (k,i-0.5)
C***  type is either unknown, inside, outside.

      integer sright, sdown, sleft, scright, scdown, scleft, noconform
      integer sscright, sscdown, sscleft, nused, sdownleft, sdownright
      integer scsdownleft, scsdownright, vright, vupright, vleft
      integer vupleft, vdownleft, vdownright
      parameter(sright=1,sdown=2,sleft=3)
      parameter(scright=4,scdown=5,scleft=6)
      parameter(sscright=7,sscdown=8,sscleft=9, noconform=20)
      parameter(nused=-32, scsdownleft=12, scsdownright=13)
      parameter(sdownleft=10, sdownright=11, vright=14, vupright=15)
      parameter(vleft=16, vupleft=17, vdownleft=18, vdownright=19)

C***  Matlab display lengend for use with geom.m script
C***  A - sright        - stretch cell to the right
C***  B - sdown         - stretch cell to downwards
C***  C - sleft         - stretch cell to the left
C***  D - scright       - self-conform to the right
C***  E - scdown        - self-conform downwards
C***  F - scleft        - self-conform to the left
C***  G - sscright      - stretch & self-conform to the right
C***  H - sscdown       - stretch & self-conform downwards
C***  I - sscleft       - stretch & self-conform to the left
C***  J - sdownleft     - special case of stretch down & left
C***  K - sdownright    - special case of stretch down & right
C***  L - scsdownleft   - self-conform and stretch down & left
C***  M - scsdownright  - self-conform and stretch down & right
C***  N - vright        - vertical wall to the right
C***  O - vupright      - vertical wall to right and stretch upwards
C***  P - vleft         - vertical wall to the left
C***  Q - vupleft       - vertical wall to left and stretch upwards
C***  R - vdownleft     - vertical wall to left and stretch downwards
C***  S - vdownright    - vertical wall to right and stretch downwards
C***  T - noconform     - a cell on boundary with hphi inside target

      integer lastoutk, lastouti, lastoutOB, lastink, lastini

      real conform_grid(1:5,start_z:end_z,1:end_r)
      integer temp_hz(1:mz,1:mr)

C***  hphiIO grid identifies inside and outside hphi fields by
C***  their k,i position which corresponds to physical (k,i-0.5)

      integer hphiIO(start_z:end_z,1:end_r), OUT, IN
      parameter(OUT = 0, IN = 1)

      integer typels, typels2, typels3, index12, index13
      real len1, len2, area, ze2, re2, zx2, rx2, ze3, re3, zx3, rx3

C***  identifies type of enter,exit relationship(see enter_exit routine)
      real ze,re,zx,rx
      integer type, sr, etr, str

      integer NPcount, defaults
      real zstep, zslope, radius

C**********************Start of Geom Analysis Code**************

C***  Initialize OnBdy vector

      totOB = 0
      do 5 i = 1,MAXCP
         OnBdy(accessk,i) = -1.0
         OnBdy(accessi,i) = -1.0
         OnBdy(accesst,i) = unknown
         OnBdy(accessNPi,i) = -1.0
 5    continue

C***  actype, acl1, acl2, acA, acNPi

C***  Initialize conform_grid array and hphiIO grid

      do 60 k=start_z,end_z
         do 70 i=1,end_r
            hphiIO(k,i) = OUT
            do 80 index1 = 1,5
               conform_grid(index1,k,i) = nused
 80         continue
 70      continue
 60   continue

C***  Initialize conform_list vector

      do 65 index1=1,MAXCP
         do 68 k = 1,9
            conform_list(k,index1) = nused
 68      continue
         borrow_list(ezt,index1) = NO
         borrow_list(ezb,index1) = NO
         borrow_list(erl,index1) = NO
         borrow_list(err,index1) = NO
 65   continue

C***  Initialize conform_hz, conform_hz_length

      do 22 index1=1,MAXCP
         conform_hz_length(index1) = nused
         conform_hz(1,index1) = nused
         conform_hz(2,index1) = nused
         conform_hz(3,index1) = nused
 22   continue

C*** Initialize conform_hr, conform_hr_length

      do 23 index1=1,MAXCP
         conform_hr_length(index1) = nused
         conform_hr(1,index1) = nused
         conform_hr(2,index1) = nused
         conform_hr(3,index1) = nused
 23   continue


      write(6,*) 'Setting up geometry...'

      write(6,'(''*Accept spacing defaults [Y=1,N=2]: '',$)')
      read(5,*) defaults

      if (defaults.eq.1) then
         xtot_sp=10
         ytot_sp=10

         xscat_sp=40
         yscat_sp=40

         xhuy_sp=2
         yhuy_sp=2

         xall_sp = xtot_sp+xscat_sp
         yall_sp = ytot_sp+yscat_sp
      else
         write(6,'(''*Enter xtot_sp [10]: '',$)')
         read(5,*) xtot_sp
         write(6,'(''*Enter ytot_sp [10]: '',$)')
         read(5,*) ytot_sp

         write(6,'(''*Enter xscat_sp [40]: '',$)')
         read(5,*) xscat_sp
         write(6,'(''*Enter yscat_sp [40]: '',$)')
         read(5,*) yscat_sp

         write(6,'(''*Enter xhuy_sp [2]: '',$)')
         read(5,*) xhuy_sp
         write(6,'(''*Enter yhuy_sp [2]: '',$)')
         read(5,*) yhuy_sp

         xall_sp = xtot_sp+xscat_sp
         yall_sp = ytot_sp+yscat_sp
      end if

      xall_sp = 50
      yall_sp = 50

c*********new geom readin procedure...

      write(6,*) 'Reading in data and analyzing geometry...'
      open(unit=11,file=fnamein,status='old',form='formatted')

      read(11,*) dz
      read(11,*) NP

      if (NP.gt.0) then

c        read(11,*) ZBt(1), RBt(1)
c        ZBt(1) = dz*round(ZBt(1)/dz)
c        ZBt(2) = ZBt(1)+dz
c        RBt(2) = RBt(1)+dz+dz/4
```

```
c          NP = NP+1
      do 333 i=1,NP
         read(11,*) ZBt(i), RBt(i)
c         if (i.eq.3) RBt(i) = RBt(i)
         ZBt(i) = dz*round(ZBt(i)/dz)
 333  continue

      NPcount = 0
      do 334 i=1,NP-1
         if (ZBt(i).ne.ZBt(i+1)) then
            zslope = (RBt(i)-RBt(i+1))/(ZBt(i)-ZBt(i+1))
            do 335 zstep = ZBt(i), ZBt(i+1)-dz/2, dz
               NPcount = NPcount+1
               ZBa(NPcount) = zstep
               RBa(NPcount) = zslope*(zstep-ZBt(i))+RBt(i)
 335        continue
         else
            NPcount = NPcount+1
            ZBa(NPcount) = ZBt(i)
            RBa(NPcount) = RBt(i)
         end if
 334  continue

      NPcount = NPcount + 1
      ZBa(NPcount) = ZBt(NP)
      RBa(NPcount) = RBt(NP)

      NP = NPcount

      else
c*****object is a sphere
         read(11,*) radius
         radius = dz*round(radius/dz)
         NPcount = 0
         do 341 zstep = 0.0,2*radius+dz/2,dz
            NPcount = NPcount+1
            ZBa(NPcount) = zstep
            RBa(NPcount) = sqrt(radius**2.0-(zstep-radius)**2.0)
 341     continue

         NP = NPcount
      end if

      close(unit=11)

      open(unit=10,file='whoknows.dat',status='unknown',
     1     form='formatted')
      do 338 i=1,NP
         write(10,*) ZBa(i), RBa(i)
 338  continue

c************



C***  Read in geometry file.
c      write(6,*) 'Reading in data and analyzing geometry...'
c      open(unit=11,file=fnamein,status='old',form='formatted')
c      read(11,*) dz
c      read(11,*) NP
c      read(11,*) (RBa(i),i=1,NP)
c      read(11,*) (ZBa(i),i=1,NP)

      i = 1
      dz = 0.0
 1    if (dz.ne.(0.0)) go to 2
         dz = ZBa(i+1) - ZBa(i)
c         print *,dz
         i = i+1
      go to 1
 2    dz = dz
c      print *,dz

C***  Check for proper geometry file that satisfies specs above.
      do 10 i = 1,NP-1
         errorcode = 100
         if (ZBa(i).gt.ZBa(i+1)) goto 980
         errorcode = 101
c         print *,dz,ZBa(i+1),ZBa(i),ZBa(i+1)-ZBa(i),toler
         if ((abs(abs(ZBa(i+1)-ZBa(i))-dz)).gt.toler.AND.
     1        (abs(ZBa(i+1)-ZBa(i)).gt.toler)) goto 980
 10   continue

      errorcode = 102
      if (abs(RBa(1)-RBa(NP)).gt.toler) goto 980

C***  Shift the BOR object so that the target is centered

      minzp = ZBa(1)
      maxzp = ZBa(1)
      minrp = RBa(1)
      maxrp = RBa(1)
      do 15 i = 1,NP
         if (ZBa(i).gt.maxzp) maxzp = ZBa(i)
         if (ZBa(i).lt.minzp) minzp = ZBa(i)
         if (RBa(i).gt.maxrp) maxrp = RBa(i)
         if (RBa(i).lt.minrp) minrp = RBa(i)
 15   continue
      len = maxzp-minzp
      obj_height = maxrp-minrp

c      spacing1 = max(90,int(1.0*c/low_freq/dz))
      spacing1 = 50
      maxz = 2*xall_sp + len/dz
      maxr = yall_sp + obj_height/dz

      if (maxz.gt.mz) then
         print *,'maxz =',maxz,' is greater than the allowed mz =',mz
         enough_memory = .FALSE.
      end if

      if (maxr.gt.mr) then
         print *,'maxr =',maxr,' is greater than the allowed mr =',mr
         enough_memory = .FALSE.
      end if

      zshift = ZBa(1)-xall_sp*dz
      rshift = RBa(1)

      do 20 i = 1,NP
         ZBa(i) = ZBa(i)-zshift
         RBa(i) = RBa(i)-rshift
 20   continue

C***  Now scale the BOR object

      maxrb = -1
      do 30 i = 1,NP
         ZB(i) = round(ZBa(i)/dz)+0.5
         RB(i) = RBa(i)/dz
c         print *,ZBa(i),ZB(i),RBa(i),RB(i)
         if (RB(i).gt.maxrb) maxrb = RB(i)
 30   continue

C***  Determine where targets cuts cells and whether the cut encloses
C***  hphi within (inside) the PEC or whether hphi is outside the PEC

      staircount = 0
      do 40 index1 = 1,NP-1
         k = ZB(index1)+0.5
         z1 = ZB(index1)
         z2 = ZB(index1+1)
         r1 = RB(index1)
         r2 = RB(index1+1)

         if (r2.gt.r1) then
            sr = 1
            etr = floor(maxrb) + 1
            str = 1
         else
            sr = floor(maxrb) + 1
            etr = 1
            str = -1
         end if

         do 50 i = sr,etr,str
            if (hphi_inside(r1,z1,r2,z2,k,i)) hphiIO(k,i) = IN
c            if (k.lt.53) then
c               print *,k,i,z1,r1,z2,r2, OnBoundQ(z1,r1,z2,r2,k,i)
c            end if
            if (OnBoundQ(z1,r1,z2,r2,k,i)) then
               staircount = staircount + 1
               if (str.eq.1) then
                  if (k.ne.staircase(ack,max(staircount-1,1)).AND.i.ne.
     1                 staircase(aci,max(staircount-1,1))) then
                     staircase(ack,staircount) = k
                     staircase(aci,staircount) = i-1
                     staircount = staircount + 1
                     staircase(ack,staircount) = k
                     staircase(aci,staircount) = i
                  else
                     if (.not.(k.eq.staircase(ack,max(staircount-1,1))
     1                    .AND.i.eq.staircase(aci,max(staircount-1,
     2                    1)))) then
                        staircase(ack,staircount) = k
                        staircase(aci,staircount) = i
                     else
                        staircount = staircount - 1
                     end if
                  end if
               else
                  if (k.ne.staircase(ack,max(staircount-1,1)).AND.i.ne.
     1                 staircase(aci,max(staircount-1,1))) then
```

```
                  staircase(ack,staircount) = k-1
                  staircase(aci,staircount) = i
                  staircount = staircount + 1
                  staircase(ack,staircount) = k
                  staircase(aci,staircount) = i
               else
                  if (.not.(k.eq.staircase(ack,max(staircount-1,1)).
     1                 AND.i.eq.staircase(aci,max(staircount-1,
     2                 1)))) then
                     staircase(ack,staircount) = k
                     staircase(aci,staircount) = i
                  else
                     staircount = staircount - 1
                  end if
               end if

            end if

            if (i.gt.r2.AND.(z2.eq.z1).AND.r2.gt.r1) then
               conform_grid(acNPi,k,i) = index1+1
            else
               conform_grid(acNPi,k,i) = index1
            end if

            if (hphi_inside(r1,z1,r2,z2,k,i)) then
c              hphiIO(k,i) = IN
               if (lastink.eq.k.AND.lastini.eq.i) then

               else
                  totOB = totOB + 1
                  OnBdy(accessk,totOB) = k
                  OnBdy(accessi,totOB) = i
                  OnBdy(accesst,totOB) = inside
                  if (i.gt.r2.AND.(z2.eq.z1).AND.r2.gt.r1) then
                     OnBdy(accessNPi,totOB) = index1+1
                     totOB = totOB + 1
                     OnBdy(accessk,totOB) = k
                     OnBdy(accessi,totOB) = i
                     OnBdy(accesst,totOB) = inside
                     OnBdy(accessNPi,totOB) = index1
                  else
                     OnBdy(accessNPi,totOB) = index1
                  end if
                  lastink = k
                  lastini = i
               end if
            else
               if (lastoutk.eq.k.AND.lastouti.eq.i) then
                  OnBdy(accessNPi,lastoutOB) = index1
               else
                  totOB = totOB + 1
                  OnBdy(accessk,totOB) = k
                  OnBdy(accessi,totOB) = i
                  OnBdy(accesst,totOB) = outside
                  OnBdy(accessNPi,totOB) = index1
                  lastoutk = k
                  lastouti = i
                  lastoutOB = totOB
               end if
            end if

         end if

 50      continue
 40   continue

C**** Write out staircase approx data
      ilen = index(dbase,' ') - 1
      write(frmt,'(a2,i4,a5)') '(a',ilen,',a10)'
      write(filnam,frmt)dbase,'/stair.dat'

      open(unit=10,file=filnam,status='unknown',form='formatted')
      do 45 index1 = 1,staircount
         write(10,*) staircase(ack,index1), staircase(aci,index1)
 45   continue
      close(unit=10)

c**** don't finish conformal gridding routine if only staircase needed
      if (.NOT.use_conformal) goto 1200

C***  check to ensure that OnBdy array was large enough.
c      print *,totOB
      errorcode = 103
      if (totOB.gt.MAXCP) go to 980

C***  Began checking inside cells.

      do 90 index1 = 1,totOB
         k = OnBdy(accessk,index1)
         i = OnBdy(accessi,index1)
c         conform_grid(acNPi,k,i) = OnBdy(accessNPi,index1)
         if (OnBdy(accesst,index1).eq.inside) then
            conform_grid(actype,k,i) = noconform

C*********  Checking for sdown, sright, or sleft. Will be sright
C*********  or sleft is abs(slope) > 1  (i.e. > 45 deg incline)

            z1 = ZB(OnBdy(accessNPi,index1))
            z2 = ZB(OnBdy(accessNPi,index1)+1)
            r1 = RB(OnBdy(accessNPi,index1))
            r2 = RB(OnBdy(accessNPi,index1)+1)
c            if (k.eq.91.AND.i.eq.7) print *,z1,r1,z2,r2
c            print *,z1,k,z2,r1,i,r2,OnBdy(accessNPi,index1)

            if (abs(z2-z1).gt.tole) then
               slope = (r2-r1+0.0)/(z2-z1+0.0)
c               if (k.eq.10.AND.i.eq.80) print *,slope,z1,r1,z2,r2

               if (abs(slope).gt.(1.0)) then
                  if (slope.gt.(0.0)) then
                     if (conform_grid(actype,k-1,i).eq.nused) then
                        conform_grid(actype,k-1,i) = sright
                     else
                        if (conform_grid(actype,k-1,i).ne.sdown) then
                           errorcode = 104
                           print *,conform_grid(actype,k-1,i)
                           go to 980
                        else
                           conform_grid(actype,k-1,i) = sdownright
                        end if
                     end if
                  else
                     if (conform_grid(actype,k+1,i).eq.nused) then
                        conform_grid(actype,k+1,i) = sleft
                     else
                        if (conform_grid(actype,k+1,i).ne.sdown) then
                           errorcode = 105
                           go to 980
                        else
                           conform_grid(actype,k+1,i) = sdownleft
                        end if
                     end if
                  end if
               else
                  if (conform_grid(actype,k,i+1).eq.nused) then
                     conform_grid(actype,k,i+1) = sdown
                  else
                     if (conform_grid(actype,k,i+1).eq.sdown) then
                        errorcode = 106
                        go to 980
                     else
                        if (conform_grid(actype,k,i+1).eq.sright) then
                           conform_grid(actype,k,i+1) = sdownright
                        else
c                           print *,'test 2'
                           conform_grid(actype,k,i+1) = sdownleft
                        end if
                     end if
                  end if
               end if
            else

C************ Vertical line
               if (r2.gt.r1) then
                  if (conform_grid(actype,k-1,i).eq.nused) then
                     if ((i+1.0).lt.(r2+1.0)) then
                        conform_grid(actype,k-1,i) = vright
                     else
                        conform_grid(actype,k-1,i) = vupright
                     end if
                  else
                     if (conform_grid(actype,k-1,i).eq.sdown.OR.
     1                    conform_grid(actype,k-1,i).eq.noconform) then

                     else
                        print *,k-1,i,conform_grid(actype,k-1,i)
                        errorcode = 107
                        go to 980
                     end if
                  end if
               else
c                  if (conform_grid(actype,k+1,i).eq.nused) then
c                     conform_grid(actype,k+1,i) = sleft
c                  else
                     errorcode = 108
                     go to 980
c                  end if
               end if
            end if

         end if
 90   continue
```

```
C***  Began checking outside cells.

      do 100 index1 = 1,totOB
         k = OnBdy(accessk,index1)
         i = OnBdy(accessi,index1)

         if (OnBdy(accesst,index1).eq.outside) then

C*********  Checking for scdown, scright, or scleft. Will be scright
C*********  or scleft is abs(slope) > 1  (i.e. > 45 deg incline)

            z1 = ZB(OnBdy(accessNPi,index1))
            z2 = ZB(OnBdy(accessNPi,index1)+1)
            r1 = RB(OnBdy(accessNPi,index1))
            r2 = RB(OnBdy(accessNPi,index1)+1)

            if (abs(z2-z1).gt.tole) then
               slope = (r2-r1+0.0)/(z2-z1+0.0)
c               if (k.eq.10.AND.i.eq.81) print *,slope,z1,r1,z2,r2

               if (abs(slope).gt.(1.0)) then
                  if (slope.gt.(0.0)) then

C*****************    Right stretch
                     if (conform_grid(actype,k,i).eq.nused) then
                        conform_grid(actype,k,i) = scright
                     else
                        temp = conform_grid(actype,k,i)
                        if (temp.ne.sright.AND.temp.ne.sdownright) then
                           errorcode = 109
                           go to 980
                        else
                           if (temp.eq.sright) then
                              conform_grid(actype,k,i) = sscright
                           end if
                           if (temp.eq.sdownright) then
                              conform_grid(actype,k,i) = scsdownright
                           end if
                        end if
                     end if
                  else
C*****************    Left stretch
                     if (conform_grid(actype,k,i).eq.nused) then
                        conform_grid(actype,k,i) = scleft
                     else
                        temp = conform_grid(actype,k,i)
                        if (temp.ne.sleft.AND.temp.ne.sdownleft) then
                           errorcode = 110
                           go to 980
                        else
                           if (temp.eq.sleft) then
                              conform_grid(actype,k,i) = sscleft
                           end if
                           if (temp.eq.sdownleft) then
                              conform_grid(actype,k,i) = scsdownleft
                           end if
                        end if
                     end if
                  end if

               else

C*****************    Downwards stretch
                  if (conform_grid(actype,k,i).eq.nused) then
                     conform_grid(actype,k,i) = scdown
                  else
                     temp = conform_grid(actype,k,i)
                     if (temp.ne.sdown.AND.temp.ne.sdownleft.AND.
     1                    temp.ne.sdownright.AND.temp.ne.scsdownleft.
     2                    AND.temp.ne.scsdownright.AND.temp.ne.vleft.
     3                    AND.temp.ne.vright) then
c                       print *,char(temp+64)
                        errorcode = 111
c                        go to 980
                     else
                        if (temp.eq.vright.OR.temp.eq.vleft) then
                           if (temp.eq.vright) then
                              conform_grid(actype,k,i) = vdownright
                           end if


                           if (temp.eq.vleft) then
c                              print *,'test 3'
                              conform_grid(actype,k,i) = vdownleft
                           end if
                        else


                           if (temp.ne.sdownleft.AND.temp.ne.sdownright.
     1                          AND.temp.ne.scsdownleft.AND.temp.ne.
     2                          scsdownright) then
c                             print *,'test 1',k,i,temp
                              conform_grid(actype,k,i) = sscdown
                           end if
                        end if
                     end if
                  end if

               else

C************ Vertical line

                  if (r2.gt.r1) then
c                     if (k.eq.10.AND.i.eq.81) print *,z1,r1,z2,r2
                     if (conform_grid(actype,k,i).eq.nused) then
                        conform_grid(actype,k,i) = scright
                     else
                        errorcode = 112
                        go to 980
                     end if
                  else
                     if (conform_grid(actype,k,i).eq.nused) then
                        if ((i+0.0).le.r1) then
                           if ((i+1.0).le.r1) then
                              conform_grid(actype,k,i) = vleft
                           else
                              conform_grid(actype,k,i) = vupleft
                           end if
                        end if
                     else
                        if (conform_grid(actype,k,i).ne.scdown.AND.
     1                       conform_grid(actype,k,i).ne.noconform.AND.
     2                       conform_grid(actype,k,i).ne.sdown) then
                           print *,k,i,conform_grid(actype,k,i)
                           errorcode = 113
                           go to 980
                        end if
                     end if
                  end if

               end if

            end if
 100  continue

C***  Write out the enter and exit calculated points
c      ilen = index(dbase,' ') - 1
c      write(frmt,'(a2,i4,a5)') '(a',ilen,',a12)'
c      write(filnam,frmt)dbase,'/ent_ext.dat'

c      open(unit=10,file=filnam,status='unknown',form='formatted')

c      do 2001 k = start_z,end_z
c         do 2101 i = 1,end_r
c            index1 = conform_grid(acNPi,k,i)
c            if (conform_grid(actype,k,i).ne.-32.AND.index1.ne.-32) then
c               call enter_exit(k,i,index1,ze,re,zx,rx,type,-99)
c               write(10,*) ze,k,zx,re,i-0.5,rx,type
c            end if
c 2101    continue
c 2001 continue

c      close(unit=10)

C***  Contour types.
      ilen = index(dbase,' ') - 1
      write(frmt,'(a2,i4,a5)') '(a',ilen,',a10)'
      write(filnam,frmt)dbase,'/cgrid.dat'

      open(unit=10,file=filnam,status='unknown',form='formatted')

      do 1001 k = start_z,end_z
         do 1101 i = 1,end_r
            if (conform_grid(actype,k,i).ne.-32.AND.conform_grid
     1           (actype,k,i).ne.noconform) then
               write (10,*) k,i,conform_grid(actype,k,i)+64
               conform_grid1(k,i)=1
            else
               conform_grid1(k,i)=0
            end if
 1101    continue
 1001 continue

      close(unit=10)



C***  At this point all conformal grid cells have been identified
C***  and classified.  The following determines the contour lengths,
C***  areas, and which fields need to be borrowed and/or set to zero.

C***  A list will now be created that contains all the conformal grid
C***  cells (k,i) location, their contour lengths, areas, which fields
C***  need to be borrowed, etc.
```

```
C***  Note: ack,aci,actype,acl1,acl2,aclA,acBorrow)

      listcount = 0

      do 110 k = start_z,end_z
         do 120 i = 1,end_r
            type = conform_grid(actype,k,i)
            if (type.ne.nused.AND.type.ne.noconform) then
               listcount = listcount + 1
               conform_list(ack,listcount) = k
               conform_list(aci,listcount) = i
               conform_list(actype,listcount) = type

C***************  check to see what needs to be borrowed.

                  if (hphiIO(k-1,i).eq.IN) then
c                     if (.not.inside_pec(k-1,i,erf)) then
                        borrow_list(erl,listcount) = YES
c                     end if
                  end if

                  if (hphiIO(k,max(i-1,1)).eq.IN) then
c                     if (.not.ez_inside(k,i)) then
                        if (ez_inside(k-1,i)) then
                           borrow_list(ezb,listcount) = YES_RIGHT
                        else
                           if (conform_list(aci,listcount-1).eq.i.AND.
     1                          borrow_list(ezb,listcount-1).eq.
     2                          YES_RIGHT) then
                              borrow_list(ezb,listcount) = YES_RIGHT
                           else
                              borrow_list(ezb,listcount) = YES_LEFT
                           end if
                        end if
c                     end if
                  end if

                  if (hphiIO(k+1,i).eq.IN.OR.hphiIO(k,i).eq.IN) then
c                     if (.not.inside_pec(k,i,erf)) then
                        borrow_list(err,listcount) = YES
c                     end if
                  end if

                  if (hphiIO(k,i+1).eq.IN.OR.hphiIO(k,i).eq.IN) then
                     if (ez_inside(k-1,i+1)) then
                        borrow_list(ezt,listcount) = YES_RIGHT
                     else
                        borrow_list(ezt,listcount) = YES_LEFT
                     end if
                  end if

C*********************************************************************
C*******************   Type SRIGHT (A) / VRIGHT (N)  *****************
C*********************************************************************

               if (type.eq.sright.OR.type.eq.vright) then
                  index1 = conform_grid(acNPi,k+1,i)
                  call enter_exit(k+1,i,index1,ze,re,zx,rx,typels,type)
                  errorcode = 120
                  if (typels.ne.parallel) go to 980

                  len1 = zx - (k-0.5)
                  len2 = ze - (k-0.5)

                  errorcode = 220
c                  print *,len1
                  if (len1.gt.(2.0).OR.len1.lt.(0.0)) go to 980
                  errorcode = 320
                  if (len2.gt.(2.0).OR.len2.lt.(0.0)) go to 980

                  area = 0.5*(len1+len2)

                  conform_list(acl1,listcount) = len1
                  conform_list(acl2,listcount) = len2
                  conform_list(acA,listcount) = area

               end if

C*********************************************************************
C*********************   Type SDOWN (B)   ****************************
C*********************************************************************

               if (type.eq.sdown) then
                  if (i.eq.1) then
                     index1 = conform_grid(acNPi,k,i)
                  else
                     index1 = conform_grid(acNPi,k,i-1)
                  end if

                  call enter_exit(k,i-1,index1,ze,re,zx,rx,typels,type)
c                  errorcode = 121
c                  if (typels.ne.parallel) go to 980

                  len1 = (i+0.0) - re
                  len2 = (i+0.0) - rx

                  errorcode = 221
                  if (len1.gt.(2.0).OR.len1.lt.(0.0)) go to 980
                  errorcode = 321
                  if (len2.gt.(2.0).OR.len2.lt.(0.0)) go to 980

                  area = 0.5*(len1+len2)

                  conform_list(acl1,listcount) = len1
                  conform_list(acl2,listcount) = len2
                  conform_list(acA,listcount) = area

               end if

C*********************************************************************
C*********************   Type SLEFT (C)   ****************************
C*********************************************************************

               if (type.eq.sleft) then
                  index1 = conform_grid(acNPi,k-1,i)
                  call enter_exit(k-1,i,index1,ze,re,zx,rx,typels,type)
                  errorcode = 122
                  if (typels.ne.parallel) go to 980

                  len1 = (k+0.5) - ze
                  len2 = (k+0.5) - zx

                  errorcode = 222
                  if (len1.gt.(2.0).OR.len1.lt.(0.0)) go to 980
                  errorcode = 322
                  if (len2.gt.(2.0).OR.len2.lt.(0.0)) go to 980

                  area = 0.5*(len1+len2)

                  conform_list(acl1,listcount) = len1
                  conform_list(acl2,listcount) = len2
                  conform_list(acA,listcount) = area

               end if


C*********************************************************************
C*********************   Type SCRIGHT (D)   **************************
C*********************************************************************

               if (type.eq.scright) then
                  index1 = conform_grid(acNPi,k,i)
                  call enter_exit(k,i,index1,ze,re,zx,rx,typels,type)

                  if (typels.eq.parallel) then
                     conform_list(acz,listcount) = -999

                     len1 = zx - (k-0.5)
                     len2 = ze - (k-0.5)

c                     print *,len1,len2,k,i,typels

                     errorcode = 223
                     if (len1.gt.(1.0).OR.len1.lt.(0.0)) go to 980
                     errorcode = 323
                     if (len2.gt.(1.0).OR.len2.lt.(0.0)) go to 980
                     area = 0.5*(len1+len2)
                  else
                     conform_list(acz,listcount) = -1001

                     len1 = (i+0.0) - rx
                     len2 = ze - (k-0.5)

c                     print *,len1,len2,k,i,typels

                     errorcode = 423
                     if (len1.gt.(1.0).OR.len1.lt.(0.0)) go to 980
                     errorcode = 523
                     if (len2.gt.(1.0).OR.len2.lt.(0.0)) go to 980

                     area = len2+len1*(1-len2)+0.5*(1-len1)*(1-len2)
                     errorcode = 623
                     if (area.gt.(1.0).OR.area.lt.(0.0)) go to 980

                  end if

                  conform_list(acl1,listcount) = len1
                  conform_list(acl2,listcount) = len2
                  conform_list(acA,listcount) = area
               end if


C*********************************************************************
C*********************   Type SCDOWN (E)   ***************************
```

```
C*********************************************************************

               if (type.eq.scdown) then
                  index1 = conform_grid(acNPi,k,i)
c                  print *,index1
                  call enter_exit(k,i,index1,ze,re,zx,rx,typels,type)

c                  errorcode = 124
c                  if (typels.ne.parallel) go to 980

                  len1 = (i+0.0) - re
                  len2 = (i+0.0) - rx

                  errorcode = 224
                  if (len1.gt.(1.0).OR.len1.lt.(0.0)) go to 980
                  errorcode = 324
                  if (len2.gt.(1.0).OR.len2.lt.(0.0)) go to 980

                  area = 0.5*(len1+len2)

                  conform_list(acl1,listcount) = len1
                  conform_list(acl2,listcount) = len2
                  conform_list(acA,listcount) = area
               end if


C*********************************************************************
C*****************   Type SCLEFT (F) / VLEFT (P)  ********************
C*********************************************************************

               if (type.eq.scleft.OR.type.eq.vleft) then
                  index1 = conform_grid(acNPi,k,i)
                  call enter_exit(k,i,index1,ze,re,zx,rx,typels,type)

                  if (typels.eq.parallel) then
                     conform_list(acz,listcount) = -999
                     len1 = (k+0.5) - ze
                     len2 = (k+0.5) - zx

c                     print *,len1,len2,k,i,typels

                     errorcode = 225
                     if (len1.gt.(1.0).OR.len1.lt.(0.0)) go to 980
                     errorcode = 325
                     if (len2.gt.(1.0).OR.len2.lt.(0.0)) go to 980
                     area = 0.5*(len1+len2)
                  else
                     conform_list(acz,listcount) = -1001
                     len1 = (i+0.0) - re
                     len2 = (k+0.5) - zx

c                     print *,len1,len2,k,i,typels

                     errorcode = 425
                     if (len1.gt.(1.0).OR.len1.lt.(0.0)) go to 980
                     errorcode = 525
                     if (len2.gt.(1.0).OR.len2.lt.(0.0)) go to 980

                     area = len2+len1*(1-len2)+0.5*(1-len1)*(1-len2)
                     errorcode = 625
                     if (area.gt.(1.0).OR.area.lt.(0.0)) go to 980
                  end if

                  conform_list(acl1,listcount) = len1
                  conform_list(acl2,listcount) = len2
                  conform_list(acA,listcount) = area
               end if

C*********************************************************************
C*********************   Type SSCRIGHT (G)   *************************
C*********************************************************************

               if (type.eq.sscright) then
                  index1 = conform_grid(acNPi,k,i)
                  index12 = conform_grid(acNPi,k+1,i)

                  call enter_exit(k,i,index1,ze,re,zx,rx,typels,type)
                  call enter_exit(k+1,i,index12,ze2,re2,zx2,rx2,typels2,
     1                 type)

                  errorcode = 126
                  if (abs(zx-ze2).gt.tole.OR.abs(rx-re2).gt.tole) go
     1                 to 980

                  len1 = zx2 - (k-0.5)
                  len2 =  ze - (k-0.5)

                  errorcode = 226
                  if (len1.gt.(2.0).OR.len1.lt.(1.0)) go to 980
                  errorcode = 326
                  if (len2.gt.(1.0).OR.len2.lt.(0.0)) go to 980

                  area = len2*(1-(i-rx)) + (zx-(k-0.5))*(i-rx) +
     1                 0.5*(zx-(k-0.5)-len2)*(1-(i-rx)) +
     2                 0.5*(len1-(zx-(k-0.5)))*(i-rx)

                  errorcode = 426
                  if (area.gt.(2.0).OR.area.lt.(0.0)) go to 980

c                  print *,k,i,len1,len2,area,zx,rx

                  conform_list(acl1,listcount) = len1
                  conform_list(acl2,listcount) = len2
                  conform_list(acA,listcount) = area
                  conform_list(acz,listcount) = zx
                  conform_list(acr,listcount) = rx
               end if


C*********************************************************************
C*********************   Type SSCLEFT (I)   **************************
C*********************************************************************

               if (type.eq.sscleft) then
                  index1 = conform_grid(acNPi,k,i)
                  index12 = conform_grid(acNPi,k-1,i)

                  call enter_exit(k,i,index1,ze,re,zx,rx,typels,type)
                  call enter_exit(k-1,i,index12,ze2,re2,zx2,rx2,typels2,
     1                 type)

                  errorcode = 127
                  if (abs(zx2-ze).gt.tole.OR.abs(rx2-re).gt.tole) go
     1                 to 980

                  len1 = (k+0.5) - ze2
                  len2 = (k+0.5) - zx

                  errorcode = 227
                  if (len1.gt.(2.0).OR.len1.lt.(1.0)) go to 980
                  errorcode = 327
                  if (len2.gt.(1.0).OR.len2.lt.(0.0)) go to 980

                  area = len2*(1-(i-re)) + ((k+0.5)-ze)*(i-re) +
     1                 0.5*(1-(i-re))*((k+0.5)-ze-len2) +
     2                 0.5*(i-re)*(len1-(k+0.5-ze))

                  errorcode = 427
                  if (area.gt.(2.0).OR.area.lt.(0.0)) go to 980

c                  print *,k,i,len1,len2,area,zx,rx
                  conform_list(acl1,listcount) = len1
                  conform_list(acl2,listcount) = len2
                  conform_list(acA,listcount) = area
                  conform_list(acz,listcount) = ze
                  conform_list(acr,listcount) = re
               end if

C*********************************************************************
C*********************   Type SSCDOWN (H)   **************************
C*********************************************************************

               if (type.eq.sscdown) then
                  index1 = conform_grid(acNPi,k,i)
                  index12 = conform_grid(acNPi,k,i-1)

                  errorcode = 128
                  if (index1.ne.index12) go to 980

c                  print *,k,i,index12,index1,conform_grid(actype,k,i-1)
                  call enter_exit(k,i,index1,ze,re,zx,rx,typels,type)
                  call enter_exit(k,i-1,index12,ze2,re2,zx2,rx2,typels2,
     1                 type)

                  slope = (rx-re)/(zx-ze)

                  if (slope.gt.(0.0)) then
                     errorcode = 228
                     if (abs(zx2-ze).gt.tole.OR.abs(rx2-re).gt.tole) go
     1                    to 980

                     len1 = (i+0.0) - re2
                     len2 = (i+0.0) - rx

                     errorcode = 328
                     if (len1.gt.(2.0).OR.len1.lt.(1.0)) go to 980
                     errorcode = 428
                     if (len2.gt.(1.0).OR.len2.lt.(0.0)) go to 980

                     area = 0.5*(len1+len2)

                  else
                     errorcode = 528
c                     print *,ze2, zx, re2, rx
```

```
                  if (abs(ze2-zx).gt.tole.OR.abs(re2-rx).gt.tole) go
     1                 to 980

                  len1 = (i+0.0) - re
                  len2 = (i+0.0) - rx2

                  errorcode = 628
                  if (len1.gt.(1.0).OR.len1.lt.(0.0)) go to 980
                  errorcode = 728
                  if (len2.gt.(2.0).OR.len2.lt.(1.0)) go to 980

                  area = 0.5*(len1+len2)

               end if

               slope = len1 - len2
               zx = (k+0.5) + (len2-1)/slope

               conform_list(acl1,listcount) = len1
               conform_list(acl2,listcount) = len2
               conform_list(acA,listcount) = area
            end if

C*********************************************************************
C*******************    Type SCSDOWNLEFT (L)   ***********************
C*********************************************************************

            if (type.eq.scsdownleft) then
               index1 = conform_grid(acNPi,k,i)
               index12 = conform_grid(acNPi,k-1,i)
               index13 = conform_grid(acNPi,k,i-1)

               errorcode = 129
               if (index1.ne.index13) go to 980

               call enter_exit(k,i,index1,ze,re,zx,rx,typels,type)
               call enter_exit(k-1,i,index12,ze2,re2,zx2,rx2,typels2,
     1                 type)
               call enter_exit(k,i-1,index13,ze3,re3,zx3,rx3,typels3,
     1                 type)

               errorcode = 229
               if (abs(zx2-ze).gt.tole.OR.abs(rx2-re).gt.tole) go
     1                 to 980
               errorcode = 329
               if (abs(zx-ze3).gt.tole.OR.abs(rx-re3).gt.tole) go
     1                 to 980

               len1 = (k+0.5) - ze2
               len2 = (i+0.0) - rx3

               errorcode = 429
               if (len1.gt.(2.0).OR.len1.lt.(1.0)) go to 980
               errorcode = 529
               if (len2.gt.(2.0).OR.len2.lt.(1.0)) go to 980

               area = (i-re) + 0.5*(len2-(i-re)) + 0.5*(i-re)*
     1                 (ze-(k+0.5-len1))

               errorcode = 529
               if (area.gt.(2.0).OR.area.lt.(0.0)) go to 980

               conform_list(acl1,listcount) = len1
               conform_list(acl2,listcount) = len2
               conform_list(acA,listcount) = area
               conform_list(acz,listcount) = ze
               conform_list(acr,listcount) = re
            end if

C*********************************************************************
C*******************    Type SCSDOWNRIGHT (M)   **********************
C*********************************************************************

            if (type.eq.scsdownright) then
               index1 = conform_grid(acNPi,k,i)
               index12 = conform_grid(acNPi,k+1,i)
               index13 = conform_grid(acNPi,k,i-1)

               errorcode = 130
               if (index1.ne.index13) go to 980

               call enter_exit(k,i,index1,ze,re,zx,rx,typels,type)
               call enter_exit(k+1,i,index12,ze2,re2,zx2,rx2,typels2,
     1                 type)
               call enter_exit(k,i-1,index13,ze3,re3,zx3,rx3,typels3,
     1                 type)

               errorcode = 230
               if (abs(zx-ze2).gt.tole.OR.abs(rx-re2).gt.tole) go
     1                 to 980
               errorcode = 330
               if (abs(zx3-ze).gt.tole.OR.abs(rx3-re).gt.tole) go
     1                 to 980

               len1 = zx2 - (k-0.5)
               len2 = (i+0.0) - re3

               errorcode = 430
               if (len1.gt.(2.0).OR.len1.lt.(1.0)) go to 980
               errorcode = 530
               if (len2.gt.(2.0).OR.len2.lt.(1.0)) go to 980

               area = (i-rx) + 0.5*((k-0.5+len1)-zx)*(i-rx)
     1                 + 0.5*(rx-(i-len2))

               errorcode = 630
               if (area.gt.(2.0).OR.area.lt.(0.0)) go to 980

               conform_list(acl1,listcount) = len1
               conform_list(acl2,listcount) = len2
               conform_list(acA,listcount) = area
               conform_list(acz,listcount) = zx
               conform_list(acr,listcount) = rx
            end if


C*********************************************************************
C*******************    Type SDOWNLEFT (J)   *************************
C*********************************************************************

            if (type.eq.sdownleft) then
               index1 = conform_grid(acNPi,k-1,i)
               index12 = conform_grid(acNPi,k-1,i-1)
               index13 = conform_grid(acNPi,k,i-1)

               errorcode = 131
               if (index1.ne.index12) go to 980

               call enter_exit(k-1,i,index1,ze,re,zx,rx,typels,type)
               call enter_exit(k-1,i-1,index12,ze2,re2,zx2,rx2,
     1                 typels2,type)
               call enter_exit(k,i-1,index13,ze3,re3,zx3,rx3,typels3,
     1                 type)

               errorcode = 231
               if (typels.ne.parallel.OR.typels2.ne.perp.OR.typels3.
     1                 ne.parallel) go to 980

               errorcode = 331
               if (abs(ze3-zx2).gt.tole.OR.abs(re3-rx2).gt.tole) go
     1                 to 980

               len1 = (k+0.5) - ze
               len2 = (i+0.0) - rx3

               errorcode = 431
               if (len1.gt.(2.0).OR.len1.lt.(1.0)) go to 980
               errorcode = 531
               if (len2.gt.(2.0).OR.len2.lt.(1.0)) go to 980

               area = 0.5*(len1-1)*(len2-(rx2-rx3)) + 0.5*
     1                 (rx2-rx3) + (len2-(rx2-rx3))

               errorcode = 631
               if (area.gt.(2.0).OR.area.lt.(0.0)) go to 980

               conform_list(acl1,listcount) = len1
               conform_list(acl2,listcount) = len2
               conform_list(acA,listcount) = area
               conform_list(acz,listcount) = ze3
               conform_list(acr,listcount) = re3
            end if

C*********************************************************************
C*******************    Type SDOWNRIGHT (K)   ************************
C*********************************************************************

            if (type.eq.sdownright) then
               index1 = conform_grid(acNPi,k+1,i)
               index12 = conform_grid(acNPi,k+1,i-1)
               index13 = conform_grid(acNPi,k,i-1)

               errorcode = 132
               if (index1.ne.index12) go to 980

               call enter_exit(k+1,i,index1,ze,re,zx,rx,typels,type)
               call enter_exit(k+1,i-1,index12,ze2,re2,zx2,rx2,
     1                 typels2,type)
               call enter_exit(k,i-1,index13,ze3,re3,zx3,rx3,typels3,
     1                 type)

               errorcode = 232
               if (typels.ne.parallel.OR.typels2.ne.perp.OR.typels3.
     1                 ne.parallel) go to 980
```

```
                errorcode = 332
                if (abs(zx3-ze2).gt.tole.OR.abs(rx3-re2).gt.tole) go
     1               to 980

                len1 = zx - (k-0.5)
                len2 = (i+0.0) - re3

                errorcode = 432
                if (len1.gt.(2.0).OR.len1.lt.(1.0)) go to 980
                errorcode = 532
                if (len2.gt.(2.0).OR.len2.lt.(1.0)) go to 980

                area = 0.5*(len1-1)*(len2-(re2-re3)) + 0.5*
     1               (re2-re3) + (len2-(re2-re3))

                errorcode = 632
                if (area.gt.(2.0).OR.area.lt.(0.0)) go to 980

                conform_list(acl1,listcount) = len1
                conform_list(acl2,listcount) = len2
                conform_list(acA,listcount) = area
                conform_list(acz,listcount) = ze2
                conform_list(acr,listcount) = re2
             end if

C*********************************************************************
C**********************   Type VUPRIGHT (U)   ************************
C*********************************************************************

             if (type.eq.vupright) then
                if (i.eq.1) then
                   index1 = conform_grid(acNPi,k+1,i)
                else
                   index1 = conform_grid(acNPi,k+1,i-1)
                end if
c                print *,index1,k,i
                ze = ZB(index1)
                re = RB(index1)
                ze2 = ZB(index1+1)
                re2 = RB(index1+1)

                errorcode = 133
c                print *,re2,re,index1
                if (re2.lt.re) go to 980

                len1 = re2 - (i-1.0)

                errorcode = 233
c                print *,len1,re2,re
c                if (len1.gt.(2.0).OR.len1.lt.(1.0)) go to 980

                area = len1

                conform_list(acl1,listcount) = len1
                conform_list(acA,listcount) = area
             end if

C*********************************************************************
C**********************   Type VUPLEFT (Q)   *************************
C*********************************************************************

             if (type.eq.vupleft) then
                if (i.eq.1) then
                   index1 = conform_grid(acNPi,k,i)
                else
                   index1 = conform_grid(acNPi,k,i-1)
                end if
                ze = ZB(index1)
                re = RB(index1)
                ze2 = ZB(index1-1)
                re2 = RB(index1-1)

c                errorcode = 134
c                print *,re2,re,index1
c                if (re2.lt.re) go to 980

c                print *,ze,re,ze2,re2,k,i
                len1 = re - (i-1.0)

                errorcode = 234
c                print *,len1
                if (len1.gt.(2.0).OR.len1.lt.(1.0)) go to 980

                if (len1.gt.(1.5)) then
                   conform_list(actype,listcount) = vleft
                   conform_list(ack,listcount) = k
                   conform_list(aci,listcount) = i
                   conform_list(acl1,listcount) = 1.0
                   conform_list(acl2,listcount) = 1.0
                   conform_list(acz,listcount) = -999
                   conform_list(acA,listcount) = 1.0
                   listcount = listcount + 1
                   conform_list(actype,listcount) = vupleft
                   conform_list(ack,listcount) = k
                   conform_list(aci,listcount) = i+1
                   conform_list(acl1,listcount) = len1 - 1.0
                   conform_list(acA,listcount) = len1 - 1.0
                else
                   area = len1

                   conform_list(acl1,listcount) = len1
                   conform_list(acA,listcount) = area

                end if

             end if


C*********************************************************************
C**********************   Type VDOWNRIGHT (S)   **********************
C*********************************************************************

             if (type.eq.vdownright) then
                index1 = conform_grid(acNPi,k+1,i)
C                print *,index1,k,i
                ze = ZB(index1)
                re = RB(index1)
                ze2 = ZB(index1+1)
                re2 = RB(index1+1)

c                print *,ze,re

                errorcode = 135
                if (re2.lt.re) go to 980

                len1 = (i+0.0) - re

                errorcode = 235
c                print *,len1,re2,re
                if (len1.gt.(2.0).OR.len1.lt.(0.0)) go to 980

                area = len1

                conform_list(acl1,listcount) = len1
                conform_list(acA,listcount) = area
             end if

C*********************************************************************
C**********************   Type VDOWNLEFT (R)   ***********************
C*********************************************************************

             if (type.eq.vdownleft) then
                index1 = conform_grid(acNPi,k,i)
                ze = ZB(index1)
                re = RB(index1)
                ze2 = ZB(index1-1)
                re2 = RB(index1-1)
c                print *,k,i,index1
c                print *,ze,re,ze2,re2

                errorcode = 136
                if (re2.lt.re) go to 980

                len1 = (i+0.0) - re

                errorcode = 236
                if (len1.gt.(2.0).OR.len1.lt.(0.0)) go to 980

                area = len1

                conform_list(acl1,listcount) = len1
                conform_list(acA,listcount) = area
             end if

          end if
 120     continue
 110  continue

C***  Write out results for display to MATLAB plot for
C***  analysis and verfication.

      do 150 k = 1,mz
         do 160 i = 1,mr
            conform_grid1(k,i) = 0
            ephi_conform1(k,i) = 0
            er_conform1(k,i) = 0
            ez_conform1(k,1) =0
 160     continue
 150  continue


C***  Contour Length and Area Calculated Values
```

```
      ilen = index(dbase,' ') - 1
      write(frmt,'(a2,i4,a5)') '(a',ilen,',a10)'
      write(filnam,frmt)dbase,'/clist.dat'

      open(unit=10,file=filnam,status='unknown',form='formatted')

      do 2010 index1 = 1,listcount
         write(10,*) int(conform_list(ack,index1)),int(conform_list
     1        (aci,index1)),int(conform_list(actype,index1)+64),
     2        conform_list(acl1,index1),conform_list(acl2,index1),
     3        conform_list(acz,index1),conform_list(acr,index1),
     4        borrow_list(ezt,index1),borrow_list(err,index1),
     5        borrow_list(ezb,index1),borrow_list(erl,index1)
     6        ,conform_list(acA,index1)

         if (borrow_list(ezt,index1).ne.NO) ez_conform1(conform_list
     1        (ack,index1),conform_list(aci,index1)+1) = 1

         if (borrow_list(ezb,index1).ne.NO) ez_conform1(conform_list
     1        (ack,index1),conform_list(aci,index1)) = 1

         if (borrow_list(err,index1).ne.NO) er_conform1(conform_list
     1        (ack,index1),conform_list(aci,index1)) = 1

         if (borrow_list(erl,index1).ne.NO) er_conform1(conform_list
     1        (ack,index1)-1,conform_list(aci,index1)) = 1


 2010 continue

      close(unit=10)


C*********************************************************************
C***************   Conformal Hz Geometry Processing   ***************
C*********************************************************************

C***  Variables for conformal Hz field.
C***  using accessk, accessi, accesst

      hzcount = 0

      do 200 index1 = 1,NP
         z1 = ZB(index1)
         r1 = RB(index1)

         if (index1.eq.NP) then
            z2 = ZB(index1-1)
            r2 = RB(index1-1)
            z3 = -999
         else
            z2 = ZB(index1+1)
            r2 = RB(index1+1)
            if (index1.ne.1) then
               z3 = ZB(index1-1)
            else
               z3 = z1
            end if
         end if

         if ((r2.ge.r1.AND.abs(z2-z1).gt.tole).OR.(r2.lt.r1.AND.
     1        abs(z1-z3).gt.tole)) then
            if (abs(z1-z3).gt.tole.OR.abs(r1-maxrb).lt.tole) then
               if (abs(r1-floor(r1)).ge.(0.0)) then
                  i = floor(r1) + 1
                  hzcount = hzcount + 1
                  conform_hz(accessk,hzcount) = floor(z1-0.5)

                  if ((i-0.5).lt.r1) then
                     conform_hz(accessi,hzcount) = i + 1
                     conform_hz(accesst,hzcount) = STRETCH_HZ
                     conform_hz_length(hzcount) = i + 1 - r1
                  else
                     conform_hz(accessi,hzcount) = i
                     conform_hz(accesst,hzcount) = SC_HZ
                     conform_hz_length(hzcount) = i - r1
                  end if

                  if (abs(z1-z2).lt.tole) then
c*****************    A vertical line.
c                     print *,'testing',z1,r1,z2,r2

                     do 220 i = floor(max(r1,r2))+1,floor(min(r1,r2))
     1                    +1,-1
                        hzcount = hzcount + 1

                        conform_hz(accessk,hzcount) = floor(z1-0.5)
                        conform_hz(accessi,hzcount) = i
                        conform_hz(accesst,hzcount) = EQZERO_HZ
 220                 continue
                  end if

               end if

            end if

         else
C********   Vertical Line
            if (abs(z1-z2).lt.tole) then
               do 210 i = floor(max(r1,r2))+1,floor(min(r1,r2))+1,-1
                  hzcount = hzcount + 1
c                  if (int(z1-0.5).eq.109.AND.i.eq.70)
c     1                 print *,'test',r1,r2,z1,z2
                  conform_hz(accessk,hzcount) = floor(z1-0.5)
                  conform_hz(accessi,hzcount) = i
                  conform_hz(accesst,hzcount) = EQZERO_HZ
 210           continue
            end if

         end if
 200  continue


C***  Write out hz conformal grid information.

      do 250 k = 1,mz
         do 260 i = 1,mr
            conform_hz1(k,i) = 0
 260     continue
 250  continue

      ilen = index(dbase,' ') - 1
      write(frmt,'(a2,i4,a4)') '(a',ilen,',a9)'
      write(filnam,frmt)dbase,'/cghz.dat'

      open(unit=10,file=filnam,status='unknown',form='formatted')

      do 3000 index1 = 1,hzcount

         write(10,*) conform_hz(accessk,index1), conform_hz(accessi,
     1        index1), conform_hz(accesst,index1), conform_hz_length
     2        (index1)

c         if (conform_hz(accesst,index1).eq.STRETCH_HZ.OR.
c     1        conform_hz(accesst,index1).eq.SC_HZ.OR.
c     2        conform_hz(accesst,index1).eq.EQZERO_HZ) then

         if (conform_hz(accesst,index1).eq.STRETCH_HZ.OR.
     1        conform_hz(accesst,index1).eq.SC_HZ) then

            conform_hz1(conform_hz(accessk,index1),conform_hz(
     1           accessi,index1)) = 1
         end if
 3000 continue

      close(unit=10)

C*********************************************************************
C***************   Conformal Hr Geometry Processing   ***************
C*********************************************************************

      hrcount = 0
      ephicount = 0

      do 600 k = 1,mz
         do 660 i = 1,mr
            temp_hz(k,i) = 0
 660     continue
 600  continue


      do 300 index1 = 1,NP-1
         k = ZB(index1)+0.5
         z1 = ZB(index1)
         z2 = ZB(index1+1)
         r1 = RB(index1)
         r2 = RB(index1+1)

         if (abs(z1-z2).lt.tole.OR.abs(r1-r2).lt.tole) then
C********   Horizontal or vertical line.

            if (abs(z1-z2).lt.tole) then
C***********   Vertical line

               do 320 i = floor(min(r1,r2)),floor(max(r1,r2))+1
                  if (i.ge.min(r1,r2).AND.i.le.max(r1,r2)) then

                     if (r2.gt.r1) then
                        if (temp_hz(int(z1-0.5),i+1).eq.NO) then
c                        if (on_pec(int(z1-0.5),i+1,ephif)) then
                           hrcount = hrcount + 1
                           conform_hr(accessk,hrcount) = int(z1-0.5)
                           conform_hr(accessi,hrcount) = i + 1
                           conform_hr(accesst,hrcount) = SRIGHT_HR
```

```
                        conform_hr_length(hrcount) = 1.0
                        temp_hz(int(z1-0.5),i+1) = YES
                     end if
                  else
                     if (temp_hz(int(z1+0.5),i+1).eq.NO) then
                        hrcount = hrcount + 1
                        conform_hr(accessk,hrcount) = int(z1+0.5)
                        conform_hr(accessi,hrcount) = i + 1
                        conform_hr(accesst,hrcount) = SLEFT_HR
                        conform_hr_length(hrcount) = 1.0
                        temp_hz(int(z1+0.5),i+1) = YES
                     end if
                  end if
               end if
 320           continue

            else
C***********    Horizontal line
               if (temp_hz(k,round(r2)+1).eq.NO) then
                  hrcount = hrcount + 1
                  conform_hr(accessk,hrcount) = k
                  conform_hr(accessi,hrcount) = round(r2) + 1
                  conform_hr(accesst,hrcount) = EQZERO_HR
                  conform_hr_length(hrcount) = -1000
                  temp_hz(k,round(r2)+1) = YES
               end if
            end if
         else
C******   All other slopes.
            do 310 i = floor(min(r1,r2)),floor(max(r1,r2))+1
               if (i.ge.min(r1,r2).AND.i.le.max(r1,r2)) then
C***************  Calculate the k point of intersection.
                  slope = (r2-r1+0.0)/(z2-z1+0.0)
                  ze = (i-r2)/slope + z2

                  if (slope.gt.(0.0)) then
                     if (k.gt.ze) then
                        if (temp_hz(k-1,i+1).eq.NO) then
                           hrcount = hrcount + 1
                           conform_hr(accessk,hrcount) = k - 1
                           conform_hr(accessi,hrcount) = i + 1
                           conform_hr(accesst,hrcount) = SRIGHT_HR_DC
                           len1 = ze - (k-1.5)
                           errorcode = 901
                           if (len1.lt.(1.0).OR.len1.gt.(2.0)) go to 980
                           conform_hr_length(hrcount) = len1
                           if (.not.inside_pec(k-1,i+1,ephif)) then
                              ephi_conform1(k-1,i+1) = YES
                              ephicount = ephicount + 1
                              conform_ephi(accessk,ephicount) = k-1
                              conform_ephi(accessi,ephicount) = i+1
                              conform_ephi(accesst,ephicount) = hrcount
                           else
c                             print *, k-1,i+1
                           end if
                           temp_hz(k-1,i+1) = YES
                        end if
                     else
                        if (temp_hz(k,i+1).eq.NO) then
                           hrcount = hrcount + 1
                           conform_hr(accessk,hrcount) = k
                           conform_hr(accessi,hrcount) = i + 1
                           conform_hr(accesst,hrcount) = SRIGHT_HR_IC
                           len1 = ze - (k-0.5)
                           errorcode = 902
                           if (len1.lt.(0.0).OR.len1.gt.(1.0)) go to 980
                           conform_hr_length(hrcount) = len1
                           if (.not.inside_pec(k-1,i+1,ephif).AND.
     1                          (inside_pec(k,i+1,hrf).OR.inside_pec
     2                          (k-1,i,hzf))) then
                              ephi_conform1(k-1,i+1) = YES
                              ephicount = ephicount + 1
                              conform_ephi(accessk,ephicount) = k-1
                              conform_ephi(accessi,ephicount) = i+1
                              conform_ephi(accesst,ephicount) = hrcount
                           else
c                             print *,k,i+1
                           end if
                           temp_hz(k,i+1) = YES
                        end if
                     end if
                  else
                     if (ze.gt.k) then
                        if (temp_hz(k+1,i+1).eq.NO) then
                           hrcount = hrcount + 1
                           conform_hr(accessk,hrcount) = k + 1
                           conform_hr(accessi,hrcount) = i + 1
                           conform_hr(accesst,hrcount) = SLEFT_HR_DC
                           len1 = (k+1.5) - ze
                           errorcode = 1032
                           if (len1.lt.(1.0).OR.len1.gt.(2.0)) go to 980
                           conform_hr_length(hrcount) = len1
                           if (.not.inside_pec(k,i+1,ephif)) then
                              ephi_conform1(k,i+1) = YES
                              ephicount = ephicount + 1
                              conform_ephi(accessk,ephicount) = k
                              conform_ephi(accessi,ephicount) = i+1
                              conform_ephi(accesst,ephicount) = hrcount
                           end if
                           temp_hz(k+1,i+1) = YES
                        end if
                     else
                        if (temp_hz(k,i+1).eq.NO) then
                           hrcount = hrcount + 1
                           conform_hr(accessk,hrcount) = k
                           conform_hr(accessi,hrcount) = i + 1
                           conform_hr(accesst,hrcount) = SLEFT_HR_IC
                           len1 = (k+0.5) - ze
                           errorcode = 1033
                           if (len1.lt.(0.0).OR.len1.gt.(1.0)) go to 980
                           conform_hr_length(hrcount) = len1
                           if (.not.inside_pec(k,i+1,ephif)) then
                              ephi_conform1(k,i+1) = YES
                              ephicount = ephicount + 1
                              conform_ephi(accessk,ephicount) = k
                              conform_ephi(accessi,ephicount) = i+1
                              conform_ephi(accesst,ephicount) = hrcount
                           end if
                           temp_hz(k,i+1) = YES
                        end if
                     end if
                  end if
               end if
 310        continue

            end if
 300  continue

C***  Write out hr conformal grid information.

      do 350 k = 1,mz
         do 360 i = 1,mr
            conform_hr1(k,i) = 0
 360     continue
 350  continue

      ilen = index(dbase,' ') - 1
      write(frmt,'(a2,i4,a4)') '(a',ilen,',a9)'
      write(filnam,frmt)dbase,'/cghr.dat'

      open(unit=10,file=filnam,status='unknown',form='formatted')

      do 4000 index1 = 1,hrcount

         write(10,*) conform_hr(accessk,index1), conform_hr(accessi,
     1        index1), conform_hr(accesst,index1), conform_hr_length
     2        (index1)

c         if (conform_hr(accesst,index1).eq.SRIGHT_HR.OR.conform_hr
c     1        (accesst,index1).eq.SLEFT_HR.OR.conform_hr(accesst,
c     2        index1).eq.EQZERO_HR.OR.conform_hr(accesst,index1).eq
c     3        .SLEFT_HR_IC.OR.conform_hr(accesst,index1).eq.
c     4        SRIGHT_HR_IC.OR.conform_hr(accesst,index1).eq
c     3        .SLEFT_HR_DC.OR.conform_hr(accesst,index1).eq.
c     4        SRIGHT_HR_DC) then


         if (conform_hr(accesst,index1).eq.SRIGHT_HR.OR.conform_hr
     1        (accesst,index1).eq.SLEFT_HR.OR.conform_hr(accesst,
     2        index1).eq.SLEFT_HR_IC.OR.conform_hr(accesst,index1).eq.
     3        SRIGHT_HR_IC.OR.conform_hr(accesst,index1).eq
     4        .SLEFT_HR_DC.OR.conform_hr(accesst,index1).eq.
     5        SRIGHT_HR_DC) then

            conform_hr1(conform_hr(accessk,index1),conform_hr(
     1           accessi,index1)) = 1
         end if
 4000 continue

      close(unit=10)

      ilen = index(dbase,' ') - 1
      write(frmt,'(a2,i4,a4)') '(a',ilen,',a9)'
      write(filnam,frmt)dbase,'/ephi.dat'

      open(unit=10,file=filnam,status='unknown',form='formatted')
      do 4500 k=1,mz
         do 4600 i=1,mr
            if (ephi_conform1(k,i).eq.YES) write(10,*) k,i
 4600    continue
 4500 continue
      close(unit=10)

      ilen = index(dbase,' ') - 1
```

```
      write(frmt,'(a2,i4,a5)') '(a',ilen,',a10)'
      write(filnam,frmt)dbase,'/ephi.dat'

      open(unit=10,file=filnam,status='unknown',form='formatted')
      do 4505 k=1,ephicount
         write(10,*) conform_ephi(accessk,k),conform_ephi(accessi,k),
     1        conform_hr(accesst,conform_ephi(accesst,k)),
     1        conform_hr_length(conform_ephi(accesst,k))
 4505 continue
      close(unit=10)

      ilen = index(dbase,' ') - 1
      write(frmt,'(a2,i4,a5)') '(a',ilen,',a10)'
      write(filnam,frmt)dbase,'/zerof.dat'

      if (hrcount.gt.MAXCP) then
         print *,'not enough memory for hrcount=',hrcount
         enough_memory = .FALSE.
      end if
      if (hzcount.gt.MAXCP) then
         print *,'not enough memory for hzcount',hzcount
         enough_memory = .FALSE.
      end if
      if (ephicount.gt.MAXCP) then
         print *,'not enough memory for ephicount',ephicount
         enough_memory = .FALSE.
      end if
      if (listcount.gt.MAXCP) then
         print *,'not enough memory for listcount',listcount
         enough_memory = .FALSE.
      end if

c**** dummy statement to jump to if staircase only
 1200 k=k

      return

 980  print *, 'Error ', errorcode, ' in reading BOR INPUT at '
      print *, k, i, ZBa(i), RBa(i), ZBa(i+1), RBa(i+1)

c      stop
      end

C**********************************************************************
C***************     Enforce Stair Case Boundary Cond    **************
C**********************************************************************

      subroutine staircase_approx

      implicit none
      include 'common.f'
      integer index1, x1, x2, x3, y1, y2, y3, ilen

c      character*72 dbase
      character filnam*1024, frmt*30

      if (time.eq.1) then
         ilen = index(dbase,' ') - 1
         write(frmt,'(a2,i4,a4)') '(a',ilen,',a7)'

         write(filnam,frmt)dbase,'/er.dat'
         open(unit=1,file=filnam,status='unknown',form='formatted')

         write(filnam,frmt)dbase,'/ez.dat'
         open(unit=3,file=filnam,status='unknown',form='formatted')
      end if

      do 20 index1 = 2,staircount
         if (staircase(aci,index1).gt.0) ephi(staircase(ack,index1),
     1        staircase(aci,index1)) = 0.0
 20   continue

      er(staircase(ack,2),staircase(aci,2)) = 0.0
      if (time.eq.1) write(1,*) staircase(ack,2), staircase(aci,2)

      er(staircase(ack,3),staircase(aci,3)) = 0.0
      if (time.eq.1) write(1,*) staircase(ack,3), staircase(aci,3)

      do 10 index1 = 3,staircount-1
         x1 = staircase(ack,index1)
         y1 = staircase(aci,index1)
         x2 = staircase(ack,index1+1)
         y2 = staircase(aci,index1+1)
         x3 = staircase(ack,index1+2)
         y3 = staircase(aci,index1+2)

         if (y2.gt.0) then
            if (y1.eq.y2.AND.y2.eq.y3) then
               if (y2.gt.0) then
                  ez(x2,y2) = 0.0
                  if (time.eq.1) write(3,*) x2, y2
               end if
            else
               if (x1.eq.x2.AND.x2.eq.x3) then
                  if (y2.gt.0) then
                     er(x2,y2) = 0.0
                     if (time.eq.1) write(1,*) x2, y2
                  end if
               else
                  if (x1.eq.x2) then
                     if (y1.gt.y2) then
                        if (y2.gt.0) then
                           er(x2,y2) = 0.0
                           if (time.eq.1) write(1,*) x2, y2
                        end if
                     end if
                  else
                     if (x2.eq.x3) then
                        if (y2.gt.y3) then
                           if (y2.gt.0) then
                              ez(x2,y2) = 0.0
                              if (time.eq.1) write(3,*) x2, y2
                           end if
                        else
                           if (y2.gt.0) then
                              ez(x2,y2) = 0.0
                              if (time.eq.1) write(3,*) x2, y2
                              er(x2,y2) = 0.0
                              if (time.eq.1) write(1,*) x2, y2
                           end if
                        end if
                     else
c                        print *,'error at index1 =', index1
                     end if
                  end if
               end if
            end if
         end if
 10   continue

c      if (staircase(aci,staircount).gt.0) then
c         er(staircase(ack,staircount),staircase(aci,staircount)) = 0.0
c         if (time.eq.1) write(1,*) staircase(ack,staircount),
c     1        staircase(aci,staircount)
c      end if

      if (time.eq.1) then
         close(unit=1)
         close(unit=3)
      end if

      return
      end


C**********************************************************************
C*****************    Enforce Boundary Conditions    *****************
C**********************************************************************

      subroutine boundary_conditions(m,ms)

      implicit none
      include 'common.f'
      integer i, k, index1, type1s, m, ms, type1
      real len1, len2, area, ez_top, ez_bot, er_rt, er_lt, c1, c2, c3
      character type
      integer borezt, borezb, borerl, borerr
      logical borw_direc_rightQ


      do 41 index1 = 1,hzcount
         k=conform_hz(accessk,index1)
         i=conform_hz(accessi,index1)
         type1=conform_hz(accesst,index1)
         len1=dz*conform_hz_length(index1)

         if (type1.eq.STRETCH_HZ.OR.type1.eq.SC_HZ) then

            c1 = (i+0.0)*dz*dt/mu
            c2 = len1*dz*(i+0.0)-0.5*len1*len1
            c3 = len1*m*dt/mu

c** see if ephi needs to be interpolated.
            if (len1.gt.(dz-dz/5)) then
               ephi(k,i) = ephi(k,i+1)*(len1-1.0*dz)/len1
            end if

         end if
 41      continue


C**   Do the calculations for the interpolated ephi fields.

      do 40 index1 = 1,ephicount
```

```
      k = conform_ephi(accessk,index1)
      i = conform_ephi(accessi,index1)
      len1 = conform_hr_length(conform_ephi(accesst,index1))
      typels = conform_hr(accesst,conform_ephi(accesst,index1))
      if (len1.gt.(2.0).OR.len1.lt.(0.0)) then
         print *,'error, in len for interpolating ephi'
         stop
      end if

      if (typels.eq.SRIGHT_HR_DC) then
         ephi(k,i) = ephi(k-1,i)*(1.0/(len1))*(len1-1.0)
      end if

      if (typels.eq.SRIGHT_HR_IC) then
         ephi(k,i) = ephi(k-1,i)*(len1/(len1+1.0))
      end if

      if (typels.eq.SLEFT_HR_DC) then
         ephi(k,i) = ephi(k+1,i)*(1.0/(len1))*(len1-1.0)
      end if

      if (typels.eq.SLEFT_HR_IC) then
         ephi(k,i) = ephi(k+1,i)*(len1/(len1+1.0))
      end if

      if (typels.ne.SLEFT_HR_DC.AND.typels.ne.SRIGHT_HR_DC.AND.
     1       typels.ne.SLEFT_HR_IC.AND.SRIGHT_HR_IC) then
         print *,'error in typing conformal ephi cell'
         stop
      end if

 40   continue


C***  Implement boundary conditions for hphi term.

      do 10 index1 = 1,listcount

         k = int(conform_list(ack,index1))
         i = int(conform_list(aci,index1))
         type = char(int(conform_list(actype,index1)+64))
         len1 = dz*conform_list(acl1,index1)
         len2 = dz*conform_list(acl2,index1)
         area = dz*dz*conform_list(acA,index1)
         typels = conform_list(acz,index1)
         borezt = borrow_list(ezt,index1)
         borezb = borrow_list(ezb,index1)
         borerl = borrow_list(erl,index1)
         borerr = borrow_list(err,index1)

         borw_direc_rightQ = type.eq.'C'.OR.type.eq.'F'.OR.type.eq.'I'.
     1        OR.type.eq.'J'.OR.type.eq.'L'.OR.type.eq.'P'.OR.type.
     2        eq.'Q'.OR.type.eq.'R'
c         print *, borw_direc_rightQ, ' ', type

         if (borezt.ne.NO) then
            if (borezt.eq.YES_RIGHT) then
               ez_top = ez(k+1,i+1)
            else
               ez_top = ez(k-1,i+1)
            end if
         else
            ez_top = ez(k,i+1)
         end if

         if (borezb.ne.NO) then
            if (borezb.eq.YES_RIGHT) then
               ez_bot = ez(k+1,i)
            else
               ez_bot = ez(k-1,i)
            end if
         else
            ez_bot = ez(k,i)
         end if


         if (borerr.eq.YES) then
            er_rt = er(k,i+1)
         else
            er_rt = er(k,i)
         end if
         er(k,i) = er_rt

         if (borerl.eq.YES) then
            er_lt = er(k-1,i+1)
         else
            er_lt = er(k-1,i)
         end if

         if (type.eq.'A'.OR.type.eq.'N') then
            ez_top = len1*ez_top
            ez_bot = len2*ez_bot
            er_lt = dz*er_lt
            er_rt = 0
         end if

         if (type.eq.'B') then
            ez_top = dz*ez_top
            ez_bot = 0
            er_lt = len1*er_lt
            er_rt = len2*er_rt
         end if

         if (type.eq.'C'.OR.type.eq.'P') then
            ez_top = len1*ez_top
            ez_bot = len2*ez_bot
            er_lt = 0
            er_rt = dz*er_rt
         end if

         if (type.eq.'D'.AND.abs(typels+999).lt.tole) then
            ez_top = len1*ez_top
            ez_bot = len2*ez_bot
            er_lt = dz*er_lt
            er_rt = 0
         end if

         if (type.eq.'D'.AND.abs(typels+1001).lt.tole) then
            er_rt = len1*er_rt
            ez_bot = len2*ez_bot
            ez_top = dz*ez_top
            er_lt = dz*er_lt
         end if

         if (type.eq.'F'.AND.abs(typels+999).lt.tole) then
            ez_top = len1*ez_top
            ez_bot = len2*ez_bot
            er_lt = 0
            er_rt = dz*er_rt
         end if

         if (type.eq.'F'.AND.abs(typels+1001).lt.tole) then
            er_lt = len1*er_lt
            ez_bot = len2*ez_bot
            ez_top = dz*ez_top
            er_rt = dz*er_rt
         end if

         if (type.eq.'H'.OR.type.eq.'E') then
            er_lt = len1*er_lt
            er_rt = len2*er_rt
            ez_top = dz*ez_top
            ez_bot = 0
         end if

         if (type.eq.'G') then
            ez_bot = len2*ez_bot
            ez_top = len1*ez_top
            er_lt = dz*er_lt
            er_rt = 0
         end if

         if (type.eq.'I') then
            ez_bot = len2*ez_bot
            ez_top = len1*ez_top
            er_rt = dz*er_rt
            er_lt = 0
         end if

         if (type.eq.'J'.OR.type.eq.'L') then
            ez_top = len1*ez_top
            ez_bot = 0
            er_rt = len2*er_rt
            er_lt = 0
         end if

         if (type.eq.'K'.OR.type.eq.'M') then
            ez_top = len1*ez_top
            ez_bot = 0
            er_rt = 0
            er_lt = len2*er_lt
         end if

         if (type.eq.'O') then
            ez_top = dz*ez_top
            ez_bot = dz*ez_bot
            er_lt = len1*er_lt
            er_rt = 0
         end if

         if (type.eq.'Q') then
            ez_top = dz*ez_top
            ez_bot = dz*ez_bot
```

```
            er_rt = len1*er_rt
            er_lt = 0
         end if

         if (type.eq.'R') then
            ez_top = dz*ez_top
            ez_bot = 0
            er_lt = 0
            er_rt = len1*er_rt
         end if

         if (type.eq.'S') then
            ez_top = dz*ez_top
            ez_bot = 0
            er_lt = len1*er_lt
            er_rt = 0
         end if

         hphi(k,i) = hphi(k,i) + eta*((dt/(mu*area))*((ez_top-ez_bot)+
     1        (er_lt-er_rt)))

         ez(k,i+1) = ez_top
         ez(k,i) = ez_bot
         er(k-1,i) = er_lt


 10   continue

C**************Hz field boundary conditions ******************

      do 20 index1 = 1,hzcount
         k=conform_hz(accessk,index1)
         i=conform_hz(accessi,index1)
         type1=conform_hz(accesst,index1)
         len1=dz*conform_hz_length(index1)

c         if (type1.eq.EQZERO_HZ) then
c            print *,'settting to zero',k,i
c            er(k,i)   = 0.0
c            ephi(k,i) = 0.0
c            hz(k,i) = 0.0
c         end if

         if (type1.eq.STRETCH_HZ.OR.type1.eq.SC_HZ) then

            c1 = (i+0.0)*dz*dt/mu
            c2 = len1*dz*(i+0.0)-0.5*len1*len1
            c3 = len1*m*dt/mu

            hz(k,i) = hz(k,i)-eta*((c1/c2)*ephi(k,i+1)-
     1           ms*(c3/c2)*er(k,i))

         end if
 20   continue


C********* Hr field boundary conditions ***********

      do 30 index1 = 1,hrcount
         k = conform_hr(accessk,index1)
         i = conform_hr(accessi,index1)

         type1=conform_hr(accesst,index1)
         len1=dz*conform_hr_length(index1)

c         if (type1.eq.EQZERO_HR) then
c            print *,'settting to zero',k,i
c            hr(k,i) = 0.0
c         end if

         if (type1.eq.SRIGHT_HR_IC.OR.type1.eq.SRIGHT_HR_DC.OR.
     1        type1.eq.SRIGHT_HR) then
            if (i.ne.1) then
               c1 = (m*dt/mu)/((i-1.0)*dz)
               c2 = dt/(mu*len1)

               hr(k,i) = hr(k,i) + eta*(-ms*c1*ez(k,i) - c2*ephi(k-1,i))
            else
c              print *,k,i
               if (m.eq.1) then
                  c1 = (m*dt)/(mu*dz)
                  c2 = dt/(mu*len1)

                  hr(k,i) = hr(k,i) + eta*(-ms*c1*ez(k,i+1) -
     1                 c2*ephi(k-1,i))
               else
                  hr(k,i) = 0.0
               end if
            end if
         end if

         if (type1.eq.SLEFT_HR_IC.OR.type1.eq.SLEFT_HR_DC.OR.
     1        type1.eq.SLEFT_HR) then
            if (i.ne.1) then
               c1 = (m*dt/mu)/((i-1.0)*dz)
               c2 = dt/(mu*len1)

               hr(k,i) = hr(k,i) + eta*(-ms*c1*ez(k,i) + c2*ephi(k,i))
            else
               if (m.eq.1) then
                  c1 = (m*dt)/(mu*dz)
                  c2 = dt/(mu*len1)
                  hr(k,i) = hr(k,i) + eta*(-ms*c1*ez(k,i+1) +
     1                 c2*ephi(k,i))
               else
                  hr(k,i) = 0.0
               end if
            end if
         end if

 30   continue

      return
      end

c**********************************************************************
c SETUP_STAIRCASE setups all the parameters needed to run the simulation
c    including a staircasing algorithm for representing the target.
c**********************************************************************

      SUBROUTINE setup_staircase

      implicit none
      include 'common.f'

      real xstair(1:MAX_STAIR_NODES),
     1     ystair(1:MAX_STAIR_NODES), index, xcomp, ycomp,
     1     dx, dy

      real zstep, radius

      integer xdir, ydir, x1, x2, y1, y2, round, defaults

      real max_x_node, max_y_node, min_x_node, min_y_node,
     1     slope, offset, dist_to_line, xnodes(1:MAX_NODES),
     2     ynodes(1:MAX_NODES), delta, current_x, current_y

      write(6,*) 'Setting up geometry...'

      write(6,'(''*Accept spacing defaults [Y=1,N=2]: '',$)')
      read(5,*) defaults

      if (defaults.eq.1) then
         xtot_sp=10
         ytot_sp=10

         xscat_sp=15
         yscat_sp=15

         xhuy_sp=2
         yhuy_sp=2

         xall_sp = xtot_sp+xscat_sp
         yall_sp = ytot_sp+yscat_sp
      else
         write(6,'(''*Enter xtot_sp [10]: '',$)')
         read(5,*) xtot_sp
         write(6,'(''*Enter ytot_sp [10]: '',$)')
         read(5,*) ytot_sp

         write(6,'(''*Enter xscat_sp [15]: '',$)')
         read(5,*) xscat_sp
         write(6,'(''*Enter yscat_sp [15]: '',$)')
         read(5,*) yscat_sp

         write(6,'(''*Enter xhuy_sp [2]: '',$)')
         read(5,*) xhuy_sp
         write(6,'(''*Enter yhuy_sp [2]: '',$)')
         read(5,*) yhuy_sp

         xall_sp = xtot_sp+xscat_sp
         yall_sp = ytot_sp+yscat_sp
      end if

      errorcount = 0

c**** Read geometry file in.

      open(unit=10,file=fnamein,status='unknown',form='formatted')

      read(10,*) dz
      delta = dz
```

```
      read(10,*) total_nodes
      NP = total_nodes

      if (NP.gt.0) then

      if (total_nodes.gt.MAX_NODES) then
         errorcount = errorcount+1
         errors(errorcount) = NODE_ERROR
         call memory_check
      end if

      do 10 index=1,total_nodes
         read(10,*) xnodes(index), ynodes(index)
 10   continue
c      do 15 index=1,total_nodes
c         read(10,*) xnodes(index)
c 15   continue

      else
c*****object is a sphere
         read(10,*) radius
         radius = dz*round(radius/dz)
         NP = 0
         do 341 zstep = 0.0,2*radius+dz/2,dz
            NP = NP+1
            xnodes(NP) = zstep
            ynodes(NP) = sqrt(radius**2.0-(zstep-radius)**2.0)
 341     continue
         total_nodes = NP
      end if

      close(unit=10)

C**** Scale, position, and round object

      max_x_node = xnodes(1)/delta
      max_y_node = ynodes(1)/delta
      min_x_node = xnodes(1)/delta
      min_y_node = ynodes(1)/delta

      do 20 index=1,total_nodes
         xnodes(index) = xnodes(index)/delta
         if (xnodes(index).gt.max_x_node) max_x_node=xnodes(index)
         if (xnodes(index).lt.min_x_node) min_x_node=xnodes(index)
         ynodes(index) = ynodes(index)/delta
c         print *,xnodes(index),ynodes(index)
         if (ynodes(index).gt.max_y_node) max_y_node=ynodes(index)
         if (ynodes(index).lt.min_y_node) min_y_node=ynodes(index)
 20   continue

      maxz = round(2.0*xall_sp + max_x_node - min_x_node)
      maxr = round(yall_sp + max_y_node - min_y_node)
      len = delta*(max_x_node - min_x_node)
      obj_height = delta*(max_y_node - min_y_node)

      if (maxz.gt.MAX_Z_CELLS) then
         errorcount = errorcount + 1
         errors(errorcount) = MAX_Z_ERROR
      end if

      if (maxr.gt.MAX_R_CELLS) then
         errorcount = errorcount+1
         errors(errorcount) = MAX_R_ERROR
      end if

      do 30 index=1,total_nodes
         print *,xnodes(index),ynodes(index), min_x_node, min_y_node,
     1        xall_sp
         xnodes(index) = round(xnodes(index) - min_x_node)+xall_sp+0.5
         ynodes(index) = round(ynodes(index) - min_y_node) + 1
         print *,xnodes(index),ynodes(index)
         RB(index) = ynodes(index)
         ZB(index) = xnodes(index)
c         print *,xnodes(index),ynodes(index), ZB(index), RB(index)
 30   continue

c**** Estimate total number of staircase nodes needed.
      dx = 0
      dy = 0
      do 500 index=1,total_nodes-1
         dx = dx + int(abs(xnodes(index)-xnodes(index+1)))
         dy = dy + int(abs(ynodes(index)-ynodes(index+1)))
 500  continue
c**** extra point needed for first point
      dy=dy+1

      if (2*(dx+dy)-1.gt.MAX_STAIR_NODES) then
         stair_node_count = 2*(dx+dy)-1
         errorcount = errorcount+1
         errors(errorcount) = MAX_STAIR_ERROR
      end if

c**** define some corner points used for referencing
      x1=50-5
      y1=1
      x2=maxz-x1
      y2=int(max_y_node)+5
      if ((2*(y2-y1)+(x2-x1)+1).gt.MAX_RCS_NODES) then
         errorcount = errorcount+1
         errors(errorcount) = MAX_RCS_ERROR
      end if
      call memory_check


c**** Generate a staircase model by digitizing each line segment.

      stair_node_count = 1
      xstair(stair_node_count) = xnodes(1)
      ystair(stair_node_count) = ynodes(1)

      do 40 index=1,total_nodes-1
         slope = (ynodes(index+1)-ynodes(index))/(xnodes(index+1)-
     1        xnodes(index))
         offset = ynodes(index)-slope*(xnodes(index))

         current_x = xnodes(index)
         current_y = ynodes(index)

 100     stair_node_count = stair_node_count

         if (abs(current_x-xnodes(index+1)).gt.tole.OR.
     1       abs(current_y-ynodes(index+1)).gt.tole) then

            xcomp = xnodes(index+1)-current_x
            ycomp = ynodes(index+1)-current_y

            if (xcomp.ne.0) then
               xdir = int(abs(xcomp)/xcomp)
            else
               xdir = 0
            end if
            if (ycomp.ne.0) then
               ydir = int(abs(ycomp)/ycomp)
            else
               ydir = 0
            end if

            if (stair_node_count.eq.1) then
               ydir = 1
               xdir = 0
            end if

            stair_node_count = stair_node_count + 1

            if (xdir.ne.0.AND.ydir.ne.0) then
               if (dist_to_line(-slope,1.0,offset,real(current_x+xdir),
     1              real(current_y)).lt.dist_to_line(-slope,1.0,offset,
     2              real(current_x),real(current_y+ydir))) then

                  xstair(stair_node_count) = current_x+xdir
                  ystair(stair_node_count) = current_y
                  current_x = current_x+xdir
               else
                  xstair(stair_node_count) = current_x
                  ystair(stair_node_count) = current_y+ydir
                  current_y = current_y+ydir
               end if
            else
               xstair(stair_node_count) = current_x+xdir
               ystair(stair_node_count) = current_y+ydir
               current_x = current_x+xdir
               current_y = current_y+ydir
            end if

            goto 100
         end if
 40   continue

      if ((dx+dy).ne.stair_node_count) then
         write(6,*) 'estimate = ', dx+dy
         write(6,*) 'actual = ', stair_node_count
      end if

c**** print out raw stair case model.
c      open(unit=10,file='rawstair.dat',status='unknown',
c     1     form='formatted')
c      do 1001 index=1,stair_node_count
c         write(10,*) xstair(index), ystair(index)
c 1001 continue
c      close(unit=10)

c**** now figure out which fields to set to zero.

c      print *,'stair_node_count = ',stair_node_count
```

```
      staircount = 1
      do 90 index = 1,stair_node_count-1
         xcomp = xstair(index+1)-xstair(index)
         ycomp = ystair(index+1)-ystair(index)
         if (ycomp.gt.0.AND.xcomp.lt.tole) then
            stair_zero(staircount,1) = int(xstair(index))
            stair_zero(staircount,2) = int(ystair(index))
            stair_zero(staircount,3) = ephif
            staircount = staircount+1
            stair_zero(staircount,1) = int(xstair(index))
            stair_zero(staircount,2) = int(ystair(index))
            stair_zero(staircount,3) = erf
            staircount = staircount+1
         else if (ycomp.lt.0.AND.xcomp.lt.tole) then
            stair_zero(staircount,1) = int(xstair(index))
            stair_zero(staircount,2) = int(ystair(index))
            stair_zero(staircount,3) = ephif
            staircount = staircount+1
            stair_zero(staircount,1) = int(xstair(index))
            stair_zero(staircount,2) = int(ystair(index))-1
            stair_zero(staircount,3) = erf
            staircount = staircount+1
         else if (xcomp.gt.0.AND.ycomp.lt.tole) then
            stair_zero(staircount,1) = int(xstair(index))
            stair_zero(staircount,2) = int(ystair(index))
            stair_zero(staircount,3) = ephif
            staircount = staircount+1
            stair_zero(staircount,1) = int(xstair(index))+1
            stair_zero(staircount,2) = int(ystair(index))
            stair_zero(staircount,3) = ezf
            staircount = staircount+1
         else if (xcomp.lt.0.AND.ycomp.lt.tole) then
            stair_zero(staircount,1) = int(xstair(index))
            stair_zero(staircount,2) = int(ystair(index))
            stair_zero(staircount,3) = ephif
            staircount = staircount+1
            stair_zero(staircount,1) = int(xstair(index))
            stair_zero(staircount,2) = int(ystair(index))
            stair_zero(staircount,3) = erf
            staircount = staircount+1
         else
            print *,'error in determing staircase type. '
            print *,' (z,x) = ', stair_zero(index,1),
     2           stair_zero(index,2), index, xcomp, ycomp
            stair_zero(index,3) = ephif
            pause
         end if
 90   continue
c**** Complete the last zero field
      stair_zero(staircount,1) = int(xstair(stair_node_count))
      stair_zero(staircount,2) = int(ystair(stair_node_count))
      stair_zero(staircount,3) = ephif
      stair_node_count = staircount
c      print *,staircount

c**** Write out staircase model.
      open(unit=10,file='stairnew.dat',status='unknown',
     1     form='formatted')

      do 1000 index=1,stair_node_count
         write(10,*) stair_zero(index,1), stair_zero(index,2),
     1        stair_zero(index,3)
 1000 continue
      close(unit=10)

      RETURN
      END

c**********************************************************************
c STAIR_BOUNDARY_CONDITIONS sets all the appropriate fields in the
c   staircase model to zero.
c**********************************************************************

      SUBROUTINE stair_boundary_conditions

      implicit none
      include 'common.f'

      integer index

c      print *,'calling stair_boundary...',stair_node_count
      do 10 index = 3,stair_node_count
         if (stair_zero(index,3).eq.ezf) then
            ez(stair_zero(index,1),stair_zero(index,2)) = 0.0
         else if (stair_zero(index,3).eq.ephif) then
            ephi(stair_zero(index,1),stair_zero(index,2)) = 0.0
         else if (stair_zero(index,3).eq.erf) then
            er(stair_zero(index,1),stair_zero(index,2)) = 0.0
         else
            print *,'unknown stair_zero type'
            print *,stair_zero(index,1),stair_zero(index,2),
     1           stair_zero(index,3)
            pause
         end if
 10   continue

      RETURN
      END


c**********************************************************************
c REAL FUNCTION DIST_TO_LINE returns the perpendicular distance from a
c point in space (x,y) to a line that is of the form Ax+By=C
c**********************************************************************

      REAL FUNCTION dist_to_line(A,B,C,x,y)

      implicit none
      real A,B,C,x,y

      dist_to_line = abs((A*x+B*y-C)/sqrt(A**2.0 + B**2.0))

      RETURN
      END


c************************************************************c
c setups cells for scattered/total field calculations.       c
c see picture in notes for numbering scheme.                 c
c************************************************************c

      subroutine setup_scat

      implicit none
      include 'common.f'

      integer k,i,x1,y1,x2,y2
      real mxr

c      mxr=RB(1)
c      do 15 i=1,NP
c         if (RB(i).gt.mxr) mxr=RB(i)
c 15   continue

      mxr=int(obj_height/dz)

c*****scat/tot rcs box region definers
      x1 = xscat_sp
      y1=1
      x2=maxz-x1
      y2=int(mxr)+ytot_sp
      mheight = y2+yhuy_sp
      rcsz1=x1-xhuy_sp
      rcsz2=x2+xhuy_sp

      do 10 k=1,maxz
         do 20 i=1,maxr
            scattot(k,i)=15
 20      continue
 10   continue

      scattot(x1,y1)=2
      scattot(x2,y1)=8
      scattot(x1,y2)=4
      scattot(x2,y2)=6

      i=y1
      do 30 k=x1+1,x2-1
         scattot(k,i)=9
 30   continue

      i=y2
      do 40 k=x1+1,x2-1
         scattot(k,i)=5
 40   continue

      k=x1
      do 50 i=y1+1,y2-1
         scattot(k,i)=3
 50   continue

      k=x2
      do 60 i=y1+1,y2-1
         scattot(k,i)=7
 60   continue

      k=x1-1
      do 70 i=y1,y2
         scattot(k,i)=1
 70   continue

      k=x2+1
      do 80 i=y1,y2
         scattot(k,i)=12
```

```
 80   continue

      i=y2+1
      do 90 k=x1,x2
         scattot(k,i)=11
 90   continue

      do 100 k=x1+1,x2-1
         do 110 i=y1+1,y2-1
            scattot(k,i)=14
110      continue
100   continue

      return
      end
```

The following, **pml.f**, contains the subroutines for implementing the BOR FD-TD PML absorbing boundary condition.

```
c*****************************************************************c
c PML Equations: right, left, top                                 c
c*****************************************************************c
c E fields                                                        c
c*****************************************************************c

      subroutine pmlEeqn(m,ms)

      implicit none
      include 'common.f'

      integer k,i,m,axis,ms
      real c1,c2,c3,c4,c5,c6
      real sigma_r,sigma_z
      axis=1
C **************Calculate Erz fields**********************

c     real region interface

      c1=dt/(eps*dz)
      do 10 i=1,pmldepth
         do 20 k=1,maxz-1

C ****CENTER TOP REGION
            erzt(k,i)=erzt(k,i)+(1/eta)*(c1*(hphizt(k,i)+hphirt(k,i)
     1           -hphizt(k+1,i)-hphirt(k+1,i)))
 20      continue

         erzt(maxz,i)=erzt(maxz,i)+(1/eta)*(c1*(hphizt(maxz,i)+hphirt
     1        (maxz,i)-hphizr(1,i+maxr)-hphirr(1,i+maxr)))
 10   continue

      do 30 i=1,pmldepth+maxr
         do 40 k=1,pmldepth
            sigma_z=sigma_max*((k+0.0)/pmldepth)**2.0
            c1=exp(-sigma_z*dt/eps)
            c2=(c1-1.0)/(sigma_z*dz)

C **********Right Side**********
            erzr(k,i)=c1*erzr(k,i)+(1/eta)*(-c2*(hphizr(k,i)+hphirr
     1           (k,i)-hphizr(k+1,i)-hphirr(k+1,i)))
C **********Left Side**********reminder:k=right.left=pmldepth.1
            if (k.eq.1) THEN
               if (i.gt.maxr) THEN
                  c3=hphizt(1,i-maxr)+hphirt(1,i-maxr)
               ELSE
                  c3=hphi(1,i)
               END IF
               erzl(k,i)=c1*erzl(k,i)+(1/eta)*(-c2*(hphizl(k,i)+hphirl
     1              (k,i)-c3))
            ELSE
               erzl(k,i)=c1*erzl(k,i)+(1/eta)*(-c2*(hphizl(k,i)+hphirl
     1              (k,i)-hphizl(k-1,i)-hphirl(k-1,i)))
            END IF
 40      continue
 30   continue

C **************Calculate Erphi fields**********************c

C ****CENTER BOTTOM & TOP REGIONS

      do 41 k=1,maxz
         do 42 i=1,pmldepth
            c4=(m*dt/eps)/((i+0.5-1.0+maxr)*dz)
            erphit(k,i)=erphit(k,i)-(1.0/eta)*(c4*(hzrt(k,i)+
     1           hzphit(k,i)))
 42      continue
 41   continue

C ****RIGHT & LEFT SIDES

      do 46 k=1,pmldepth
         do 47 i=1,pmldepth+maxr
            c4=(m*dt/eps)/((i+0.5-1.0)*dz)
            erphil(k,i)=erphil(k,i)-(c4*(hzrl(k,i)+hzphil(k,i)))/eta
            erphir(k,i)=erphir(k,i)-(c4*(hzrr(k,i)+hzphir(k,i)))/eta
 47      continue
 46   continue

C **************Calculate Ephiz fields**********************c

C ****CENTER BOTTOM & TOP REGIONS

      c1=dt/(eps*dz)
      do 50 i=1,pmldepth
         do 60 k=1,maxz-1
            ephizt(k,i)=ephizt(k,i)+(c1*(hrzt(k+1,i)+hrphit(k+1,i)-
     1           hrzt(k,i)-hrphit(k,i)))/eta
 60      continue

         ephizt(maxz,i)=ephizt(maxz,i)+(c1*(hrzr(1,i+maxr)+hrphir
     1        (1,i+maxr)-hrzt(k,i)-hrphit(k,i)))/eta
 50   continue

      do 70 k=1,pmldepth
         sigma_z=sigma_max*((k+0.0+0.5)/pmldepth)**2.0
         c1=exp(-sigma_z*dt/eps)
         c2=(c1-1.0)/(sigma_z*dz)
c         print *,'sigma_z',c1,c2
         do 80 i=1,pmldepth+maxr

C **********Right Side**********
            if (abs(m).ne.1.AND.i.eq.axis) THEN
               ephizr(k,i)=0.0
            ELSE
               ephizr(k,i)=c1*ephizr(k,i)-(c2*(hrzr(k+1,i)+hrphir
     1              (k+1,i)-hrzr(k,i)-hrphir(k,i)))/eta
            END IF
C **********Left Side**********
            if (k.eq.1) THEN
               if (i.gt.maxr) THEN
                  c3=hrzt(1,i-maxr)+hrphit(1,i-maxr)
               ELSE
                  c3=hr(1,i)
               END IF

               if (abs(m).ne.1.AND.i.eq.axis) THEN
                  ephizl(k,i)=0.0
               ELSE
                  ephizl(k,i)=c1*ephizl(k,i)-(c2*(c3-hrzl(k,i)-
     1                 hrphil(k,i)))/eta
               END IF

            ELSE
               if (abs(m).ne.1.AND.i.eq.axis) THEN
                  ephizl(k,i)=0.0
               ELSE
                  ephizl(k,i)=c1*ephizl(k,i)-(c2*(hrzl(k-1,i)+
     1                 hrphil(k-1,i)-hrzl(k,i)-hrphil(k,i)))/eta
               END IF
            END IF

 80      continue
 70   continue

C **************Calculate Ephir fields**********************c

C ****Sigma_r region

      do 90 i=1,pmldepth
            sigma_r=sigma_max*((i+0.0)/pmldepth)**2.0
            c1=exp(-sigma_r*dt/eps)
            c2=(c1-1.0)/(sigma_r*dz)

c top center region
         do 100 k=1,maxz
            if (i.eq.1) THEN
               ephirt(k,i)=c1*ephirt(k,i)-(c2*(hz(k,maxr)-hzrt
     1              (k,i)-hzphit(k,i)))/eta
            ELSE
               ephirt(k,i)=c1*ephirt(k,i)-(c2*(hzrt(k,i-1)+hzphit
     1              (k,i-1)-hzrt(k,i)-hzphit(k,i)))/eta
            END IF
100      continue

c right and left top regions
         do 110 k=1,pmldepth
            ephirr(k,i+maxr)=c1*ephirr(k,i+maxr)-(c2*(hzrr
     1           (k,i-1+maxr)+hzphir(k,i-1+maxr)-hzrr(k,i+maxr)-
     2           hzphir(k,i+maxr)))/eta
            ephirl(k,i+maxr)=c1*ephirl(k,i+maxr)-(c2*(hzrl
```

```
     1           (k,i-1+maxr)+hzphil(k,i-1+maxr)-hzrl(k,i+maxr)-
     2           hzphil(k,i+maxr)))/eta
 110     continue
 90    continue

C ****Right and Left Center Regions (no sigmas!)

       c5=dt/(eps*dz)
       do 120 k=1,pmldepth
          do 130 i=1,maxr
             if (i.eq.1) THEN
                c4=0.0
                c3=0.0
             ELSE
                c4=hzrr(k,i-1)+hzphir(k,i-1)
                c3=hzrl(k,i-1)+hzphil(k,i-1)
             END IF

             if (i.eq.axis) THEN
                if (abs(m).ne.1) THEN
                   ephirr(k,i)=0.0
                   ephirl(k,i)=0.0
                ELSE
                   c6=2*dt/(eps*dz)
                   ephirr(k,i)=ephirr(k,i)-(c6*(hzphir(k,i)+
     1                 hzrr(k,i)))/eta
                   ephirl(k,i)=ephirl(k,i)-(c6*(hzphil(k,i)+
     1                 hzrl(k,i)))/eta
                END IF
             ELSE
                ephirr(k,i)=ephirr(k,i)+(c5*(c4-hzrr(k,i)-
     1              hzphir(k,i)))/eta
                ephirl(k,i)=ephirl(k,i)+(c5*(c3-hzrl(k,i)-
     1              hzphil(k,i)))/eta
             END IF
 130     continue
 120   continue

C **************Calculate Ezr fields**********************c

C ****Calculate TOP(right,left,center) REGIONS, ie sigma_r regions

       do 140 i=1,pmldepth
          sigma_r=sigma_max*((i+0.0)/pmldepth)**2.0
          c1=exp(-sigma_r*dt/eps)
          c2=(c1-1.0)/(sigma_r*dz)/(i+maxr-1.0)

c middle region

          do 150 k=1,maxz
             if (i.eq.1) THEN
                ezrt(k,i)=c1*ezrt(k,i)-(c2/eta)*((i-0.5+maxr)*
     1               (hphizt(k,i)+hphirt(k,i))-(i-1.5+maxr)*hphi(k,maxr))
             ELSE
                ezrt(k,i)=c1*ezrt(k,i)-(c2/eta)*((i-0.5+maxr)*
     1               (hphizt(k,i)+hphirt(k,i))-(i+maxr-1.5)*
     2               (hphizt(k,i-1)+hphirt(k,i-1)))
             END IF

 150     continue

          do 160 k=1,pmldepth
             ezrl(k,i+maxr)=c1*ezrl(k,i+maxr)-(c2/eta)*((i-0.5+maxr)*
     1            (hphizl(k,i+maxr)+hphirl(k,i+maxr))-(i+maxr-1.5)*
     2            (hphizl(k,i-1+maxr)+hphirl(k,i-1+maxr)))

             ezrr(k,i+maxr)=c1*ezrr(k,i+maxr)-(c2/eta)*((i-0.5+maxr)*
     1            (hphizr(k,i+maxr)+hphirr(k,i+maxr))-(i+maxr-1.5)*
     2            (hphizr(k,i-1+maxr)+hphirr(k,i-1+maxr)))

 160     continue
 140   continue

C ****Right and Left Center Regions (no sigmas!)

       do 125 i=1,maxr

          do 135 k=1,pmldepth
             if (i.eq.axis) THEN
                if (abs(m).eq.0) THEN
                   c4=4*dt/(eps*dz)
                   ezrl(k,i)=ezrl(k,i)+(c4/eta)*(hphirl(k,i)+hphizl(k,i))
                   ezrr(k,i)=ezrr(k,i)+(c4/eta)*(hphirr(k,i)+hphizr(k,i))
                ELSE
                   ezrl(k,i)=0.0
                   ezrr(k,i)=0.0
                END IF
             ELSE
                c5=dt*(i+0.5-1.0)/((i+0.0-1.0)*dz*eps)
                c6=dt*(i-0.5-1.0)/((i+0.0-1.0)*dz*eps)
                ezrl(k,i)=ezrl(k,i)+(c5/eta)*(hphirl(k,i)+hphizl(k,i))
     1               -(c6/eta)*(hphirl(k,i-1)+hphizl(k,i-1))
                ezrr(k,i)=ezrr(k,i)+(c5/eta)*(hphirr(k,i)+hphizr(k,i))
     1               -(c6/eta)*(hphirr(k,i-1)+hphizr(k,i-1))
             END IF
 135     continue
 125   continue

C **************Calculate Ezphi fields**********************c

C ***TOP PML

       do 170 i=1,pmldepth
          c1=m*dt/(eps*(i+0.0+maxr-1.0)*dz)
          do 180 k=1,maxz
             ezphit(k,i)=ezphit(k,i)+(c1/eta)*(hrphit(k,i)+hrzt(k,i))
 180     continue
 170   continue

C ***Right/Left PML

       do 190 i=1,pmldepth+maxr
          do 200 k=1,pmldepth
             if (i.eq.axis) THEN
                ezphir(k,i)=0.0
                ezphil(k,i)=0.0
             ELSE
                c1=m*dt/(eps*(i+0.0-1.0)*dz)
                ezphir(k,i)=ezphir(k,i)+(c1/eta)*(hrphir(k,i)+hrzr(k,i))
                ezphil(k,i)=ezphil(k,i)+(c1/eta)*(hrphil(k,i)+hrzl(k,i))
             END IF
 200     continue
 190   continue

       return
       end

c**************************************************************c
c H fields                                                     c
c**************************************************************c

       subroutine pmlHeqn(m,ms)

       implicit none
       include 'common.f'

       integer k,i,m,axis,ms
       real c1,c2,c3,c4,c5,c6
       real sigma_r,sigma_rs,sigma_z,sigma_zs
       axis=1

C **************Calculate Hrz fields**********************c

C ****The Right & Left Reigions of PML
       do 210 k=1,pmldepth
          sigma_z=sigma_max*((k+0.0)/pmldepth)**2.0
          sigma_zs=sigma_z*(mu/eps)
          c1=exp(-sigma_zs*dt/mu)
          c2=(c1-1.0)/(sigma_zs*dz)
c          print *,'sigma_zs',c1,c2

          do 220 i=1,pmldepth+maxr
             if (k.eq.1) THEN
                if (i.gt.maxr) THEN
                   hrzr(k,i)=c1*hrzr(k,i)-eta*c2*(ephizr(k,i)+ephirr
     1                  (k,i)-ephirt(maxz,i-maxr)-ephizt(maxz,i-maxr))
                ELSE
                   if (i.eq.axis.AND.abs(m).ne.1) THEN
                      hrzr(k,i)=0.0
                   ELSE
                      hrzr(k,i)=c1*hrzr(k,i)-eta*c2*(ephizr(k,i)+ephirr
     1                     (k,i)-ephi(maxz,i))
                   END IF
                END IF
             ELSE
                if (i.eq.axis.AND.abs(m).ne.1) THEN
                   hrzr(k,i)=0.0
                ELSE
                   hrzr(k,i)=c1*hrzr(k,i)-eta*c2*(ephizr(k,i)+ephirr
     1                  (k,i)-ephizr(k-1,i)-ephirr(k-1,i))
                END IF
             END IF

             if (i.eq.axis.AND.abs(m).ne.1) THEN
                hrzl(k,i)=0.0
             ELSE
                hrzl(k,i)=c1*hrzl(k,i)-eta*c2*(ephizl(k,i)+ephirl(k,i)-
     1               ephizl(k+1,i)-ephirl(k+1,i))
             END IF

 220     continue
 210   continue

C ****The Up/Down Center Region PML
```

```
      c5=dt/(mu*dz)

      do 230 k=1,maxz
         do 240 i=1,pmldepth
            if (k.eq.1) THEN
               hrzt(k,i)=hrzt(k,i)+eta*c5*(ephizt(k,i)+ephirt(k,i)-
     1              ephizl(1,i+maxr)-ephirl(1,i+maxr))
            ELSE
               hrzt(k,i)=hrzt(k,i)+eta*c5*(ephizt(k,i)+ephirt(k,i)-
     1              ephizt(k-1,i)-ephirt(k-1,i))
            END IF

 240     continue
 230  continue

C **************Calculate Hrphi fields**********************c

C ****The Right/Left Regions PML

      do 250 i=1,maxr+pmldepth
         do 260 k=1,pmldepth
            if (i.ne.axis) THEN
               c1=m*dt/(mu*(i+0.0-1.0)*dz)
               hrphir(k,i)=hrphir(k,i)-eta*c1*(ezphir(k,i)+ezrr(k,i))
               hrphil(k,i)=hrphil(k,i)-eta*c1*(ezphil(k,i)+ezrl(k,i))
            ELSE
               if (abs(m).ne.1) THEN
                  hrphir(k,i)=0.0
                  hrphil(k,i)=0.0
               ELSE
                  c6=dt/(mu*dz)
                  hrphir(k,i)=hrphir(k,i)+eta*ms*c6*(ezphir(k,i+1)+
     1                 ezrr(k,i+1))
                  hrphil(k,i)=hrphil(k,i)+eta*ms*c6*(ezphil(k,i+1)+
     1                 ezrl(k,i+1))
               END IF
            END IF
 260     continue
 250  continue

C ****The Up/Down Regions PML

      do 270 i=1,pmldepth
         c1=m*dt/(mu*(i+maxr+0.0-1.0)*dz)
         do 280 k=1,maxz
            hrphit(k,i)=hrphit(k,i)-eta*c1*(ezphit(k,i)+ezrt(k,i))
 280     continue
 270  continue

C **************Calculate Hphiz fields**********************c

C ****The Right/Left PML

      do 290 k=1,pmldepth
         sigma_z=sigma_max*((k+0.0)/pmldepth)**2.0
         sigma_zs=sigma_z*(mu/eps)
         c1=exp(-sigma_zs*dt/mu)
         c2=eta*(c1-1.0)/(sigma_zs*dz)

         do 300 i=1,pmldepth+maxr
            if (k.eq.1) THEN
               if (i.gt.maxr) THEN
                  hphizr(k,i)=c1*hphizr(k,i)-c2*(erzt(maxz,i-maxr)+
     1                 erphit(maxz,i-maxr)-erzr(k,i)-erphir(k,i))
               ELSE
                  hphizr(k,i)=c1*hphizr(k,i)-c2*(er(maxz,i)
     1                 -erzr(k,i)-erphir(k,i))
               END IF
            ELSE
               hphizr(k,i)=c1*hphizr(k,i)-c2*(erzr(k-1,i)+erphir(k-1,i)
     1              -erzr(k,i)-erphir(k,i))
            END IF

            hphizl(k,i)=c1*hphizl(k,i)-c2*(erzl(k+1,i)+erphil(k+1,i)
     1           -erzl(k,i)-erphil(k,i))

 300     continue
 290  continue

C ****The Up/Down PML

      c3=eta*dt/(mu*dz)

      do 310 k=1,maxz
         do 320 i=1,pmldepth
            if (k.eq.1) THEN
               hphizt(k,i)=hphizt(k,i)+c3*(erphil(1,i+maxr)+
     1              erzl(1,i+maxr)-erphit(k,i)-erzt(k,i))
            ELSE
               hphizt(k,i)=hphizt(k,i)+c3*(erphit(k-1,i)+erzt(k-1,i)-
     1              erphit(k,i)-erzt(k,i))
            END IF
 320     continue
 310  continue

C **************Calculate Hphir fields**********************c

C ****Bottom/Top (sigma_r) Regions PML

      do 330 i=1,pmldepth
         sigma_r=sigma_max*((i+0.0+0.5)/pmldepth)**2.0
         sigma_rs=sigma_r*(mu/eps)
         c1=exp(-sigma_rs*dt/mu)
         c2=eta*(c1-1.0)/(sigma_rs*dz)

C *******Center Top Region
         do 340 k=1,maxz
            hphirt(k,i)=c1*hphirt(k,i)-c2*(ezrt(k,i+1)+ezphit(k,i+1)-
     1           ezrt(k,i)-ezphit(k,i))
 340     continue

c ****right/left corners
         do 350 k=1,pmldepth
            hphirr(k,i+maxr)=c1*hphirr(k,i+maxr)-c2*(ezrr(k,i+1+maxr)+
     1           ezphir(k,i+1+maxr)-ezrr(k,i+maxr)-ezphir(k,i+maxr))
            hphirl(k,i+maxr)=c1*hphirl(k,i+maxr)-c2*(ezrl(k,i+1+maxr)+
     1           ezphil(k,i+1+maxr)-ezrl(k,i+maxr)-ezphil(k,i+maxr))
 350     continue
 330  continue

C ****Right/Left Center Regions (no sigmas!!)

         c4=eta*dt/(mu*dz)
         do 360 k=1,pmldepth
            do 370 i=1,maxr
               hphirr(k,i)=hphirr(k,i)+c4*(ezrr(k,i+1)+ezphir(k,i+1)
     1              -ezrr(k,i)-ezphir(k,i))

               hphirl(k,i)=hphirl(k,i)+c4*(ezrl(k,i+1)+ezphil(k,i+1)
     1              -ezrl(k,i)-ezphil(k,i))

 370        continue
 360     continue

C **************Calculate Hzr fields**********************c

      do 380 i=1,pmldepth
         sigma_r=sigma_max*((i+0.0+0.5)/pmldepth)**2.0
         sigma_rs=sigma_r*(mu/eps)
         c1=exp(-sigma_rs*dt/mu)
         c2=eta*(c1-1.0)/(sigma_rs*dz)
         c3=c2/(i+maxr+0.5-1.0)

C *******Middle Top/Bottom Region
         do 390 k=1,maxz
            hzrt(k,i)=c1*hzrt(k,i)+c3*((i+maxr+0.0)*(ephizt(k,i+1)+
     1           ephirt(k,i+1))-(i+maxr-1.0)*(ephizt(k,i)+ephirt(k,i)))
 390     continue

c *******right/left corners
         do 400 k=1,pmldepth
            hzrr(k,i+maxr)=c1*hzrr(k,i+maxr)+c3*((i+maxr+0.0)*
     1           (ephizr(k,i+maxr+1)+ephirr(k,i+maxr+1))-(i+maxr-1.0)*
     1           (ephizr(k,i+maxr)+ephirr(k,i+maxr)))

            hzrl(k,i+maxr)=c1*hzrl(k,i+maxr)+c3*((i+maxr+0.0)*
     1           (ephizl(k,i+maxr+1)+ephirl(k,i+maxr+1))-(i+maxr-1.0)*
     2           (ephizl(k,i+maxr)+ephirl(k,i+maxr)))

 400     continue
 380  continue

C ****Right/Left Center Regions (no sigmas!!)

      do 410 i=1,maxr
         c5=eta*(i+0.0-1.0)*dt/(mu*(i+0.5-1.0)*dz)
         c6=eta*(i+1.0-1.0)*dt/(mu*(i+0.5-1.0)*dz)
         do 420 k=1,pmldepth

            hzrl(k,i)=hzrl(k,i)+c5*(ephizl(k,i)+ephirl(k,i))-c6*
     1           (ephizl(k,i+1)+ephirl(k,i+1))
            hzrr(k,i)=hzrr(k,i)+c5*(ephizr(k,i)+ephirr(k,i))-c6*
     1           (ephizr(k,i+1)+ephirr(k,i+1))
 420     continue
 410  continue

C **************Calculate Hzphi fields**********************c

C ****Top/Bottom PML Regions

      do 430 i=1,pmldepth
         c1=m*dt/(mu*dz)
         c2=eta*c1/(i+maxr+0.5-1.0)
```

```
         do 440 k=1,maxz
            hzphit(k,i)=hzphit(k,i)+c2*(erphit(k,i)+erzt(k,i))
 440     continue
 430  continue

C ****Right/Left PML Regions

      do 450 i=1,pmldepth+maxr
         c1=eta*m*dt/(mu*dz*(i+0.5-1.0))

         do 460 k=1,pmldepth
            hzphir(k,i)=hzphir(k,i)+c1*(erphir(k,i)+erzr(k,i))
            hzphil(k,i)=hzphil(k,i)+c1*(erphil(k,i)+erzl(k,i))
 460     continue
 450  continue

      return
      end
```

The following, **gquad.f**, contains the subroutines for calculating the Fourier components of the incident wave using a Gaussian quadrature technique.

```
c*********************************************************************c
c Calculates an numerical integral using Gaussian Quadrature        c
c in order to determines the coef of the Fourier series for         c
c the incident plane wave.                                          c
c Intno: -4-Ermu;   2-Ephimu; -6-Ezmu;  4-Ermv; -2-Ephimv;  6-Ezmv c
c         7-Hrmu; -11-Hphimu;  9-Hzmu; -7-Hrmv; 11-Hphimv; -9-Hzmv c
c*********************************************************************c

      real function Gquad(a,b,IntNo,m,t,r,zg,theta)

      implicit none
      include 'common.f'

      real a,b,t,r,zg,theta,Intgrl,value,y,z,weight
      real Ermu,Ephimu,Ezmu,Ermv,Ephimv,Ezmv,cossq,z20
      real Hrmu,Hphimu,Hzmu,Hrmv,Hphimv,Hzmv,sinsq
      real weight20, dx, e1, e2, h, mid, steps
      integer j,IntNo,m,AIN,i

      dimension z(10), weight(10), z20(20), weight20(20)
      DATA (z(j), j=1,10)/-.9739065285,-.8650633667,-.6794095683,
     1     -.4333953941, -.1488743390,.1488743390,.4333953941,
     2     .6794095683, .8650633667,.9739065285/

      DATA (weight(j), j=1,10)
     1     /.0666713443,.1494513492,.2190863625,.2692667193,
     2      .2955242247,.295524247,.2692667193,.2190863625,
     3      .1494513492,.0666713443/

      DATA (z20(j), j=1,20)
     1     /-0.99312859919241, -0.96397192726078,
     2      -0.91223442826796, -0.83911697181213,
     3      -0.74633190646476, -0.63605368072468,
     4      -0.51086700195146, -0.37370608871528,
     5      -0.22778585114165, -0.07652652113350,
     6       0.07652652113350,  0.22778585114165,
     7       0.37370608871528,  0.51086700195146,
     8       0.63605368072468,  0.74633190646476,
     9       0.83911697181213,  0.91223442826796,
     1       0.96397192726078,  0.99312859919241/


      DATA (weight20(j), j=1,20)
     1     /0.01761400713536,
     1     0.04060142981029, 0.06267204829089,
     2     0.08327674159386, 0.10193011980641,
     3     0.11819453199405, 0.13168863844930,
     4     0.14209610916487, 0.14917298630417,
     5     0.15275338717117, 0.15275338723120,
     6     0.14917298659407, 0.14209610937519,
     7     0.13168863843930, 0.11819453196154,
     8     0.10193011980823, 0.08327674160932,
     9     0.06267204829828, 0.04060142982019,
     1     0.01761400714091/

C*****Expressions to account for "real" distance from orgin
C*****of field values.  It calculates field distances for 1/2 lattice
C*****points, and since "grid" i=1,maxr <=> "real" i=0,(maxr-1)*dr

      r=r-dz

      AIN = abs(IntNo)
      if (AIN.eq.4.OR.AIN.eq.9.OR.AIN.eq.11)
     1     r=r+dz/(2.0)
      if (AIN.eq.4.OR.AIN.eq.9.OR.AIN.eq.2)
     1     zg=zg+dz/(2.0)
      if (AIN.eq.7.OR.AIN.eq.9.OR.AIN.eq.11)
     1     t = t-dt
c     1     t = t
      if (AIN.eq.4.OR.AIN.eq.6.OR.AIN.eq.2)
     1     t = t-dt/2.0
c     1     t = t+dt/2.0


C*****************************************************************
C* Integration by 20-point Gauss-Legendre quadrature.  A and B   *
C* are the limits of integration, and FUNC is the user-supplied  *
C* function to be integrated.  The result is returned in INTGRL  *
c* Incident waves of mode m are divided in m+1 regions which     *
c* are each computed by 20-point gquad.                          *
C*****************************************************************

      INTGRL = 0.0
      dx = (b-a)/(m+1)
      do 5 steps = 1,(m+1)
         e1 = a + (steps-1)*dx
         e2 = a + steps*dx
         h = (e2-e1)/2
         mid = (e1+e2)/2

         do 10 I = 1,20
            Y = z20(I)*h + mid
            if (IntNo.eq.2) value = Ephimu(Y,m,t,r,zg,theta)
            if (IntNo.eq.4) value = Ermv(Y,m,t,r,zg,theta)
            if (IntNo.eq.6) value = Ezmv(Y,m,t,r,zg,theta)
            if (IntNo.eq.7) value = Hrmu(Y,m,t,r,zg,theta)
            if (IntNo.eq.9) value = Hzmu(Y,m,t,r,zg,theta)
            if (IntNo.eq.11) value = Hphimv(Y,m,t,r,zg,theta)

            if (IntNo.eq.-2) value = Ephimv(Y,m,t,r,zg,theta)
            if (IntNo.eq.-4) value = Ermu(Y,m,t,r,zg,theta)
            if (IntNo.eq.-6) value = Ezmu(Y,m,t,r,zg,theta)
            if (IntNo.eq.-7) value = Hrmv(Y,m,t,r,zg,theta)
            if (IntNo.eq.-9) value = Hzmv(Y,m,t,r,zg,theta)
            if (IntNo.eq.-11) value = Hphimu(Y,m,t,r,zg,theta)

            if (IntNo.eq.45) value = cossq(Y)
            if (IntNo.eq.46) value = sinsq(Y)

            INTGRL = INTGRL + h*weight20(I)*value
 10      continue
 5    continue

      if (m.eq.0) THEN
         gquad = INTGRL*5.0/2.0
      ELSE
         gquad = INTGRL*5.0
      END IF

c      if (abs(gquad).gt.1e-6) print *,gquad,IntNo,m,t,r,zg,theta

c      if (abs(gquad).gt.1) then
c         print *, IntNo, t, r, zg, sdev, theta, m
c      end if


      RETURN
      END

c*********************************************************************c
c All the incident wave functions to be integrated by Gaussian
c Quadrature.
c*********************************************************************c

      real function cossq(phi)

      implicit none
      include 'common.f'

      real phi

      cossq = cos(6*phi)*cos(phi)*exp(-(3+cos(phi))**2.0)*100

      return
      end

C*************************************************************
      real function sinsq(phi)

      implicit none
      include 'common.f'

      real phi

      sinsq = (1/pi)*(sin(phi))**2.0
```

```
      return
      end
C*************************************************************

      real function Ermu(phi,m,t,r,zg,theta)

      implicit none
      include 'common.f'

      real phi,t,r,zg,theta
      integer m

      Ermu=(1/(pi*sqrt(2*pi)))*cos(m*phi)*(Ehg*cos(phi)*cos
     1     (theta)+Evg*sin(phi))*exp(-(((t-gd)+((zg*cos(theta)
     2     +r*sin(theta)*cos(phi))/c))**2)/(sdev**2))*
     3     ((sin(2*pi*modfreq*((t-gd)+((zg*cos(theta)+r*sin(theta)*
     4     cos(phi))/c))))*modulate+abs(modulate-1))

      return
      end

c*****************************************************************c

      real function Ermv(phi,m,t,r,zg,theta)

      implicit none
      include 'common.f'

      real phi,t,r,zg,theta
      integer m

      Ermv=(1/(pi*sqrt(2*pi)))*sin(m*phi)*(Ehg*cos(phi)*cos(theta)
     1     +Evg*sin(phi))*exp(-(((t-gd)+((zg*cos(theta)+r*sin(theta)
     2     *cos(phi))/c))**2)/(sdev**2))*
     3     ((sin(2*pi*modfreq*((t-gd)+((zg*cos(theta)+r*sin(theta)*
     4     cos(phi))/c))))*modulate+abs(modulate-1))

      return
      end

c*****************************************************************c

      real function Ephimu(phi,m,t,r,zg,theta)

      implicit none
      include 'common.f'

      real phi,t,r,zg,theta
      integer m

      Ephimu=(1/(pi*sqrt(2*pi)))*cos(m*phi)*(-Ehg*sin(phi)*cos(theta)
     1     +Evg*cos(phi))*exp(-(((t-gd)+((zg*cos(theta)+r*sin(theta)
     2     *cos(phi))/c))**2)/(sdev**2))*
     3     ((sin(2*pi*modfreq*((t-gd)+((zg*cos(theta)+r*sin(theta)*
     4     cos(phi))/c))))*modulate+abs(modulate-1))

      return
      end

c*****************************************************************c

      real function Ephimv(phi,m,t,r,zg,theta)

      implicit none
      include 'common.f'

      real phi,t,r,zg,theta
      integer m

      Ephimv=(1/(pi*sqrt(2*pi)))*sin(m*phi)*(-Ehg*sin(phi)*cos(theta)
     1     +Evg*cos(phi))*exp(-(((t-gd)+((zg*cos(theta)+r*sin(theta)
     2     *cos(phi))/c))**2)/(sdev**2))*
     3     ((sin(2*pi*modfreq*((t-gd)+((zg*cos(theta)+r*sin(theta)*
     4     cos(phi))/c))))*modulate+abs(modulate-1))

      return
      end

c*****************************************************************c

      real function Ezmu(phi,m,t,r,zg,theta)

      implicit none
      include 'common.f'

      real phi,t,r,zg,theta
      integer m

      Ezmu=(1/(pi*sqrt(2*pi)))*cos(m*phi)*(-Ehg*sin(theta))*
     1     exp((-((t-gd)+(zg*cos(theta)+r*sin(theta)*cos
     2     (phi))/c)**2)/(sdev**2))*
     3     ((sin(2*pi*modfreq*((t-gd)+((zg*cos(theta)+r*sin(theta)*
     4     cos(phi))/c))))*modulate+abs(modulate-1))

      return
      end

c*****************************************************************c

      real function Ezmv(phi,m,t,r,zg,theta)

      implicit none
      include 'common.f'

      real phi,t,r,zg,theta
      integer m

      Ezmv=(1/(pi*sqrt(2*pi)))*sin(m*phi)*(-Ehg*sin(theta))*
     1     exp((-((t-gd)+(zg*cos(theta)+r*sin(theta)*cos(phi))
     2     /c)**2)/(sdev**2))*
     3     ((sin(2*pi*modfreq*((t-gd)+((zg*cos(theta)+r*sin(theta)*
     4     cos(phi))/c))))*modulate+abs(modulate-1))

      return
      end

c*****************************************************************c

      real function Hrmu(phi,m,t,r,zg,theta)

      implicit none
      include 'common.f'

      real phi,t,r,zg,theta
      integer m

      Hrmu=(1/(pi*sqrt(2*pi)))*cos(m*phi)*(Evg*cos(theta)*
     1     cos(phi)-Ehg*sin(phi))*exp((-((t-gd)+(zg*cos(theta)+r*
     2     sin(theta)*cos(phi))/c)**2)/(sdev**2))*
     3     ((sin(2*pi*modfreq*((t-gd)+((zg*cos(theta)+r*sin(theta)*
     4     cos(phi))/c))))*modulate+abs(modulate-1))

      return
      end

c*****************************************************************c

      real function Hrmv(phi,m,t,r,zg,theta)

      implicit none
      include 'common.f'

      real phi,t,r,zg,theta
      integer m

      Hrmv=(1/(pi*sqrt(2*pi)))*sin(m*phi)*(Evg*cos(theta)*
     1     cos(phi)-Ehg*sin(phi))*exp((-((t-gd)+(zg*cos(theta)+r*
     2     sin(theta)*cos(phi))/c)**2)/(sdev**2))*
     3     ((sin(2*pi*modfreq*((t-gd)+((zg*cos(theta)+r*sin(theta)*
     4     cos(phi))/c))))*modulate+abs(modulate-1))

      return
      end

c*****************************************************************c

      real function Hphimu(phi,m,t,r,zg,theta)

      implicit none
      include 'common.f'

      real phi,t,r,zg,theta
      integer m

      Hphimu=(1/(pi*sqrt(2*pi)))*cos(m*phi)*(-Evg*cos(theta)*
     1     sin(phi)-Ehg*cos(phi))*exp((-((t-gd)+(zg*cos(theta)+r*
     2     sin(theta)*cos(phi))/c)**2)/(sdev**2))*
     3     ((sin(2*pi*modfreq*((t-gd)+((zg*cos(theta)+r*sin(theta)*
     4     cos(phi))/c))))*modulate+abs(modulate-1))

      return
      end

c*****************************************************************c

      real function Hphimv(phi,m,t,r,zg,theta)

      implicit none
      include 'common.f'

      real phi,t,r,zg,theta
      integer m

      Hphimv=(1/(pi*sqrt(2*pi)))*sin(m*phi)*(-Evg*cos(theta)*
```

```
     1     sin(phi)-Ehg*cos(phi))*exp((-((t-gd)+(zg*cos(theta)+r*
     2     sin(theta)*cos(phi))/c)**2)/(sdev**2))*
     3     ((sin(2*pi*modfreq*((t-gd)+((zg*cos(theta)+r*sin(theta)*
     4     cos(phi))/c))))*modulate+abs(modulate-1))

      return
      end

c*****************************************************************c

      real function Hzmu(phi,m,t,r,zg,theta)

      implicit none
      include 'common.f'

      real phi,t,r,zg,theta
      integer m

      Hzmu=(1/(pi*sqrt(2*pi)))*cos(m*phi)*(-Evg*sin(theta))*
     1     exp((-((t-gd)+(zg*cos(theta)+r*sin(theta)*cos(phi))
     2     /c)**2)/(sdev**2))*
     3     ((sin(2*pi*modfreq*((t-gd)+((zg*cos(theta)+r*sin(theta)*
     4     cos(phi))/c))))*modulate+abs(modulate-1))

      return
      end

c*****************************************************************c

      real function Hzmv(phi,m,t,r,zg,theta)

      implicit none
      include 'common.f'

      real phi,t,r,zg,theta
      integer m

      Hzmv=(1/(pi*sqrt(2*pi)))*sin(m*phi)*(-Evg*sin(theta))*
     1     exp((-((t-gd)+(zg*cos(theta)+r*sin(theta)*cos(phi))
     2     /c)**2)/(sdev**2))*
     3     ((sin(2*pi*modfreq*((t-gd)+((zg*cos(theta)+r*sin(theta)*
     4     cos(phi))/c))))*modulate+abs(modulate-1))

      return
      end
```

The following, **rcs.f**, contains the subroutines for performing the DFT on the fly of the fields as well as those for computing the radar cross sections.

```
C*********************************************************************C
C Performs the dft on the fly.There are 12 field values per grid per   C
C mode cell that will be stored (i.e. eru, erv, ephiu, ephiv, etc.)    C
C They are stored in the complex arrays feru, ferv, fephiu, fphiv,     C
C etc.  Since there are only six arrays at any given time holding      C
C field values (i.e. er, ephi, ez, hr, hphi, hz) the subroutine        C
C updates the appropiate complex arrays based on the input variables   C
C mode (what Fourier is being calculated) and eqset (which equation    C
C set is being used).                                                  C
C                                                                      C
C Equation set 1 contains erv, ephiu, ezv, hru, hzu, hphiv             C
C Equation set 2 contains eru, ephiv, ezu, hrv, hzv, hphiu             C
C                                                                      C
C Adjacent field values are averaged in order to approximate their     C
C values along the lattice points (k,i) (Note: hr and ez are never     C
C averaged since they lie on the lattice points)                       C
C                                                                      C
C*********************************************************************C

      SUBROUTINE update_dft(mode,eqset)

      implicit none
      include 'common.f'

      integer k, i, j, mode, eqset
      real temp, tempfreq

      if (eqset.eq.1) THEN
         k=rcsz1
C     ***loop cycles through first mheight-1 points, left side of box
         do 10 i=1,mheight-1
C     ******loop cycles through all frequencies of interest.
            do 11 j=minf,maxf,stepf
c              tempfreq = low_freq+dfreq*(j+0.0)
               tempfreq = freqlist(j,1)

               if (i.eq.1) then
                  temp = er(k,i)
               else
                  temp = (er(k,i)+er(k,i-1))/2.0
               end if
               ferv(mode,i,j)=ferv(mode,i,j)+temp*exp(2*pi*
     1              (0,1)*tempfreq*dt*time)*dt

               temp=(hz(k,i)+hz(k-1,i))/2.0
               fhzu(mode,i,j)=fhzu(mode,i,j)+temp*exp(2*pi*
     1              (0,1)*tempfreq*dt*time)*dt

               temp=(ephi(k,i)+ephi(k-1,i))/2.0
               fephiu(mode,i,j)=fephiu(mode,i,j)+temp*exp(2*pi*
     1              (0,1)*tempfreq*dt*time)*dt

               temp=hr(k,i)
               fhru(mode,i,j)=fhru(mode,i,j)+temp*exp(2*pi*
     1              (0,1)*tempfreq*dt*time)*dt

               temp=ez(k,i)
               fezv(mode,i,j)=fezv(mode,i,j)+temp*exp(2*pi*(0,1)
     1              *tempfreq*dt*time)*dt

               if (i.eq.1) then
                  temp = hphi(k,i)
               else
                  temp=(hphi(k,i)+hphi(k,i-1))/2.0
               end if
               fhphiv(mode,i,j)=fhphiv(mode,i,j)+temp*exp(2*pi*(0,1)
     1              *tempfreq*dt*time)*dt

 11         continue
 10      continue

         i=mheight
C     ***loop cycles through mheight,mheight+z2-z1 points, top of box
         do 20 k=rcsz1,rcsz2
C     ******loop cycles through all frequencies of interest.
            do 21 j=minf,maxf,stepf
c              tempfreq = low_freq+dfreq*(j+0.0)
               tempfreq = freqlist(j,1)

               temp=(er(k,i)+er(k,i-1))/2.0
               ferv(mode,mheight+k-rcsz1,j)=ferv(mode,mheight+k-rcsz1,
     1              j)+temp*exp(2*pi*(0,1)*tempfreq*dt*time)*dt

               temp=(hz(k,i)+hz(k-1,i))/2.0
               fhzu(mode,mheight+k-rcsz1,j)=fhzu(mode,mheight+k-rcsz1,
     1              j)+temp*exp(2*pi*(0,1)*tempfreq*dt*time)*dt

               temp=(hphi(k,i)+hphi(k,i-1))/2.0
               fhphiv(mode,mheight+k-rcsz1,j)=fhphiv(mode,mheight+k-
     1              rcsz1,j)+temp*exp(2*pi*(0,1)*tempfreq*dt*time)*dt

               temp=hr(k,i)
               fhru(mode,mheight+k-rcsz1,j)=fhru(mode,mheight+k-rcsz1,
     1              j)+temp*exp(2*pi*(0,1)*tempfreq*dt*time)*dt

               temp=ez(k,i)
               fezv(mode,mheight+k-rcsz1,j)=fezv(mode,mheight+k-rcsz1,
     1              j)+temp*exp(2*pi*(0,1)*tempfreq*dt*time)*dt

               temp=(ephi(k,i)+ephi(k-1,i))/2.0
               fephiu(mode,mheight+k-rcsz1,j)=fephiu(mode,mheight+k-
     1              rcsz1,j)+temp*exp(2*pi*(0,1)*tempfreq*dt*time)*dt
 21         continue
 20      continue

         k=rcsz2
C     ***loop cycles through last mheight-1 points, right side of box
         do 30 i=1,mheight-1
C     ******loop cycles through all frequencies of interest.
            do 31 j=minf,maxf,stepf
c              tempfreq = low_freq+dfreq*(j+0.0)
               tempfreq = freqlist(j,1)

               if (i.eq.1) then
                  temp = er(k,i)
               else
                  temp = (er(k,i)+er(k,i-1))/2.0
               end if
               ferv(mode,2*mheight-i+rcsz2-rcsz1,j)=ferv(mode,2*
     1              mheight-i+rcsz2-rcsz1,j)+temp*exp(2*pi*(0,1)*
     2              tempfreq*dt*time)*dt

               temp=(hz(k,i)+hz(k-1,i))/2.0
               fhzu(mode,2*mheight-i+rcsz2-rcsz1,j)=fhzu(mode,2*
     1              mheight-i+rcsz2-rcsz1,j)+temp*exp(2*pi*(0,1)*
     2              tempfreq*dt*time)*dt

               temp=(ephi(k,i)+ephi(k-1,i))/2.0
               fephiu(mode,2*mheight-i+rcsz2-rcsz1,j)=fephiu(mode
     1              ,2*mheight-i+rcsz2-rcsz1,j)+temp*exp(2*pi*
```

```
     2              (0,1)*tempfreq*dt*time)*dt

                 temp=hr(k,i)
                 fhru(mode,2*mheight-i+rcsz2-rcsz1,j)=fhru(mode
     1                ,2*mheight-i+rcsz2-rcsz1,j)+temp*exp(2*pi*
     2                (0,1)*tempfreq*dt*time)*dt

                 temp=ez(k,i)
                 fezv(mode,2*mheight-i+rcsz2-rcsz1,j)=fezv(mode
     1                ,2*mheight-i+rcsz2-rcsz1,j)+temp*exp(2*pi*
     2                (0,1)*tempfreq*dt*time)*dt

                 if (i.eq.1) then
                    temp = hphi(k,i)
                 else
                    temp=(hphi(k,i)+hphi(k,i-1))/2.0
                 end if
                 fhphiv(mode,2*mheight-i+rcsz2-rcsz1,j)=fhphiv(mode
     1                ,2*mheight-i+rcsz2-rcsz1,j)+temp*exp(2*pi*
     2                (0,1)*tempfreq*dt*time)*dt

  31           continue
  30         continue

       ELSE
C****Eqset number 2

           k=rcsz1
C        ***loop cycles through first mheight-1 points, left side of box
           do 110 i=1,mheight-1
C        ******loop cycles through all frequencies of interest.
              do 111 j=minf,maxf,stepf
c                tempfreq = low_freq+dfreq*(j+0.0)
                 tempfreq = freqlist(j,1)

                 if (i.eq.1) then
                    temp = er(k,i)
                 else
                    temp = (er(k,i)+er(k,i-1))/2.0
                 end if
                 feru(mode,i,j)=feru(mode,i,j)+temp*exp(2*pi*
     1                (0,1)*tempfreq*dt*time)*dt

                 temp=(hz(k,i)+hz(k-1,i))/2.0
c                 print *,'hz',temp
                 fhzv(mode,i,j)=fhzv(mode,i,j)+temp*exp(2*pi*
     1                (0,1)*tempfreq*dt*time)*dt

                 temp=(ephi(k,i)+ephi(k-1,i))/2.0
c                 print *,'ephi', temp
                 fephiv(mode,i,j)=fephiv(mode,i,j)+temp*
     1                exp(2*pi*(0,1)*tempfreq*dt*time)*dt

                 temp=hr(k,i)
c                 print *,'hr',temp
                 fhrv(mode,i,j)=fhrv(mode,i,j)+temp*
     1                exp(2*pi*(0,1)*tempfreq*dt*time)*dt

                 temp=ez(k,i)
c                 print *,'ez',temp
                 fezu(mode,i,j)=fezu(mode,i,j)+temp*
     1                exp(2*pi*(0,1)*tempfreq*dt*time)*dt

                 if (i.eq.1) then
                    temp = hphi(k,i)
                 else
                    temp=(hphi(k,i)+hphi(k,i-1))/2.0
                 end if
c                print *,'hphi',temp
                 fhphiu(mode,i,j)=fhphiu(mode,i,j)+temp*
     1                exp(2*pi*(0,1)*tempfreq*dt*time)*dt

  111          continue
  110        continue

           i=mheight
C        ***loop cycles through mheight,mheight+z2-z1 points, top of box
           do 120 k=rcsz1,rcsz2
C        ******loop cycles through all frequencies of interest.
              do 121 j=minf,maxf,stepf
c                tempfreq = low_freq+dfreq*(j+0.0)
                 tempfreq = freqlist(j,1)

                 temp=(er(k,i)+er(k,i-1))/2.0
c                 if (k.eq.rcsz1) write(81,*) temp
                 feru(mode,mheight+k-rcsz1,j)=feru(mode,mheight
     1                +k-rcsz1,j)+temp*exp(2*pi*(0,1)*tempfreq*
     2                dt*time)*dt

                 temp=(hz(k,i)+hz(k-1,i))/2.0
                 fhzv(mode,mheight+k-rcsz1,j)=fhzv(mode,mheight
     1                +k-rcsz1,j)+temp*exp(2*pi*(0,1)*tempfreq*
     2                dt*time)*dt

                 temp=(hphi(k,i)+hphi(k,i-1))/2.0
                 fhphiu(mode,mheight+k-rcsz1,j)=fhphiu(mode,mheight
     1                +k-rcsz1,j)+temp*exp(2*pi*(0,1)*tempfreq*
     2                dt*time)*dt

                 temp=hr(k,i)
                 fhrv(mode,mheight+k-rcsz1,j)=fhrv(mode,mheight
     1                +k-rcsz1,j)+temp*exp(2*pi*(0,1)*tempfreq*
     2                dt*time)*dt

                 temp=ez(k,i)
                 fezu(mode,mheight+k-rcsz1,j)=fezu(mode,mheight
     1                +k-rcsz1,j)+temp*exp(2*pi*(0,1)*tempfreq*
     2                dt*time)*dt

                 temp=(ephi(k,i)+ephi(k-1,i))/2.0
                 fephiv(mode,mheight+k-rcsz1,j)=fephiv(mode,mheight
     1                +k-rcsz1,j)+temp*exp(2*pi*(0,1)*tempfreq*
     2                dt*time)*dt
  121          continue
  120        continue

           k=rcsz2
C        ***loop cycles through last mheight-1 points, right side of box
           do 130 i=1,mheight-1
C        ******loop cycles through all frequencies of interest.
              do 131 j=minf,maxf,stepf
c                tempfreq = low_freq+dfreq*(j+0.0)
                 tempfreq = freqlist(j,1)

                 if (i.eq.1) then
                    temp = er(k,i)
                 else
                    temp = (er(k,i)+er(k,i-1))/2.0
                 end if
                 feru(mode,2*mheight-i+rcsz2-rcsz1,j)=feru(mode,2*
     1                mheight-i+rcsz2-rcsz1,j)+temp*exp(2*pi*(0,1)*
     2                tempfreq*dt*time)*dt

                 temp=(hz(k,i)+hz(k-1,i))/2.0
                 fhzv(mode,2*mheight-i+rcsz2-rcsz1,j)=fhzv(mode,2*
     1                mheight-i+rcsz2-rcsz1,j)+temp*exp(2*pi*(0,1)*
     2                tempfreq*dt*time)*dt

                 temp=(ephi(k,i)+ephi(k-1,i))/2.0
                 fephiv(mode,2*mheight-i+rcsz2-rcsz1,j)=fephiv(mode
     1                ,2*mheight-i+rcsz2-rcsz1,j)+temp*exp(2*pi*
     2                (0,1)*tempfreq*dt*time)*dt

                 temp=hr(k,i)
                 fhrv(mode,2*mheight-i+rcsz2-rcsz1,j)=fhrv(mode
     1                ,2*mheight-i+rcsz2-rcsz1,j)+temp*exp(2*pi*
     2                (0,1)*tempfreq*dt*time)*dt

                 temp=ez(k,i)
                 fezu(mode,2*mheight-i+rcsz2-rcsz1,j)=fezu(mode
     1                ,2*mheight-i+rcsz2-rcsz1,j)+temp*exp(2*pi*
     2                (0,1)*tempfreq*dt*time)*dt

                 if (i.eq.1) then
                    temp = hphi(k,i)
                 else
                    temp=(hphi(k,i)+hphi(k,i-1))/2.0
                 end if
                 fhphiu(mode,2*mheight-i+rcsz2-rcsz1,j)=fhphiu(mode
     1                ,2*mheight-i+rcsz2-rcsz1,j)+temp*exp(2*pi*
     2                (0,1)*tempfreq*dt*time)*dt

  131          continue
  130        continue

       END IF

       return
       end

C*************************************************************C
C initialize all frequency field values to zero               C
C*************************************************************C

       SUBROUTINE init_freq

       implicit none
       include 'common.f'

       integer m,k,i

       do 10 m=mode_start,mode_end
          do 20 k=1,mxdp
             do 30 i=1,MAX_FREQS
```

```
               fephiu(m,k,i)=0.0
               fephiv(m,k,i)=0.0
               feru(m,k,i)=0.0
               ferv(m,k,i)=0.0
               fezu(m,k,i)=0.0
               fezv(m,k,i)=0.0
               fhphiu(m,k,i)=0.0
               fhphiv(m,k,i)=0.0
               fhru(m,k,i)=0.0
               fhrv(m,k,i)=0.0
               fhzu(m,k,i)=0.0
               fhzv(m,k,i)=0.0
 30          continue
 20       continue
 10    continue

       return
       end

C***************************************************************C
C write out phasor values to a file.                            C
C***************************************************************C

       SUBROUTINE write_phasors

       implicit none
       include 'common.f'

C*****pm: the current mode being written out.

       integer pm,i,k,fi
       complex temp

       write(6,*) 'Writing out frequency data...'
       open(unit=9,file='fdata/info.dat',status='unknown',
      1     form='formatted')
       write(9,*) dt
       write(9,*) dz
       write(9,*) N
       write(9,*) inc_ang
       write(9,*) gd
       write(9,*) sdev
       write(9,*) rcsz1
       write(9,*) rcsz2
       write(9,*) mheight
       write(9,*) mode_start
       write(9,*) mode_end
       write(9,*) modulate
       write(9,*) modfreq
       write(9,*) num_freqs
       do 130 fi=minf,maxf
          write(9,*) freqlist(fi,1), freqlist(fi,2)
 130   continue
       close(unit=9)

 100   format(F12.8, ' ', F12.8)

       open(unit=9,file='fdata/feru.dat',status='unknown',
      1     form='formatted')
       do 10 pm = mode_start,mode_end
          do 20 i = 1,2*mheight+rcsz2-rcsz1-1
             do 30 k = minf,maxf,stepf
                temp = feru(pm,i,k)
                write(9, *) real(temp), aimag(temp)
 30          continue
 20       continue
 10    continue
       close(unit=9)

       open(unit=9,file='fdata/ferv.dat',status='unknown',
      1     form='formatted')
       do 101 pm = mode_start,mode_end
          do 201 i = 1,2*mheight + rcsz2 - rcsz1 - 1
             do 301 k = minf,maxf,stepf
                temp = ferv(pm,i,k)
                write(9, *) real(temp), aimag(temp)
 301         continue
 201      continue
 101   continue
       close(unit=9)

       open(unit=9,file='fdata/fezu.dat',status='unknown',
      1     form='formatted')
       do 102 pm = mode_start,mode_end
          do 202 i = 1,2*mheight + rcsz2 - rcsz1 - 1
             do 302 k = minf,maxf,stepf
                temp = fezu(pm,i,k)
                write(9, *) real(temp), aimag(temp)
 302         continue
 202      continue
 102   continue
       close(unit=9)

       open(unit=9,file='fdata/fezv.dat',status='unknown',
      1     form='formatted')
       do 103 pm = mode_start,mode_end
          do 203 i = 1,2*mheight + rcsz2 - rcsz1 - 1
             do 303 k = minf,maxf,stepf
                temp = fezv(pm,i,k)
                write(9, *) real(temp), aimag(temp)
 303         continue
 203      continue
 103   continue
       close(unit=9)

       open(unit=9,file='fdata/fephiu.dat',status='unknown',
      1     form='formatted')
       do 104 pm = mode_start,mode_end
          do 204 i = 1,2*mheight + rcsz2 - rcsz1 - 1
             do 304 k = minf,maxf,stepf
                temp = fephiu(pm,i,k)
                write(9, *) real(temp), aimag(temp)
 304         continue
 204      continue
 104   continue
       close(unit=9)

       open(unit=9,file='fdata/fephiv.dat',status='unknown',
      1     form='formatted')
       do 105 pm = mode_start,mode_end
          do 205 i = 1,2*mheight + rcsz2 - rcsz1 - 1
             do 305 k = minf,maxf,stepf
                temp = fephiv(pm,i,k)
                write(9, *) real(temp), aimag(temp)
 305         continue
 205      continue
 105   continue
       close(unit=9)

       open(unit=9,file='fdata/fhru.dat',status='unknown',
      1     form='formatted')
       do 106 pm = mode_start,mode_end
          do 206 i = 1,2*mheight + rcsz2 - rcsz1 - 1
             do 306 k = minf,maxf,stepf
                temp = fhru(pm,i,k)
                write(9, *) real(temp), aimag(temp)
 306         continue
 206      continue
 106   continue
       close(unit=9)

       open(unit=9,file='fdata/fhrv.dat',status='unknown',
      1     form='formatted')
       do 107 pm = mode_start,mode_end
          do 207 i = 1,2*mheight + rcsz2 - rcsz1 - 1
             do 307 k = minf,maxf,stepf
                temp = fhrv(pm,i,k)
                write(9, *) real(temp), aimag(temp)
 307         continue
 207      continue
 107   continue
       close(unit=9)

       open(unit=9,file='fdata/fhzu.dat',status='unknown',
      1     form='formatted')
       do 108 pm = mode_start,mode_end
          do 208 i = 1,2*mheight + rcsz2 - rcsz1 - 1
             do 308 k = minf,maxf,stepf
                temp = fhzu(pm,i,k)
               write(9, *) real(temp), aimag(temp)
 308         continue
 208      continue
 108   continue
       close(unit=9)

       open(unit=9,file='fdata/fhzv.dat',status='unknown',
      1     form='formatted')
        do 109 pm = mode_start,mode_end
          do 209 i = 1,2*mheight + rcsz2 - rcsz1 - 1
             do 309 k = minf,maxf,stepf
                temp = fhzv(pm,i,k)
                write(9, *) real(temp), aimag(temp)
 309         continue
 209      continue
 109   continue
       close(unit=9)

       open(unit=9,file='fdata/fhphiu.dat',status='unknown',
      1     form='formatted')
       do 110 pm = mode_start,mode_end
          do 210 i = 1,2*mheight + rcsz2 - rcsz1 - 1
             do 310 k = minf,maxf,stepf
                temp = fhphiu(pm,i,k)
                write(9, *) real(temp), aimag(temp)
```

```
310         continue
210       continue
110    continue
       close(unit=9)

       open(unit=9,file='fdata/fhphiv.dat',status='unknown',
     1      form='formatted')
       do 111 pm = mode_start,mode_end
          do 211 i = 1,2*mheight + rcsz2 - rcsz1 - 1
             do 311 k = minf,maxf,stepf
                temp = fhphiv(pm,i,k)
                write(9, *) real(temp), aimag(temp)
311         continue
211       continue
111    continue
       close(unit=9)

      return
      end

C*************************************************************C
C read out phasor values from a file.                         C
C*************************************************************C

      SUBROUTINE read_phasors

      implicit none
      include 'common.f'

C*****pm: the current mode being written out.

      integer pm,i,k,fi
      real tempr, tempi

      write(6,*) 'Reading in frequency data...'

      open(unit=9,file='fdata/info.dat',status='old',
     1     form='formatted')
      read(9,*) dt
      read(9,*) dz
      read(9,*) N
      read(9,*) low_freq
      read(9,*) high_freq
      read(9,*) dfreq
      read(9,*) inc_ang
      read(9,*) gd
      read(9,*) sdev
      read(9,*) rcsz1
      read(9,*) rcsz2
      read(9,*) mheight
      read(9,*) mode_start
      read(9,*) mode_end
      read(9,*) modulate
      read(9,*) modfreq
      read(9,*) num_freqs
      do 130 fi=1,num_freqs
         read(9,*) freqlist(fi,1), freqlist(fi,2)
130    continue
      close(unit=9)

      minf = 0
      maxf = int((high_freq-low_freq)/dfreq)
      stepf = 1

      print *,low_freq,high_freq,dfreq
      print *,minf,maxf,stepf

      if (nm.gt.mode_start) then
         write(6,*)
         print *,'nm =',nm,' is greater than the starting mode'
         print *,'number', mode_start, '.  Adjust the nm parameter'
         enough_memory = .FALSE.
      end if

      if (mm.lt.mode_end) then
         write(6,*)
         print *,'mm =',mm,' is less than the ending mode'
         print *,'number', mode_end, '.  Adjust the mm parameter'
         print *,'in the common.f file'
         enough_memory = .FALSE.
      end if

      if ((maxf-minf+1).gt.MAX_FREQS) then
         print *, 'too many frequencies, lower number of freq'
         print *, 'from ', maxf-minf+1, ' to less than ',MAX_FREQS
         print *, 'or increase MAX_FREQS variable in the common.f file.'
         enough_memory = .FALSE.
      end if

      if ((2*mheight+rcsz2-rcsz1-1).gt.mxdp) then
         print *,'error not enough memory for RCS components'
         print *,'set the parameter mxdp higher than',
     1        2*mheight+rcsz2-rcsz1-1
         enough_memory = .FALSE.
      end if

      if (.NOT.enough_memory) then
         print *,'Not enough memory, must allocate more by altering'
         print *,'parms in common.f file'
         stop
      end if

      print *, 'reading in freq data'

100   format(F12.8, ' ', F12.8)

      open(unit=9,file='fdata/feru.dat',status='old',
     1     form='formatted')
      do 10 pm = mode_start,mode_end
         do 20 i = 1,2*mheight+rcsz2-rcsz1-1
            do 30 k = minf,maxf,stepf
               read(9, *) tempr, tempi
               feru(pm,i,k) = tempr + (0,1)*tempi
30          continue
20       continue
10    continue
      close(unit=9)

      open(unit=9,file='fdata/ferv.dat',status='old',
     1     form='formatted')
      do 101 pm = mode_start,mode_end
         do 201 i = 1,2*mheight + rcsz2 - rcsz1 - 1
            do 301 k = minf,maxf,stepf
               read(9, *) tempr, tempi
               ferv(pm,i,k) = tempr + (0,1)*tempi
301         continue
201      continue
101   continue
      close(unit=9)

      open(unit=9,file='fdata/fezu.dat',status='old',
     1     form='formatted')
      do 102 pm = mode_start,mode_end
         do 202 i = 1,2*mheight + rcsz2 - rcsz1 - 1
            do 302 k = minf,maxf,stepf
               read(9, *) tempr, tempi
               fezu(pm,i,k) = tempr + (0,1)*tempi
302         continue
202      continue
102   continue
      close(unit=9)

      open(unit=9,file='fdata/fezv.dat',status='old',
     1     form='formatted')
      do 103 pm = mode_start,mode_end
         do 203 i = 1,2*mheight + rcsz2 - rcsz1 - 1
            do 303 k = minf,maxf,stepf
               read(9, *) tempr, tempi
               fezv(pm,i,k) = tempr + (0,1)*tempi
303         continue
203      continue
103   continue
      close(unit=9)

      open(unit=9,file='fdata/fephiu.dat',status='old',
     1     form='formatted')
      do 104 pm = mode_start,mode_end
         do 204 i = 1,2*mheight + rcsz2 - rcsz1 - 1
            do 304 k = minf,maxf,stepf
               read(9, *) tempr, tempi
               fephiu(pm,i,k) = tempr + (0,1)*tempi
304         continue
204      continue
104   continue
      close(unit=9)

      open(unit=9,file='fdata/fephiv.dat',status='old',
     1     form='formatted')
      do 105 pm = mode_start,mode_end
         do 205 i = 1,2*mheight + rcsz2 - rcsz1 - 1
            do 305 k = minf,maxf,stepf
               read(9, *) tempr, tempi
               fephiv(pm,i,k) = tempr + (0,1)*tempi
305         continue
205      continue
105   continue
      close(unit=9)

      open(unit=9,file='fdata/fhru.dat',status='old',
     1     form='formatted')
      do 106 pm = mode_start,mode_end
         do 206 i = 1,2*mheight + rcsz2 - rcsz1 - 1
            do 306 k = minf,maxf,stepf
               read(9, *) tempr, tempi
```

```
               fhru(pm,i,k) = tempr + (0,1)*tempi
306         continue
206      continue
106   continue
      close(unit=9)

      open(unit=9,file='fdata/fhrv.dat',status='old',
     1     form='formatted')
      do 107 pm = mode_start,mode_end
         do 207 i = 1,2*mheight + rcsz2 - rcsz1 - 1
            do 307 k = minf,maxf,stepf
               read(9, *) tempr, tempi
               fhrv(pm,i,k) = tempr + (0,1)*tempi
307         continue
207      continue
107   continue
      close(unit=9)

      open(unit=9,file='fdata/fhzu.dat',status='old',
     1     form='formatted')
      do 108 pm = mode_start,mode_end
         do 208 i = 1,2*mheight + rcsz2 - rcsz1 - 1
            do 308 k = minf,maxf,stepf
               read(9, *) tempr, tempi
               fhzu(pm,i,k) = tempr + (0,1)*tempi
308         continue
208      continue
108   continue
      close(unit=9)

      open(unit=9,file='fdata/fhzv.dat',status='old',
     1     form='formatted')
      do 109 pm = mode_start,mode_end
         do 209 i = 1,2*mheight + rcsz2 - rcsz1 - 1
            do 309 k = minf,maxf,stepf
               read(9, *) tempr, tempi
               fhzv(pm,i,k) = tempr + (0,1)*tempi
309         continue
209      continue
109   continue
      close(unit=9)

      open(unit=9,file='fdata/fhphiu.dat',status='old',
     1     form='formatted')
      do 110 pm = mode_start,mode_end
         do 210 i = 1,2*mheight + rcsz2 - rcsz1 - 1
            do 310 k = minf,maxf,stepf
               read(9, *) tempr, tempi
               fhphiu(pm,i,k) = tempr + (0,1)*tempi
310         continue
210      continue
110   continue
      close(unit=9)

      open(unit=9,file='fdata/fhphiv.dat',status='old',
     1     form='formatted')
      do 111 pm = mode_start,mode_end
         do 211 i = 1,2*mheight + rcsz2 - rcsz1 - 1
            do 311 k = minf,maxf,stepf
               read(9, *) tempr, tempi
               fhphiv(pm,i,k) = tempr + (0,1)*tempi
311         continue
211      continue
111   continue
      close(unit=9)

      print *, 'Currently you are calculating the RCS at',maxf-minf+1
      print *, 'frequencies.  Enter the new step size (1 for all freq):'
      read(5,*) stepf

      return
      end

C*****************************************************************C
C Calculate far-field E and H fields using Huygens' Principle   C
C*****************************************************************C

      subroutine calc_rcs

      implicit none
      include 'common.f'

      real besselj, kwave, rho, kps, cz, RCS, RCSDB
      real obs_phi, obs_theta, eincsq, temp, targ
      real cosp, sinp, cost, sint, sinmp, cosmp, PDIV, tempfreq
      integer pt_rB, pt_rB0, pt_z1, pt_z2, pt_rC, pt_rC0,t,phase_z
      integer pt_index, freq_index, mode_index

      real dp_kwave, dutheta, tempang, dp_obs_theta
      real kps_tole

      complex Escat_theta_A, Escat_theta_B, Escat_theta_C,
     1     einc(1:MAX_FREQS),
     1     At, Escat_phi_A, Escat_phi_B, Escat_phi_C, Ap, A, uniti, I1,
     2     I3, I5, c1, c2, c3, c4, c5, RCSc, eincc
      complex ferup, fervp, fephiup, fephivp, fezup, fezvp,
     1     fhrup, fhrvp, fhphiup, fhphivp, fhzup, fhzvp

      character filnam*1024, frmt*30
      integer ilen

      parameter(PDIV=1.0,uniti=(0.0,1.0),kps_tole=1.0e-7)

      write(6,*) 'Calculating RCS...'

      ilen = index(dbase,' ') - 1
      write(frmt,'(a2,i4,a4)') '(a',ilen,',a8)'
      write(filnam,frmt)dbase,'/rcs.dat'

      open(unit=9,file='rcs.dat',status='unknown',form='formatted')
      open(unit=12,file='scat.dat',status='unknown',form='formatted')

C*****Some reference points to define
C*****pt_z1 index of first point of integral A
C*****pt_z2 index of last point of integral A
C*****pt_rB index of first point of integral B -left side
C*****pt_rB0 index of last point of integral B -left side
C*****pt_rC index of first point of integral C -right side
C*****pt_rC0 index of last point of integral C -right side

      pt_rB = 1
      pt_rB0 = mheight
      pt_z1 = mheight
      pt_z2 = mheight + rcsz2 - rcsz1
C*****low point (i.e. right side botom corner)
      pt_rC = 2*mheight + rcsz2 - rcsz1 - 1
C*****high point (i.e. right side top corner)
      pt_rC0 = mheight + rcsz2 - rcsz1

      print *,pt_rB,pt_rB0,pt_z1,pt_z2,pt_rC,pt_rC0

      do 1 freq_index = 1,MAX_FREQS
         einc(freq_index) = 0.0
 1    continue

C*****Calculate DFT of incident field for RCS calculation.
      do 5 t = 1,N
         targ = (t*dt-gd)+(rcsz1*dz*cos(inc_ang)+10*dz*sin(inc_ang))/c
         temp=((Ehg**2)+(Evg**2))*(1/(sqrt(2*pi)))*exp(-(targ**2.0)/
     1        ((sdev)**2.0))*((sin(2*pi*modfreq*targ))*modulate+
     2        abs(modulate-1))*5.0

         do 8 freq_index=minf,maxf,stepf
c            tempfreq = low_freq + freq_index*dfreq
            tempfreq = freqlist(freq_index,1)
c            print *,tempfreq
            einc(freq_index)=einc(freq_index)+temp*exp(2*pi*uniti*
     1           tempfreq*dt*t)*dt
 8       continue
 5    continue

      do 10 freq_index=minf,maxf,stepf
c        print *,minf,maxf,freq_index
         eincc = einc(freq_index)
         eincsq = (abs(einc(freq_index)))**2.0
c         kwave = ((low_freq+freq_index*dfreq)/c)*(2*pi)
         kwave = (freqlist(freq_index,1)/c)*(2*pi)
         if (calc_bist) then
            dp_kwave = kwave
            dutheta = dtheta
         else
c            print *,freq_index,mono_nang,int((freq_index-1)/
c     1           ((mono_nang+1)/2))+1
            dp_kwave = (freqlist(mono_freq_ind(int((freq_index-1)/
     1           ((mono_nang+1)/2))+1),1)/c)*(2*pi)
            tempang = dtheta*(freq_index-mono_freq_ind(int((freq_index
     1           -1)/((mono_nang+1)/2))+1))
            low_theta = real(inc_ang/pi*180-tempang*2)
            high_theta = real(inc_ang/pi*180+tempang*2)
            dutheta = high_theta-low_theta
            if (abs(dutheta).lt.eps) dutheta = 1.0
c            print *,dtheta,tempang,low_theta,high_theta,
c     1           (inc_ang/pi*180+low_theta)/2.,
c     2           (inc_ang/pi*180+high_theta)/2.
         end if

         do 20 obs_phi=low_phi,high_phi,dphi
            sinp = sin(obs_phi/180*pi)
            cosp = cos(obs_phi/180*pi)

            do 30 obs_theta=low_theta,high_theta,dutheta
               if (calc_bist) then
                  dp_obs_theta = obs_theta
               else
```

```
                dp_obs_theta = (inc_ang/pi*180+obs_theta)/2.0
              end if
              sint = sin(obs_theta/180*pi)
              cost = cos(obs_theta/180*pi)

C***************Initialize integral values

              Escat_theta_A = 0.0
              Escat_phi_A = 0.0
              Escat_theta_B = 0.0
              Escat_phi_B = 0.0
              Escat_theta_C = 0.0
              Escat_phi_C = 0.0

              do 40 mode_index = nm,mm
                 sinmp = sin(mode_index*obs_phi/180*pi)
                 cosmp = cos(mode_index*obs_phi/180*pi)

C******************Three different integrals to evaluate
                 c3 = 2*pi*exp(uniti*mode_index*1.5*pi)
                 c4 = 2*pi*exp(uniti*(mode_index+1)*1.5*pi)

C******************Integral A: z1 --> z2 -center integral at r0
                 rho = (mheight - 1) * dz
                 kps = kwave * rho * sint

                    if (abs(kps).lt.kps_tole) then
                       if (mode_index.eq.1) then
                          I1=0.0
                          I3=pi
                          I5=pi
                       else
                          I1=0.0
                          I3=0.0
                          I5=0.0
                       end if
                       if (mode_index.eq.0) then
                          I1=2*pi
                       end if
                    else
                       c2 = 2.0*pi*uniti*mode_index/kps
                       c5 = c2*exp(uniti*mode_index*1.5*pi)
                       I1 = c3*besselj(kps,mode_index)
                       I3 = c4*besselj(kps,mode_index+1)+c5*
     1                      besselj(kps,mode_index)
                       I5 = c5*besselj(kps,mode_index)
                    end if

                 do 50 pt_index = pt_z1,pt_z2
                    ferup = feru(mode_index,pt_index,freq_index)*cosmp
     1                    +ferv(mode_index,pt_index,freq_index)*sinmp
                    fervp = ferv(mode_index,pt_index,freq_index)*cosmp
     1                    -feru(mode_index,pt_index,freq_index)*sinmp
                    fezup = fezu(mode_index,pt_index,freq_index)*cosmp
     1                    +fezv(mode_index,pt_index,freq_index)*sinmp
                    fezvp = fezv(mode_index,pt_index,freq_index)*cosmp
     1                    -fezu(mode_index,pt_index,freq_index)*sinmp
                    fephiup = fephiu(mode_index,pt_index,freq_index)*
     1                    cosmp+fephiv(mode_index,pt_index,freq_index)*
     1                    sinmp
                    fephivp = fephiv(mode_index,pt_index,freq_index)*
     1                    cosmp-fephiu(mode_index,pt_index,freq_index)*
     1                    sinmp
                    fhrup = fhru(mode_index,pt_index,freq_index)*cosmp
     1                    +fhrv(mode_index,pt_index,freq_index)*sinmp
                    fhrvp = fhrv(mode_index,pt_index,freq_index)*cosmp
     1                    -fhru(mode_index,pt_index,freq_index)*sinmp
                    fhzup = fhzu(mode_index,pt_index,freq_index)*cosmp
     1                    +fhzv(mode_index,pt_index,freq_index)*sinmp
                    fhzvp = fhzv(mode_index,pt_index,freq_index)*cosmp
     1                    -fhzu(mode_index,pt_index,freq_index)*sinmp
                    fhphiup = fhphiu(mode_index,pt_index,freq_index)*
     1                    cosmp+fhphiv(mode_index,pt_index,freq_index)*
     1                    sinmp
                    fhphivp = fhphiv(mode_index,pt_index,freq_index)*
     1                    cosmp-fhphiu(mode_index,pt_index,freq_index)*
     1                    sinmp
                    do 55 phase_z = 0,(int(PDIV)-1)
                       cz = (rcsz1+pt_index-pt_z1)*dz+phase_z*dz/PDIV
                       c1 = exp(-uniti*kwave*cz*cost)

                       Escat_theta_A = (dz/PDIV)*rho*c1*(-sint*fhphiup
     1                       *c3*besselj(kps, mode_index)+fezup*I3+
     2                       cost*fhzvp*I5)+Escat_theta_A

                       Escat_phi_A = (dz/PDIV)*rho*c1*(-fhzup*I3-sint*
     1                       fephiup*c3*besselj(kps,mode_index)+cost*
     2                       fezvp*I5)+Escat_phi_A

 55                 continue
 50                 continue
C*********************************************************

C******************Integral B: 0 --> r0 -left integral at z1

                 cz = rcsz1*dz
                 c1 = exp(-uniti*kwave*cz*cost)

                 do 60 pt_index = pt_rB, pt_rB0
                    ferup = feru(mode_index,pt_index,freq_index)*cosmp
     1                    +ferv(mode_index,pt_index,freq_index)*sinmp
                    fervp = ferv(mode_index,pt_index,freq_index)*cosmp
     1                    -feru(mode_index,pt_index,freq_index)*sinmp
                    fezup = fezu(mode_index,pt_index,freq_index)*cosmp
     1                    +fezv(mode_index,pt_index,freq_index)*sinmp
                    fezvp = fezv(mode_index,pt_index,freq_index)*cosmp
     1                    -fezu(mode_index,pt_index,freq_index)*sinmp
                    fephiup = fephiu(mode_index,pt_index,freq_index)*
     1                    cosmp+fephiv(mode_index,pt_index,freq_index)*
     1                    sinmp
                    fephivp = fephiv(mode_index,pt_index,freq_index)*
     1                    cosmp-fephiu(mode_index,pt_index,freq_index)*
     1                    sinmp
                    fhrup = fhru(mode_index,pt_index,freq_index)*cosmp
     1                    +fhrv(mode_index,pt_index,freq_index)*sinmp
                    fhrvp = fhrv(mode_index,pt_index,freq_index)*cosmp
     1                    -fhru(mode_index,pt_index,freq_index)*sinmp
                    fhzup = fhzu(mode_index,pt_index,freq_index)*cosmp
     1                    +fhzv(mode_index,pt_index,freq_index)*sinmp
                    fhzvp = fhzv(mode_index,pt_index,freq_index)*cosmp
     1                    -fhzu(mode_index,pt_index,freq_index)*sinmp
                    fhphiup = fhphiu(mode_index,pt_index,freq_index)*
     1                    cosmp+fhphiv(mode_index,pt_index,freq_index)*
     1                    sinmp
                    fhphivp = fhphiv(mode_index,pt_index,freq_index)*
     1                    cosmp-fhphiu(mode_index,pt_index,freq_index)*
     1                    sinmp

                    rho = (pt_index-1)*dz
                    kps = kwave * rho * sint
c                    print *,obs_theta,kps,sint

                    if (abs(kps).lt.kps_tole) then
                       if (mode_index.eq.1) then
                          I1=0.0
                          I3=pi
                          I5=pi
                       else
                          I1=0.0
                          I3=0.0
                          I5=0.0
                       end if
                       if (mode_index.eq.0) then
                          I1=2*pi
                       end if
                    else
                       c2 = 2.0*pi*uniti*mode_index/kps
                       c5 = c2*exp(uniti*mode_index*1.5*pi)
                       I1 = c3*besselj(kps,mode_index)
                       I3 = c4*besselj(kps,mode_index+1)+c5*
     1                      besselj(kps,mode_index)
                       I5 = c5*besselj(kps,mode_index)
                    end if

                    Escat_theta_B = -dz*rho*c1*(-cost*fhphiup*I3-ferup
     1                    *I3-cost*fhrvp*I5+fephivp*I5)+Escat_theta_B

                    Escat_phi_B = -dz*rho*c1*((fhrup-cost*fephiup)*I3+
     1                    (-fhphivp-cost*fervp)*I5)+Escat_phi_B

 60                 continue
C*********************************************************

C******************Integral C: 0 --> r0 -right integral at z2

                 cz = rcsz2*dz
                 c1 = exp(-uniti*kwave*cz*cost)

                 do 70 pt_index = pt_rC0, pt_rC
                    ferup = feru(mode_index,pt_index,freq_index)*cosmp
     1                    +ferv(mode_index,pt_index,freq_index)*sinmp
                    fervp = ferv(mode_index,pt_index,freq_index)*cosmp
     1                    -feru(mode_index,pt_index,freq_index)*sinmp
                    fezup = fezu(mode_index,pt_index,freq_index)*cosmp
     1                    +fezv(mode_index,pt_index,freq_index)*sinmp
                    fezvp = fezv(mode_index,pt_index,freq_index)*cosmp
     1                    -fezu(mode_index,pt_index,freq_index)*sinmp
                    fephiup = fephiu(mode_index,pt_index,freq_index)*
     1                    cosmp+fephiv(mode_index,pt_index,freq_index)*
     1                    sinmp
                    fephivp = fephiv(mode_index,pt_index,freq_index)*
     1                    cosmp-fephiu(mode_index,pt_index,freq_index)*
     1                    sinmp
                    fhrup = fhru(mode_index,pt_index,freq_index)*cosmp
```

```
     1                    +fhrv(mode_index,pt_index,freq_index)*sinmp
                     fhrvp = fhrv(mode_index,pt_index,freq_index)*cosmp
     1                    -fhru(mode_index,pt_index,freq_index)*sinmp
                     fhzup = fhzu(mode_index,pt_index,freq_index)*cosmp
     1                    +fhzv(mode_index,pt_index,freq_index)*sinmp
                     fhzvp = fhzv(mode_index,pt_index,freq_index)*cosmp
     1                    -fhzu(mode_index,pt_index,freq_index)*sinmp
                     fhphiup = fhphiu(mode_index,pt_index,freq_index)*
     1                    cosmp+fhphiv(mode_index,pt_index,freq_index)*
     1                    sinmp
                     fhphivp = fhphiv(mode_index,pt_index,freq_index)*
     1                    cosmp-fhphiu(mode_index,pt_index,freq_index)*
     1                    sinmp
                     rho = (pt_rC-pt_index)*dz
                     kps = kwave * rho * sint

                     if (abs(kps).lt.kps_tole) then
                        if (mode_index.eq.1) then
                           I1=0.0
                           I3=pi
                           I5=pi
                        else
                           I1=0.0
                           I3=0.0
                           I5=0.0
                        end if
                        if (mode_index.eq.0) then
                           I1=2*pi
                        end if
                     else
                        c2 = 2.0*pi*uniti*mode_index/kps
                        c5 = c2*exp(uniti*mode_index*1.5*pi)
                        I1 = c3*besselj(kps,mode_index)
                        I3 = c4*besselj(kps,mode_index+1)+c5*
     1                       besselj(kps,mode_index)
                        I5 = c5*besselj(kps,mode_index)
                     end if

                     Escat_theta_C = dz*rho*c1*(-cost*fhphiup*I3-ferup*
     1                    I3-cost*fhrvp*I5+fephivp*I5)+Escat_theta_C

                     Escat_phi_C = dz*rho*c1*((fhrup-cost*fephiup)*I3+
     3                    (-fhphivp-cost*fervp)*I5)+Escat_phi_C

 70               continue
C*************************************************************

 40            continue

               At = Escat_theta_A + Escat_theta_B + Escat_theta_C

               Ap = Escat_phi_A + Escat_phi_B + Escat_phi_C

               A = At*((cost*cost*cosp+sint*sint)*Ehg+(cost*sinp)*Evg)
     1              +Ap*((-cost*sinp)*Ehg+cosp*Evg)

               RCSc = ((kwave**2)*(A**2))/(4.0*pi*eincc**2)
               RCS = ((kwave**2)*((abs(A))**2))/(4.0*pi*eincsq)
               if (abs(RCS).lt.1e-7) then
                  RCSDB = -200.0
               else
                  RCSDB = 10*ALOG10(RCS)
               end if

               write(9,*) dp_kwave,obs_phi,dp_obs_theta,RCSDB,
     1              abs(RCSc),atan2(imag(RCSc),real(RCSc))


 30         continue
 20      continue
 10    continue

 99    format(F25.15,'    ',F25.15)

       close(unit=9)
       close(unit=12)
       return
       end
```

The following, **lib.f**, contains several output subroutines including those to generate an FD-TD movie.

```
c*************************************************************c
c This is the library file for the BOR program.  It contains  c
c many of the subroutines needed to run the BOR FDTD program  c
c*************************************************************c

c*************************************************************c
```

```
      SUBROUTINE PLOTB(X,Y,N,NC,NR)
C
C WRITTEN 2/14/74   BY J. M. PUTNAM       DEPT 220     X23877
C
C THIS ROUTINE PRODUCES A LINEAR XY PLOT.
C    N IS THE NUMBER OF POINTS TO BE PLOTTED.
C    NR IS THE NUMBER OF ROWS TO BE USED FOR THE Y-AXIS.
C    NC IS THE NUMBER OF COLUMNS TO BE USED FOR THE X-AXIS.
C       NOTE, NC-1 MUST BE DIVISIBLE BY 10 AND LESS THAN 102.
C

      REAL X(161),Y(161),HEAD(10)
      INTEGER  LINE(101),BLANK,STAR
      DATA BLANK,STAR /1H ,1H*/
      N10=(NC-1)/10
      WRITE(9,500)
 500  FORMAT(//,17H1BODY COORDINATES)
      WRITE(9,504)
      XMIN=X(1)
      XMAX=X(1)
      YMIN=Y(1)
      YMAX=Y(1)
      DO 6 I=1,N
      IF(X(I).LT.XMIN) XMIN=X(I)
      IF(X(I).GT.XMAX) XMAX=X(I)
      IF(Y(I).LT.YMIN) YMIN=Y(I)
      IF(Y(I).GT.YMAX) YMAX=Y(I)
 6    CONTINUE
      DEL=XMAX-XMIN
      IF(YMAX-YMIN.GT.DEL) DEL=YMAX-YMIN
      XMAX=XMIN+DEL
      YMAX=YMIN+DEL
      DO 5 I =1,N10
      Z=I
    5 HEAD(I)=(XMAX-XMIN)*Z/N10+XMIN
      DY=(YMAX-YMIN)/(NR-1)
      Z=YMAX+DY
      YL=Z-DY/2.
      DO 7 J=1,NR
      DO 8 K=1,NC
 8    LINE(K)=BLANK
      Z=Z-DY
      YU=YL
      YL=Z-DY/2.
      DO 9 I=1,N
      IF(Y(I).GE.YU) GO TO 9
      IF(Y(I).LT.YL) GO TO 9
      K=(X(I)-XMIN)/(XMAX-XMIN)*(NC-1)+1.5
      IF(K.GT.NC) K=NC
      LINE(K)=STAR
 9    CONTINUE
      WRITE(9,508) Z,(LINE(K),K=1,NC)
 7    CONTINUE
      WRITE(9,504)
      WRITE(9,3002)
      WRITE(9,507) XMIN,(HEAD(I),I=1,N10)
C     *************************
      RETURN
  504 FORMAT ( 1X, 14(1H-), 1H., 10(5H----.), 1H- )
  507 FORMAT(10X,11(F10.4))
  508 FORMAT (1X, F12.4,1X, 1HI, 51A1, 1HI )
 3002 FORMAT(4X,7HRB / ZB,4X,1HI,5(9X,1HI))
      END
```

```
c*****************************************************************c
c setups the display for the EM "movie"                           c
c*****************************************************************c

      subroutine setup_movie(totsteps)

      implicit none
      include 'common.f'
c      character*72 mhname, mfname
      integer totsteps

      open(unit=4,file=mfname,status='unknown',form='formatted')
      open(unit=7,file=mhname,status='unknown',form='formatted')

c     *** variables for wrtraw movie generator        ***
c     *** need to call rw2pnm on raw files to process ***

      base='/willow/home/pacheco/movie_data/frame'

      write(4,*) maxz
      write (4,81) 'new.image'
      write(7,296) maxz, maxr, 64, totsteps, 'new.image.Z'

      write (7,*) 1
      write (7,81) 'a '
      write (7,81) 'b '
      close(unit=7)
```

```
 81   format(a)
 296  format(i4,x,i4,x,i2,x,i4,x,a)

      return
      end

c*************************************************************c
c subroutine to select display for er, ez, or ephi field      c
c based on input movie_num                                    c
c*************************************************************c

      subroutine movie(mode,ms)

      implicit none
      include 'common.f'
      integer mode,ms
      logical b
      character a(1:mz+2*pmldepth)

      if (movie_type.eq.1) THEN
         if (movie_num.eq.1) call movie_gen(erphil,erzl,erphir,erzr,
     1        erphit,erzt,er,4*ms,mode,a)
         if (movie_num.eq.2) call movie_gen(ezphil,ezrl,ezphir,ezrr,
     1        ezphit,ezrt,ez,6*ms,mode,a)
         if (movie_num.eq.3) call movie_gen(ephirl,ephizl,ephirr,ephizr,
     1        ephirt,ephizt,ephi,2*ms,mode,a)

         if (movie_num.eq.4) call movie_gen(hrphil,hrzl,hrphir,hrzr,
     1        hrphit,hrzt,hr,7*ms,mode,a)
         if (movie_num.eq.5) call movie_gen(hzphil,hzrl,hzphir,hzrr,
     1        hzphit,hzrt,hz,9*ms,mode,a)
         if (movie_num.eq.6) call movie_gen(hphirl,hphizl,hphirr,hphizr,
     1        hphirt,hphizt,hphi,11*ms,mode,a)
      END IF

c      b=movie_type.eq.2.OR.movie_type.eq.3

c      if (b.AND.time.gt.0) THEN
c         if (movie_num.eq.1) call wrtraw(erphil,erzl,erphir,erzr,
c     1        erphit,erzt,er,4*ms,mode,a)
c         if (movie_num.eq.2) call wrtraw(ezphil,ezrl,ezphir,ezrr,
c     1        ezphit,ezrt,ez,6*ms,mode,a)
c         if (movie_num.eq.3) call wrtraw(ephirl,ephizl,ephirr,ephizr,
c     1        ephirt,ephizt,ephi,2*ms,mode,a)
c      END IF

      return
      end

c*************************************************************c
c subroutine to display a "movie" of the Electric or          c
c magnetic fields with N time steps: this does the Er field   c
c*************************************************************c

      subroutine movie_gen(eabl,eacl,eabr,eacr,eabt,eact,ea,inceq,
     1     mode,a)

      implicit none
      include 'common.f'

      real maxtest
      character a(1:maxz)
      real eabl(1:pmldepth+1,0:pmldepth+mr+1),
     1     eacl(1:pmldepth+1,0:pmldepth+mr+1),
     2     eabr(1:pmldepth+1,0:pmldepth+mr+1),
     3     eacr(1:pmldepth+1,0:pmldepth+mr+1),
     4     eabt(1:mz,1:pmldepth+1),
     5     eact(1:mz,1:pmldepth+1),
     6     ea(1:mz,1:mr)

      real inc,gquad

      logical inside
      integer i,k,ia,inceq,mode
c      integer nsplit,ns,l
      integer hlevels,numcolors1,center,topcolor,nctshift,ngray
c     topcolor: location of top of colorbar
c     numcolors1: 1 less than # of colors in colorbar
      parameter (numcolors1=128,topcolor=243)
c     nctshift: = color table shift
      parameter(nctshift = topcolor-numcolors1)
      parameter(ngray = topcolor-numcolors1-1)
      parameter (hlevels=numcolors1/4)
      parameter (center = 2*hlevels + 0.5)
      character frmt*30

      write(frmt,'(a1,i4,a5)') '(', (maxz), '(a1))'
      maxtest = -1.0
      do 50 i=maxr,1,-1
         do 60 k=1,maxz
            if (abs(ea(k,i)).gt.maxtest) maxtest = abs(ea(k,i))
            if (scattot(k,i).ne.15) then
c              inc = -gquad(0.0,2*pi,inceq,mode,time*dt,(i-1)*dz,
c     1           k*dz,inc_ang)
               inc = 0.0
               ia=int((inc+ea(k,i))*hlevels+center+0.5)
            else
               ia=int(ea(k,i)*hlevels+center+0.5)
            end if
            if (ia .lt. 0) ia=0
            if (ia .gt. numcolors1) ia=numcolors1
            a(k)=char(ia+nctshift)
 60      continue

         write(4,frmt) a
 50   continue

      print *,'maxtest=',maxtest

      return
      end

c*************************************************************c
c The subroutine floors a real number, ie truncates it        c
c*************************************************************c

      integer function floor(x)

      implicit none
      real x
      integer temp

      if (x.lt.0) then
         temp = x
         floor = temp - 1
      else
         floor = int(x)
      end if

      return
      end

c*************************************************************c
c The subroutine swaps two integer numbers                    c
c*************************************************************c

      subroutine swap(i,j)

      implicit none
      integer temp,i,j

      temp=i
      i=j
      j=temp

      return
      end

c*************************************************************c
c The subroutine swaps two real numbers                       c
c*************************************************************c

      subroutine swapr(i,j)

      implicit none
      real temp,i,j

      temp=i
      i=j
      j=temp

      return
      end
```

The following, **besselj.f**, contains the subroutine for calculating bessel functions of integer order and real arguements that is used by the BOR FD-TD and BOR PWE programs.

```
c***********************************************************************
c BESSELJ computes bessel function of the first kind, order n, for
c    real argument x. Uses fortran numerical recipe routines.
c***********************************************************************

      real function besselj(x,n)

      implicit none

      real x,bessj0,bessj1,bessj
      integer n
```

```
      if (n.eq.0) besselj=bessj0(x)
      if (n.eq.1) besselj=bessj1(x)
      if (n.gt.1) besselj=bessj(n,x)
c      besselj = 0.01

      RETURN
      END

C  (C) Copr. 1986-92 Numerical Recipes Software ]2+r9,6)!.
      FUNCTION bessj(n,x)
      INTEGER n,IACC
      REAL bessj,x,BIGNO,BIGNI
      PARAMETER (IACC=40,BIGNO=1.e10,BIGNI=1.e-10)
CU    USES bessj0,bessj1
      INTEGER j,jsum,m
      REAL ax,bj,bjm,bjp,sum,tox,bessj0,bessj1
      if(n.lt.2)pause 'bad argument n in bessj'
      ax=abs(x)
      if(ax.eq.0.)then
        bessj=0.
      else if(ax.gt.float(n))then
        tox=2./ax
        bjm=bessj0(ax)
        bj=bessj1(ax)
        do 11 j=1,n-1
          bjp=j*tox*bj-bjm
          bjm=bj
          bj=bjp
11      continue
        bessj=bj
      else
        tox=2./ax
        m=2*((n+int(sqrt(float(IACC*n))))/2)
        bessj=0.
        jsum=0
        sum=0.
        bjp=0.
        bj=1.
        do 12 j=m,1,-1
          bjm=j*tox*bj-bjp
          bjp=bj
          bj=bjm
          if(abs(bj).gt.BIGNO)then
            bj=bj*BIGNI
            bjp=bjp*BIGNI
            bessj=bessj*BIGNI
            sum=sum*BIGNI
          endif
          if(jsum.ne.0)sum=sum+bj
          jsum=1-jsum
          if(j.eq.n)bessj=bjp
12      continue
        sum=2.*sum-bj
        bessj=bessj/sum
      endif
      if(x.lt.0..and.mod(n,2).eq.1)bessj=-bessj
      return
      END
C  (C) Copr. 1986-92 Numerical Recipes Software ]2+r9,6)!.


      FUNCTION bessj0(x)
      REAL bessj0,x
      REAL ax,xx,z
      DOUBLE PRECISION p1,p2,p3,p4,p5,q1,q2,q3,q4,q5,r1,r2,r3,r4,r5,r6,
     *s1,s2,s3,s4,s5,s6,y
      SAVE p1,p2,p3,p4,p5,q1,q2,q3,q4,q5,r1,r2,r3,r4,r5,r6,s1,s2,s3,s4,
     *s5,s6
      DATA p1,p2,p3,p4,p5/1.d0,-.1098628627d-2,.2734510407d-4,
     *-.2073370639d-5,.2093887211d-6/, q1,q2,q3,q4,q5/-.1562499995d-1,
     *.1430488765d-3,-.6911147651d-5,.7621095161d-6,-.934945152d-7/
      DATA r1,r2,r3,r4,r5,r6/57568490574.d0,-13362590354.d0,
     *651619640.7d0,-11214424.18d0,77392.33017d0,-184.9052456d0/,s1,s2,
     *s3,s4,s5,s6/57568490411.d0,1029532985.d0,9494680.718d0,
     *59272.64853d0,267.8532712d0,1.d0/
      if(abs(x).lt.8.)then
        y=x**2
        bessj0=(r1+y*(r2+y*(r3+y*(r4+y*(r5+y*r6))))/(s1+y*(s2+y*(s3+y*
     *(s4+y*(s5+y*s6))))
      else
        ax=abs(x)
        z=8./ax
        y=z**2
        xx=ax-.785398164
        bessj0=sqrt(.636619772/ax)*(cos(xx)*(p1+y*(p2+y*(p3+y*(p4+y*
     *p5))))-z*sin(xx)*(q1+y*(q2+y*(q3+y*(q4+y*q5)))))
      endif
      return
      END
C  (C) Copr. 1986-92 Numerical Recipes Software ]2+r9,6)!.

      FUNCTION bessj1(x)
      REAL bessj1,x
      REAL ax,xx,z
      DOUBLE PRECISION p1,p2,p3,p4,p5,q1,q2,q3,q4,q5,r1,r2,r3,r4,r5,r6,
     *s1,s2,s3,s4,s5,s6,y
      SAVE p1,p2,p3,p4,p5,q1,q2,q3,q4,q5,r1,r2,r3,r4,r5,r6,s1,s2,s3,s4,
     *s5,s6
      DATA r1,r2,r3,r4,r5,r6/72362614232.d0,-7895059235.d0,
     *242396853.1d0,-2972611.439d0,15704.48260d0,-30.16036606d0/,s1,s2,
     *s3,s4,s5,s6/144725228442.d0,2300535178.d0,18583304.74d0,
     *99447.43394d0,376.9991397d0,1.d0/
      DATA p1,p2,p3,p4,p5/1.d0,.183105d-2,-.3516396496d-4,
     *.2457520174d-5,-.240337019d-6/, q1,q2,q3,q4,q5/.04687499995d0,
     *-.2002690873d-3,.8449199096d-5,-.88228987d-6,.105787412d-6/
      if(abs(x).lt.8.)then
        y=x**2
        bessj1=x*(r1+y*(r2+y*(r3+y*(r4+y*(r5+y*r6))))/(s1+y*(s2+y*(s3+
     *y*(s4+y*(s5+y*s6)))))
      else
        ax=abs(x)
        z=8./ax
        y=z**2
        xx=ax-2.356194491
        bessj1=sqrt(.636619772/ax)*(cos(xx)*(p1+y*(p2+y*(p3+y*(p4+y*
     *p5))))-z*sin(xx)*(q1+y*(q2+y*(q3+y*(q4+y*q5)))))*sign(1.,x)
      endif
      return
      END
C  (C) Copr. 1986-92 Numerical Recipes Software ]2+r9,6)!.
```

The following, **common.f**, is used to create the common blocks that are included in most of the subroutines used by the BOR FD-TD program.

```
c*****************************************************************c
c This is the common file for the BOR FD-TD program.  It contains
c all the global variables and constants used in the program
c*****************************************************************c

integer mz, mr, maxpt, MAX_FREQS, mm, mxdp, nm, MAXCP,
1    MAX_STAIR_NODES, MAX_Z_CELLS, MAX_R_CELLS, MAX_RCS_NODES,
2    MAX_NODES

C***** ADJUSTABLE PARAMETERS TO ALLOCATE MEMORY NEEDED

parameter(mz = 650)
parameter(MAX_Z_CELLS = mz)
   parameter(mr = 300)
parameter(MAX_R_CELLS = mr)
parameter(maxpt = 1300)
parameter(nm = 0)
parameter(mm = 10)
parameter(mxdp = 800)
  parameter(MAX_FREQS = 105)
parameter(MAXCP = 4*maxpt)
parameter(MAX_STAIR_NODES=1460)
parameter(MAX_RCS_NODES=mxdp)
parameter(MAX_NODES=maxpt)

C*****  DO NOT CHANGE BELOW ************************************

real sigma_max,dz,freq,len,tole, dt, sdev
real Ehg,Evg,gd,modfreq,maxf_v,inc_ang,obj_height
real low_freq,high_freq,dfreq,sim_duration

real eta, mu, eps, c, pi

logical enough_memory

integer N, time, pmldepth, NP, maxz, maxr,modes,ps
integer movie_num,movie_type,nframe,gquad_count,mheight
integer modulate,rcsz1,rcsz2,minf,maxf,stepf
        integer eqset_start, eqset_end, mode_start, mode_end

c*******# cells in total field region
integer xtot_sp, ytot_sp

c*******# cells in scattered field region
integer xscat_sp, yscat_sp

c*******# cells between total fields and Huygens' surface
integer xhuy_sp, yhuy_sp

c*******# cells from object to PML region
integer xall_sp, yall_sp

c*******xall_sp = xtot_sp+xscat_sp
c*******yall_sp = ytot_sp+yscat_sp

        character base*80
```

```
      character*72 fnamein, dnamefdata, mhname, mfname,
     1     dbase

parameter(c=2.99792458E8, mu=1.25663706144E-6)
parameter(pi=3.1415926535, eps=8.8541874E-12,eta=376.73031)
parameter(pmldepth=15)

parameter(tole=1e-12)

C*****  Geometry readin routine parameters and variables.

C*****  RB,ZB: translated points; RBa, ZBa: original data points
real RBa(1:maxpt), ZBa(maxpt)
real RBt(1:maxpt), ZBt(maxpt)
real RB(maxpt), ZB(maxpt)

C*****  Parameters giving starting position of the target.
integer start_z,end_z,end_r
parameter(start_z = 40, end_z=mz-40,end_r=mr-40)

integer accessk, accessi, accesst
parameter(accessk=1,accessi=2,accesst=3)

integer actype, acl1, acl2, acA, acNPi, ack, aci
integer acz, acr, YES, NO, YES_RIGHT, YES_LEFT
parameter(actype=1,acl1=2,acl2=3,acA=4,acNPi=5)
parameter(ack=6, aci=7, acz=8, acr=9)
parameter(YES=1, NO=0, YES_RIGHT=2, YES_LEFT=3)

integer conform_grid1(1:mz,1:mr)
real conform_list(1:9,MAXCP)
integer borrow_list(1:4,MAXCP), listcount
integer ezt, ezb, erl, err
parameter(ezt=1,ezb=3,erl=4, err=2)

integer parallel, perp
parameter(parallel=2,perp=1)

C*****  Variables for conformal Hz field.
C*****  using accessk, accessi, accesst

integer conform_hz(1:3,MAXCP), EQZERO_HZ, STRETCH_HZ, SC_HZ,
1    hzcount, conform_hz1(1:mz,1:mr)

real conform_hz_length(MAXCP)
parameter(EQZERO_HZ=1, STRETCH_HZ=2, SC_HZ=3)

C*****  Variables and parameters for conformal Hr field.

integer conform_hr(1:3,MAXCP), EQZERO_HR, SRIGHT_HR, SLEFT_HR,
1    hrcount, conform_hr1(1:mz,1:mr), SLEFT_HR_DC, SRIGHT_HR_DC,
2    SRIGHT_HR_IC, SLEFT_HR_IC

real conform_hr_length(MAXCP)
parameter(EQZERO_HR=1, SRIGHT_HR=2, SLEFT_HR=3, SRIGHT_HR_DC=4,
1    SLEFT_HR_DC=5, SRIGHT_HR_IC=6, SLEFT_HR_IC=7)

integer erf,ezf,ephif,hrf,hzf,hphif,hzfo,hrfo,ezsc,ersc
parameter(erf=1,ezf=2,ephif=3,hrf=4,hzf=5,hphif=6,hzfo=7)
       parameter(hrfo=8, ezsc=9,ersc=10)

integer ephi_conform1(1:mz,1:mr), ephicount, conform_ephi(1:3,
1    MAXCP)

integer ez_conform1(1:mz,1:mr), er_conform1(1:mz,1:mr)

integer staircase(6:7,1:MAXCP),staircount
integer total_nodes, stair_node_count,
1    stair_zero(1:MAX_STAIR_NODES,1:3)

integer movie_step
logical store_movie, use_conformal, use_stair2

integer errorcount, errors(10),
1    NODE_ERROR, MAX_Z_ERROR, MAX_R_ERROR, MAX_STAIR_ERROR,
2    MAX_RCS_ERROR

parameter(NODE_ERROR=1, MAX_Z_ERROR=2, MAX_R_ERROR=3,
1    MAX_STAIR_ERROR=4, MAX_RCS_ERROR=5)

C ***********************************************************************

C*****  Cells in the free space region

real er(1:mz,1:mr), ez(1:mz,1:mr)
real ephi(1:mz,1:mr), hr(1:mz,1:mr)
real hz(1:mz,1:mr), hphi(1:mz,1:mr)

C ******Note: the array scattot indicates whether the cell is in a
C ******      scattering field points dictated by picture. see chart in
C ******      README file. Tot Fields: 2-9, 14; Scat Fields: 1,11,12,15

integer scattot(1:mz, 1:mr)

C ******Cells in left Region of PML (includes top-bottom left corners)

real   erzl(1:pmldepth+1,0:pmldepth+mr+1)
real ephizl(1:pmldepth+1,0:pmldepth+mr+1)
real   ezrl(1:pmldepth+1,0:pmldepth+mr+1)
real   hrzl(1:pmldepth+1,0:pmldepth+mr+1)
real hphizl(1:pmldepth+1,0:pmldepth+mr+1)
real   hzrl(1:pmldepth+1,0:pmldepth+mr+1)

real erphil(1:pmldepth+1,0:pmldepth+mr+1)
real ephirl(1:pmldepth+1,0:pmldepth+mr+1)
real ezphil(1:pmldepth+1,0:pmldepth+mr+1)
real hrphil(1:pmldepth+1,0:pmldepth+mr+1)
real hphirl(1:pmldepth+1,0:pmldepth+mr+1)
real hzphil(1:pmldepth+1,0:pmldepth+mr+1)

C ******Cells in the right Region of PML (incl. top-bot right corners)

real   erzr(1:pmldepth+1,0:pmldepth+mr+1)
real ephizr(1:pmldepth+1,0:pmldepth+mr+1)
real   ezrr(1:pmldepth+1,0:pmldepth+mr+1)
real   hrzr(1:pmldepth+1,0:pmldepth+mr+1)
real hphizr(1:pmldepth+1,0:pmldepth+mr+1)
real   hzrr(1:pmldepth+1,0:pmldepth+mr+1)

real erphir(1:pmldepth+1,0:pmldepth+mr+1)
real ephirr(1:pmldepth+1,0:pmldepth+mr+1)
real ezphir(1:pmldepth+1,0:pmldepth+mr+1)
real hrphir(1:pmldepth+1,0:pmldepth+mr+1)
real hphirr(1:pmldepth+1,0:pmldepth+mr+1)
real hzphir(1:pmldepth+1,0:pmldepth+mr+1)

C ******Cells in the top Region of PML (no corners)

real   erzt(1:mz,1:pmldepth+1)
real ephizt(1:mz,1:pmldepth+1)
real   ezrt(1:mz,1:pmldepth+1)
real   hrzt(1:mz,1:pmldepth+1)
real hphizt(1:mz,1:pmldepth+1)
real   hzrt(1:mz,1:pmldepth+1)

real erphit(1:mz,1:pmldepth+1)
real ephirt(1:mz,1:pmldepth+1)
real ezphit(1:mz,1:pmldepth+1)
real hrphit(1:mz,1:pmldepth+1)
real hphirt(1:mz,1:pmldepth+1)
real hzphit(1:mz,1:pmldepth+1)

C ******Frequency components
C ******  mxf = maximum number of frequencies to store.
C ******   mm = maximum number of modes to store.
C ******mxdp = maximum number of points to calculate far-field with.

real low_phi, high_phi, dphi, low_theta, high_theta, dtheta

integer num_freqs
complex feru(nm:mm,1:mxdp,1:MAX_FREQS),
1    ferv(nm:mm,1:mxdp,1:MAX_FREQS),
2    fephiu(nm:mm,1:mxdp,1:MAX_FREQS),
3    fephiv(nm:mm,1:mxdp,1:MAX_FREQS),
4    fezu(nm:mm,1:mxdp,1:MAX_FREQS),
5    fezv(nm:mm,1:mxdp,1:MAX_FREQS),
6    fhru(nm:mm,1:mxdp,1:MAX_FREQS),
7    fhrv(nm:mm,1:mxdp,1:MAX_FREQS),
8    fhphiu(nm:mm,1:mxdp,1:MAX_FREQS),
9    fhphiv(nm:mm,1:mxdp,1:MAX_FREQS),
1    fhzu(nm:mm,1:mxdp,1:MAX_FREQS),
2    fhzv(nm:mm,1:mxdp,1:MAX_FREQS)
real freqlist(1:MAX_FREQS,1:2)
c****** gives the starting index in freqlist of extra freqs for
c****** use in approximating the monostatic RCS
integer mono_freq_ind(1:MAX_FREQS), mono_nang
logical calc_bist

C ******Common Block

common N,time,NP,sigma_max,RB,ZB,dz,sdev,freq,len,er,ephi,ez
common hr,hphi,hz,erzl,erphil,ephizl,ephirl,ezrl,ezphil,hrzl
common hrphil,hphizl,hphirl,hzrl,hzphil,movie_num,movie_type
common erzr,erphir,ephizr,ephirr,ezrr,ezphir,hrzr,hrphir
common hphizr,hphirr,hzrr,hzphir,erzt,erphit,ephizt,ephirt
common ezrt,ezphit,hrzt,hrphit,hphizt,hphirt,hzrt,hzphit
common scattot,maxz,maxr,dt,obj_height,sim_duration, zbt, rbt
common zba,rba,modes,ps,gd,base,nframe,inc_ang, low_theta
common gquad_count,mheight,modfreq,low_freq,high_freq,dfreq
common modulate,maxf_v, rcsz1, rcsz2, low_phi, high_phi
common dtheta, dphi, er_conform1,enough_memory,high_theta
common feru, ferv, fhzu, fhzv, fephiu, fephiv, fezu, fezv
       common fhru, fhrv, fhphiu, fhphiv,minf,maxf, stepf, store_movie
common conform_list, borrow_list, listcount, conform_hz
```

```
        common calc_bist, mono_freq_ind, mono_nang, Ehg, Evg
common hzcount, conform_hz_length, conform_grid1, conform_hz1
common conform_hr, conform_hr1, hrcount, conform_hr_length
        common ephi_conform1, ephicount, freqlist, num_freqs
common conform_ephi,staircase,staircount,fnamein, dnamefdata
        common mhname, mfname, dbase, ez_conform1
        common eqset_start, eqset_end, mode_start, mode_end, movie_step
common use_conformal, use_stair2

common total_nodes, stair_node_count, stair_zero, errorcount,
1    errors

common xtot_sp, ytot_sp, xscat_sp, yscat_sp,
1    xhuy_sp, yhuy_sp, xall_sp, yall_sp
```

D.2 2D FD-TD Program for TE Mode

The 2D FD-TD program calculates monostatic or bistatic radar cross sections of PEC two dimensional objects of arbitrary shape for the TE mode. Similar to the BOR FD-TD program, bistatic signatures are calculated exactly while monostatic signatures are estimated using the monostatic bistatic equivalence principle, and both can be calculated over an extended bandwidth. The user can specify whether or not to include an infinite ground plane. If the ground plane is included the first point defining the PEC object is assumed to be flush with the ground plane.

The following, **fdtd_2d.f**, contains the subroutines used for reading in the input parameters from the user.

```
** THIS is the E polarization or VV code.

*************************************************************************
* 2D-FDTD CODE:                                                         *
*     This program computes the EM scattering in two dimensions. It     *
*     assumes that objects and EM fields have NO variation in the z-    *
*     direction                                                         *
*************************************************************************

      program fdtd_2d

      implicit none
      integer menu_choice

 10   write(6,*)
      write(6,*) 'What would you like do?'
      write(6,*) '1 = FDTD, WRITE FREQ, RCS'
      write(6,*) '2 = FDTD, RCS'
      write(6,*) '3 = READ FREQ, RCS'
      write(6,*) '4 = FDTD'

      read(5,*) menu_choice
      if (menu_choice.lt.1.OR.menu_choice.gt.4) goto 10

      if (menu_choice.eq.1) then
         call get_primary_input
         call get_rcs_output_ranges(.FALSE.)
         call init_fields
         call init_freqs
         call fdtd_loop(.TRUE.)
         call write_out_freqs
         call calc_rcs
      else if (menu_choice.eq.2) then
         call get_primary_input
         call get_rcs_output_ranges(.FALSE.)
         call init_fields
         call init_freqs
         call fdtd_loop(.TRUE.)
         call calc_rcs
      else if (menu_choice.eq.3) then
         call read_in_freqs
         call get_rcs_output_ranges(.TRUE.)
         call calc_rcs
      else if (menu_choice.eq.4) then
         call get_primary_input
         call init_fields
         call fdtd_loop(.FALSE.)
      end if


      END

c**********************************************************************
c GET_PRIMARY_INPUT gets info from user about geomfile name, incident
c    wave, duration of simulation, output file names (including movie).
c**********************************************************************
      SUBROUTINE get_primary_input

      implicit none
      include 'common.f'

      character*72 geomfile,mhname, mfname
      integer movie_test, ground_plane_test

c      write(6,*) 'get_primary_input'

      write(6,'(''*Enter geometry file name: '', $)')
      read(5,*) geomfile

      write(6, '(''*Include ground plane? (1=Y,2=N): '', $)')
      read(5,*) ground_plane_test
      include_ground_plane = (ground_plane_test.eq.1)

      write(6,'(''*Enter number of time steps to run: '', $)')
      read(5,*) tot_time_steps

      write(6,'(''*Store for movie? (1=Y, 2=N): '', $)')
      read(5,*) movie_test
      store_movie = (movie_test.eq.1)

      if (store_movie) then
         write(6,'(''*Movie header name: '',$)')
         read(5,*) mhname
         write(6,'(''*Movie imgfile name: '', $)')
         read(5,*) mfname
         write(6,'(''*Number of time steps between each frame: '', $)')
         read(5,*) movie_step
      end if

      write(6,'(''*Enter incident angle in degrees: '', $)')
      read(5,*) inc_ang

      write(6,'(''*Enter modulation frequency (0=unmodulated): '',$)')
      read(5,*) modfreq

      if (abs(modfreq).lt.tole) then
         modulate = 0
      else
         modulate = 1
      end if

      call setup_geometry(geomfile)

      if (store_movie) call setup_movie(mhname,mfname)

      RETURN
      END


c**********************************************************************
c GET_RCS_OUTPUT_RANGES gets info from user about what angles and freqs
c    to calc the RCS for.
c**********************************************************************

      SUBROUTINE get_rcs_output_ranges(skip_fd)

      implicit none
      include 'common.f'

      integer nang, fi, mi
      logical skip_fd
      real low_freq, dfreq, high_freq

c      write(6,*) 'get_rcs_out_ranges'

      if (.NOT.skip_fd) then
 100  write(6,*)
      write(6,*) '1. Calculate bistatic RCS for multiple freqs'
      write(6,*) '2. Estimate monostatic RCS versus angle for one freq'
```

```
      write(6,'(''*Enter your choice: '',$)')
      read(5,*) mono_bi
      if (mono_bi.ne.1.AND.mono_bi.ne.2) goto 100
      else
         mono_bi = 1
      end if

      if (mono_bi.eq.1) then
c******* calculate bistatic RCS for multiple freqs...
         if (.NOT.skip_fd) then
            write(6,'(''*Enter lowest frequency of interest: '',$)')
            read(5,*) low_freq
            write(6,'(''*Enter highest frequency of interest: '',$)')
            read(5,*) high_freq

            if (abs(low_freq-high_freq).gt.tole) then

 10            write(6,'(''*Enter the number of frequencies: '',$)')
               read(5,*) num_freqs

               if (num_freqs.gt.MAX_FREQS) then
                  write(6,*) 'Error. Number of freqs must be ',
     1                 'less than ', MAX_FREQS, ' or raise ',
     2                 'MAX_FREQS parmaeter'
                  write(6,*)
                  goto 10
               end if

               minf = 1
               maxf = num_freqs
               dfreq = (high_freq-low_freq)/(num_freqs-1.0)

               do 20 fi = minf, maxf
                  freqlist(fi,1) = low_freq + dfreq*(fi-1.0)
c**Define type as normal RCS freq
                  freqlist(fi,2) = 0
 20            continue

               stepf = 1
            else
               freqlist(1,1) = low_freq
c**Define type as normal RCS freq
               freqlist(1,2) = 0
               num_freqs = 1
               minf = 1
               maxf = 1
               stepf = 1
            end if
         else
            write(6,*) 'Currently you are calculating the RCS at ',
     1           num_freqs, ' between ', freqlist(1,1),' and ',
     2           freqlist(num_freqs,1),'. Enter the new step',
     3           ' (1 for all).'
            read(5,*) stepf
         end if

c******* Read in angles at which to calculate 2D RCS

 200     write(6,*) 'Bistatic RCS angles (in degrees)'
         write(6,'(''*Enter initial and final phi: '',$,$)')
         read(5,*) low_phi,high_phi
         if (include_ground_plane) then
            if (low_phi.lt.0.OR.high_phi.gt.180) then
               write(6,*)
               write(6,*) '*****************************************'
               write(6,*) 'Error. With ground plane, only upper half'
               write(6,*) 'plane results valid. Phi must be between'
               write(6,*) '0 and 180 degs. Re-enter angles.'
               write(6,*)
               goto 200
            end if
         end if

         if (abs(low_phi-high_phi).lt.tole) then
            dphi = high_phi-low_phi+1.0
         else
            write(6,'(''*Enter number of angles: '',$)')
            read(5,*) nang
            dphi = (high_phi-low_phi)/ real(nang-1.0)
         end if

      else
c******* Here goes all input info for mono estimation routines.

         low_phi = 1
         high_phi = 1
         dphi = 1

         write(6,99) int(inc_ang-45),int(inc_ang+45)

 99      format('Monostatic RCS angles range: ',I3, ' to ', I3,
     1        ' in one degree increments.')

         write(6,*) 'Enter lowest frequency of interest.'
         read(5,*) low_freq
         write(6,*) 'Enter highest frequency of interest.'
         read(5,*) high_freq

         if (abs(high_freq-low_freq).gt.tole) then
 30         write(6,*) 'Enter the number of frequencies.'
            read(5,*) num_freqs

            if ((MNANG*num_freqs).gt.MAX_FREQS) then
               write(6,*) 'Error. Number of freqs must be ',
     1              'less than or equal to ', MAX_FREQS/MNANG,
     1              ' or raise the MAX_FREQS parameter'
               write(6,*)
               goto 30
            end if
         else
            num_freqs = 1
         end if

         minf = 1
         maxf = num_freqs*MNANG
         print *,minf,maxf
         stepf = 1
         if (num_freqs.ne.1) then
            dfreq = (high_freq-low_freq)/(num_freqs-1.0)
         else
            dfreq = 0.0
         end if

         do 50 fi = 1,num_freqs
            do 40 mi = 1,MNANG
               freqlist(MNANG*(fi-1)+mi,1) = (low_freq+dfreq*
     1              (fi-1.0))*(1/cos((mi-46.0)*pi/180))
c               print *,freqlist(MNANG*(fi-1)+mi,1)
               freqlist(MNANG*(fi-1)+mi,2) = mi
 40         continue
 50      continue

      end if

      RETURN
      END
```

The following, **setup.f**, contains the subroutines used for setting up the computational domain as well as the staircase representation of the target.

```
c**** This is the E polarization or VV code.

c**********************************************************************
c SETUP_GEOMETRY reads in geometry file and setups up staircase model.
c**********************************************************************

      SUBROUTINE setup_geometry(geomfile)

      implicit none
      include 'common.f'

      character*72 geomfile

      integer xstair(1:MAX_STAIR_NODES),
     1     ystair(1:MAX_STAIR_NODES), index, round, spacing, current_x,
     2     current_y, xcomp, ycomp, xdir, ydir, dx, dy

      real max_x_node, max_y_node, min_x_node, min_y_node,
     1     slope, offset, dist_to_line

      parameter(spacing = 40)

      write(6,*) 'Setting up geometry...'

      errorcount = 0

c**** Read geometry file in.

      open(unit=10,file=geomfile,status='unknown',form='formatted')

      read(10,*) delta

      read(10,*) total_nodes
      if (total_nodes.gt.MAX_NODES) then
         errorcount = errorcount+1
         errors(errorcount) = NODE_ERROR
         call memory_check
      end if

      do 10 index=1,total_nodes
```

```
         read(10,*) xnodes(index), ynodes(index)
 10   continue
      close(unit=10)

C**** Scale, position, and round object

      max_x_node = xnodes(1)/delta
      max_y_node = ynodes(1)/delta
      min_x_node = xnodes(1)/delta
      min_y_node = ynodes(1)/delta

      do 20 index=1,total_nodes
         xnodes(index) = xnodes(index)/delta
         if (xnodes(index).gt.max_x_node) max_x_node=xnodes(index)
         if (xnodes(index).lt.min_x_node) min_x_node=xnodes(index)
         ynodes(index) = ynodes(index)/delta
         if (ynodes(index).gt.max_y_node) max_y_node=ynodes(index)
         if (ynodes(index).lt.min_y_node) min_y_node=ynodes(index)
 20   continue

c**** If including ground plane, object will be horizontally centered
c**** but not vertically centered. This allows the user to define
c**** exactly how high above the ground plane the object is.
c**** If no ground plane, object will be horizontally and vertically
c**** centered.

      if (include_ground_plane) then
         max_x = round(2.0*spacing + max_x_node - min_x_node)
         max_y = round(1.0*spacing + max_y_node) + 1
      else
         max_x = round(2.0*spacing + max_x_node - min_x_node)
         max_y = round(2.0*spacing + max_y_node - min_y_node)
      end if

      if (max_x.gt.MAX_X_CELLS) then
         errorcount = errorcount+1
         errors(errorcount) = MAX_X_ERROR
      end if

      if (max_y.gt.MAX_Y_CELLS) then
         errorcount = errorcount+1
         errors(errorcount) = MAX_Y_ERROR
      end if

      if (include_ground_plane) then
         do 31 index=1,total_nodes
            xnodes(index) = round(xnodes(index) - min_x_node) + spacing
            ynodes(index) = round(ynodes(index)) + 1
 31      continue
      else
         do 30 index=1,total_nodes
            xnodes(index) = round(xnodes(index) - min_x_node) + spacing
            ynodes(index) = round(ynodes(index) - min_y_node) + spacing
 30      continue
      end if


C**** define tot/scat field boundary points and setup fields
      x1=spacing-5
      if (include_ground_plane) then
         y1 = 1
      else
         y1=spacing-5
      end if
      x2=max_x-spacing+5
      y2=max_y-spacing+5

c**** Generate a staircase model by digitizing each line segment.

c**** Estimate total number of staircase nodes needed.
      dx = 0
      dy = 0
      do 50 index=1,total_nodes-1
         dx = dx + int(abs(xnodes(index)-xnodes(index+1)))
         dy = dy + int(abs(ynodes(index)-ynodes(index+1)))
 50   continue
c**** extra point needed for first point
      dy=dy+1

      if ((dx+dy).gt.MAX_STAIR_NODES) then
         stair_node_count = dx+dy
         errorcount = errorcount+1
         errors(errorcount) = MAX_STAIR_ERROR
      end if
      call define_tot_scat

      stair_node_count = 1
      xstair(stair_node_count) = int(xnodes(1))
      ystair(stair_node_count) = int(ynodes(1))

      do 40 index=1,total_nodes-1
         slope = (ynodes(index+1)-ynodes(index))/(xnodes(index+1)-
     1       xnodes(index))
         offset = ynodes(index)-slope*xnodes(index)

         current_x = int(xnodes(index))
         current_y = int(ynodes(index))

 100     stair_node_count = stair_node_count

         if (current_x.ne.int(xnodes(index+1)).OR.
     1       current_y.ne.int(ynodes(index+1))) then

            xcomp = int(xnodes(index+1))-current_x
            ycomp = int(ynodes(index+1))-current_y

            if (xcomp.ne.0) then
               xdir = int(abs(xcomp)/xcomp)
            else
               xdir = 0
            end if
            if (ycomp.ne.0) then
               ydir = int(abs(ycomp)/ycomp)
            else
               ydir = 0
            end if

            stair_node_count = stair_node_count + 1

            if (xdir.ne.0.AND.ydir.ne.0) then
               if (dist_to_line(-slope,1.0,offset,real(current_x+xdir),
     1             real(current_y)).lt.dist_to_line(-slope,1.0,offset,
     2             real(current_x),real(current_y+ydir))) then

                  xstair(stair_node_count) = current_x+xdir
                  ystair(stair_node_count) = current_y
                  current_x = current_x+xdir
               else
                  xstair(stair_node_count) = current_x
                  ystair(stair_node_count) = current_y+ydir
                  current_y = current_y+ydir
               end if
            else
               xstair(stair_node_count) = current_x+xdir
               ystair(stair_node_count) = current_y+ydir
               current_x = current_x+xdir
               current_y = current_y+ydir
            end if

            goto 100
         end if
 40   continue

      if ((dx+dy).ne.stair_node_count) then
         write(6,*) 'estimate = ', dx+dy
         write(6,*) 'actual = ', stair_node_count
      end if

c**** Now take digitized line & figure out which fields to set to zero

      call generate_tm_stair_model(xstair,ystair,stair_node_count)

      open(unit=10,file='stair.dat',status='unknown',
     1     form='formatted')

      do 1000 index=1,stair_node_count
         write(10,*) xstair(index), ystair(index)
 1000 continue
      close(unit=10)

c**** Calculate some important variables

c**** Time Step based on 2D stability requirements
      dt = 0.86*(delta/c/sqrt(2.0))

c**** Width of the Gaussian Pulse
 300  width = 2*sqrt(8.0)/(pi*(c/15/delta-modfreq))
      if (width.gt.(50*dt).OR.width.lt.(0.0)) then
         write(6,*) '*********************************************'
         if (width.lt.(0.0)) then
            write(6,*) 'Mod freq too high for grid resolution'
         else
            write(6,*) 'Mod freq implies too large a width'
            write(6,*) 'width/dt=', width/dt
         end if
         write(6,*) 'Enter new modulation frequency (0=unmodulated)'
         read(5,*) modfreq

         if (abs(modfreq).lt.tole) then
            modulate = 0
         else
            modulate = 1
         end if
         goto 300
```

```
      end if

c**** The maximum sigma needed in the PML regions
      reflection = -40.0
      sigma_max = -reflection*3/eta/40./0.434294481903/(PML_DEPTH*delta)

      RETURN
      END

c**************************************************************************
c INIT_FIELDS initializes all fields in normal and PML regions to zero.
c**************************************************************************

      SUBROUTINE init_fields

      implicit none
      include 'common.f'

      integer i,j

      write(6,*) 'Initializing field data...'

      do 10 i=1,max_x
         do 20 j=1,max_y
            hx(i,j) = 0.0
            hy(i,j) = 0.0
            ez(i,j) = 0.0
 20      continue
 10   continue

      do 30 i=1,max_x
         do 40 j=1,PML_DEPTH
            ezxtt(i,j) = 0.0
            ezytt(i,j) = 0.0
            hxxtt(i,j) = 0.0
            hyytt(i,j) = 0.0
 40      continue
 30   continue

      do 50 i=1,max_x
         do 60 j=1,PML_DEPTH
            ezxbb(i,j) = 0.0
            ezybb(i,j) = 0.0
            hxxbb(i,j) = 0.0
            hyybb(i,j) = 0.0
 60      continue
 50   continue

      do 70 i=1,PML_DEPTH
         do 80 j=1,max_y
            ezxrr(i,j) = 0.0
            ezyrr(i,j) = 0.0
            hxxrr(i,j) = 0.0
            hyyrr(i,j) = 0.0
 80      continue
 70   continue

      do 90 i=1,PML_DEPTH
         do 100 j=1,max_y
            ezxll(i,j) = 0.0
            ezyll(i,j) = 0.0
            hxxll(i,j) = 0.0
            hyyll(i,j) = 0.0
 100     continue
 90   continue

      do 110 i=1,PML_DEPTH
         do 120 j=1,PML_DEPTH
            ezxtr(i,j) = 0.0
            ezytr(i,j) = 0.0
            hxxtr(i,j) = 0.0
            hyytr(i,j) = 0.0
 120     continue
 110  continue

      do 130 i=1,PML_DEPTH
         do 140 j=1,PML_DEPTH
            ezxtl(i,j) = 0.0
            ezytl(i,j) = 0.0
            hxxtl(i,j) = 0.0
            hyytl(i,j) = 0.0
 140     continue
 130  continue

      do 150 i=1,PML_DEPTH
         do 160 j=1,PML_DEPTH
            ezxbr(i,j) = 0.0
            ezybr(i,j) = 0.0
            hxxbr(i,j) = 0.0
            hyybr(i,j) = 0.0
 160     continue
 150  continue

      do 170 i=1,PML_DEPTH
         do 180 j=1,PML_DEPTH
            ezxbl(i,j) = 0.0
            ezybl(i,j) = 0.0
            hxxbl(i,j) = 0.0
            hyybl(i,j) = 0.0
 180     continue
 170  continue

      RETURN
      END

c**************************************************************************
c INIT_FREQS initializes all freq field comps to zero.
c**************************************************************************

      SUBROUTINE init_freqs

      implicit none
      include 'common.f'

      integer fi, k, k2, ytemp

      write(6,*) 'Initializing frequency data...'

      do 10 fi=minf,maxf,stepf
         if (include_ground_plane) then
            ytemp = 4*y2+2*(x2-x1)-4+8*rcs_space+1
         else
            ytemp = 2*(y2-y1)+2*(x2-x1)+8*rcs_space+1
         end if
         do 20 k=1, ytemp
            ezfreq(fi,k) = 0.0
            hxfreq(fi,k) = 0.0
            hyfreq(fi,k) = 0.0
 20      continue
         do 30 k2=1,stair_node_count
            Jsfreq(fi,k2) = 0.0
 30      continue
 10   continue

      RETURN
      END

c**************************************************************************
c READ_IN_FREQS reads in the previously saved frequency field comps.
c**************************************************************************

      SUBROUTINE read_in_freqs

      implicit none
      include 'common.f'

      integer fi, k, num_freqs, rcs_nodes, rbeg, rend
      real tempr, tempi

      write(6,*) 'Reading in frequency data...'

      open(unit=10,file='ezf.dat',status='unknown',form='formatted')
      open(unit=11,file='hxf.dat',status='unknown',form='formatted')
      open(unit=12,file='hyf.dat',status='unknown',form='formatted')

c**** Restore state

      open(unit=13,file='info.dat',status='unknown',form='formatted')

      read(13,*) max_x
      read(13,*) max_y
      read(13,*) delta
      read(13,*) dt
      read(13,*) tot_time_steps
      read(13,*) x1,y1,x2,y2
      read(13,*) inc_ang
      read(13,*) modulate
      read(13,*) modfreq
      read(13,*) delay
      read(13,*) width
      read(13,*) include_ground_plane
      read(13,*) num_freqs

c**** Assuming bistatic calculation
      mono_bi = 1
      do 30 fi = 1, num_freqs
         read(13,*) freqlist(fi,1)
         freqlist(fi,2) = 0
 30   continue

      close(unit=13)

      cost = cos(inc_ang*pi/180)
      sint = sin(inc_ang*pi/180)
```

```
      if (abs(freqlist(num_freqs,1)-freqlist(1,1)).gt.tole) then
         if (num_freqs.gt.MAX_FREQS) then
            write(6,*) 'Error. Set MAX_FREQS parmeter higher.'
            write(6,*)
            stop
         end if

         minf = 1
         maxf = num_freqs
         stepf = 1
      else
         num_freqs = 1
         minf = 1
         maxf = 1
         stepf = 1
      end if

      if (include_ground_plane) then
         rbeg = y2+rcs_space
         rend = 3*y2+x2-x1-2+5*rcs_space
      else
         rbeg = 1
         rend = 2*(x2-x1)+2*(y2-y1)+8*rcs_space+1
      end if

      do 10 fi = minf,maxf
         do 20 k=rbeg, rend
            read(10,*) tempr,tempi
            ezfreq(fi,k) = tempr + (0.0,1.0)*tempi

            read(11,*) tempr,tempi
            hxfreq(fi,k) = tempr + (0.0,1.0)*tempi

            read(12,*) tempr,tempi
            hyfreq(fi,k) = tempr + (0.0,1.0)*tempi
 20      continue

 10   continue

      close(unit=10)
      close(unit=11)
      close(unit=12)

      RETURN
      END

c**********************************************************************
c DEFINE_TOT_SCAT defines each cell on the grid as either a total field
c   or a scattered field.
c**********************************************************************

      SUBROUTINE define_tot_scat

      implicit none
      include 'common.f'

      integer i,j

c     15 15 15 15 15 15 15 15 15 15
c     15 11 11 11 11 11 11 11 11 15
c       -------------------------
c     01|04 05 05 05 05 05 06|12
c     01|03 14 14 14 14 14 07|12
c     01|03 14 14 14 14 14 07|12
c     01|03 14 14 14 14 14 07|12
c     01|03 14 14 14 14 14 07|12
c     01|03 14 14 14 14 14 07|12
c     01|03 14 14 14 14 14 07|12
c     01|02 09 09 09 09 09 08|12
c       -------------------------
c     15 00 00 00 00 00 00 00 15
c     15 15 15 15 15 15 15 15 15

c     02,03,04,05,06,07,08,09,14 are total fields
c     00,01,11,12,15 are scattered fields.

c      write(6,*) 'define_tot_scat'

      if (include_ground_plane) then
         if ((4*y2+2*(x2-x1)-4+8*rcs_space+1).gt.MAX_RCS_NODES) then
            errorcount = errorcount+1
            errors(errorcount) = MAX_RCS_ERROR
         end if
      else
         if ((2*(x2-x1)+2*(y2-y1)+8*rcs_space+1).gt.MAX_RCS_NODES) then
            errorcount = errorcount + 1
            errors(errorcount) = MAX_RCS_ERROR
         end if
      end if

      call memory_check

      do 10 i=1,max_x
         do 20 j=1,max_y
            tot_scat(i,j) = 15
 20      continue
 10   continue

      do 70 i=x1+1,x2-1
         do 80 j=y1+1,y2-1
            tot_scat(i,j) = 14
 80      continue
 70   continue

      tot_scat(x1,y1) = 2
      tot_scat(x1,y2) = 4
      tot_scat(x2,y2) = 6
      tot_scat(x2,y1) = 8

      do 30 j=y1+1,y2-1
         tot_scat(x1,j) = 3
         tot_scat(x2,j) = 7
 30   continue

      do 40 i=x1+1,x2-1
         tot_scat(i,y1) = 9
         tot_scat(i,y2) = 5
 40   continue

      do 50 j=y1,y2
         tot_scat(x1-1,j) = 1
         tot_scat(x2+1,j) = 12
 50   continue

      do 60 i=x1,x2
         if (.NOT.include_ground_plane) tot_scat(i,y1-1) = 0
         tot_scat(i,y2+1) = 11
 60   continue

c      open(unit=14,file='totsc.dat',status='unknown',form='formatted')

c      do 170 j=y2+5,y1-5,-1
c         do 180 i=x1-5,x2+5
c            write(14,*) tot_scat(i,j)
c 180     continue
c 170  continue

c      close(unit=14)

      RETURN
      END

c**********************************************************************
c SETUP_INCIDENT_FIELD does some prelim calcs to prepare for using the
c   one-dimensional source look-up table method.
c**********************************************************************

      SUBROUTINE setup_incident_field

      implicit none
      include 'common.f'

      integer STEPS_TO_DELAY
      parameter(STEPS_TO_DELAY=150)

      if (abs(inc_ang-0.0).lt.tole) then
         sint = 0.0
         cost = 1.0
      elseif (abs(inc_ang-90.0).lt.tole) then
         sint = 1.0
         cost = 0.0
      elseif (abs(inc_ang-180.0).lt.tole) then
         sint = 0.0
         cost = -1.0
      elseif (abs(inc_ang-270.0).lt.tole) then
         sint = -1.0
         cost = 0.0
      else
         sint = sin((inc_ang/180.)*pi)
         cost = cos((inc_ang/180.)*pi)
      end if
c      print *,inc_ang,sint,cost

c***  calculate time delay

 10   if (inc_ang.ge.360) then
         inc_ang = inc_ang-360.0
         goto 10
      end if

 20   if (inc_ang.lt.0) then
         inc_ang = inc_ang+360.0
         goto 20
```

```
      end if

      if (inc_ang.ge.0.AND.inc_ang.lt.90) then
         delay = -(x2*delta*cost+y2*delta*sint)/c - STEPS_TO_DELAY*dt
     1        + width/2.0
      elseif (inc_ang.ge.90.AND.inc_ang.lt.180) then
         delay = -(x1*delta*cost+y2*delta*sint)/c - STEPS_TO_DELAY*dt
     1        - width/2.0
      elseif (inc_ang.ge.180.AND.inc_ang.lt.270) then
         delay = -(x1*delta*cost+y1*delta*sint)/c - STEPS_TO_DELAY*dt
     1        - width/2.0
      else
         delay = -(x2*delta*cost+y1*delta*sint)/c - STEPS_TO_DELAY*dt
     1        + width/2.0
      end if

c***  calculate numerical phase velocity at theta=0

c***  calculate numerical phase velocity at theta=inc_ang

      RETURN
      END


c**********************************************************************
c SUBROUTINE SETUP_MOVIE
c**********************************************************************

      SUBROUTINE setup_movie(mhname,mfname)

      implicit none
      include 'common.f'
      character*72 mhname, mfname

      open(unit=4,file=mfname,status='unknown',form='formatted')
      open(unit=7,file=mhname,status='unknown',form='formatted')

      write(4,*) min(max_x,100)
      write (4,100) 'new.image'
      write(7,200) min(100,max_x), min(100,max_y), 64, tot_time_steps,
     1     'new.image.Z'

      dummy = min(1000,max_x)
      write (7,*) 1
      write (7,100) 'a '
      write (7,100) 'b '
      close(unit=7)

 100  format(a)
 200  format(i4,x,i4,x,i2,x,i5,x,a)

      RETURN
      END


c**********************************************************************
c MEMORY_CHECK checks if enough memory has been allocated and reports
c   all errors stored in error buffer
c**********************************************************************

      SUBROUTINE memory_check

      implicit none
      include 'common.f'

      integer i, id

      if (errorcount.gt.0) then
         write(6,*) '*********************************************'
         write(6,*) 'Insufficient memory to begin simulation. The'
         write(6,*) 'following parameter(s) in the common.f file'
         write(6,*) 'need to be adjusted:'

         do 10 i=1,errorcount
            id = errors(i)
            write(6,*)
            if (id.eq.NODE_ERROR) then
               write(6,*) 'Set MAX_NODES to at least',total_nodes
            else if (id.eq.MAX_X_ERROR) then
               write(6,*) 'Set MAX_X_CELLS to at least',max_x
            else if (id.eq.MAX_Y_ERROR) then
               write(6,*) 'Set MAX_Y_CELLS to at least',max_y
            else if (id.eq.MAX_STAIR_ERROR) then
               write(6,*) 'Set MAX_STAIR_NODES to at least',
     1              stair_node_count
            else if (id.eq.MAX_RCS_ERROR) then
               if (include_ground_plane) then
                  write(6,*) 'Set MAX_RCS_NODES to at least',
     1                 2*(x2-x1)+4*y2-4+8*rcs_space+1
               else
                  write(6,*) 'Set MAX_RCS_NODES to at least',
     1                 2*(x2-x1)+2*(y2-y1)+8*rcs_space+1
               end if
            end if

 10      continue
         write(6,*) '*********************************************'
         stop

      end if

      RETURN
      END


c**********************************************************************
c INTEGER FUNCTION ROUND returns the integer nearest in absolute
c   distance to the real arguement
c**********************************************************************

      INTEGER FUNCTION round(x)

      real x, fpart
      integer ipart

      ipart = int(x)
      fpart = x-aint(x)

      if (fpart.gt.0.5) then
         round = ipart+1
      else if (fpart.gt.0.0) then
         round = ipart
      else if (fpart.ge.-0.5) then
         round = ipart
      else
         round = ipart-1
      end if

      RETURN
      END


c**********************************************************************
c REAL FUNCTION DIST_TO_LINE returns the perpendicular distance from a
c point in space (x,y) to a line that is of the form Ax+By=C
c**********************************************************************

      REAL FUNCTION dist_to_line(A,B,C,x,y)

      implicit none
      real A,B,C,x,y

      dist_to_line = abs((A*x+B*y-C)/sqrt(A**2.0 + B**2.0))

      RETURN
      END


c**********************************************************************
c LOGICAL FUNCTION INSIDE_PEC(xlist,ylist,x,y) returns TRUE if the point
c   (x,y) is inside the polygon described by the points in x,y lists
c**********************************************************************

      LOGICAL FUNCTION inside_pec(xlist,ylist,count,px,py)

      implicit none
      include 'common.f'

      integer count
      real xlist(1:count),ylist(1:count),xp1,yp1,px,py,slope

      integer maxx, minx, maxy, miny, index, above, below, connect

c**** if polygon not closed, close it.
      if (xlist(1).ne.xlist(count).OR.ylist(1).ne.ylist(count)) then
         connect = 0
      else
         connect = -1
      end if

      maxx = xlist(1)
      minx = xlist(1)
      maxy = ylist(1)
      miny = ylist(1)

      do 10 index=1,count
         if (xlist(index).gt.maxx) maxx=xlist(index)
         if (ylist(index).gt.maxy) maxy=ylist(index)
         if (xlist(index).lt.minx) minx=xlist(index)
         if (ylist(index).lt.miny) miny=ylist(index)
 10   continue

      if (px.gt.maxx.OR.px.lt.minx.OR.py.gt.maxy.OR.py.lt.miny) then
         inside_pec = .FALSE.
```

```
      else

c**** count the intersections

         above = 0
         below = 0

         do 20 index = 1,count+connect
            if (index.eq.count) then
               xp1 = xlist(1)
               yp1 = ylist(1)
            else
               xp1 = xlist(index+1)
               yp1 = ylist(index+1)
            end if

            if ( (abs(px-xlist(index)).lt.tole).AND.(abs(px-
     1           xlist(index+1)).lt.tole)) then
               if (((py.le.ylist(index)).AND.(py.ge.ylist(index+1))).
     1              OR.((py.ge.ylist(index)).AND.(py.le.ylist(index
     2              +1))) ) then
                  inside_pec = .TRUE.
                  RETURN
               end if
            end if

            if ( ((abs(px-xlist(index)).lt.tole).AND.(abs(py-
     1           ylist(index)).lt.tole)).OR.((abs(px-xlist(index+1)).
     2           lt.tole).AND.(abs(py-ylist(index+1)).lt.tole))) then
               inside_pec = .TRUE.
               RETURN
            end if

            if ((px.le.xlist(index).AND.px.ge.xp1).OR.
     1           (px.ge.xlist(index).AND.px.le.xp1)) then
               slope = (ylist(index)-yp1)/(xlist(index)-xp1)
               if (abs((slope*(px-xlist(index))+ylist(index))-py).
     1              lt.tole) then
                  inside_pec = .TRUE.
                  RETURN
               else if ((slope*(px-xlist(index))+ylist(index)).gt.
     1                 py) then
                  above = above+1
               else
                  below = below+1
               end if
            end if
 20      continue

         if (mod(above,2).eq.1.AND.mod(below,2).eq.1) then
            inside_pec = .TRUE.
         else
            inside_pec = .FALSE.
         end if
      end if

      RETURN
      END

c**********************************************************************
c GENERATE_TM_STAIR_MODEL generates the TM staircase model by compiling
c   a list of all the Ez fields that need to be set to zero.
c**********************************************************************

      SUBROUTINE generate_tm_stair_model(xstair,ystair)

      implicit none
      include 'common.f'

      integer xstair(1:MAX_STAIR_NODES), ystair(1:MAX_STAIR_NODES),
     1     index

      do 10 index = 1,stair_node_count
         stair_zero(index,1) = int(xstair(index))
         stair_zero(index,2) = int(ystair(index))
         stair_zero(index,3) = ezf
 10   continue

      RETURN
      END
```

The following, **calc.f**, contains the core FDTD subroutines used in updating the fields. It includes the implementation of Berenger's PML absorbing boundary condition.

```
c***This is the E polarization or VV code.

c**********************************************************************
c FDTD_LOOP controls the flow of the wave propagations calculations. It
c   calls neccessary subroutines including E_FIELDS, H_FIELDS, E_PML,
c   and H_PML.
c**********************************************************************

      SUBROUTINE fdtd_loop(store_freq)

      implicit none
      include 'common.f'

      character a(1:MAX_X_CELLS)
      logical store_freq
      integer time_step, movie_frame, freq_frame

      open(unit=18,file='effscat.dat',status='unknown',form='formatted')

      movie_frame = 0
      freq_frame = 1
      max_field = 0.0
      call setup_incident_field
      call write_out_all_parms(store_freq)
      call memory_check
      write(6,*) 'Running simulation...'
      do 10 time_step = 1,tot_time_steps
         write(6,*) time_step
         call h_fields(time_step)
         call h_pml
         call e_fields(time_step)
         call e_pml
         call boundary_conditions
         write(18,*) ez(30,50), ez(33,1)
         if (store_movie) then
            if (movie_frame.eq.movie_step) then
               call write_out_movie_frame(a,time_step)
               movie_frame = 0
            else
               movie_frame = movie_frame + 1
            end if
         end if
         if (freq_frame.eq.1) then
            if (store_freq) call update_freqs(time_step)
c            call Js_freq(time_step)
            freq_frame = 1
         else
            freq_frame = freq_frame + 1
         end if
 10   continue

      close(unit=18)
      close(unit=4)
c      call Jsfreq_out

      RETURN
      END

c**********************************************************************
c E_FIELDS updates the E field comps in the normal region.
c**********************************************************************

      SUBROUTINE e_fields(time_step)

      implicit none
      include 'common.f'

      integer time_step, i, j, ct
      real t1, t2, hyinc, hxinc, hyref, hxref

      do 10 i=1,max_x
         do 20 j=1,max_y
            ct = tot_scat(i,j)

            if (i.eq.max_x) then
               t1=hyyrr(1,j)
            else
               t1=hy(i+1,j)
            end if

            if (j.eq.max_y) then
               t2=hxxtt(i,1)
            else
               t2=hx(i,j+1)
            end if

            if (ct.eq.6.OR.ct.eq.7.OR.ct.eq.8) then
               t1=t1+hyinc(i+1,j,time_step)
               if (include_ground_plane) t1=t1+hyref(i+1,j,time_step)
            end if
            if (ct.eq.1) then
               t1=t1-hyinc(i+1,j,time_step)
               if (include_ground_plane) t1=t1-hyref(i+1,j,time_step)
            end if
```

```
            if (ct.eq.4.OR.ct.eq.5.OR.ct.eq.6) then
               t2=t2+hxinc(i,j+1,time_step)
               if (include_ground_plane) t2=t2+hxref(i,j+1,time_step)
            end if
            if (ct.eq.0) t2=t2-hxinc(i,j+1,time_step)

            ez(i,j)=ez(i,j)+(c*dt/delta)*((t1-hy(i,j))-(t2-hx(i,j)))
 20      continue
 10   continue

      RETURN
      END

c**************************************************************************
c H_FIELDS updates the H field comps in the normal region.
c**************************************************************************

      SUBROUTINE h_fields(time_step)

      implicit none
      include 'common.f'

      integer time_step, i, j, ct
      real t1, ezinc, ezref

c**** Hy fields ********************************************************

      do 10 i=1,max_x
         do 20 j=1,max_y
            ct=tot_scat(i,j)

            if (i.eq.1) then
               t1=ezxll(PML_DEPTH,j)+ezyll(PML_DEPTH,j)
            else
               t1=ez(i-1,j)
            end if

            if (ct.eq.2.OR.ct.eq.3.OR.ct.eq.4) then
               t1=t1+ezinc(i-1,j,time_step)
               if (include_ground_plane) t1=t1+ezref(i-1,j,time_step)
            end if

            if (ct.eq.12) then
               t1=t1-ezinc(i-1,j,time_step)
               if (include_ground_plane) t1=t1-ezref(i-1,j,time_step)
            end if

            hy(i,j)=hy(i,j) + (c*dt/delta)*(ez(i,j)-t1)
 20      continue
 10   continue

c**** Hx fields ********************************************************

      do 30 i=1,max_x
         do 40 j=1,max_y
            ct=tot_scat(i,j)

            if (j.eq.1) then
               t1=ezxbb(i,PML_DEPTH)+ezybb(i,PML_DEPTH)
            else
               t1=ez(i,j-1)
            end if

c********** with ground plane, set correct boundary conditions.
            if (ct.eq.2.OR.ct.eq.9.OR.ct.eq.8) then
               if (include_ground_plane) then
                  t1=0.0
               else
                  t1=t1+ezinc(i,j-1,time_step)
               end if
            end if

            if (ct.eq.11) then
               t1=t1-ezinc(i,j-1,time_step)
               if (include_ground_plane) t1=t1-ezref(i,j-1,time_step)
            end if

            hx(i,j)=hx(i,j) - (c*dt/delta)*(ez(i,j)-t1)
 40      continue
 30   continue

      RETURN
      END

c**************************************************************************
c E_PML updates the E field comps in the PML regions.
c**************************************************************************

      SUBROUTINE e_pml

      implicit none
      include 'common.f'

      integer i,j
      real c1, c2, c3, c4, sigma_x, sigma_y
      real t1, t2

c**** Bottom Left Region (BL), sigma_x & sigma_y nonzero

      do 10 i=1,PML_DEPTH
         sigma_x = sigma_max*((PML_DEPTH+1.0-i)/PML_DEPTH)**2.0
         c1 = exp(-sigma_x*dt/eps)
         c2 = (1-c1)/(sigma_x*delta)

         do 20 j=1,PML_DEPTH
            sigma_y = sigma_max*((PML_DEPTH+1.0-j)/PML_DEPTH)**2.0
            c3 = exp(-sigma_y*dt/eps)
            c4 = (1-c3)/(sigma_y*delta)

            if (i.eq.PML_DEPTH) then
               t1=hyybb(1,j)
            else
               t1=hyybl(i+1,j)
            end if

            if (j.eq.PML_DEPTH) then
               t2=hxxll(i,1)
            else
               t2=hxxbl(i,j+1)
            end if

            ezxbl(i,j)=c1*ezxbl(i,j)+c2*((1/eta)*(t1-hyybl(i,j)))
            ezybl(i,j)=c3*ezybl(i,j)-c4*((1/eta)*(t2-hxxbl(i,j)))
 20      continue
 10   continue

c**** Top Left Region (TL), sigma_x & sigma_y nonzero

      do 30 i=1,PML_DEPTH
         sigma_x = sigma_max*((PML_DEPTH+1.0-i)/PML_DEPTH)**2.0
         c1 = exp(-sigma_x*dt/eps)
         c2 = (1-c1)/(sigma_x*delta)

         do 40 j=1,PML_DEPTH
            sigma_y = sigma_max*((j+0.0)/PML_DEPTH)**2.0
            c3 = exp(-sigma_y*dt/eps)
            c4 = (1-c3)/(sigma_y*delta)

            if (i.eq.PML_DEPTH) then
               t1=hyytt(1,j)
            else
               t1=hyytl(i+1,j)
            end if

            if (j.eq.PML_DEPTH) then
               t2=0.0
            else
               t2=hxxtl(i,j+1)
            end if

            ezxtl(i,j)=c1*ezxtl(i,j)+c2*((1/eta)*(t1-hyytl(i,j)))
            ezytl(i,j)=c3*ezytl(i,j)-c4*((1/eta)*(t2-hxxtl(i,j)))
 40      continue
 30   continue

c**** Top Right Region (TR), sigma_x & sigma_y nonzero

      do 50 i=1,PML_DEPTH
         sigma_x = sigma_max*((i+0.0)/PML_DEPTH)**2.0
         c1 = exp(-sigma_x*dt/eps)
         c2 = (1-c1)/(sigma_x*delta)

         do 60 j=1,PML_DEPTH
            sigma_y = sigma_max*((j+0.0)/PML_DEPTH)**2.0
            c3 = exp(-sigma_y*dt/eps)
            c4 = (1-c3)/(sigma_y*delta)

            if (i.eq.PML_DEPTH) then
               t1=0.0
            else
               t1=hyytr(i+1,j)
            end if

            if (j.eq.PML_DEPTH) then
               t2=0.0
            else
               t2=hxxtr(i,j+1)
            end if

            ezxtr(i,j)=c1*ezxtr(i,j)+c2*((1/eta)*(t1-hyytr(i,j)))
            ezytr(i,j)=c3*ezytr(i,j)-c4*((1/eta)*(t2-hxxtr(i,j)))
 60      continue
 50   continue
```

```
c**** Bottom Right Region (BR), sigma_x & sigma_y nonzero

      do 70 i=1,PML_DEPTH
         sigma_x = sigma_max*((i+0.0)/PML_DEPTH)**2.0
         c1 = exp(-sigma_x*dt/eps)
         c2 = (1-c1)/(sigma_x*delta)

         do 80 j=1,PML_DEPTH
            sigma_y = sigma_max*((PML_DEPTH+1.0-j)/PML_DEPTH)**2.0
            c3 = exp(-sigma_y*dt/eps)
            c4 = (1-c3)/(sigma_y*delta)

            if (i.eq.PML_DEPTH) then
               t1=0.0
            else
               t1=hyybr(i+1,j)
            end if

            if (j.eq.PML_DEPTH) then
               t2=hxxrr(i,1)
            else
               t2=hxxbr(i,j+1)
            end if

            ezxbr(i,j)=c1*ezxbr(i,j)+c2*((1/eta)*(t1-hyybr(i,j)))
            ezybr(i,j)=c3*ezybr(i,j)-c4*((1/eta)*(t2-hxxbr(i,j)))
 80      continue
 70   continue

c**** Bottom Center Region (BB), sigma_y nonzero

      do 90 i=1,max_x
         do 100 j=1,PML_DEPTH
            sigma_y = sigma_max*((PML_DEPTH+1.0-j)/PML_DEPTH)**2.0
            c3 = exp(-sigma_y*dt/eps)
            c4 = (1-c3)/(sigma_y*delta)

            if (i.eq.max_x) then
               t1=hyybr(1,j)
            else
               t1=hyybb(i+1,j)
            end if

            if (j.eq.PML_DEPTH) then
               t2=hx(i,1)
            else
               t2=hxxbb(i,j+1)
            end if

            ezxbb(i,j)=ezxbb(i,j)+((c*dt)/delta)*(t1-hyybb(i,j))
            ezybb(i,j)=c3*ezybb(i,j)-c4*((1/eta)*(t2-hxxbb(i,j)))
 100        continue
 90      continue

c**** Top Center Region, sigma_y nonzero

      do 110 i=1,max_x
         do 120 j=1,PML_DEPTH
            sigma_y = sigma_max*((j+0.0)/PML_DEPTH)**2.0
            c3 = exp(-sigma_y*dt/eps)
            c4 = (1-c3)/(sigma_y*delta)

            if (i.eq.max_x) then
               t1=hyytr(1,j)
            else
               t1=hyytt(i+1,j)
            end if

            if (j.eq.PML_DEPTH) then
               t2=0.0
            else
               t2=hxxtt(i,j+1)
            end if

            ezxtt(i,j)=ezxtt(i,j)+((c*dt)/delta)*(t1-hyytt(i,j))
            ezytt(i,j)=c3*ezytt(i,j)-c4*((1/eta)*(t2-hxxtt(i,j)))
 120        continue
 110     continue

c**** Right Center Region (RR), sigma_x nonzero

      do 130 i=1,PML_DEPTH
         sigma_x = sigma_max*((i+0.0)/PML_DEPTH)**2.0
         c1 = exp(-sigma_x*dt/eps)
         c2 = (1-c1)/(sigma_x*delta)

         do 140 j=1,max_y

            if (i.eq.PML_DEPTH) then
               t1=0.0
            else
               t1=hyyrr(i+1,j)
            end if

            if (j.eq.max_y) then
               t2=hxxtr(i,1)
            else
               t2=hxxrr(i,j+1)
            end if

            ezxrr(i,j)=c1*ezxrr(i,j)+c2*((1/eta)*(t1-hyyrr(i,j)))
            ezyrr(i,j)=ezyrr(i,j)-((c*dt)/delta)*(t2-hxxrr(i,j))
 140     continue
 130  continue

c**** Left Center Region (LL), sigma_x nonzero

      do 150 i=1,PML_DEPTH
         sigma_x = sigma_max*((PML_DEPTH+1.0-i)/PML_DEPTH)**2.0
         c1 = exp(-sigma_x*dt/eps)
         c2 = (1-c1)/(sigma_x*delta)

         do 160 j=1,max_y

            if (i.eq.PML_DEPTH) then
               t1=hy(i,j)
            else
               t1=hyyll(i+1,j)
            end if

            if (j.eq.max_y) then
               t2=hxxtl(i,1)
            else
               t2=hxxll(i,j+1)
            end if

            ezxll(i,j)=c1*ezxll(i,j)+c2*((1/eta)*(t1-hyyll(i,j)))
            ezyll(i,j)=ezyll(i,j)-((c*dt)/delta)*(t2-hxxll(i,j))
 160      continue
 150   continue

      RETURN
      END

c**********************************************************************
c H_PML updates the H field comps in the PML regions.
c**********************************************************************

      SUBROUTINE h_pml

      implicit none
      include 'common.f'

      integer i,j
      real c1, c2, c3, c4, sigma_x, sigma_y
      real t1, t2, t3

c**** Bottom Left Region (BL), sigma_x & sigma_y nonzero

      do 10 i=1,PML_DEPTH
         sigma_x = sigma_max*((PML_DEPTH+1.0-i)/PML_DEPTH)**2.0
         c1 = exp(-sigma_x*dt/eps)
         c2 = (1-c1)/(sigma_x*delta*eta*eta)

         do 20 j=1,PML_DEPTH
            sigma_y = sigma_max*((PML_DEPTH+1.0-j)/PML_DEPTH)**2.0
            c3 = exp(-sigma_y*dt/eps)
            c4 = (1-c3)/(sigma_y*delta*eta*eta)

            t1=ezxbl(i,j)+ezybl(i,j)

            if (i.eq.1) then
               t2=0
            else
               t2=ezxbl(i-1,j)+ezybl(i-1,j)
            end if

            if (j.eq.1) then
               t3=0
            else
               t3=ezxbl(i,j-1)+ezybl(i,j-1)
            end if

            hyybl(i,j)=c1*hyybl(i,j)+c2*eta*(t1-t2)
            hxxbl(i,j)=c3*hxxbl(i,j)-c4*eta*(t1-t3)
 20      continue
 10   continue

c**** Top Left Region (TL), sigma_x & sigma_y nonzero

      do 30 i=1,PML_DEPTH
         sigma_x = sigma_max*((PML_DEPTH+1.0-i)/PML_DEPTH)**2.0
         c1 = exp(-sigma_x*dt/eps)
         c2 = (1-c1)/(sigma_x*delta*eta*eta)
```

```
         do 40 j=1,PML_DEPTH
            sigma_y = sigma_max*((j+0.0)/PML_DEPTH)**2.0
            c3 = exp(-sigma_y*dt/eps)
            c4 = (1-c3)/(sigma_y*delta*eta*eta)

            t1=ezxtl(i,j)+ezytl(i,j)

            if (i.eq.1) then
               t2=0
            else
               t2=ezxtl(i-1,j)+ezytl(i-1,j)
            end if

            if (j.eq.1) then
               t3=ezxll(i,max_y)+ezyll(i,max_y)
            else
               t3=ezxtl(i,j-1)+ezytl(i,j-1)
            end if

            hyytl(i,j)=c1*hyytl(i,j)+c2*eta*(t1-t2)
            hxxtl(i,j)=c3*hxxtl(i,j)-c4*eta*(t1-t3)
 40      continue
 30    continue

c**** Top Right Region (TR), sigma_x & sigma_y nonzero

       do 50 i=1,PML_DEPTH
          sigma_x = sigma_max*((i+0.0)/PML_DEPTH)**2.0
          c1 = exp(-sigma_x*dt/eps)
          c2 = (1-c1)/(sigma_x*delta*eta*eta)

          do 60 j=1,PML_DEPTH
             sigma_y = sigma_max*((j+0.0)/PML_DEPTH)**2.0
             c3 = exp(-sigma_y*dt/eps)
             c4 = (1-c3)/(sigma_y*delta*eta*eta)
             t1=ezxtr(i,j)+ezytr(i,j)

             if (i.eq.1) then
                t2=ezxtt(max_x,j)+ezytt(max_x,j)
             else
                t2=ezxtr(i-1,j)+ezytr(i-1,j)
             end if

             if (j.eq.1) then
                t3=ezxrr(i,max_y)+ezyrr(i,max_y)
             else
                t3=ezxtr(i,j-1)+ezytr(i,j-1)
             end if

             hyytr(i,j)=c1*hyytr(i,j)+c2*eta*(t1-t2)
             hxxtr(i,j)=c3*hxxtr(i,j)-c4*eta*(t1-t3)
 60       continue
 50    continue

c**** Bottom Right Region (BR), sigma_x & sigma_y nonzero

       do 70 i=1,PML_DEPTH
          sigma_x = sigma_max*((i+0.0)/PML_DEPTH)**2.0
          c1 = exp(-sigma_x*dt/eps)
          c2 = (1-c1)/(sigma_x*delta*eta*eta)

          do 80 j=1,PML_DEPTH
             sigma_y = sigma_max*((PML_DEPTH+1.0-j)/PML_DEPTH)**2.0
             c3 = exp(-sigma_y*dt/eps)
             c4 = (1-c3)/(sigma_y*delta*eta*eta)
             t1=ezxbr(i,j)+ezybr(i,j)

             if (i.eq.1) then
                t2=ezxbb(max_x,j)+ezybb(max_x,j)
             else
                t2=ezxbr(i-1,j)+ezybr(i-1,j)
             end if

             if (j.eq.1) then
                t3=0
             else
                t3=ezxbr(i,j-1)+ezybr(i,j-1)
             end if

             hyybr(i,j)=c1*hyybr(i,j)+c2*eta*(t1-t2)
             hxxbr(i,j)=c3*hxxbr(i,j)-c4*eta*(t1-t3)
 80       continue
 70    continue

c**** Bottom Center Region (BB), sigma_y nonzero

       do 90 i=1,max_x
          do 100 j=1,PML_DEPTH
             sigma_y = sigma_max*((PML_DEPTH+1.0-j)/PML_DEPTH)**2.0
             c3 = exp(-sigma_y*dt/eps)
             c4 = (1-c3)/(sigma_y*delta*eta*eta)

             t1=ezxbb(i,j)+ezybb(i,j)

             if (i.eq.1) then
                t2=ezxbl(PML_DEPTH,j)+ezybl(PML_DEPTH,j)
             else
                t2=ezxbb(i-1,j)+ezybb(i-1,j)
             end if

             if (j.eq.1) then
                t3=0
             else
                t3=ezxbb(i,j-1)+ezybb(i,j-1)
             end if

             hyybb(i,j)=hyybb(i,j)+(c*dt/delta)*(t1-t2)
             hxxbb(i,j)=c3*hxxbb(i,j)-c4*eta*(t1-t3)
 100         continue
 90       continue

c**** Top Center Region (TT), sigma_y nonzero

       do 110 i=1,max_x
          do 120 j=1,PML_DEPTH
             sigma_y = sigma_max*((j+0.0)/PML_DEPTH)**2.0
             c3 = exp(-sigma_y*dt/eps)
             c4 = (1-c3)/(sigma_y*delta*eta*eta)

             t1=ezxtt(i,j)+ezytt(i,j)

             if (i.eq.1) then
                t2=ezxtl(PML_DEPTH,j)+ezytl(PML_DEPTH,j)
             else
                t2=ezxtt(i-1,j)+ezytt(i-1,j)
             end if

             if (j.eq.1) then
                t3=ez(i,max_y)
             else
                t3=ezxtt(i,j-1)+ezytt(i,j-1)
             end if

             hyytt(i,j)=hyytt(i,j)+(c*dt/delta)*(t1-t2)
             hxxtt(i,j)=c3*hxxtt(i,j)-c4*eta*(t1-t3)
 120         continue
 110      continue

c**** Right Center Region (RR), sigma_x nonzero

       do 130 i=1,PML_DEPTH
          sigma_x = sigma_max*((i+0.0)/PML_DEPTH)**2.0
          c1 = exp(-sigma_x*dt/eps)
          c2 = (1-c1)/(sigma_x*delta*eta*eta)

          do 140 j=1,max_y

             t1=ezxrr(i,j)+ezyrr(i,j)

             if (i.eq.1) then
                t2=ez(max_x,j)
             else
                t2=ezxrr(i-1,j)+ezyrr(i-1,j)
             end if

             if (j.eq.1) then
                t3=ezxbr(i,PML_DEPTH)+ezybr(i,PML_DEPTH)
             else
                t3=ezxrr(i,j-1)+ezyrr(i,j-1)
             end if

             hyyrr(i,j)=c1*hyyrr(i,j)+c2*eta*(t1-t2)
             hxxrr(i,j)=hxxrr(i,j)-(c*dt/delta)*(t1-t3)
 140      continue
 130   continue

c**** Left Center Region (LL), sigma_x nonzero

       do 150 i=1,PML_DEPTH
          sigma_x = sigma_max*((PML_DEPTH+1.0-i)/PML_DEPTH)**2.0
          c1 = exp(-sigma_x*dt/eps)
          c2 = (1-c1)/(sigma_x*delta*eta*eta)

          do 160 j=1,max_y

             t1=ezxll(i,j)+ezyll(i,j)

             if (i.eq.1) then
                t2=0
             else
                t2=ezxll(i-1,j)+ezyll(i-1,j)
             end if
```

```
            if (j.eq.1) then
               t3=ezxbl(i,PML_DEPTH)+ezybl(i,PML_DEPTH)
            else
               t3=ezxll(i,j-1)+ezyll(i,j-1)
            end if

            hyyll(i,j)=c1*hyyll(i,j)+c2*eta*(t1-t2)
            hxxll(i,j)=hxxll(i,j)-(c*dt/delta)*(t1-t3)
 160      continue
 150   continue

       RETURN
       END

c**************************************************************************
c BOUNDARY_CONDITIONS sets all the appropriate fields in the staircase
c   model to zero.
c**************************************************************************

       SUBROUTINE boundary_conditions

       implicit none
       include 'common.f'

       integer index

       do 10 index = 1,stair_node_count
          if (stair_zero(index,3).eq.ezf) then
             ez(stair_zero(index,1),stair_zero(index,2)) = 0.0
          end if
 10    continue

c**** Set conditions for ground plane.
       if (include_ground_plane) then
          do 20 index = 1,max_x
             ez(index,1) = 0.0
 20       continue
       end if

       RETURN
       END

c**************************************************************************
c GENERATE_INCIDENT_FIELD_LOOKUP_TABLE calculates the propagation of
c   the one-dimensional wave along the k-vector at the current time
c   step that is used to calculate incident at all locations.
c**************************************************************************

c      SUBROUTINE generate_incident_field_lookup_table(time_step)

c      implicit none
c      include 'common.f'

c      integer time_step

c      RETURN
c      END


c**************************************************************************
c REAL FUNCTION EZINC returns the value of the Ez incident field at the
c   given location and time.
c**************************************************************************

       REAL FUNCTION ezinc(i,j,time_step)

       implicit none
       include 'common.f'

       integer i,j,time_step
       real x, y, t, amplitude

       x=(i+0.5)*delta
       y=(j+0.5)*delta-1.5*delta
       t=(time_step-0.0)*dt

       ezinc = amplitude(delay+t+(x*cost+y*sint)/c)

       RETURN
       END


c**************************************************************************
c REAL FUNCTION HXINC returns the value eta times the Hx incident field
c at the given location and time.
c**************************************************************************

       REAL FUNCTION hxinc(i,j,time_step)

       implicit none
       include 'common.f'

       integer i,j,time_step
       real x, y, t, amplitude

       x=(i+0.5)*delta
       y=(j+0.0)*delta-1.5*delta
       t=(time_step+0.5)*dt

       hxinc = -sint*amplitude(delay+t+(x*cost+y*sint)/c)

       RETURN
       END

c**************************************************************************
c REAL FUNCTION HYINC returns the value eta times the Hy incident field
c   at the given location and time.
c**************************************************************************

       REAL FUNCTION hyinc(i,j,time_step)

       implicit none
       include 'common.f'

       integer i,j,time_step

       real x, y, t, amplitude

       x=(i+0.0)*delta
       y=(j+0.5)*delta-1.5*delta
       t=(time_step+0.5)*dt

       hyinc = cost*amplitude(delay+t+(x*cost+y*sint)/c)

       RETURN
       END

c**************************************************************************
c REAL FUNCTION EZREF returns the value of the Ez reflected field at the
c   given location and time.
c**************************************************************************

       REAL FUNCTION ezref(i,j,time_step)

       implicit none
       include 'common.f'

       integer i,j,time_step
       real x, y, t, amplitude

       x=(i+0.5)*delta
       y=(j+0.5)*delta-1.5*delta
       t=(time_step-0.0)*dt

       ezref = -amplitude(delay+t+(x*cost-y*sint)/c)

       RETURN
       END


c**************************************************************************
c REAL FUNCTION HXREF returns the value eta times the Hx reflected field
c at the given location and time.
c**************************************************************************

       REAL FUNCTION hxref(i,j,time_step)

       implicit none
       include 'common.f'

       integer i,j,time_step
       real x, y, t, amplitude

       x=(i+0.5)*delta
       y=(j+0.0)*delta-1.5*delta
       t=(time_step+0.5)*dt

       hxref = -sint*amplitude(delay+t+(x*cost-y*sint)/c)

       RETURN
       END

c**************************************************************************
c REAL FUNCTION HYREF returns the value eta times the Hy reflected field
c   at the given location and time.
c**************************************************************************

       REAL FUNCTION hyref(i,j,time_step)

       implicit none
       include 'common.f'

       integer i,j,time_step
```

```
      real x, y, t, amplitude

      x=(i+0.0)*delta
      y=(j+0.5)*delta-1.5*delta
      t=(time_step+0.5)*dt

      hyref = -cost*amplitude(delay+t+(x*cost-y*sint)/c)

      RETURN
      END

c**********************************************************************
c REAL FUNCTION AMPLITUDE returns the value of the envelope at a given
c   space-time locations
c**********************************************************************

      REAL FUNCTION amplitude(x)

      implicit none
      include 'common.f'

      real x

      if (modulate.eq.1) then
         amplitude = 15*exp(-((2*sqrt(2.5)*x/width)**2.0))*
     1        sin(2*pi*modfreq*x)
      else
         amplitude = 15*exp(-((2*sqrt(2.5)*x/width)**2.0))
      end if

      RETURN
      END
```

The following, **post.f**, contains the subroutines for performing the DFT on the fly of the fields as well as those for computing the radar cross sections.

```
c**********************************************************************
c UPDATE_FREQS performs the DFT on the fly along a virtual surface
c   that encloses the scatterer source.
c**********************************************************************

      SUBROUTINE update_freqs(time_step)

      implicit none
      include 'common.f'

      integer i, j, time_step, k, fi, ytemp
      real temp, cfreq
      complex tempfactor

      do 10 fi = minf,maxf,stepf
         if (include_ground_plane) then
            k = y2+rcs_space-1
         else
            k = 0
         end if

c         cfreq = low_freq+dfreq*(fi+0.0)
         cfreq = freqlist(fi,1)

         tempfactor = exp(2.0*pi*(0.0,1.0)*dt*time_step*cfreq)
c         print *,2.0*pi*(0.0,1.0)*dt*time_step*(low_freq+
c     1        dfreq*(fi+0.0))

         i = x1-rcs_space
         if (include_ground_plane) then
            ytemp = y1
         else
            ytemp = y1-rcs_space
         end if
         do 20 j=ytemp,y2+(rcs_space-1)
c            if (j.eq.(y1-rcs_space)) print *,k+1
            k=k+1

            temp = ez(i,j)
            ezfreq(fi,k)=ezfreq(fi,k)+temp*tempfactor

            temp = 0.5*(hx(i,j)+hx(i,j+1))
            hxfreq(fi,k)=hxfreq(fi,k)+temp*tempfactor

            temp = 0.5*(hy(i,j)+hy(i+1,j))
            hyfreq(fi,k)=hyfreq(fi,k)+temp*tempfactor
 20      continue

         j=y2+rcs_space
         do 30 i=x1-rcs_space,x2+rcs_space-1
c            if (i.eq.(x1-rcs_space)) print *,k+1
            k=k+1

            temp = ez(i,j)
            ezfreq(fi,k)=ezfreq(fi,k)+temp*tempfactor

            temp = 0.5*(hx(i,j)+hx(i,j+1))
            hxfreq(fi,k)=hxfreq(fi,k)+temp*tempfactor

            temp = 0.5*(hy(i,j)+hy(i+1,j))
            hyfreq(fi,k)=hyfreq(fi,k)+temp*tempfactor
 30      continue

         i=x2+rcs_space
         if (include_ground_plane) then
            ytemp = y1
         else
            ytemp=y1-(rcs_space-1)
         end if
         do 40 j=y2+rcs_space,ytemp,-1
c            if (j.eq.(y2+rcs_space)) print *,k+1
            k=k+1

            temp = ez(i,j)
            ezfreq(fi,k)=ezfreq(fi,k)+temp*tempfactor

            temp = 0.5*(hx(i,j)+hx(i,j+1))
            hxfreq(fi,k)=hxfreq(fi,k)+temp*tempfactor

            temp = 0.5*(hy(i,j)+hy(i+1,j))
            hyfreq(fi,k)=hyfreq(fi,k)+temp*tempfactor
 40      continue

         if (.NOT.include_ground_plane) then
            j=y1-rcs_space
            do 50 i=x2+rcs_space,x1-rcs_space,-1
c               if (i.eq.(x2+rcs_space)) print *,k+1
               k=k+1

               temp = ez(i,j)
               ezfreq(fi,k)=ezfreq(fi,k)+temp*tempfactor

               temp = 0.5*(hx(i,j)+hx(i,j+1))
               hxfreq(fi,k)=hxfreq(fi,k)+temp*tempfactor

               temp = 0.5*(hy(i,j)+hy(i+1,j))
               hyfreq(fi,k)=hyfreq(fi,k)+temp*tempfactor
 50         continue
         end if

 10   continue

      RETURN
      END


c**********************************************************************
c JS_FREQ calculates frequency components along surface of target
c**********************************************************************

      SUBROUTINE Js_freq(time_step)

      implicit none
      include 'common.f'

      integer k, sx1, sy1, time_step, fi
      complex tempfactor
      real x(1:MAX_STAIR_NODES), y(1:MAX_STAIR_NODES), temp
      logical inside_pec

      do 100 k = 1,stair_node_count
         x(k) = real(stair_zero(k,1))
         y(k) = real(stair_zero(k,2))
 100  continue

      do 10 fi = minf,maxf,stepf
         tempfactor = exp(2.0*pi*(0.0,1.0)*dt*time_step*freqlist(fi,1))

         do 20 k = 1,stair_node_count-1

            sx1 = stair_zero(k,1)
            sy1 = stair_zero(k,2)

            if (.NOT.inside_pec(x,y,stair_node_count,sx1+0.5,sy1)) then
               temp = hy(int(sx1+1),int(sy1))
               Jsfreq(fi,k) = Jsfreq(fi,k) + temp*tempfactor
            else if (.NOT.inside_pec(x,y,stair_node_count,sx1,sy1+0.5))
     1              then
               temp = hx(int(sx1),int(sy1+0.5))
               Jsfreq(fi,k) = Jsfreq(fi,k) + temp*tempfactor
            else if (.NOT.inside_pec(x,y,stair_node_count,sx1-0.5,sy1))
     1              then
               temp = -hy(int(sx1),int(sy1))
```

```
               Jsfreq(fi,k) = Jsfreq(fi,k) + temp*tempfactor
            else if (.NOT.inside_pec(x,y,stair_node_count,sx1,sy1-0.5))
     1              then
               temp = -hx(int(sx1),int(sy1))
               Jsfreq(fi,k) = Jsfreq(fi,k) + temp*tempfactor
            else
               Jsfreq(fi,k) = (0.0,0.0)
            end if
 20      continue

 10   continue


      RETURN
      END

c**********************************************************************
c JSFREQ_OUT writes out surface currents on target for debugging.
c**********************************************************************

      SUBROUTINE Jsfreq_out

      implicit none
      include 'common.f'

      integer k, sx1, sy1, fi,index1
      real phi, radius, cfreq, amplitude, time, refx, refy
      complex temp, eincfreq

      refx = real(max_x)/2.0
      refy = real(max_y)/2.0

      open(unit=10,file='Js.dat',status='unknown',form='formatted')

      do 10 fi = minf,maxf,stepf
c         cfreq = low_freq+dfreq*(fi+0.0)
         cfreq = freqlist(fi,1)

         eincfreq = 0.0
         do 20 index1 = 1, tot_time_steps
            time = (index1+0.0)*dt
            eincfreq = eincfreq + amplitude(delay+time+(delta*max_x*
     1           cost/2.0+delta*max_y*sint/2.0)/c)*cexp(2.0*pi*
     1           (0.0,1.0)*(time*cfreq))
 20      continue

         do 30 k = 1, stair_node_count-1
            sx1 = stair_zero(k,1)
            sy1 = stair_zero(k,2)
            temp = Jsfreq(fi,k)/eta/abs(eincfreq)
c            temp = Jsfreq(fi,k)
            radius = sqrt((real(sx1-refx))**2.0+(real(sy1-refy))**2.0)
            phi = atan2((real(sy1-refy)),(real(sx1-refx)))
            write(10,*) cfreq,radius,phi,real(temp),imag(temp)
 30      continue

 10   continue

      close(unit=10)

      RETURN
      END


c**********************************************************************
c WRITE_OUT_FREQS writes out all freq field comp data.
c**********************************************************************

      SUBROUTINE write_out_freqs

      implicit none
      include 'common.f'

      integer fi, k, rbeg, rend
      complex temp

      write(6,*) 'Writing out frequency data...'

      open(unit=10,file='ezf.dat',status='unknown',form='formatted')
      open(unit=11,file='hxf.dat',status='unknown',form='formatted')
      open(unit=12,file='hyf.dat',status='unknown',form='formatted')

c**** Save state

      open(unit=13,file='info.dat',status='unknown',form='formatted')

      write(13,*) max_x
      write(13,*) max_y
      write(13,*) delta
      write(13,*) dt
      write(13,*) tot_time_steps
      write(13,*) x1,y1,x2,y2
      write(13,*) inc_ang
      write(13,*) modulate
      write(13,*) modfreq
      write(13,*) delay
      write(13,*) width
      write(13,*) include_ground_plane
      write(13,*) num_freqs
      do 30 fi = minf,maxf
         write(13,*) freqlist(fi,1)
 30   continue

      close(unit=13)

      if (include_ground_plane) then
         rbeg = y2+rcs_space
         rend = 3*y2+x2-x1-2+5*rcs_space
      else
         rbeg = 1
         rend = 2*(x2-x1)+2*(y2-y1)+8*rcs_space+1
      end if
      do 10 fi = minf,maxf,stepf
         do 20 k=rbeg,rend
            temp = ezfreq(fi,k)
            write(10,*) real(temp),aimag(temp)
            temp = hxfreq(fi,k)
            write(11,*) real(temp),aimag(temp)
            temp = hyfreq(fi,k)
            write(12,*) real(temp),aimag(temp)
 20      continue
 10   continue

      close(unit=10)
      close(unit=11)
      close(unit=12)

      RETURN
      END

c**********************************************************************
c CALC_RCS calculates the 2D RCS at the requested output locations.
c**********************************************************************

      SUBROUTINE calc_rcs

      implicit none
      include 'common.f'

      integer fi,index1,llc,ulc,urc,lrc, tfreq
      real cfreq, kwave, sinp, cosp, phi_obs, time, cphi_obs
      real xphys, yphys, amplitude, rcs, dpfreq, dkwave,iphi_obs
c      real tempr, tempi

      complex eincfreq, I1, I2, I3, I4, Fphi, uniti, phase, rcsi

      write(6,*) 'Calculating RCS...'

c**** Define unit imaginary number
      uniti = (0.0,1.0)

c**** Redefine x1,x2,y1,y2 so that are the corners RCS box

      if (include_ground_plane) y1=-y2+2

      x1 = x1-rcs_space
      x2 = x2+rcs_space
      y1 = y1-rcs_space
      y2 = y2+rcs_space

c**** Determine labels for corners based on update_freq
c**** llc = lower left corner, ulc = upper left corner
c**** lrc = lower right corner, urc = upper right corner

c**** this one has two labels, 1, and what's given below
      llc = 2*(x2-x1)+2*(y2-y1)+1

      ulc = y2-y1+1
      urc = x2-x1+y2-y1+1
      lrc = 2*(y2-y1)+x2-x1+1
      print *, x1,y1,x2,y2
      print *, 1, ulc, urc, lrc, llc

c**** with ground plane use image theory to gen. fields for lower half
c**** of huygens' surface. Equiv. to using layered Green's functions.
      if (include_ground_plane) then
         do 500 fi=minf,maxf,stepf
            do 510 index1=1,y2-1
c************* Reflect left top side of Huygens' surface down
               ezfreq(fi,index1) = -ezfreq(fi,-index1+2*y2)
               hxfreq(fi,index1+1) = hxfreq(fi,-(index1+1)+2*y2+1)
               hyfreq(fi,index1) = -hyfreq(fi,-index1+2*y2)

c************* Reflect right topside of Huygens' surface down
```

```
               ezfreq(fi,lrc-index1+1) = -ezfreq(fi,urc+index1-1)
               hxfreq(fi,lrc-index1) = hxfreq(fi,urc+index1-1)
               hyfreq(fi,lrc-index1+1) = -hyfreq(fi,urc+index1-1)
 510        continue
c********** Approximations b/c nothing to reflect
            hxfreq(fi,1) = hxfreq(fi,2)
            hxfreq(fi,lrc) = hxfreq(fi,lrc-1)

            do 520 index1=ulc+1,urc-1
c************* Reflect middle top of Huygens' surface down
               ezfreq(fi,llc-index1+ulc) = -ezfreq(fi,index1)
               hxfreq(fi,llc-index1+ulc) = hxfreq(fi,index1)
               hyfreq(fi,llc-index1+ulc) = -hyfreq(fi,index1)
 520        continue
 500     continue
      end if

      open(unit=10,file='rcs.dat',status='unknown',form='formatted')
      open(unit=14,file='inc.dat',status='unknown',form='formatted')
      open(unit=15,file='debug.dat',status='unknown',form='formatted')

c*********************** READ IN EXACT DATA
c      open(unit=11,file='ez.dat',status='unknown',form='formatted')
c      open(unit=12,file='hx.dat',status='unknown',form='formatted')
c      open(unit=13,file='hy.dat',status='unknown',form='formatted')
c
c      do 200 index1 = 1, llc-1
c         read(11,*) tempr, tempi
c         ezfreq(0,index1) = (tempr+(0.0,1.0)*tempi)
c         read(12,*) tempr, tempi
c         hxfreq(0,index1) = (tempr+(0.0,1.0)*tempi)
c         read(13,*) tempr, tempi
c         hyfreq(0,index1) = (tempr+(0.0,1.0)*tempi)
c 200  continue
c      close(unit=11)
c      close(unit=12)
c      close(unit=13)
c
c      ezfreq(0,llc) = ezfreq(0,1)
c      hxfreq(0,llc) = hxfreq(0,1)
c      hyfreq(0,llc) = hyfreq(0,1)
c
c*********************** END OF READING IN EXACT DATA

c**** loop through all freqs of interest.

      do 10 fi = minf, maxf, stepf
c         cfreq = low_freq+dfreq*(fi+0.0)

c******* Note 1: tfreq = 0 if normal frequency.
c*******         tfreq = i (i=1..MNANG) cfreq(46,1) contains the true
c*******         freq.
c******* Note 2: the monostatic angles calculated are always
c*******         inc_ang-45, ...., inc_ang+45 in 1 deg increments
c******* Note 3: The variables dpfreq and dkwave are determined based
c*******         on the current frequency at which the RCS is actually
c*******         being calculated at.

         cfreq = freqlist(fi,1)
         tfreq = freqlist(fi,2)
         if (tfreq.eq.0) then
            dpfreq = cfreq
         else
            dpfreq = freqlist(46+int((fi-1.0)/(MNANG+0.0))*MNANG,1)
            if (freqlist(46+int((tfreq-1.0)/(MNANG+0.0))*MNANG,2).ne.
     1           46) then
               print *, 'error, bad location of actual freq'
            end if
         end if

         dkwave = 2.0*pi*dpfreq/c
         kwave = 2.0*pi*cfreq/c

         eincfreq = 0.0
         do 20 index1 = 1, tot_time_steps
            time = (index1+0.0)*dt
            eincfreq = eincfreq + amplitude(delay+time+(delta*max_x*
     1           cost/2.0+delta*max_y*sint/2.0)/c)*cexp(2.0*pi*uniti*
     1           (time*cfreq))
 20      continue
c         print *,cfreq,eincfreq

c**** loop through all scattering angles of interest

c**** if mono_bi == 1, choose normal bistatic RCS angles
c****   in this case, phi_obs and cphi_obs are equal since the reported
c****   angle is the same as the angle used in the calculations.

c**** if mono_bi == 2, choose angles based on freqs for monostatic RCS
c****   in this case, phi_obs represents the monostatic angle we are
c****   trying to calculate the RCS at, while cphi_obs represents the
c****   bistatic angle we use to approx the monostatic angle we need

         do 30 iphi_obs = low_phi, high_phi, dphi
            phi_obs = iphi_obs
            if (mono_bi.eq.1) then
               cphi_obs = phi_obs
            else
               phi_obs = inc_ang + freqlist(fi,2)-46.0
               cphi_obs = 2*phi_obs-inc_ang

c*********      verify that we are using the correct cfreq (debug mode)
               if (abs(cfreq-dpfreq*(1.0/cos((phi_obs-inc_ang)*
     1              pi/180))).gt.tole) then
                  print *, 'error, miscalculated frequency'
c                  stop
               end if
c               write(15,*) cfreq,dpfreq,phi_obs,cphi_obs
            end if

            cosp = cos((cphi_obs/180.0)*pi)
            sinp = sin((cphi_obs/180.0)*pi)

c**** Initialize values of integrals.
            I1 = (0.0,0.0)
            I2 = (0.0,0.0)
            I3 = (0.0,0.0)
            I4 = (0.0,0.0)

c*****************************************************************
c************************* INTEGRAL 4 ****************************
c*****************************************************************
C**** Compute Integral over y2->y1 at x1

            do 40 index1 = 1,ulc
               yphys = real(y1+index1-1)*delta
               xphys = real(x1)*delta
               phase = -uniti*kwave*yphys*sinp
               if (index1.gt.((1+ulc)/2)) then
               I4 = I4 + (-hyfreq(fi,index1) + ezfreq(fi,index1)*
     1              cosp)*cexp(+phase)*delta
               end if
 40         continue

            I4 = I4*cexp(-uniti*kwave*xphys*cosp)

c*****************************************************************
c************************* INTEGRAL 3 ****************************
c*****************************************************************
c**** Compute Integral over x2->x1 at y2

            do 60 index1 = ulc,urc
               yphys = real(y2)*delta
               xphys = real(x1+index1-ulc)*delta
               phase = -uniti*kwave*xphys*cosp

               I3 = I3 + (-hxfreq(fi,index1)-ezfreq(fi,index1)*
     1              sinp)*cexp(+phase)*delta
 60         continue

            I3 = I3*cexp(-uniti*kwave*yphys*sinp)

c*****************************************************************
c************************* INTEGRAL 2 ****************************
c*****************************************************************
c**** Compute Integral over y1->y2 at x2

            do 80 index1 = urc,lrc
               yphys = real(y2-index1+urc)*delta
               xphys = real(x2)*delta
               phase = -uniti*kwave*yphys*sinp
               if (index1.lt.((urc+lrc)/2)) then
               I2 = I2 + (hyfreq(fi,index1)-ezfreq(fi,index1)*
     1              cosp)*cexp(+phase)*delta
               end if
 80         continue

            I2 = I2*cexp(-uniti*kwave*xphys*cosp)

c*****************************************************************
c************************* INTEGRAL 1 ****************************
c*****************************************************************
c**** Compute Integral over x1->x2 at y1

            do 100 index1 = lrc,llc
               yphys = real(y1)*delta
               xphys = real(x2-index1+lrc)*delta
               phase = -uniti*kwave*xphys*cosp

               I1 = I1 + (hxfreq(fi,index1)+ezfreq(fi,index1)*
     1              sinp)*cexp(+phase)*delta
 100        continue

            I1 = I1*cexp(-uniti*kwave*yphys*sinp)
```

```
C**** Sum Integrals

           Fphi = cexp(-uniti*pi/4.0)*sqrt(kwave/8.0/pi)*(I1+
     1           I2+I3+I4)

c           print *,cfreq,phi_obs,Fphi
C**** Calculate RCS

c           print *,eincfreq
           rcs = 2*pi*((cabs(Fphi))**2.0)/((cabs(eincfreq))**2.0)
           rcsi = sqrt(2*pi)*Fphi/eincfreq

c**** For use when reading in exact near field data.
c           rcs = 2*pi*((cabs(Fphi))**2.0)

c**** Write out results (RCS given in dBm)
c**** Note extra column of zeros written out to allow compatibility
c**** with MATLAB scripts that read RCS data and plot them.

           write(10,*) dkwave,0.0,phi_obs,10*ALOG10(rcs),real(rcsi),
     1           imag(rcsi)
           write(14,*) dkwave,0.0,phi_obs,abs(rcsi),atan2(imag(rcsi),
     1           real(rcsi))


 30      continue
 10   continue

      close(unit=10)
      close(unit=14)
      close(unit=15)

      RETURN
      END

c***********************************************************************
c WRITE_OUT_MOVIE_FRAME
c***********************************************************************

      SUBROUTINE write_out_movie_frame(a,time_step)

      implicit none
      include 'common.f'

      character a(1:dummy)

      integer i,j,ia,time_step
      real ezinc

c      integer nsplit,ns,l
      integer hlevels,numcolors1,center,topcolor,nctshift,ngray
c     topcolor: location of top of colorbar
c     numcolors1: 1 less than # of colors in colorbar
      parameter (numcolors1=128,topcolor=243)
c     nctshift: = color table shift
      parameter(nctshift = topcolor-numcolors1)
      parameter(ngray = topcolor-numcolors1-1)
c     hlevels = numcolors1/4
      parameter (hlevels=numcolors1/4)
c     center = 2*hlevels + 1/2
      parameter (center = 2*hlevels + 0.5)
      character frmt*30

      write(frmt,'(a1,i4,a5)') '(', max_x, '(a1))'

      do 10 j=min(1000,max_y),1,-1
         do 20 i=1,min(1000,max_x)
c           ia=int((ez(i,j))*hlevels+center+0.5)

c           if (tot_scat(i,j).eq.0.OR.tot_scat(i,j).eq.1.OR.
c     1           tot_scat(i,j).eq.11.OR.tot_scat(i,j).eq.12.OR.
c     1           tot_scat(i,j).eq.15) then
c             ia=int(((ez(i,j)+ezinc(i,j,time_step+1))/4.0)*hlevels+
c     1             center+0.5)
c           else
             ia=int((ez(i,j)/4.0)*hlevels+center+0.5)
c           end if
c           if (abs(ez(i,j).gt.max_field)) then
c              max_field=abs(ez(i,j))
c              print *,time_step,i,j,max_field
c           end if

           if (ia .lt. 0) ia=0
           if (ia .gt. numcolors1) ia=numcolors1
           a(i)=char(ia+nctshift)
 20      continue

         write(4,frmt) a
 10   continue

      RETURN
      END

c***********************************************************************
c WRITE_OUT_ALL_PARMS outputs to a file all important parameters used
c   in running the simulation.
c***********************************************************************

      SUBROUTINE write_out_all_parms(store_freq)

      implicit none
      include 'common.f'

      integer index
      real x(1:MAX_STAIR_NODES), y(1:MAX_STAIR_NODES)
      logical store_freq

      do 10 index = 1,stair_node_count
         x(index) = real(stair_zero(index,1))
         y(index) = real(stair_zero(index,2))
 10   continue

      open(unit=10,file='fdtd.out',status='unknown',form='formatted')

      write(10,*) 'max_x = ', max_x,';'
      write(10,*) 'max_y = ', max_y,';'
      write(10,*) 'delta = ', delta,';'
      write(10,*) 'dt = ', dt,';'
      write(10,*) 'N = ', tot_time_steps,';'
      write(10,*) 'inc_ang = ', inc_ang,';'
      write(10,*) 'modulate = ', modulate,';'
      write(10,*) 'modfreq = ', modfreq,';'
      write(10,*) 'delay = ', delay,';'
      write(10,*) 'width = ', width,';'
      write(10,*)
      write(10,*) 'width/dt = ', width/dt
      write(10,*) 'stair_node_count = ', stair_node_count
      write(10,*) 'sigma_max = ', sigma_max
      write(10,*) 'x1,y1,x2,y2= ',x1,y1,x2,y2
      if (include_ground_plane) then
         write(10,*) 'rcs_nodes = ',2*(x2-x1)+4*y2-4+8*rcs_space+1
      else
         write(10,*) 'rcs_nodes = ', 2*(y2-y1)+2*(x2-x1)+8*rcs_space+1
      end if
      if (store_freq) then
         write(10,*) 'low_freq (GHz) = ', freqlist(1,1)/1.0e9
         write(10,*) 'high_freq (GHz) = ', freqlist(num_freqs,1)/1.0e9
         write(10,*) 'num_freqs = ', maxf-minf+1
      end if
      call plotb(x,y,stair_node_count,51,41,10)

      close(unit=10)

      RETURN
      END


c***********************************************************************
c PLOTB takes a set of data points (X,Y) and makes and ascii plot out
c   of them
c***********************************************************************

      SUBROUTINE PLOTB(X,Y,N,NC,NR,FID)
C
C WRITTEN 2/14/74   BY J. M. PUTNAM       DEPT 220     X23877
C
C THIS ROUTINE PRODUCES A LINEAR XY PLOT.
C    N IS THE NUMBER OF POINTS TO BE PLOTTED.
C    NR IS THE NUMBER OF ROWS TO BE USED FOR THE Y-AXIS.
C    NC IS THE NUMBER OF COLUMNS TO BE USED FOR THE X-AXIS.
C       NOTE, NC-1 MUST BE DIVISIBLE BY 10 AND LESS THAN 102.
C

      REAL X(161),Y(161),HEAD(10)
      INTEGER  LINE(101),BLANK,STAR,FID
      DATA BLANK,STAR /1H ,1H*/
      N10=(NC-1)/10
      WRITE(FID,500)
 500  FORMAT(//,17H1BODY COORDINATES)
      WRITE(FID,504)
      XMIN=X(1)
      XMAX=X(1)
      YMIN=Y(1)
      YMAX=Y(1)
      DO 6 I=1,N
      IF(X(I).LT.XMIN) XMIN=X(I)
      IF(X(I).GT.XMAX) XMAX=X(I)
      IF(Y(I).LT.YMIN) YMIN=Y(I)
      IF(Y(I).GT.YMAX) YMAX=Y(I)
 6    CONTINUE
      DEL=XMAX-XMIN
      IF(YMAX-YMIN.GT.DEL) DEL=YMAX-YMIN
      XMAX=XMIN+DEL
```

```
      YMAX=YMIN+DEL
      DO 5 I =1,N10
      Z=I
    5 HEAD(I)=(XMAX-XMIN)*Z/N10+XMIN
      DY=(YMAX-YMIN)/(NR-1)
      Z=YMAX+DY
      YL=Z-DY/2.
      DO 7 J=1,NR
      DO 8 K=1,NC
 8    LINE(K)=BLANK
      Z=Z-DY
      YU=YL
      YL=Z-DY/2.
      DO 9 I=1,N
      IF(Y(I).GE.YU) GO TO 9
      IF(Y(I).LT.YL) GO TO 9
      K=(X(I)-XMIN)/(XMAX-XMIN)*(NC-1)+1.5
      IF(K.GT.NC) K=NC
      LINE(K)=STAR
 9    CONTINUE
      WRITE(FID,508) Z,(LINE(K),K=1,NC)
 7    CONTINUE
      WRITE(FID,504)
      WRITE(FID,3002)
      WRITE(FID,507) XMIN,(HEAD(I),I=1,N10)
C     ************************
      RETURN
  504 FORMAT ( 1X, 14(1H-), 1H., 10(5H----.), 1H- )
  507 FORMAT(10X,11(F10.4))
  508 FORMAT (1X, F12.4,1X, 1HI, 51A1, 1HI )
 3002 FORMAT(4X,7HRB / ZB,4X,1HI,5(9X,1HI))

      END
```

The following, **common.f**, is used to create the common blocks that are included in most of the subroutines used by the 2D FD-TD TE program.

```
c**** common.include file.  Contains all global variable declarations.

c**** Variables changed to allocate memory

      integer MAX_X_CELLS, MAX_Y_CELLS, MAX_NODES, MAX_STAIR_NODES,
     1     MAX_RCS_NODES, MAX_FREQS

      parameter(MAX_X_CELLS=680)
      parameter(MAX_Y_CELLS=520)
      parameter(MAX_NODES=17)
      parameter(MAX_STAIR_NODES=2481)
      parameter(MAX_RCS_NODES=1737)
      parameter(MAX_FREQS=102)

c*************************************************************************
c*****  DO NOT CHANGE BELOW ***********************************************
c*************************************************************************

c**** Important contants
      real c, mu, pi, eps, eta

      parameter(c=2.9979247917E8, mu=1.25663706144E-6, pi=3.1415926535,
     1     eps=8.8541874E-12, eta=376.73032)

      real tole
      parameter(tole = 1.0e-8)

c**** Simulation Variables

      real delta, dt, sigma_max, reflection, ij
      integer tot_time_steps, dummy, movie_step
      logical store_movie

c**** Grid layout variables.

      integer PML_DEPTH

      parameter(PML_DEPTH=12)
      integer tot_scat(1:MAX_X_CELLS,1:MAX_Y_CELLS)

      integer max_x, max_y

c**** Points 1 & 2 define the corners of a tot/field region
      integer x1,y1,x2,y2

c**** Geometry data points
      real xnodes(1:MAX_NODES), ynodes(1:MAX_NODES)
      integer total_nodes, stair_node_count,
     1     stair_zero(1:MAX_STAIR_NODES,1:3), ezf, exf, eyf

      logical include_ground_plane

      parameter(ezf=1,exf=2,eyf=3)

      integer errorcount, errors(10),
     1     NODE_ERROR, MAX_X_ERROR, MAX_Y_ERROR, MAX_STAIR_ERROR,
     2     MAX_RCS_ERROR

      parameter(NODE_ERROR=1, MAX_X_ERROR=2, MAX_Y_ERROR=3,
     1     MAX_STAIR_ERROR=4, MAX_RCS_ERROR=5)

c**** Incident Wave parameters
      real inc_ang, modfreq, sint, cost, delay, width
      integer modulate

c**** Fields: TM Case

c**** Incident fields

      integer MAX_M_CELLS
      parameter(MAX_M_CELLS = (MAX_X_CELLS+MAX_Y_CELLS))

      real Hinc(1:MAX_M_CELLS), Einc(1:MAX_M_CELLS)

c**** Normal fields

      real hx(1:MAX_X_CELLS,1:MAX_Y_CELLS),
     1     hy(1:MAX_X_CELLS,1:MAX_Y_CELLS),
     2     ez(1:MAX_X_CELLS,1:MAX_Y_CELLS)

c**** PML fields

c**** Top and Center fields
      real ezxtt(1:MAX_X_CELLS,1:PML_DEPTH),
     1     ezytt(1:MAX_X_CELLS,1:PML_DEPTH),
     1     hxxtt(1:MAX_X_CELLS,1:PML_DEPTH),
     1     hyytt(1:MAX_X_CELLS,1:PML_DEPTH)

c**** Bottom and Center fields
      real ezxbb(1:MAX_X_CELLS,1:PML_DEPTH),
     1     ezybb(1:MAX_X_CELLS,1:PML_DEPTH),
     1     hxxbb(1:MAX_X_CELLS,1:PML_DEPTH),
     1     hyybb(1:MAX_X_CELLS,1:PML_DEPTH)

c**** Right and Center fields
      real ezxrr(1:PML_DEPTH,1:MAX_Y_CELLS),
     1     ezyrr(1:PML_DEPTH,1:MAX_Y_CELLS),
     1     hxxrr(1:PML_DEPTH,1:MAX_Y_CELLS),
     1     hyyrr(1:PML_DEPTH,1:MAX_Y_CELLS)

c**** Left and Center fields
      real ezxll(1:PML_DEPTH,1:MAX_Y_CELLS),
     1     ezyll(1:PML_DEPTH,1:MAX_Y_CELLS),
     1     hxxll(1:PML_DEPTH,1:MAX_Y_CELLS),
     1     hyyll(1:PML_DEPTH,1:MAX_Y_CELLS)

c**** Top and Right fields
      real ezxtr(1:PML_DEPTH,1:PML_DEPTH),
     1     ezytr(1:PML_DEPTH,1:PML_DEPTH),
     1     hxxtr(1:PML_DEPTH,1:PML_DEPTH),
     1     hyytr(1:PML_DEPTH,1:PML_DEPTH)

c**** Top and Left fields
      real ezxtl(1:PML_DEPTH,1:PML_DEPTH),
     1     ezytl(1:PML_DEPTH,1:PML_DEPTH),
     1     hxxtl(1:PML_DEPTH,1:PML_DEPTH),
     1     hyytl(1:PML_DEPTH,1:PML_DEPTH)

c**** Bottom and Right fields
      real ezxbr(1:PML_DEPTH,1:PML_DEPTH),
     1     ezybr(1:PML_DEPTH,1:PML_DEPTH),
     1     hxxbr(1:PML_DEPTH,1:PML_DEPTH),
     1     hyybr(1:PML_DEPTH,1:PML_DEPTH)

c**** Bottom and Left fields
      real ezxbl(1:PML_DEPTH,1:PML_DEPTH),
     1     ezybl(1:PML_DEPTH,1:PML_DEPTH),
     1     hxxbl(1:PML_DEPTH,1:PML_DEPTH),
     1     hyybl(1:PML_DEPTH,1:PML_DEPTH)

c**** Frequency domain points
      integer minf, maxf, stepf, num_freqs, rcs_space
      parameter(rcs_space=2)

      real freqlist(1:MAX_FREQS,1:2)
      integer mono_bi
      complex ezfreq(1:MAX_FREQS, 1:MAX_RCS_NODES),
     1     hxfreq(1:MAX_FREQS, 1:MAX_RCS_NODES),
     2     hyfreq(1:MAX_FREQS, 1:MAX_RCS_NODES),
     3     Jsfreq(1:MAX_FREQS, 1:MAX_STAIR_NODES)

C**** Maximum number of monostatic observation points. Currently
```

```
C**** set to angle resolution of 1 degree.
      integer MNANG
      parameter(MNANG=91)

c**** Output information
      real low_phi, high_phi, dphi

c**** Debug output

      complex debugfreq(1:200)
      common debugfreq

c*********************************************************************c

c**** Fields Common Block ************

      common hx, hy, ez,
     1     ezxtt, ezytt, hxxtt, hyytt,
     1     ezxbb, ezybb, hxxbb, hyybb,
     1     ezxll, ezyll, hxxll, hyyll,
     1     ezxrr, ezyrr, hxxrr, hyyrr,
     1     ezxtr, ezytr, hxxtr, hyytr,
     1     ezxtl, ezytl, hxxtl, hyytl,
     1     ezxbr, ezybr, hxxbr, hyybr,
     1     ezxbl, ezybl, hxxbl, hyybl,
     1     tot_scat, Hinc, Einc, ezfreq,
     1     hxfreq, hyfreq, Jsfreq, freqlist

c**** Incident Wave Parms Common Block *******

      common inc_ang, modulate, modfreq,
     1     sint, cost, delay, width

c**** Simulation & Grid Layout Common Block *******

      common delta, dt, tot_time_steps, store_movie, movie_step,
     1     max_x, max_y, sigma_max, reflection,
     2     x1, y1, x2, y2, xnodes, ynodes, total_nodes,
     3     stair_zero, stair_node_count, errors, errorcount,
     4     dummy, ij, include_ground_plane

c**** Output variables common block

      real max_field

      common low_phi, high_phi, dphi, num_freqs,
     1     minf, maxf, stepf, max_field, mono_bi
```

D.3 2D FD-TD Program for TM Mode

Similar to the 2D FD-TD program for the TE mode, except that the TM mode radar cross sections are calculated. It should be noted that in the following code, E field variable names are actually H fields, while H field variable names are actually E fields. This is due to the fact that the TE mode code was modified using the principles of duality to form this code.

The following, **fdtd_2d.f**, contains the subroutines used for reading in the input parameters from the user.

```
c****************************************
c**** 2D FDTD H-polarization code ******
c****        FDTD_2D.F            ******
c****************************************

************************************************************************
* 2D-FDTD CODE:                                                        *
*     This program computes the EM scattering in two dimensions. It    *
*     assumes that objects and EM fields have NO variation in the z-   *
*     direction.                                                       *
************************************************************************

      program fdtd_2d

      implicit none
      integer menu_choice

 10   write(6,*)
      write(6,*) 'What would you like do?'
      write(6,*) '1 = FDTD, WRITE FREQ, RCS'
      write(6,*) '2 = FDTD, RCS'
      write(6,*) '3 = READ FREQ, RCS'
      write(6,*) '4 = FDTD'

      read(5,*) menu_choice
      if (menu_choice.lt.1.OR.menu_choice.gt.4) goto 10

      if (menu_choice.eq.1) then
         call get_primary_input
         call get_rcs_output_ranges(.FALSE.)
         call init_fields
         call init_freqs
         call fdtd_loop(.TRUE.)
         call write_out_freqs
         call calc_rcs
      else if (menu_choice.eq.2) then
         call get_primary_input
         call get_rcs_output_ranges(.FALSE.)
         call init_fields
         call init_freqs
         call fdtd_loop(.TRUE.)
         call calc_rcs
      else if (menu_choice.eq.3) then
         call read_in_freqs
         call get_rcs_output_ranges(.TRUE.)
         call calc_rcs
      else if (menu_choice.eq.4) then
         call get_primary_input
         call init_fields
         call fdtd_loop(.FALSE.)
      end if

      END

c**********************************************************************
c GET_PRIMARY_INPUT gets info from user about geomfile name, incident
c    wave, duration of simulation, output file names (including movie).
c**********************************************************************
      SUBROUTINE get_primary_input

      implicit none
      include 'common.f'

      character*72 geomfile,mhname, mfname
      integer movie_test, ground_plane_test

c      write(6,*) 'get_primary_input'

      write(6,'(''*Enter geometry file name: '', $)')
      read(5,*) geomfile

      write(6, '(''*Include ground plane? (1=Y,2=N): '', $)')
      read(5,*) ground_plane_test
      include_ground_plane = (ground_plane_test.eq.1)

      write(6,'(''*Enter number of time steps to run: '', $)')
      read(5,*) tot_time_steps

      write(6,'(''*Store for movie? (1=Y, 2=N): '', $)')
      read(5,*) movie_test
      store_movie = (movie_test.eq.1)

      if (store_movie) then
         write(6,'(''*Movie header name: '',$)')
         read(5,*) mhname
         write(6,'(''*Movie imgfile name: '', $)')
         read(5,*) mfname
         write(6,'(''*Number of time steps between each frame: '', $)')
         read(5,*) movie_step
      end if

      write(6,'(''*Enter incident angle in degrees: '', $)')
      read(5,*) inc_ang

      write(6,'(''*Enter modulation frequency (0=unmodulated): '',$)')
      read(5,*) modfreq

      if (abs(modfreq).lt.tole) then
         modulate = 0
      else
         modulate = 1
      end if

      call setup_geometry(geomfile)

      if (store_movie) call setup_movie(mhname,mfname)

      RETURN
      END


c**********************************************************************
c GET_RCS_OUTPUT_RANGES gets info from user about what angles and freqs
c    to calc the RCS for.
c**********************************************************************

      SUBROUTINE get_rcs_output_ranges(skip_fd)

      implicit none
      include 'common.f'

      integer nang, fi, mi
      logical skip_fd
      real low_freq, dfreq, high_freq

c      write(6,*) 'get_rcs_out_ranges'

      if (.NOT.skip_fd) then
 100  write(6,*)
      write(6,*) '1. Calculate bistatic RCS for multiple freqs'
      write(6,*) '2. Estimate monostatic RCS versus angle for one freq'
      write(6,'(''*Enter your choice: '',$)')
      read(5,*) mono_bi
      if (mono_bi.ne.1.AND.mono_bi.ne.2) goto 100
      else
         mono_bi =1
      end if

      if (mono_bi.eq.1) then
c******* calculate bistatic RCS for multiple freqs...
         if (.NOT.skip_fd) then
            write(6,'(''*Enter lowest frequency of interest: '',$)')
            read(5,*) low_freq
```

```
            write(6,'(''*Enter highest frequency of interest: '',$)')
            read(5,*) high_freq

            if (abs(low_freq-high_freq).gt.tole) then

 10            write(6,'(''*Enter the number of frequencies: '',$)')
               read(5,*) num_freqs

               if (num_freqs.gt.MAX_FREQS) then
                  write(6,*) 'Error. Number of freqs must be ',
     1                 'less than ', MAX_FREQS, ' or raise ',
     2                 'MAX_FREQS parmaeter'
                  write(6,*)
                  goto 10
               end if

               minf = 1
               maxf = num_freqs
               dfreq = (high_freq-low_freq)/(num_freqs-1.0)

               do 20 fi = minf, maxf
                  freqlist(fi,1) = low_freq + dfreq*(fi-1.0)
c**Define type as normal RCS freq
                  freqlist(fi,2) = 0
 20            continue

               stepf = 1
            else
               freqlist(1,1) = low_freq
c**Define type as normal RCS freq
               freqlist(1,2) = 0
               num_freqs = 1
               minf = 1
               maxf = 1
               stepf = 1
            end if
         else
            write(6,*) 'Currently you are calculating the RCS at ',
     1           num_freqs, ' between ', freqlist(1,1),' and ',
     2           freqlist(num_freqs,1),'. Enter the new step',
     3           ' (1 for all).'
            read(5,*) stepf
         end if

c******* Read in angles at which to calculate 2D RCS

 200     write(6,*) 'Bistatic RCS angles (in degrees):'
         write(6,'(''*Enter initial and final phi: '',$,$)')
         read(5,*) low_phi,high_phi
         if (include_ground_plane) then
            if (low_phi.lt.0.OR.high_phi.gt.180) then
               write(6,*)
               write(6,*) '*****************************************'
               write(6,*) 'Error. With ground plane, only upper half'
               write(6,*) 'plane results valid. Phi must be between'
               write(6,*) '0 and 180 degs. Re-enter angles.'
               write(6,*)
               goto 200
            end if
         end if

         if (abs(low_phi-high_phi).lt.tole) then
            dphi = high_phi-low_phi+1.0
         else
            write(6,'(''*Enter number of angles: '',$)')
            read(5,*) nang
            dphi = (high_phi-low_phi)/ real(nang-1.0)
         end if

      else
c******* Here goes all input info for mono estimation routines.

         low_phi = 1
         high_phi = 1
         dphi = 1

         write(6,99) int(inc_ang-45),int(inc_ang+45)

 99      format('Monostatic RCS angles range: ',I3, ' to ', I3,
     1        ' in one degree increments.')
         write(6,*) 'Enter lowest frequency of interest.'
         read(5,*) low_freq
         write(6,*) 'Enter highest frequency of interest.'
         read(5,*) high_freq

         if (abs(high_freq-low_freq).gt.tole) then
 30         write(6,*) 'Enter the number of frequencies.'
            read(5,*) num_freqs

            if ((MNANG*num_freqs).gt.MAX_FREQS) then
               write(6,*) 'Error. Number of freqs must be ',
     1              'less than or equal to ', MAX_FREQS/MNANG,
     1              ' or raise the MAX_FREQS parameter'
               write(6,*)
               goto 30
            end if
         else
            num_freqs = 1
         end if

         minf = 1
         maxf = num_freqs*MNANG
         print *,minf,maxf
         stepf = 1
         if (num_freqs.ne.1) then
            dfreq = (high_freq-low_freq)/(num_freqs-1.0)
         else
            dfreq = 0.0
         end if

         do 50 fi = 1,num_freqs
            do 40 mi = 1,MNANG
               freqlist(MNANG*(fi-1)+mi,1) = (low_freq+dfreq*
     1              (fi-1.0))*(1/cos((mi-46.0)*pi/180))
c               print *,freqlist(MNANG*(fi-1)+mi,1)
               freqlist(MNANG*(fi-1)+mi,2) = mi
 40         continue
 50      continue

      end if

      RETURN
      END
```

The following, **setup.f**, contains the subroutines used for setting up the computational domain as well as the staircase representation of the target.

```
c**************************************
c**** 2D FDTD H-polarization code ******
c****          SETUP.F           ******
c**************************************

c***********************************************************************
c SETUP_GEOMETRY reads in geometry file and setups up staircase model.
c***********************************************************************

      SUBROUTINE setup_geometry(geomfile)

      implicit none
      include 'common.f'

      character*72 geomfile

      integer xstair(1:MAX_STAIR_NODES),
     1     ystair(1:MAX_STAIR_NODES), index, round, spacing, current_x,
     2     current_y, xcomp, ycomp, xdir, ydir, dx, dy

      real max_x_node, max_y_node, min_x_node, min_y_node,
     1     slope, offset, dist_to_line

      parameter(spacing = 40)

      write(6,*) 'Setting up geometry...'

      errorcount = 0

c**** Read geometry file in.

      open(unit=10,file=geomfile,status='unknown',form='formatted')

      read(10,*) delta

      read(10,*) total_nodes
      if (total_nodes.gt.MAX_NODES) then
         errorcount = errorcount+1
         errors(errorcount) = NODE_ERROR
         call memory_check
      end if

      do 10 index=1,total_nodes
         read(10,*) xnodes(index), ynodes(index)
 10   continue
      close(unit=10)

C**** Scale, position, and round object

      max_x_node = xnodes(1)/delta
      max_y_node = ynodes(1)/delta
```

```
      min_x_node = xnodes(1)/delta
      min_y_node = ynodes(1)/delta

      do 20 index=1,total_nodes
         xnodes(index) = xnodes(index)/delta
         if (xnodes(index).gt.max_x_node) max_x_node=xnodes(index)
         if (xnodes(index).lt.min_x_node) min_x_node=xnodes(index)
         ynodes(index) = ynodes(index)/delta
         if (ynodes(index).gt.max_y_node) max_y_node=ynodes(index)
         if (ynodes(index).lt.min_y_node) min_y_node=ynodes(index)
 20   continue


c**** If including ground plane, object will be horizontally centered
c**** but not vertically centered. This allows the user to define
c**** exactly how high above the ground plane the object is.
c**** If no ground plane, object will be horizontally and vertically
c**** centered.

      if (include_ground_plane) then
         max_x = round(2.0*spacing + max_x_node - min_x_node)
         max_y = round(1.0*spacing + max_y_node) + 1
      else
         max_x = round(2.0*spacing + max_x_node - min_x_node)
         max_y = round(2.0*spacing + max_y_node - min_y_node)
      end if

      if (max_x.gt.MAX_X_CELLS) then
         errorcount = errorcount+1
         errors(errorcount) = MAX_X_ERROR
      end if

      if (max_y.gt.MAX_Y_CELLS) then
         errorcount = errorcount+1
         errors(errorcount) = MAX_Y_ERROR
      end if

      if (include_ground_plane) then
         do 31 index=1,total_nodes
            xnodes(index) = round(xnodes(index) - min_x_node) + spacing
            ynodes(index) = round(ynodes(index)) + 1
 31      continue
      else
         do 30 index=1,total_nodes
            xnodes(index) = round(xnodes(index) - min_x_node) + spacing
            ynodes(index) = round(ynodes(index) - min_y_node) + spacing
 30      continue
      end if

C**** define tot/scat field boundary points and setup fields
      x1=spacing-5
      if (include_ground_plane) then
         y1 = 1
      else
         y1=spacing-5
      end if
      x2=max_x-spacing+5
      y2=max_y-spacing+5

c**** Generate a staircase model by digitizing each line segment.

c**** Estimate total number of staircase nodes needed.
      dx = 0
      dy = 0
      do 50 index=1,total_nodes-1
         dx = dx + int(abs(xnodes(index)-xnodes(index+1)))
         dy = dy + int(abs(ynodes(index)-ynodes(index+1)))
 50   continue
c**** extra point needed for first point
      dy=dy+1

      if ((dx+dy).gt.MAX_STAIR_NODES) then
         stair_node_count = dx+dy
         errorcount = errorcount+1
         errors(errorcount) = MAX_STAIR_ERROR
      end if
      call define_tot_scat

      stair_node_count = 1
      xstair(stair_node_count) = int(xnodes(1))
      ystair(stair_node_count) = int(ynodes(1))

      do 40 index=1,total_nodes-1
         slope = (ynodes(index+1)-ynodes(index))/(xnodes(index+1)-
     1         xnodes(index))
         offset = ynodes(index)-slope*xnodes(index)

         current_x = int(xnodes(index))
         current_y = int(ynodes(index))

 100     stair_node_count = stair_node_count

         if (current_x.ne.int(xnodes(index+1)).OR.
     1       current_y.ne.int(ynodes(index+1))) then

            xcomp = int(xnodes(index+1))-current_x
            ycomp = int(ynodes(index+1))-current_y

            if (xcomp.ne.0) then
               xdir = int(abs(xcomp)/xcomp)
            else
               xdir = 0
            end if
            if (ycomp.ne.0) then
               ydir = int(abs(ycomp)/ycomp)
            else
               ydir = 0
            end if

            stair_node_count = stair_node_count + 1

            if (xdir.ne.0.AND.ydir.ne.0) then
               if (dist_to_line(-slope,1.0,offset,real(current_x+xdir),
     1              real(current_y)).lt.dist_to_line(-slope,1.0,offset,
     2              real(current_x),real(current_y+ydir))) then

                  xstair(stair_node_count) = current_x+xdir
                  ystair(stair_node_count) = current_y
                  current_x = current_x+xdir
               else
                  xstair(stair_node_count) = current_x
                  ystair(stair_node_count) = current_y+ydir
                  current_y = current_y+ydir
               end if
            else
               xstair(stair_node_count) = current_x+xdir
               ystair(stair_node_count) = current_y+ydir
               current_x = current_x+xdir
               current_y = current_y+ydir
            end if

            goto 100
         end if
 40   continue

      if ((dx+dy).ne.stair_node_count) then
         write(6,*) 'estimate = ', dx+dy
         write(6,*) 'actual = ', stair_node_count
      end if

c**** Now take digitized line & figure out which fields to set to zero

      call generate_te_stair_model(xstair,ystair)

      open(unit=10,file='stair.dat',status='unknown',
     1     form='formatted')

      do 1000 index=1,stair_node_count
         write(10,*) xstair(index), ystair(index)
 1000 continue
      close(unit=10)

c**** Calculate some important variables

c**** Time Step based on 2D stability requirements
      dt = 0.86*(delta/c/sqrt(2.0))

c**** Width of the Gaussian Pulse
 300  width = 2*sqrt(8.0)/(pi*(c/15/delta-modfreq))
      if (width.gt.(100*dt).OR.width.lt.(0.0)) then
         write(6,*) '***********************************************'
         write(6,*) 'Modulation frequency too high for grid resolution'
         write(6,*) 'Enter new modulation frequency (0=unmodulated)'
         read(5,*) modfreq

         if (abs(modfreq).lt.tole) then
            modulate = 0
         else
            modulate = 1
         end if
         goto 300
      end if

c**** The maximum sigma needed in the PML regions
      reflection = -50.0
      sigma_max = -reflection*3/eta/40./0.434294481903/(PML_DEPTH*delta)

      RETURN
      END

c**********************************************************************
c INIT_FIELDS initializes all fields in normal and PML regions to zero.
c**********************************************************************
```

```
      SUBROUTINE init_fields

      implicit none
      include 'common.f'

      integer i,j

      write(6,*) 'Initializing field data...'

      do 10 i=1,max_x
         do 20 j=1,max_y
            hx(i,j) = 0.0
            hy(i,j) = 0.0
            ez(i,j) = 0.0
20       continue
10    continue

      do 30 i=1,max_x
         do 40 j=1,PML_DEPTH
            ezxtt(i,j) = 0.0
            ezytt(i,j) = 0.0
            hxxtt(i,j) = 0.0
            hyytt(i,j) = 0.0
40       continue
30    continue

      do 50 i=1,max_x
         do 60 j=1,PML_DEPTH
            ezxbb(i,j) = 0.0
            ezybb(i,j) = 0.0
            hxxbb(i,j) = 0.0
            hyybb(i,j) = 0.0
60       continue
50    continue

      do 70 i=1,PML_DEPTH
         do 80 j=1,max_y
            ezxrr(i,j) = 0.0
            ezyrr(i,j) = 0.0
            hxxrr(i,j) = 0.0
            hyyrr(i,j) = 0.0
80       continue
70    continue

      do 90 i=1,PML_DEPTH
         do 100 j=1,max_y
            ezxll(i,j) = 0.0
            ezyll(i,j) = 0.0
            hxxll(i,j) = 0.0
            hyyll(i,j) = 0.0
100      continue
90    continue

      do 110 i=1,PML_DEPTH
         do 120 j=1,PML_DEPTH
            ezxtr(i,j) = 0.0
            ezytr(i,j) = 0.0
            hxxtr(i,j) = 0.0
            hyytr(i,j) = 0.0
120      continue
110   continue

      do 130 i=1,PML_DEPTH
         do 140 j=1,PML_DEPTH
            ezxtl(i,j) = 0.0
            ezytl(i,j) = 0.0
            hxxtl(i,j) = 0.0
            hyytl(i,j) = 0.0
140      continue
130   continue

      do 150 i=1,PML_DEPTH
         do 160 j=1,PML_DEPTH
            ezxbr(i,j) = 0.0
            ezybr(i,j) = 0.0
            hxxbr(i,j) = 0.0
            hyybr(i,j) = 0.0
160      continue
150   continue

      do 170 i=1,PML_DEPTH
         do 180 j=1,PML_DEPTH
            ezxbl(i,j) = 0.0
            ezybl(i,j) = 0.0
            hxxbl(i,j) = 0.0
            hyybl(i,j) = 0.0
180      continue
170   continue

      RETURN
      END
```

```
c*************************************************************************
c INIT_FREQS initializes all freq field comps to zero.
c*************************************************************************

      SUBROUTINE init_freqs

      implicit none
      include 'common.f'

      integer fi, k, ytemp

      write(6,*) 'Initializing frequency data...'

      do 10 fi=minf,maxf,stepf
         if (include_ground_plane) then
            ytemp = 4*y2+2*(x2-x1)-2+8*rcs_space+1
         else
            ytemp = 2*(y2-y1)+2*(x2-x1)+8*rcs_space+1
         end if
         do 20 k=1,ytemp
            ezfreq(fi,k) = 0.0
            hxfreq(fi,k) = 0.0
            hyfreq(fi,k) = 0.0
20       continue
10    continue

      RETURN
      END

c*************************************************************************
c READ_IN_FREQS reads in the previously saved frequency field comps.
c*************************************************************************

      SUBROUTINE read_in_freqs

      implicit none
      include 'common.f'

      integer fi, k, rcs_nodes, rbeg, rend
      real tempr, tempi

      write(6,*) 'Reading in frequency data...'

      open(unit=10,file='ezf.dat',status='unknown',form='formatted')
      open(unit=11,file='hxf.dat',status='unknown',form='formatted')
      open(unit=12,file='hyf.dat',status='unknown',form='formatted')

c**** Restore state

      open(unit=13,file='info.dat',status='unknown',form='formatted')

      read(13,*) max_x
      read(13,*) max_y
      read(13,*) delta
      read(13,*) dt
      read(13,*) tot_time_steps
      read(13,*) x1,y1,x2,y2
      read(13,*) inc_ang
      read(13,*) modulate
      read(13,*) modfreq
      read(13,*) delay
      read(13,*) width
      read(13,*) include_ground_plane
      read(13,*) num_freqs
c**** Assuming bistatic calculation
      mono_bi = 1
      do 30 fi = 1, num_freqs
         read(13,*) freqlist(fi,1)
         freqlist(fi,2) = 0
30    continue

      close(unit=13)

      cost = cos(inc_ang*180./pi)
      sint = sin(inc_ang*180./pi)

      if (abs(freqlist(num_freqs,1)-freqlist(1,1)).gt.tole) then
         if (num_freqs.gt.MAX_FREQS) then
            write(6,*) 'Error. Set MAX_FREQS parmeter higher.'
            write(6,*)
            stop
         end if

         minf = 1
         maxf = num_freqs
         stepf = 1
      else
         num_freqs = 1
         minf = 1
         maxf = 1
         stepf = 1
      end if
```

```
      if (include_ground_plane) then
         rbeg = y2+rcs_space+1
         rend = 3*y2+x2-x1-1+5*rcs_space
      else
         rbeg = 1
         rend = 2*(x2-x1)+2*(y2-y1)+8*rcs_space+1
      end if

      do 10 fi = minf,maxf
         do 20 k=rbeg,rend
            read(10,*) tempr,tempi
            ezfreq(fi,k) = tempr + (0.0,1.0)*tempi

            read(11,*) tempr,tempi
            hxfreq(fi,k) = tempr + (0.0,1.0)*tempi

            read(12,*) tempr,tempi
            hyfreq(fi,k) = tempr + (0.0,1.0)*tempi
 20      continue

 10   continue

      close(unit=10)
      close(unit=11)
      close(unit=12)

      RETURN
      END

c**********************************************************************
c DEFINE_TOT_SCAT defines each cell on the grid as either a total field
c   or a scattered field.
c**********************************************************************

      SUBROUTINE define_tot_scat

      implicit none
      include 'common.f'

      integer i,j

c     15 15 15 15 15 15 15 15 15 15
c     15 11 11 11 11 11 11 11 11 15
c       -------------------------
c     01|04 05 05 05 05 05 05 06|12
c     01|03 14 14 14 14 14 14 07|12
c     01|03 14 14 14 14 14 14 07|12
c     01|03 14 14 14 14 14 14 07|12
c     01|03 14 14 14 14 14 14 07|12
c     01|03 14 14 14 14 14 14 07|12
c     01|03 14 14 14 14 14 14 07|12
c     01|02 09 09 09 09 09 09 08|12
c       -------------------------
c     15 00 00 00 00 00 00 00 00 15
c     15 15 15 15 15 15 15 15 15 15

c     02,03,04,05,06,07,08,09,14 are total fields
c     00,01,11,12,15 are scattered fields.

c      write(6,*) 'define_tot_scat'

      if (include_ground_plane) then
         if ((2*(x2-x1)+4*y2-2+8*rcs_space+1).gt.MAX_RCS_NODES) then
            errorcount = errorcount + 1
            errors(errorcount) = MAX_RCS_ERROR
         end if
      else
         if ((2*(x2-x1)+2*(y2-y1)+8*rcs_space+1).gt.MAX_RCS_NODES) then
            errorcount = errorcount + 1
            errors(errorcount) = MAX_RCS_ERROR
         end if
      end if

      call memory_check

      do 10 i=1,max_x
         do 20 j=1,max_y
            tot_scat(i,j) = 15
 20      continue
 10   continue

      do 70 i=x1+1,x2-1
         do 80 j=y1+1,y2-1
            tot_scat(i,j) = 14
 80      continue
 70   continue

      tot_scat(x1,y1) = 2
      tot_scat(x1,y2) = 4
      tot_scat(x2,y2) = 6
      tot_scat(x2,y1) = 8

      do 30 j=y1+1,y2-1
         tot_scat(x1,j) = 3
         tot_scat(x2,j) = 7
 30   continue

      do 40 i=x1+1,x2-1
         tot_scat(i,y1) = 9
         tot_scat(i,y2) = 5
 40   continue

      do 50 j=y1,y2
         tot_scat(x1-1,j) = 1
         tot_scat(x2+1,j) = 12
 50   continue

      do 60 i=x1,x2
         if (.NOT.include_ground_plane) tot_scat(i,y1-1) = 0
         tot_scat(i,y2+1) = 11
 60   continue

      RETURN
      END

c**********************************************************************
c SETUP_INCIDENT_FIELD does some prelim calcs to prepare for using the
c   one-dimensional source look-up table method.
c**********************************************************************

      SUBROUTINE setup_incident_field

      implicit none
      include 'common.f'

      integer STEPS_TO_DELAY
      parameter(STEPS_TO_DELAY=150)

      if (abs(inc_ang-0.0).lt.tole) then
         sint = 0.0
         cost = 1.0
      elseif (abs(inc_ang-90.0).lt.tole) then
         sint = 1.0
         cost = 0.0
      elseif (abs(inc_ang-180.0).lt.tole) then
         sint = 0.0
         cost = -1.0
      elseif (abs(inc_ang-270.0).lt.tole) then
         sint = -1.0
         cost = 0.0
      else
         sint = sin((inc_ang/180.)*pi)
         cost = cos((inc_ang/180.)*pi)
      end if
c      print *,inc_ang,sint,cost

c***  calculate time delay

 10   if (inc_ang.ge.360) then
         inc_ang = inc_ang-360.0
         goto 10
      end if

 20   if (inc_ang.lt.0) then
         inc_ang = inc_ang+360.0
         goto 20
      end if

      if (inc_ang.ge.0.AND.inc_ang.lt.90) then
         delay = -(x2*delta*cost+y2*delta*sint)/c - STEPS_TO_DELAY*dt
     1         + width/2.0
      elseif (inc_ang.ge.90.AND.inc_ang.lt.180) then
         delay = -(x1*delta*cost+y2*delta*sint)/c - STEPS_TO_DELAY*dt
     1         - width/2.0
      elseif (inc_ang.ge.180.AND.inc_ang.lt.270) then
         delay = -(x1*delta*cost+y1*delta*sint)/c - STEPS_TO_DELAY*dt
     1         - width/2.0
      else
         delay = -(x2*delta*cost+y1*delta*sint)/c - STEPS_TO_DELAY*dt
     1         + width/2.0
      end if

c***  calculate numerical phase velocity at theta=0

c***  calculate numerical phase velocity at theta=inc_ang

      RETURN
      END


c**********************************************************************
c SUBROUTINE SETUP_MOVIE
c**********************************************************************
```

```
      SUBROUTINE setup_movie(mhname,mfname)

      implicit none
      include 'common.f'
      character*72 mhname, mfname

      open(unit=4,file=mfname,status='unknown',form='formatted')
      open(unit=7,file=mhname,status='unknown',form='formatted')

      write(4,*) min(max_x,100)
      write (4,100) 'new.image'
      write(7,200) min(100,max_x), min(100,max_y), 64, tot_time_steps,
     1     'new.image.Z'

      dummy = min(1000,max_x)
      write (7,*) 1
      write (7,100) 'a '
      write (7,100) 'b '
      close(unit=7)

 100  format(a)
 200  format(i4,x,i4,x,i2,x,i5,x,a)

      RETURN
      END


c**********************************************************************
c MEMORY_CHECK checks if enough memory has been allocated and reports
c   all errors stored in error buffer
c**********************************************************************

      SUBROUTINE memory_check

      implicit none
      include 'common.f'

      integer i, id

      if (errorcount.gt.0) then
         write(6,*) '*********************************************'
         write(6,*) 'Insufficient memory to begin simulation. The'
         write(6,*) 'following parameter(s) in the common.f file'
         write(6,*) 'need to be adjusted:'

         do 10 i=1,errorcount
            id = errors(i)
            write(6,*)
            if (id.eq.NODE_ERROR) then
               write(6,*) 'Set MAX_NODES to at least',total_nodes
            else if (id.eq.MAX_X_ERROR) then
               write(6,*) 'Set MAX_X_CELLS to at least',max_x
            else if (id.eq.MAX_Y_ERROR) then
               write(6,*) 'Set MAX_Y_CELLS to at least',max_y
            else if (id.eq.MAX_STAIR_ERROR) then
               write(6,*) 'Set MAX_STAIR_NODES to at least',
     1              stair_node_count
            else if (id.eq.MAX_RCS_ERROR) then
               if (include_ground_plane) then
                  write(6,*) 'Set MAX_RCS_NODES to at least',
     1                 2*(x2-x1)+4*y2-2+8*rcs_space+1
               else
                  write(6,*) 'Set MAX_RCS_NODES to at least',
     1                 2*(x2-x1)+2*(y2-y1)+8*rcs_space+1
               end if
            end if

 10      continue
         write(6,*) '*********************************************'
         stop

      end if

      RETURN
      END


c**********************************************************************
c INTEGER FUNCTION ROUND returns the integer nearest in absolute
c   distance to the real arguement
c**********************************************************************

      INTEGER FUNCTION round(x)

      real x, fpart
      integer ipart

      ipart = int(x)
      fpart = x-aint(x)

      if (fpart.gt.0.5) then
         round = ipart+1
      else if (fpart.gt.0.0) then
         round = ipart
      else if (fpart.ge.-0.5) then
         round = ipart
      else
         round = ipart-1
      end if

      RETURN
      END


c**********************************************************************
c REAL FUNCTION DIST_TO_LINE returns the perpendicular distance from a
c point in space (x,y) to a line that is of the form Ax+By=C
c**********************************************************************

      REAL FUNCTION dist_to_line(A,B,C,x,y)

      implicit none
      real A,B,C,x,y

      dist_to_line = abs((A*x+B*y-C)/sqrt(A**2.0 + B**2.0))

      RETURN
      END


c**********************************************************************
c LOGICAL FUNCTION INSIDE_PEC(xlist,ylist,x,y) returns TRUE if the point
c   (x,y) is inside the polygon described by the points in x,y lists
c**********************************************************************

      LOGICAL FUNCTION inside_pec(xlist,ylist,count,px,py)

      implicit none
      include 'common.f'

      integer count
      real xlist(1:count),ylist(1:count),xp1,yp1,px,py,slope

      integer maxx, minx, maxy, miny, index, above, below, connect

c**** if polygon not closed, close it.
      if (xlist(1).ne.xlist(count).OR.ylist(1).ne.ylist(count)) then
         connect = 0
      else
         connect = -1
      end if

      maxx = xlist(1)
      minx = xlist(1)
      maxy = ylist(1)
      miny = ylist(1)

      do 10 index=1,count
         if (xlist(index).gt.maxx) maxx=xlist(index)
         if (ylist(index).gt.maxy) maxy=ylist(index)
         if (xlist(index).lt.minx) minx=xlist(index)
         if (ylist(index).lt.miny) miny=ylist(index)
 10   continue

      if (px.gt.maxx.OR.px.lt.minx.OR.py.gt.maxy.OR.py.lt.miny) then
         inside_pec = .FALSE.
      else

c**** count the intersections

         above = 0
         below = 0

         do 20 index = 1,count+connect
            if (index.eq.count) then
               xp1 = xlist(1)
               yp1 = ylist(1)
            else
               xp1 = xlist(index+1)
               yp1 = ylist(index+1)
            end if

            if ( (abs(px-xlist(index)).lt.tole).AND.(abs(px-
     1           xlist(index+1)).lt.tole)) then
               if (((py.le.ylist(index)).AND.(py.ge.ylist(index+1))).
     1              OR.((py.ge.ylist(index)).AND.(py.le.ylist(index
     2              +1))) ) then
                  inside_pec = .TRUE.
                  RETURN
               end if
            end if

            if ( ((abs(px-xlist(index)).lt.tole).AND.(abs(py-
```

```
     1          ylist(index)).lt.tole)).OR.((abs(px-xlist(index+1)).
     2          lt.tole).AND.(abs(py-ylist(index+1)).lt.tole))) then
                inside_pec = .TRUE.
                RETURN
             end if

             if ((px.le.xlist(index).AND.px.ge.xp1).OR.
     1           (px.ge.xlist(index).AND.px.le.xp1)) then
                slope = (ylist(index)-yp1)/(xlist(index)-xp1)
                if (abs((slope*(px-xlist(index))+ylist(index))-py).
     1              lt.tole) then
                   inside_pec = .TRUE.
                   RETURN
                else if ((slope*(px-xlist(index))+ylist(index)).gt.
     1                 py) then
                   above = above+1
                else
                   below = below+1
                end if
             end if
 20       continue

          if (mod(above,2).eq.1.AND.mod(below,2).eq.1) then
             inside_pec = .TRUE.
          else
             inside_pec = .FALSE.
          end if
       end if

       RETURN
       END

c**********************************************************************
c GENERATE_TE_STAIR_MODEL generates the TE staircase model by compiling
c   a list of all the Ex/Ey fields that need to be set to zero.
c**********************************************************************

       SUBROUTINE generate_te_stair_model(xstair,ystair)

       implicit none
       include 'common.f'

       integer xt1,xt2,yt1,yt2

       integer xstair(1:MAX_STAIR_NODES), ystair(1:MAX_STAIR_NODES),
      1     index

       do 10 index = 1,stair_node_count-1
          xt1 = int(xstair(index))
          yt1 = int(ystair(index))
          xt2 = int(xstair(index+1))
          yt2 = int(ystair(index+1))

          if ((yt2-yt1).gt.0.AND.(xt2-xt1).eq.0) then
             stair_zero(index,1) = int(xstair(index))
             stair_zero(index,2) = int(ystair(index))
             stair_zero(index,3) = eyf
          else if ((yt2-yt1).lt.0.AND.(xt2-xt1).eq.0) then
             stair_zero(index,1) = int(xstair(index))
             stair_zero(index,2) = int(ystair(index))-1
             stair_zero(index,3) = eyf
          else if ((xt2-xt1).gt.0.AND.(yt2-yt1).eq.0) then
             stair_zero(index,1) = int(xstair(index))
             stair_zero(index,2) = int(ystair(index))
             stair_zero(index,3) = exf
          else if ((xt2-xt1).lt.0.AND.(yt2-yt1).eq.0) then
             stair_zero(index,1) = int(xstair(index))-1
             stair_zero(index,2) = int(ystair(index))
             stair_zero(index,3) = exf
          else
             print *,'error in staircasing point ',index,' to ',
     1           'point ', index+1
          end if

 10    continue

       open(unit=14,file='eys.dat',status='unknown',form='formatted')

       do 20 index = 1,stair_node_count-1
          write(14,*) stair_zero(index,1), stair_zero(index,2),
     1        stair_zero(index,3)
 20    continue

       close(unit=14)

       RETURN
       END
```

The following, **calc.f**, contains the core FD-TD subroutines used in updating the fields. It includes the implementation of Berenger's PML absorbing boundary condition.

```
c**************************************
c**** 2D FDTD H-polarization code ******
c****          CALC.F            ******
c**************************************

c***********************************************c
c********* IMPORTANT! PLEASE READ BELOW *********c
c***********************************************
c Note that in the code below, H fields are      c
c labeled as E fields, and E fields are labeled  c
c as H fields.  Signs and other parameters have  c
c been adjusted to correctly represent Maxwell's c
c equations. The reason for this is that the     c
c 2D FDTD E-polarization case has already been   c
c done, and we use duality rather than go        c
c through and change every variable name.        c
c***********************************************c
c***********************************************c

c**********************************************************************
c FDTD_LOOP controls the flow of the wave propagations calculations. It
c   calls neccessary subroutines including E_FIELDS, H_FIELDS, E_PML,
c   and H_PML.
c**********************************************************************

       SUBROUTINE fdtd_loop(store_freq)

       implicit none
       include 'common.f'

       character a(1:MAX_X_CELLS)
       logical store_freq
       integer time_step, movie_frame, freq_frame

       open(unit=18,file='effscat.dat',status='unknown',form='formatted')

       movie_frame = 0
       freq_frame = 1
       max_field = 0.0
       call setup_incident_field
       call write_out_all_parms(store_freq)
       call memory_check
       write(6,*) 'Running simulation...'
       do 10 time_step = 1,tot_time_steps
c          write(6,*) time_step
          call h_fields(time_step)
          call h_pml
          call boundary_conditions
          call e_fields(time_step)
          call e_pml

          write(18,*) ez(48,24),ez(48,28)
          if (store_movie) then
             if (movie_frame.eq.movie_step) then
                call write_out_movie_frame(a,time_step)
                movie_frame = 0
             else
                movie_frame = movie_frame + 1
             end if
          end if
          if (freq_frame.eq.1) then
             if (store_freq) call update_freqs(time_step)
             freq_frame = 1
          else
             freq_frame = freq_frame + 1
          end if
 10    continue

       close(unit=18)
       close(unit=4)
       RETURN
       END

c**********************************************************************
c E_FIELDS updates the E field comps in the normal region.
c**********************************************************************

       SUBROUTINE e_fields(time_step)

       implicit none
       include 'common.f'

       integer time_step, i, j, ct
       real t1, t2, hyinc, hxinc, hyref, hxref

       do 10 i=1,max_x
          do 20 j=1,max_y
             ct = tot_scat(i,j)
```

```
            if (i.eq.max_x) then
               t1=hyyrr(1,j)
            else
               t1=hy(i+1,j)
            end if

            if (j.eq.max_y) then
               t2=hxxtt(i,1)
            else
               t2=hx(i,j+1)
            end if

            if (ct.eq.6.OR.ct.eq.7.OR.ct.eq.8) then
               t1=t1+hyinc(i+1,j,time_step)
               if (include_ground_plane) t1=t1+hyref(i+1,j,time_step)
            end if
            if (ct.eq.1) then
               t1=t1-hyinc(i+1,j,time_step)
               if (include_ground_plane) t2=t2+hyref(i+1,j,time_step)
            end if

            if (ct.eq.4.OR.ct.eq.5.OR.ct.eq.6) then
               t2=t2+hxinc(i,j+1,time_step)
               if (include_ground_plane) t2=t2+hxref(i,j+1,time_step)
            end if
            if (ct.eq.0) t2=t2-hxinc(i,j+1,time_step)

            ez(i,j)=ez(i,j)+(c*dt/delta)*(-(t1-hy(i,j))+(t2-hx(i,j)))
 20      continue
 10   continue

      RETURN
      END

c**************************************************************************
c H_FIELDS updates the H field comps in the normal region.
c**************************************************************************

      SUBROUTINE h_fields(time_step)

      implicit none
      include 'common.f'

      integer time_step, i, j, ct
      real t1, ezinc, ezref

c**** Hy fields ********************************************************

      do 10 i=1,max_x
         do 20 j=1,max_y
            ct=tot_scat(i,j)

            if (i.eq.1) then
               t1=ezxll(PML_DEPTH,j)+ezyll(PML_DEPTH,j)
            else
               t1=ez(i-1,j)
            end if

            if (ct.eq.2.OR.ct.eq.3.OR.ct.eq.4) then
               t1=t1+ezinc(i-1,j,time_step)
               if (include_ground_plane) t1=t1+ezref(i-1,j,time_step)
            end if
            if (ct.eq.12) then
               t1=t1-ezinc(i-1,j,time_step)
               if (include_ground_plane) t1=t1-ezref(i-1,j,time_step)
            end if

            hy(i,j)=hy(i,j) - (c*dt/delta)*(ez(i,j)-t1)
 20      continue
 10   continue

c**** Hx fields ********************************************************

      do 30 i=1,max_x
         do 40 j=1,max_y
            ct=tot_scat(i,j)

            if (j.eq.1) then
               t1=ezxbb(i,PML_DEPTH)+ezybb(i,PML_DEPTH)
            else
               t1=ez(i,j-1)
            end if

            if (.NOT.include_ground_plane) then
               if (ct.eq.2.OR.ct.eq.9.OR.ct.eq.8) t1=t1+
     1            ezinc(i,j-1,time_step)
            end if
            if (ct.eq.11) then
               t1=t1-ezinc(i,j-1,time_step)
               if (include_ground_plane) t1=t1-ezref(i,j-1,time_step)
            end if

            hx(i,j)=hx(i,j) + (c*dt/delta)*(ez(i,j)-t1)
 40      continue
 30   continue

      RETURN
      END

c**************************************************************************
c E_PML updates the E field comps in the PML regions.
c**************************************************************************

      SUBROUTINE e_pml

      implicit none
      include 'common.f'

      integer i,j
      real c1, c2, c3, c4, sigma_x, sigma_y
      real t1, t2

c**** Bottom Left Region (BL), sigma_x & sigma_y nonzero

      do 10 i=1,PML_DEPTH
         sigma_x = sigma_max*((PML_DEPTH+1.0-i)/PML_DEPTH)**2.0
         c1 = exp(-sigma_x*dt/eps)
         c2 = (1-c1)/(sigma_x*delta*eta*eta)

         do 20 j=1,PML_DEPTH
            sigma_y = sigma_max*((PML_DEPTH+1.0-j)/PML_DEPTH)**2.0
            c3 = exp(-sigma_y*dt/eps)
            c4 = (1-c3)/(sigma_y*delta*eta*eta)

            if (i.eq.PML_DEPTH) then
               t1=hyybb(1,j)
            else
               t1=hyybl(i+1,j)
            end if

            if (j.eq.PML_DEPTH) then
               t2=hxxll(i,1)
            else
               t2=hxxbl(i,j+1)
            end if

            ezxbl(i,j)=c1*ezxbl(i,j)-c2*((eta)*(t1-hyybl(i,j)))
            ezybl(i,j)=c3*ezybl(i,j)+c4*((eta)*(t2-hxxbl(i,j)))
 20      continue
 10   continue

c**** Top Left Region (TL), sigma_x & sigma_y nonzero

      do 30 i=1,PML_DEPTH
         sigma_x = sigma_max*((PML_DEPTH+1.0-i)/PML_DEPTH)**2.0
         c1 = exp(-sigma_x*dt/eps)
         c2 = (1-c1)/(sigma_x*delta*eta*eta)

         do 40 j=1,PML_DEPTH
            sigma_y = sigma_max*((j+0.0)/PML_DEPTH)**2.0
            c3 = exp(-sigma_y*dt/eps)
            c4 = (1-c3)/(sigma_y*delta*eta*eta)

            if (i.eq.PML_DEPTH) then
               t1=hyytt(1,j)
            else
               t1=hyytl(i+1,j)
            end if

            if (j.eq.PML_DEPTH) then
               t2=0.0
            else
               t2=hxxtl(i,j+1)
            end if

            ezxtl(i,j)=c1*ezxtl(i,j)-c2*((eta)*(t1-hyytl(i,j)))
            ezytl(i,j)=c3*ezytl(i,j)+c4*((eta)*(t2-hxxtl(i,j)))
 40      continue
 30   continue

c**** Top Right Region (TR), sigma_x & sigma_y nonzero

      do 50 i=1,PML_DEPTH
         sigma_x = sigma_max*((i+0.0)/PML_DEPTH)**2.0
         c1 = exp(-sigma_x*dt/eps)
         c2 = (1-c1)/(sigma_x*delta*eta*eta)

         do 60 j=1,PML_DEPTH
            sigma_y = sigma_max*((j+0.0)/PML_DEPTH)**2.0
            c3 = exp(-sigma_y*dt/eps)
            c4 = (1-c3)/(sigma_y*delta*eta*eta)

            if (i.eq.PML_DEPTH) then
```

```
            t1=0.0
         else
            t1=hyytr(i+1,j)
         end if

         if (j.eq.PML_DEPTH) then
            t2=0.0
         else
            t2=hxxtr(i,j+1)
         end if

         ezxtr(i,j)=c1*ezxtr(i,j)-c2*((eta)*(t1-hyytr(i,j)))
         ezytr(i,j)=c3*ezytr(i,j)+c4*((eta)*(t2-hxxtr(i,j)))
 60    continue
 50  continue

c**** Bottom Right Region (BR), sigma_x & sigma_y nonzero

      do 70 i=1,PML_DEPTH
         sigma_x = sigma_max*((i+0.0)/PML_DEPTH)**2.0
         c1 = exp(-sigma_x*dt/eps)
         c2 = (1-c1)/(sigma_x*delta*eta*eta)

         do 80 j=1,PML_DEPTH
            sigma_y = sigma_max*((PML_DEPTH+1.0-j)/PML_DEPTH)**2.0
            c3 = exp(-sigma_y*dt/eps)
            c4 = (1-c3)/(sigma_y*delta*eta*eta)

            if (i.eq.PML_DEPTH) then
               t1=0.0
            else
               t1=hyybr(i+1,j)
            end if

            if (j.eq.PML_DEPTH) then
               t2=hxxrr(i,1)
            else
               t2=hxxbr(i,j+1)
            end if

            ezxbr(i,j)=c1*ezxbr(i,j)-c2*((eta)*(t1-hyybr(i,j)))
            ezybr(i,j)=c3*ezybr(i,j)+c4*((eta)*(t2-hxxbr(i,j)))
 80      continue
 70   continue

c**** Bottom Center Region (BB), sigma_y nonzero

      do 90 i=1,max_x
         do 100 j=1,PML_DEPTH
            sigma_y = sigma_max*((PML_DEPTH+1.0-j)/PML_DEPTH)**2.0
            c3 = exp(-sigma_y*dt/eps)
            c4 = (1-c3)/(sigma_y*delta*eta*eta)

            if (i.eq.max_x) then
               t1=hyybr(1,j)
            else
               t1=hyybb(i+1,j)
            end if

            if (j.eq.PML_DEPTH) then
               t2=hx(i,1)
            else
               t2=hxxbb(i,j+1)
            end if

            ezxbb(i,j)=ezxbb(i,j)-((c*dt)/delta)*(t1-hyybb(i,j))
            ezybb(i,j)=c3*ezybb(i,j)+c4*((eta)*(t2-hxxbb(i,j)))
 100        continue
 90      continue

c**** Top Center Region, sigma_y nonzero

      do 110 i=1,max_x
         do 120 j=1,PML_DEPTH
            sigma_y = sigma_max*((j+0.0)/PML_DEPTH)**2.0
            c3 = exp(-sigma_y*dt/eps)
            c4 = (1-c3)/(sigma_y*delta*eta*eta)

            if (i.eq.max_x) then
               t1=hyytr(1,j)
            else
               t1=hyytt(i+1,j)
            end if

            if (j.eq.PML_DEPTH) then
               t2=0.0
            else
               t2=hxxtt(i,j+1)
            end if

            ezxtt(i,j)=ezxtt(i,j)-((c*dt)/delta)*(t1-hyytt(i,j))
            ezytt(i,j)=c3*ezytt(i,j)+c4*((eta)*(t2-hxxtt(i,j)))
 120        continue
 110     continue

c**** Right Center Region (RR), sigma_x nonzero

      do 130 i=1,PML_DEPTH
         sigma_x = sigma_max*((i+0.0)/PML_DEPTH)**2.0
         c1 = exp(-sigma_x*dt/eps)
         c2 = (1-c1)/(sigma_x*delta*eta*eta)

         do 140 j=1,max_y

            if (i.eq.PML_DEPTH) then
               t1=0.0
            else
               t1=hyyrr(i+1,j)
            end if

            if (j.eq.max_y) then
               t2=hxxtr(i,1)
            else
               t2=hxxrr(i,j+1)
            end if

            ezxrr(i,j)=c1*ezxrr(i,j)-c2*((eta)*(t1-hyyrr(i,j)))
            ezyrr(i,j)=ezyrr(i,j)+((c*dt)/delta)*(t2-hxxrr(i,j))
 140     continue
 130  continue

c**** Left Center Region (LL), sigma_x nonzero

      do 150 i=1,PML_DEPTH
         sigma_x = sigma_max*((PML_DEPTH+1.0-i)/PML_DEPTH)**2.0
         c1 = exp(-sigma_x*dt/eps)
         c2 = (1-c1)/(sigma_x*delta*eta*eta)

         do 160 j=1,max_y

            if (i.eq.PML_DEPTH) then
               t1=hy(1,j)
            else
               t1=hyyll(i+1,j)
            end if

            if (j.eq.max_y) then
               t2=hxxtl(i,1)
            else
               t2=hxxll(i,j+1)
            end if

            ezxll(i,j)=c1*ezxll(i,j)-c2*((eta)*(t1-hyyll(i,j)))
            ezyll(i,j)=ezyll(i,j)+((c*dt)/delta)*(t2-hxxll(i,j))
 160      continue
 150   continue

      RETURN
      END

c**********************************************************************
c H_PML updates the H field comps in the PML regions.
c**********************************************************************

      SUBROUTINE h_pml

      implicit none
      include 'common.f'

      integer i,j
      real c1, c2, c3, c4, sigma_x, sigma_y
      real t1, t2, t3

c**** Bottom Left Region (BL), sigma_x & sigma_y nonzero

      do 10 i=1,PML_DEPTH
         sigma_x = sigma_max*((PML_DEPTH+1.0-i)/PML_DEPTH)**2.0
         c1 = exp(-sigma_x*dt/eps)
         c2 = (1-c1)/(sigma_x*delta)

         do 20 j=1,PML_DEPTH
            sigma_y = sigma_max*((PML_DEPTH+1.0-j)/PML_DEPTH)**2.0
            c3 = exp(-sigma_y*dt/eps)
            c4 = (1-c3)/(sigma_y*delta)

            t1=ezxbl(i,j)+ezybl(i,j)

            if (i.eq.1) then
               t2=0
            else
               t2=ezxbl(i-1,j)+ezybl(i-1,j)
            end if

            if (j.eq.1) then
```

```
            t3=0
         else
            t3=ezxbl(i,j-1)+ezybl(i,j-1)
         end if

         hyybl(i,j)=c1*hyybl(i,j)-c2*(1/eta)*(t1-t2)
         hxxbl(i,j)=c3*hxxbl(i,j)+c4*(1/eta)*(t1-t3)
 20   continue
 10   continue

c**** Top Left Region (TL), sigma_x & sigma_y nonzero

      do 30 i=1,PML_DEPTH
         sigma_x = sigma_max*((PML_DEPTH+1.0-i)/PML_DEPTH)**2.0
         c1 = exp(-sigma_x*dt/eps)
         c2 = (1-c1)/(sigma_x*delta)

         do 40 j=1,PML_DEPTH
            sigma_y = sigma_max*((j+0.0)/PML_DEPTH)**2.0
            c3 = exp(-sigma_y*dt/eps)
            c4 = (1-c3)/(sigma_y*delta)

            t1=ezxtl(i,j)+ezytl(i,j)

            if (i.eq.1) then
               t2=0
            else
               t2=ezxtl(i-1,j)+ezytl(i-1,j)
            end if

            if (j.eq.1) then
               t3=ezxll(i,max_y)+ezyll(i,max_y)
            else
               t3=ezxtl(i,j-1)+ezytl(i,j-1)
            end if

            hyytl(i,j)=c1*hyytl(i,j)-c2*(1/eta)*(t1-t2)
            hxxtl(i,j)=c3*hxxtl(i,j)+c4*(1/eta)*(t1-t3)
 40      continue
 30   continue

c**** Top Right Region (TR), sigma_x & sigma_y nonzero

      do 50 i=1,PML_DEPTH
         sigma_x = sigma_max*((i+0.0)/PML_DEPTH)**2.0
         c1 = exp(-sigma_x*dt/eps)
         c2 = (1-c1)/(sigma_x*delta)

         do 60 j=1,PML_DEPTH
            sigma_y = sigma_max*((j+0.0)/PML_DEPTH)**2.0
            c3 = exp(-sigma_y*dt/eps)
            c4 = (1-c3)/(sigma_y*delta)
            t1=ezxtr(i,j)+ezytr(i,j)

            if (i.eq.1) then
               t2=ezxtt(max_x,j)+ezytt(max_x,j)
            else
               t2=ezxtr(i-1,j)+ezytr(i-1,j)
            end if

            if (j.eq.1) then
               t3=ezxrr(i,max_y)+ezyrr(i,max_y)
            else
               t3=ezxtr(i,j-1)+ezytr(i,j-1)
            end if

            hyytr(i,j)=c1*hyytr(i,j)-c2*(1/eta)*(t1-t2)
            hxxtr(i,j)=c3*hxxtr(i,j)+c4*(1/eta)*(t1-t3)
 60      continue
 50   continue

c**** Bottom Right Region (BR), sigma_x & sigma_y nonzero

      do 70 i=1,PML_DEPTH
         sigma_x = sigma_max*((i+0.0)/PML_DEPTH)**2.0
         c1 = exp(-sigma_x*dt/eps)
         c2 = (1-c1)/(sigma_x*delta)

         do 80 j=1,PML_DEPTH
            sigma_y = sigma_max*((PML_DEPTH+1.0-j)/PML_DEPTH)**2.0
            c3 = exp(-sigma_y*dt/eps)
            c4 = (1-c3)/(sigma_y*delta)
            t1=ezxbr(i,j)+ezybr(i,j)

            if (i.eq.1) then
               t2=ezxbb(max_x,j)+ezybb(max_x,j)
            else
               t2=ezxbr(i-1,j)+ezybr(i-1,j)
            end if

            if (j.eq.1) then
               t3=0
            else
               t3=ezxbr(i,j-1)+ezybr(i,j-1)
            end if

            hyybr(i,j)=c1*hyybr(i,j)-c2*(1/eta)*(t1-t2)
            hxxbr(i,j)=c3*hxxbr(i,j)+c4*(1/eta)*(t1-t3)
 80      continue
 70   continue

c**** Bottom Center Region (BB), sigma_y nonzero

      do 90 i=1,max_x
         do 100 j=1,PML_DEPTH
            sigma_y = sigma_max*((PML_DEPTH+1.0-j)/PML_DEPTH)**2.0
            c3 = exp(-sigma_y*dt/eps)
            c4 = (1-c3)/(sigma_y*delta)

            t1=ezxbb(i,j)+ezybb(i,j)

            if (i.eq.1) then
               t2=ezxbl(PML_DEPTH,j)+ezybl(PML_DEPTH,j)
            else
               t2=ezxbb(i-1,j)+ezybb(i-1,j)
            end if

            if (j.eq.1) then
               t3=0
            else
               t3=ezxbb(i,j-1)+ezybb(i,j-1)
            end if

            hyybb(i,j)=hyybb(i,j)-(c*dt/delta)*(t1-t2)
            hxxbb(i,j)=c3*hxxbb(i,j)+c4*(1/eta)*(t1-t3)
 100        continue
 90      continue

c**** Top Center Region (TT), sigma_y nonzero

      do 110 i=1,max_x
         do 120 j=1,PML_DEPTH
            sigma_y = sigma_max*((j+0.0)/PML_DEPTH)**2.0
            c3 = exp(-sigma_y*dt/eps)
            c4 = (1-c3)/(sigma_y*delta)

            t1=ezxtt(i,j)+ezytt(i,j)

            if (i.eq.1) then
               t2=ezxtl(PML_DEPTH,j)+ezytl(PML_DEPTH,j)
            else
               t2=ezxtt(i-1,j)+ezytt(i-1,j)
            end if

            if (j.eq.1) then
               t3=ez(i,max_y)
            else
               t3=ezxtt(i,j-1)+ezytt(i,j-1)
            end if

            hyytt(i,j)=hyytt(i,j)-(c*dt/delta)*(t1-t2)
            hxxtt(i,j)=c3*hxxtt(i,j)+c4*(1/eta)*(t1-t3)
 120        continue
 110     continue

c**** Right Center Region (RR), sigma_x nonzero

      do 130 i=1,PML_DEPTH
         sigma_x = sigma_max*((i+0.0)/PML_DEPTH)**2.0
         c1 = exp(-sigma_x*dt/eps)
         c2 = (1-c1)/(sigma_x*delta)

         do 140 j=1,max_y

            t1=ezxrr(i,j)+ezyrr(i,j)

            if (i.eq.1) then
               t2=ez(max_x,j)
            else
               t2=ezxrr(i-1,j)+ezyrr(i-1,j)
            end if

            if (j.eq.1) then
               t3=ezxbr(i,PML_DEPTH)+ezybr(i,PML_DEPTH)
            else
               t3=ezxrr(i,j-1)+ezyrr(i,j-1)
            end if

            hyyrr(i,j)=c1*hyyrr(i,j)-c2*(1/eta)*(t1-t2)
            hxxrr(i,j)=hxxrr(i,j)+(c*dt/delta)*(t1-t3)
 140     continue
 130  continue

c**** Left Center Region (LL), sigma_x nonzero
```

```
      do 150 i=1,PML_DEPTH
         sigma_x = sigma_max*((PML_DEPTH+1.0-i)/PML_DEPTH)**2.0
         c1 = exp(-sigma_x*dt/eps)
         c2 = (1-c1)/(sigma_x*delta)

         do 160 j=1,max_y

            t1=ezxll(i,j)+ezyll(i,j)

            if (i.eq.1) then
               t2=0
            else
               t2=ezxll(i-1,j)+ezyll(i-1,j)
            end if

            if (j.eq.1) then
               t3=ezxbl(i,PML_DEPTH)+ezybl(i,PML_DEPTH)
            else
               t3=ezxll(i,j-1)+ezyll(i,j-1)
            end if

            hyyll(i,j)=c1*hyyll(i,j)-c2*(1/eta)*(t1-t2)
            hxxll(i,j)=hxxll(i,j)+(c*dt/delta)*(t1-t3)
 160      continue
 150   continue

      RETURN
      END


c**********************************************************************
c BOUNDARY_CONDITIONS sets all the appropriate fields in the staircase
c   model to zero.
c**********************************************************************

      SUBROUTINE boundary_conditions

      implicit none
      include 'common.f'

      integer index

      do 10 index = 1,stair_node_count-1
         if (stair_zero(index,3).eq.exf) then
            hx(stair_zero(index,1),stair_zero(index,2)) = 0.0
         else if (stair_zero(index,3).eq.eyf) then
            hy(stair_zero(index,1),stair_zero(index,2)) = 0.0
         end if
 10   continue

c**** Set conditions for ground plane.
      if (include_ground_plane) then
         do 20 index = 1,max_x
            hx(index,1) = 0.0
 20      continue
      end if

      RETURN
      END

c**********************************************************************
c GENERATE_INCIDENT_FIELD_LOOKUP_TABLE calculates the propagation of
c   the one-dimensional wave along the k-vector at the current time
c   step that is used to calculate incident at all locations.
c**********************************************************************

      SUBROUTINE generate_incident_field_lookup_table(time_step)

      implicit none
      include 'common.f'

      integer time_step

      RETURN
      END


c**********************************************************************
c REAL FUNCTION EZINC returns the value of the Ez incident field at the
c   given location and time.
c**********************************************************************

      REAL FUNCTION ezinc(i,j,time_step)

      implicit none
      include 'common.f'

      integer i,j,time_step
      real x, y, t, amplitude

      x=(i+0.5)*delta
      y=(j+0.5)*delta-delta
      t=(time_step-0.0)*dt

      ezinc = amplitude(delay+t+(x*cost+y*sint)/c)

      RETURN
      END


c**********************************************************************
c REAL FUNCTION HXINC returns the value eta times the Hx incident field
c at the given location and time.
c**********************************************************************

      REAL FUNCTION hxinc(i,j,time_step)

      implicit none
      include 'common.f'

      integer i,j,time_step
      real x, y, t, amplitude

      x=(i+0.5)*delta
      y=(j+0.0)*delta-delta
      t=(time_step+0.5)*dt

      hxinc = sint*amplitude(delay+t+(x*cost+y*sint)/c)

      RETURN
      END

c**********************************************************************
c REAL FUNCTION HYINC returns the value eta times the Hy incident field
c   at the given location and time.
c**********************************************************************

      REAL FUNCTION hyinc(i,j,time_step)

      implicit none
      include 'common.f'

      integer i,j,time_step

      real x, y, t, amplitude

      x=(i+0.0)*delta
      y=(j+0.5)*delta-delta
      t=(time_step+0.5)*dt

      hyinc = -cost*amplitude(delay+t+(x*cost+y*sint)/c)

      RETURN
      END

c**********************************************************************
c REAL FUNCTION EZREF returns the value of the Ez reflected field at the
c   given location and time.
c**********************************************************************

      REAL FUNCTION ezref(i,j,time_step)

      implicit none
      include 'common.f'

      integer i,j,time_step
      real x, y, t, amplitude

      x=(i+0.5)*delta
      y=(j+0.5)*delta-delta
      t=(time_step-0.0)*dt

      ezref = amplitude(delay+t+(x*cost-y*sint)/c)

      RETURN
      END


c**********************************************************************
c REAL FUNCTION HXREF returns the value eta times the Hx reflected field
c at the given location and time.
c**********************************************************************

      REAL FUNCTION hxref(i,j,time_step)

      implicit none
      include 'common.f'

      integer i,j,time_step
      real x, y, t, amplitude

      x=(i+0.5)*delta
      y=(j+0.0)*delta-delta
```

```
      t=(time_step+0.5)*dt

      hxref = -sint*amplitude(delay+t+(x*cost-y*sint)/c)

      RETURN
      END

c**********************************************************************
c REAL FUNCTION HYREF returns the value eta times the Hy reflected field
c   at the given location and time.
c**********************************************************************

      REAL FUNCTION hyref(i,j,time_step)

      implicit none
      include 'common.f'

      integer i,j,time_step

      real x, y, t, amplitude

      x=(i+0.0)*delta
      y=(j+0.5)*delta-delta
      t=(time_step+0.5)*dt

      hyref = -cost*amplitude(delay+t+(x*cost-y*sint)/c)

      RETURN
      END

c**********************************************************************
c REAL FUNCTION AMPLITUDE returns the value of the envelope at a given
c   space-time locations
c**********************************************************************

      REAL FUNCTION amplitude(x)

      implicit none
      include 'common.f'

      real x

      if (modulate.eq.1) then
         amplitude = 15*exp(-((2*sqrt(2.5)*x/width)**2.0))*
     1        sin(2*pi*modfreq*x)
      else
         amplitude = 15*exp(-((2*sqrt(2.5)*x/width)**2.0))
      end if

      RETURN
      END
```

The following, **post.f**, contains the subroutines for performing the DFT on the fly of the fields as well as those for computing the radar cross sections.

```
c****************************************
c**** 2D FDTD H-polarization code ******
c****         POST.F              ******
c****************************************

c************************************************c
c********* IMPORTANT! PLEASE READ BELOW *********c
c************************************************c
c Note that in the code below, H fields are      c
c labeled as E fields, and E fields are labeled  c
c as H fields.  Signs and other parameters have  c
c been adjusted to correctly represent Maxwell's c
c equations. The reason for this is that the     c
c 2D FDTD E-polarization case has already been   c
c done, and we use duality rather than go        c
c through and change every variable name.        c
c************************************************c
c************************************************c

c**********************************************************************
c UPDATE_FREQS performs the DFT on the fly along a virtual surface
c   that encloses the scatterer source.
c**********************************************************************

      SUBROUTINE update_freqs(time_step)

      implicit none
      include 'common.f'

      integer i, j, time_step, k, fi, ytemp
      real temp, cfreq
      complex tempfactor

      do 10 fi = minf,maxf,stepf
         if (include_ground_plane) then
            k = y2+rcs_space
         else
            k = 0
         end if

c         cfreq = low_freq+dfreq*(fi+0.0)
         cfreq = freqlist(fi,1)

         tempfactor = exp(2.0*pi*(0.0,1.0)*dt*time_step*cfreq)
c         print *,2.0*pi*(0.0,1.0)*dt*time_step*(low_freq+
c     1        dfreq*(fi+0.0))

         i = x1-rcs_space
         if (include_ground_plane) then
            ytemp = y1
         else
            ytemp = y1-rcs_space
         end if
         do 20 j=ytemp,y2+(rcs_space-1)
c            if (j.eq.(y1-rcs_space)) print *,k+1
            k=k+1

            temp = ez(i,j)
            ezfreq(fi,k)=ezfreq(fi,k)+temp*tempfactor

            temp = 0.5*(hx(i,j)+hx(i,j+1))
            hxfreq(fi,k)=hxfreq(fi,k)+temp*tempfactor

            temp = 0.5*(hy(i,j)+hy(i+1,j))
            hyfreq(fi,k)=hyfreq(fi,k)+temp*tempfactor
 20      continue

         j=y2+rcs_space
         do 30 i=x1-rcs_space,x2+rcs_space-1
c            if (i.eq.(x1-rcs_space)) print *,k+1
            k=k+1

            temp = ez(i,j)
            ezfreq(fi,k)=ezfreq(fi,k)+temp*tempfactor

            temp = 0.5*(hx(i,j)+hx(i,j+1))
            hxfreq(fi,k)=hxfreq(fi,k)+temp*tempfactor

            temp = 0.5*(hy(i,j)+hy(i+1,j))
            hyfreq(fi,k)=hyfreq(fi,k)+temp*tempfactor
 30      continue

         i=x2+rcs_space
         if (include_ground_plane) then
            ytemp = y1
         else
            ytemp = y1-(rcs_space-1)
         end if
         do 40 j=y2+rcs_space,ytemp,-1
c            if (j.eq.(y2+rcs_space)) print *,k+1
            k=k+1

            temp = ez(i,j)
            ezfreq(fi,k)=ezfreq(fi,k)+temp*tempfactor

            temp = 0.5*(hx(i,j)+hx(i,j+1))
            hxfreq(fi,k)=hxfreq(fi,k)+temp*tempfactor

            temp = 0.5*(hy(i,j)+hy(i+1,j))
            hyfreq(fi,k)=hyfreq(fi,k)+temp*tempfactor
 40      continue

         if (.NOT.include_ground_plane) then
            j=y1-rcs_space
            do 50 i=x2+rcs_space,x1-rcs_space,-1
c               if (i.eq.(x2+rcs_space)) print *,k+1
               k=k+1

               temp = ez(i,j)
               ezfreq(fi,k)=ezfreq(fi,k)+temp*tempfactor

               temp = 0.5*(hx(i,j)+hx(i,j+1))
               hxfreq(fi,k)=hxfreq(fi,k)+temp*tempfactor

               temp = 0.5*(hy(i,j)+hy(i+1,j))
               hyfreq(fi,k)=hyfreq(fi,k)+temp*tempfactor
 50         continue
         end if

 10   continue

      RETURN
```

```
      END

c**************************************************************************
c WRITE_OUT_FREQS writes out all freq field comp data.
c**************************************************************************

      SUBROUTINE write_out_freqs

      implicit none
      include 'common.f'

      integer fi, k, rbeg, rend
      complex temp

      write(6,*) 'Writing out frequency data...'

      open(unit=10,file='ezf.dat',status='unknown',form='formatted')
      open(unit=11,file='hxf.dat',status='unknown',form='formatted')
      open(unit=12,file='hyf.dat',status='unknown',form='formatted')

c**** Save state

      open(unit=13,file='info.dat',status='unknown',form='formatted')

      write(13,*) max_x
      write(13,*) max_y
      write(13,*) delta
      write(13,*) dt
      write(13,*) tot_time_steps
      write(13,*) x1,y1,x2,y2
      write(13,*) inc_ang
      write(13,*) modulate
      write(13,*) modfreq
      write(13,*) delay
      write(13,*) width
      write(13,*) include_ground_plane
      write(13,*) num_freqs
      do 30 fi = minf,maxf
         write(13,*) freqlist(fi,1)
 30   continue

      close(unit=13)

      if (include_ground_plane) then
         rbeg = y2+rcs_space+1
         rend = 3*y2+x2-x1-1+5*rcs_space
      else
         rbeg = 1
         rend = 2*(x2-x1)+2*(y2-y1)+8*rcs_space+1
      end if
      do 10 fi = minf,maxf,stepf
         do 20 k=rbeg, rend
            temp = ezfreq(fi,k)
            write(10,*) real(temp),aimag(temp)
            temp = hxfreq(fi,k)
            write(11,*) real(temp),aimag(temp)
            temp = hyfreq(fi,k)
            write(12,*) real(temp),aimag(temp)
 20      continue
 10   continue

      close(unit=10)
      close(unit=11)
      close(unit=12)

      RETURN
      END

c**************************************************************************
c CALC_RCS calculates the 2D RCS at the requested output locations.
c**************************************************************************

      SUBROUTINE calc_rcs

      implicit none
      include 'common.f'

      integer fi,index1,llc,ulc,urc,lrc, tfreq
      real cfreq, kwave, sinp, cosp, phi_obs, time, cphi_obs
      real xphys, yphys, amplitude, rcs, dpfreq, dkwave,iphi_obs
c      real tempr, tempi

      complex eincfreq, I1, I2, I3, I4, Fphi, uniti, phase

      write(6,*) 'Calculating RCS...'

c**** Define unit imaginary number
      uniti = (0.0,1.0)

c**** Redefine x1,x2,y1,y2 so that are the corners RCS box

      if (include_ground_plane) y1=-y2+1

      x1 = x1-rcs_space
      x2 = x2+rcs_space
      y1 = y1-rcs_space
      y2 = y2+rcs_space

c**** Determine labels for corners based on update_freq
c**** llc = lower left corner, ulc = upper left corner
c**** lrc = lower right corner, urc = upper right corner

c**** this one has two labels, 1, and what's given below
      llc = 2*(x2-x1)+2*(y2-y1)+1

      ulc = y2-y1+1
      urc = x2-x1+y2-y1+1
      lrc = 2*(y2-y1)+x2-x1+1
      print *, x1,y1,x2,y2
      print *, 1, ulc, urc, lrc, llc

c**** with ground plane use image theory to gen. fields for lower half
c**** of huygens' surface. Equiv. to using layered Green's functions.
      if (include_ground_plane) then
         do 500 fi=minf,maxf,stepf
            do 510 index1=1,y2
c************* Reflect left top side of Huygens' surface down
               ezfreq(fi,index1) = ezfreq(fi,2*y2+1-index1)
               hxfreq(fi,index1+1) = -hxfreq(fi,2*y2+1-index1)
               hyfreq(fi,index1) = hyfreq(fi,2*y2+1-index1)

c************* Reflect right topside of Huygens' surface down
               ezfreq(fi,lrc-index1+1) = ezfreq(fi,urc+index1-1)
               hxfreq(fi,lrc-index1) = -hxfreq(fi,urc+index1-1)
               hyfreq(fi,lrc-index1+1) = hyfreq(fi,urc+index1-1)
 510        continue
c********** Approximations b/c nothing to reflect
            hxfreq(fi,1) = hxfreq(fi,2)
            hxfreq(fi,lrc) = hxfreq(fi,lrc-1)
            do 520 index1=ulc+1,urc-1
c************* Reflect middle top of Huygens' surface down
               ezfreq(fi,llc-index1+ulc) = ezfreq(fi,index1)
               hxfreq(fi,llc-index1+ulc) = -hxfreq(fi,index1)
               hyfreq(fi,llc-index1+ulc) = hyfreq(fi,index1)
 520        continue
 500     continue
      end if

      open(unit=10,file='rcs.dat',status='unknown',form='formatted')
      open(unit=14,file='inc.dat',status='unknown',form='formatted')
      open(unit=15,file='debug.dat',status='unknown',form='formatted')

c********************** READ IN EXACT DATA
c      open(unit=11,file='ez.dat',status='unknown',form='formatted')
c      open(unit=12,file='hx.dat',status='unknown',form='formatted')
c      open(unit=13,file='hy.dat',status='unknown',form='formatted')
c
c      do 200 index1 = 1, llc-1
c         read(11,*) tempr, tempi
c         ezfreq(0,index1) = (tempr+(0.0,1.0)*tempi)
c         read(12,*) tempr, tempi
c         hxfreq(0,index1) = (tempr+(0.0,1.0)*tempi)
c         read(13,*) tempr, tempi
c         hyfreq(0,index1) = (tempr+(0.0,1.0)*tempi)
c 200  continue
c      close(unit=11)
c      close(unit=12)
c      close(unit=13)
c
c      ezfreq(0,llc) = ezfreq(0,1)
c      hxfreq(0,llc) = hxfreq(0,1)
c      hyfreq(0,llc) = hyfreq(0,1)
c
c********************** END OF READING IN EXACT DATA

c**** loop through all freqs of interest.

      do 10 fi = minf, maxf, stepf
c         cfreq = low_freq+dfreq*(fi+0.0)

c******* Note 1: tfreq = 0 if normal frequency.
c*******         tfreq = i (i=1..MNANG) cfreq(46,1) contains the true
c*******         freq.
c******* Note 2: the monostatic angles calculated are always
c*******         inc_ang-45, ...., inc_ang+45 in 1 deg increments
c******* Note 3: The variables dpfreq and dkwave are determined based
c*******         on the current frequency at which the RCS is actually
c*******         being calculated at.

         cfreq = freqlist(fi,1)
         tfreq = freqlist(fi,2)
         if (tfreq.eq.0) then
            dpfreq = cfreq
         else
```

```
            dpfreq = freqlist(46+int((fi-1.0)/(MNANG+0.0))*MNANG,1)
            if (freqlist(46+int((tfreq-1.0)/(MNANG+0.0))*MNANG,2).ne.
     1           46) then
               print *, 'error, bad location of actual freq'
            end if
         end if

         dkwave = 2.0*pi*dpfreq/c
         kwave = 2.0*pi*cfreq/c

         eincfreq = 0.0
         do 20 index1 = 1, tot_time_steps
            time = (index1+0.0)*dt
            eincfreq = eincfreq + amplitude(delay+time+(delta*max_x*
     1           cost/2.0+delta*max_y*sint/2.0)/c)*cexp(2.0*pi*uniti*
     1           (time*cfreq))
 20      continue
c         print *,cfreq,eincfreq

c**** loop through all scattering angles of interest

c**** if mono_bi == 1, choose normal bistatic RCS angles
c****   in this case, phi_obs and cphi_obs are equal since the reported
c****   angle is the same as the angle used in the calculations.

c**** if mono_bi == 2, choose angles based on freqs for monostatic RCS
c****   in this case, phi_obs represents the monostatic angle we are
c****   trying to calculate the RCS at, while cphi_obs represents the
c****   bistatic angle we use to approx the monostatic angle we need

         do 30 iphi_obs = low_phi, high_phi, dphi
            phi_obs = iphi_obs
            if (mono_bi.eq.1) then
               cphi_obs = phi_obs
            else
               phi_obs = inc_ang + freqlist(fi,2)-46.0
               cphi_obs = 2*phi_obs-inc_ang

c*********      verify that we are using the correct cfreq (debug mods)
               if (abs(cfreq-dpfreq*(1.0/cos((phi_obs-inc_ang)*
     1              pi/180))).gt.tole) then
                  print *, 'error, miscalculated frequency'
c                 stop
               end if
c              write(15,*) cfreq,dpfreq,phi_obs,cphi_obs
            end if

            cosp = cos((cphi_obs/180.0)*pi)
            sinp = sin((cphi_obs/180.0)*pi)

c**** Initialize values of integrals.
            I1 = (0.0,0.0)
            I2 = (0.0,0.0)
            I3 = (0.0,0.0)
            I4 = (0.0,0.0)

c*************************************************************************
c************************** INTEGRAL 4 ***********************************
c*************************************************************************
C**** Compute Integral over y' at x1

            do 40 index1 = 1,ulc
               yphys = real(y1+index1-1)*delta
               xphys = real(x1)*delta
               phase = -uniti*kwave*yphys*sinp

               I4 = I4 + (hyfreq(fi,index1) + ezfreq(fi,index1)*
     1              cosp)*cexp(+phase)*delta
 40         continue

            I4 = I4*cexp(-uniti*kwave*xphys*cosp)

c*************************************************************************
c************************** INTEGRAL 3 ***********************************
c*************************************************************************
c**** Compute Integral over x' at y2

            do 60 index1 = ulc,urc
               yphys = real(y2)*delta
               xphys = real(x1+index1-ulc)*delta
               phase = -uniti*kwave*xphys*cosp

               I3 = I3 + (hxfreq(fi,index1)-ezfreq(fi,index1)*
     1              sinp)*cexp(+phase)*delta
 60         continue

            I3 = I3*cexp(-uniti*kwave*yphys*sinp)

c*************************************************************************
c************************** INTEGRAL 2 ***********************************
c*************************************************************************
c**** Compute Integral over y' at x2

            do 80 index1 = urc,lrc
               yphys = real(y2-index1+urc)*delta
               xphys = real(x2)*delta
               phase = -uniti*kwave*yphys*sinp

               I2 = I2 + (-hyfreq(fi,index1)-ezfreq(fi,index1)*
     1              cosp)*cexp(+phase)*delta
 80         continue

            I2 = I2*cexp(-uniti*kwave*xphys*cosp)

c*************************************************************************
c************************** INTEGRAL 1 ***********************************
c*************************************************************************
c**** Compute Integral over x' at y1

            do 100 index1 = lrc,llc
               yphys = real(y1)*delta
               xphys = real(x2-index1+lrc)*delta
               phase = -uniti*kwave*xphys*cosp

               I1 = I1 + (-hxfreq(fi,index1)+ezfreq(fi,index1)*
     1              sinp)*cexp(+phase)*delta
 100        continue

            I1 = I1*cexp(-uniti*kwave*yphys*sinp)

C**** Sum Integrals

            Fphi = cexp(uniti*pi/4.0)*sqrt(kwave/8.0/pi)*(I1+
     1           I2+I3+I4)

c            print *,cfreq,phi_obs,Fphi
C**** Calculate RCS

            rcs = 2*pi*((cabs(Fphi))**2.0)/((cabs(eincfreq))**2.0)

c**** For use when reading in exact near field data.
c           rcs = 2*pi*((cabs(Fphi))**2.0)

c**** Write out results (RCS given in dBm)
c**** Note extra column of zeros written out to allow compatibility
c**** with MATLAB scripts that read RCS data and plot them.

            write(10,*) dkwave,0.0,phi_obs,10*ALOG10(rcs)
            write(14,*) dkwave,real(eincfreq),imag(eincfreq),real(Fphi),
     1           imag(Fphi)


 30      continue
 10   continue

      close(unit=10)
      close(unit=14)
      close(unit=15)

      RETURN
      END

c*************************************************************************
c WRITE_OUT_MOVIE_FRAME
c*************************************************************************

      SUBROUTINE write_out_movie_frame(a,time_step)

      implicit none
      include 'common.f'

      character a(1:dummy)

      integer i,j,ia,time_step
      real ezinc

c      integer nsplit,ns,l
      integer hlevels,numcolors1,center,topcolor,nctshift,ngray
c     topcolor: location of top of colorbar
c     numcolors1: 1 less than # of colors in colorbar
      parameter (numcolors1=128,topcolor=243)
c     nctshift: = color table shift
      parameter(nctshift = topcolor-numcolors1)
      parameter(ngray = topcolor-numcolors1-1)
c     hlevels = numcolors1/4
      parameter (hlevels=numcolors1/4)
c     center = 2*hlevels + 1/2
      parameter (center = 2*hlevels + 0.5)
      character frmt*30

      write(frmt,'(a1,i4,a5)') '(', max_x, '(a1))'

      do 10 j=min(1000,max_y),1,-1
         do 20 i=1,min(1000,max_x)
```

```
c            ia=int((ez(i,j))*hlevels+center+0.5)

c            if (tot_scat(i,j).eq.0.OR.tot_scat(i,j).eq.1.OR.
c     1           tot_scat(i,j).eq.11.OR.tot_scat(i,j).eq.12.OR.
c     1           tot_scat(i,j).eq.15) then
c               ia=int(((ez(i,j)+ezinc(i,j,time_step+1))/4.0)*hlevels+
c     1              center+0.5)
c            else
               ia=int((ez(i,j)/4.0)*hlevels+center+0.5)
c            end if
c            if (abs(ez(i,j).gt.max_field)) then
c               max_field=abs(ez(i,j))
c               print *,time_step,i,j,max_field
c            end if

            if (ia .lt. 0) ia=0
            if (ia .gt. numcolors1) ia=numcolors1
            a(i)=char(ia+nctshift)
 20      continue

         write(4,frmt) a
 10    continue

       RETURN
       END

c**********************************************************************
c WRITE_OUT_ALL_PARMS outputs to a file all important parameters used
c   in running the simulation.
c**********************************************************************

       SUBROUTINE write_out_all_parms(store_freq)

       implicit none
       include 'common.f'

       integer index
       real x(1:MAX_STAIR_NODES), y(1:MAX_STAIR_NODES)
       logical store_freq

c       open(unit=10,file='st2.dat',status='unknown',form='formatted')

       do 10 index = 1,stair_node_count-1
          x(index) = real(stair_zero(index,1))
          y(index) = real(stair_zero(index,2))
c          write(10,*) x(index),y(index),stair_zero(index,3)
 10    continue
c       close(unit=10)

       open(unit=10,file='fdtd.out',status='unknown',form='formatted')

       write(10,*) 'max_x = ', max_x,';'
       write(10,*) 'max_y = ', max_y,';'
       write(10,*) 'delta = ', delta,';'
       write(10,*) 'dt = ', dt,';'
       write(10,*) 'N = ', tot_time_steps,';'
       write(10,*) 'inc_ang = ', inc_ang,';'
       write(10,*) 'modulate = ', modulate,';'
       write(10,*) 'modfreq = ', modfreq,';'
       write(10,*) 'delay = ', delay,';'
       write(10,*) 'width = ', width,';'
       write(10,*)
       write(10,*) 'width/dt = ', width/dt
       write(10,*) 'stair_node_count = ', stair_node_count
       write(10,*) 'sigma_max = ', sigma_max
       write(10,*) 'x1,y1,x2,y2= ',x1,y1,x2,y2
       if (include_ground_plane) then
          write(10,*) 'rcs_nodes = ',2*(x2-x1)+4*y2-2+8*rcs_space+1
       else
          write(10,*) 'rcs_nodes = ', 2*(y2-y1)+2*(x2-x1)+8*rcs_space+1
       end if
       if (store_freq) then
          write(10,*) 'low_freq (GHz) = ', freqlist(1,1)/1.0e9
          write(10,*) 'high_freq (GHz) = ', freqlist(num_freqs,1)/1.0e9
          write(10,*) 'num_freqs = ', maxf-minf+1
       end if
       call plotb(x,y,stair_node_count-1,51,41,10)

       close(unit=10)

       RETURN
       END


c**********************************************************************
c PLOTB takes a set of data points (X,Y) and makes and ascii plot out
c   of them
c**********************************************************************

       SUBROUTINE PLOTB(X,Y,N,NC,NR,FID)
C
C WRITTEN 2/14/74   BY J. M. PUTNAM       DEPT 220     X23877
C
C THIS ROUTINE PRODUCES A LINEAR XY PLOT.
C    N IS THE NUMBER OF POINTS TO BE PLOTTED.
C    NR IS THE NUMBER OF ROWS TO BE USED FOR THE Y-AXIS.
C    NC IS THE NUMBER OF COLUMNS TO BE USED FOR THE X-AXIS.
C       NOTE, NC-1 MUST BE DIVISIBLE BY 10 AND LESS THAN 102.
C

       REAL X(161),Y(161),HEAD(10)
       INTEGER  LINE(101),BLANK,STAR,FID
       DATA BLANK,STAR /1H ,1H*/
       N10=(NC-1)/10
       WRITE(FID,500)
 500   FORMAT(//,17H1BODY COORDINATES)
       WRITE(FID,504)
       XMIN=X(1)
       XMAX=X(1)
       YMIN=Y(1)
       YMAX=Y(1)
       DO 6 I=1,N
       IF(X(I).LT.XMIN) XMIN=X(I)
       IF(X(I).GT.XMAX) XMAX=X(I)
       IF(Y(I).LT.YMIN) YMIN=Y(I)
       IF(Y(I).GT.YMAX) YMAX=Y(I)
 6     CONTINUE
       DEL=XMAX-XMIN
       IF(YMAX-YMIN.GT.DEL) DEL=YMAX-YMIN
       XMAX=XMIN+DEL
       YMAX=YMIN+DEL
       DO 5 I =1,N10
       Z=I
    5  HEAD(I)=(XMAX-XMIN)*Z/N10+XMIN
       DY=(YMAX-YMIN)/(NR-1)
       Z=YMAX+DY
       YL=Z-DY/2.
       DO 7 J=1,NR
       DO 8 K=1,NC
 8     LINE(K)=BLANK
       Z=Z-DY
       YU=YL
       YL=Z-DY/2.
       DO 9 I=1,N
       IF(Y(I).GE.YU) GO TO 9
       IF(Y(I).LT.YL) GO TO 9
       K=(X(I)-XMIN)/(XMAX-XMIN)*(NC-1)+1.5
       IF(K.GT.NC) K=NC
       LINE(K)=STAR
 9     CONTINUE
       WRITE(FID,508) Z,(LINE(K),K=1,NC)
 7     CONTINUE
       WRITE(FID,504)
       WRITE(FID,3002)
       WRITE(FID,507) XMIN,(HEAD(I),I=1,N10)
C      *************************
       RETURN
  504  FORMAT ( 1X, 14(1H-), 1H., 10(5H----.), 1H- )
  507  FORMAT(10X,11(F10.4))
  508  FORMAT (1X, F12.4,1X, 1HI, 51A1, 1HI )
 3002  FORMAT(4X,7HRB / ZB,4X,1HI,5(9X,1HI))

       END
```

The following, **common.f**, is used to create the common blocks that are included in most of the subroutines used by the 2D FD-TD TM program.

```
c***************************************
c**** 2D FDTD H-polarization code ******
c****        COMMON.F             ******
c***************************************

c**** common.include file.  Contains all global variable declarations.

c**** Variables changed to allocate memory

       integer MAX_X_CELLS, MAX_Y_CELLS, MAX_NODES, MAX_STAIR_NODES,
      1     MAX_RCS_NODES, MAX_FREQS

       parameter(MAX_X_CELLS=100)
       parameter(MAX_Y_CELLS=100)
       parameter(MAX_NODES=10)
       parameter(MAX_STAIR_NODES=81)
       parameter(MAX_RCS_NODES=169)
       parameter(MAX_FREQS=546)

c**********************************************************************
c*****  DO NOT CHANGE BELOW ********************************************
```

```
c***********************************************************************

c**** Important contants
      real c, mu, pi, eps, eta

      parameter(c=2.9979247917E8, mu=1.25663706144E-6, pi=3.1415926535,
     1     eps=8.8541874E-12, eta=376.73032)

      real tole
      parameter(tole = 1.0e-8)

c**** Simulation Variables

      real delta, dt, sigma_max, reflection, ij
      integer tot_time_steps, dummy, movie_step
      logical store_movie

c**** Grid layout variables.

      integer PML_DEPTH

      parameter(PML_DEPTH=15)
      integer tot_scat(1:MAX_X_CELLS,1:MAX_Y_CELLS)

      integer max_x, max_y

c**** Points 1 & 2 define the corners of a tot/field region
      integer x1,y1,x2,y2

c**** Geometry data points
      real xnodes(1:MAX_NODES), ynodes(1:MAX_NODES)
      integer total_nodes, stair_node_count,
     1     stair_zero(1:MAX_STAIR_NODES,1:3), ezf, exf, eyf
      logical include_ground_plane

      parameter(ezf=1,exf=2,eyf=3)

      integer errorcount, errors(10),
     1     NODE_ERROR, MAX_X_ERROR, MAX_Y_ERROR, MAX_STAIR_ERROR,
     2     MAX_RCS_ERROR

      parameter(NODE_ERROR=1, MAX_X_ERROR=2, MAX_Y_ERROR=3,
     1     MAX_STAIR_ERROR=4, MAX_RCS_ERROR=5)

c**** Incident Wave parameters
      real inc_ang, modfreq, sint, cost, delay, width
      integer modulate

c**** Fields: TM Case

c**** Incident fields

      integer MAX_M_CELLS
      parameter(MAX_M_CELLS = (MAX_X_CELLS+MAX_Y_CELLS))

      real Hinc(1:MAX_M_CELLS), Einc(1:MAX_M_CELLS)

c**** Normal fields

      real hx(1:MAX_X_CELLS,1:MAX_Y_CELLS),
     1     hy(1:MAX_X_CELLS,1:MAX_Y_CELLS),
     2     ez(1:MAX_X_CELLS,1:MAX_Y_CELLS)

c**** PML fields

c**** Top and Center fields
      real ezxtt(1:MAX_X_CELLS,1:PML_DEPTH),
     1     ezytt(1:MAX_X_CELLS,1:PML_DEPTH),
     1     hxxtt(1:MAX_X_CELLS,1:PML_DEPTH),
     1     hyytt(1:MAX_X_CELLS,1:PML_DEPTH)

c**** Bottom and Center fields
      real ezxbb(1:MAX_X_CELLS,1:PML_DEPTH),
     1     ezybb(1:MAX_X_CELLS,1:PML_DEPTH),
     1     hxxbb(1:MAX_X_CELLS,1:PML_DEPTH),
     1     hyybb(1:MAX_X_CELLS,1:PML_DEPTH)

c**** Right and Center fields
      real ezxrr(1:PML_DEPTH,1:MAX_Y_CELLS),
     1     ezyrr(1:PML_DEPTH,1:MAX_Y_CELLS),
     1     hxxrr(1:PML_DEPTH,1:MAX_Y_CELLS),
     1     hyyrr(1:PML_DEPTH,1:MAX_Y_CELLS)

c**** Left and Center fields
      real ezxll(1:PML_DEPTH,1:MAX_Y_CELLS),
     1     ezyll(1:PML_DEPTH,1:MAX_Y_CELLS),
     1     hxxll(1:PML_DEPTH,1:MAX_Y_CELLS),
     1     hyyll(1:PML_DEPTH,1:MAX_Y_CELLS)

c**** Top and Right fields
      real ezxtr(1:PML_DEPTH,1:PML_DEPTH),
     1     ezytr(1:PML_DEPTH,1:PML_DEPTH),
     1     hxxtr(1:PML_DEPTH,1:PML_DEPTH),
     1     hyytr(1:PML_DEPTH,1:PML_DEPTH)

c**** Top and Left fields
      real ezxtl(1:PML_DEPTH,1:PML_DEPTH),
     1     ezytl(1:PML_DEPTH,1:PML_DEPTH),
     1     hxxtl(1:PML_DEPTH,1:PML_DEPTH),
     1     hyytl(1:PML_DEPTH,1:PML_DEPTH)

c**** Bottom and Right fields
      real ezxbr(1:PML_DEPTH,1:PML_DEPTH),
     1     ezybr(1:PML_DEPTH,1:PML_DEPTH),
     1     hxxbr(1:PML_DEPTH,1:PML_DEPTH),
     1     hyybr(1:PML_DEPTH,1:PML_DEPTH)

c**** Bottom and Left fields
      real ezxbl(1:PML_DEPTH,1:PML_DEPTH),
     1     ezybl(1:PML_DEPTH,1:PML_DEPTH),
     1     hxxbl(1:PML_DEPTH,1:PML_DEPTH),
     1     hyybl(1:PML_DEPTH,1:PML_DEPTH)

c**** Frequency domain points
      integer minf, maxf, stepf, num_freqs, rcs_space
      parameter(rcs_space=2)

      real freqlist(1:MAX_FREQS,1:2)
      integer mono_bi
      complex ezfreq(1:MAX_FREQS, 1:MAX_RCS_NODES),
     1     hxfreq(1:MAX_FREQS, 1:MAX_RCS_NODES),
     2     hyfreq(1:MAX_FREQS, 1:MAX_RCS_NODES)

C**** Maximum number of monostatic observation points. Currently
C**** set to angle resolution of 1 degree.
      integer MNANG
      parameter(MNANG=91)

c**** Output information
      real low_phi, high_phi, dphi

c***********************************************************************c

c**** Fields Common Block ************

      common hx, hy, ez,
     1     ezxtt, ezytt, hxxtt, hyytt,
     1     ezxbb, ezybb, hxxbb, hyybb,
     1     ezxll, ezyll, hxxll, hyyll,
     1     ezxrr, ezyrr, hxxrr, hyyrr,
     1     ezxtr, ezytr, hxxtr, hyytr,
     1     ezxtl, ezytl, hxxtl, hyytl,
     1     ezxbr, ezybr, hxxbr, hyybr,
     1     ezxbl, ezybl, hxxbl, hyybl,
     1     tot_scat, Hinc, Einc, ezfreq,
     1     hxfreq, hyfreq, freqlist

c**** Incident Wave Parms Common Block *******

      common inc_ang, modulate, modfreq,
     1     sint, cost, delay, width

c**** Simulation & Grid Layout Common Block *******

      common delta, dt, tot_time_steps, store_movie, movie_step,
     1     max_x, max_y, sigma_max, reflection,
     2     x1, y1, x2, y2, xnodes, ynodes, total_nodes,
     3     stair_zero, stair_node_count, errors, errorcount,
     4     dummy, ij, include_ground_plane

c**** Output variables common block

      real max_field

      common low_phi, high_phi, dphi, num_freqs,
     1     minf, maxf, stepf, max_field, mono_bi
```

D.4 BOR PWE Program

The BOR PWE program calculates bistatic radar cross sections of a PEC body of revolution with arbitrary cross section. The paraxial direction may be specified in the $\pm\hat{z}$ directions. The incident wave direction may be specified independently of the paraxial direction. The program can loop through a set of frequncies to obtain the RCS over an extended bandwidth.

The following, **pwe.f**, contains the core subroutines for the BOR PWE including the input routines, the range marching/linear system setup subroutines, and the RCS calculation subroutine.

```
************************************************************************
* PWE --                                                               *
*     This program uses the narrow angle BOR PWE technique to          *
*     calculate the bistatic RCS of BOR targets. The current           *
*     implementation restricts the paraxial direction to be in the     *
*     +/- z directions, but arbitrary incident wave direction.         *
************************************************************************

      program PWE

      implicit none
      include 'common.f'

      real freq

      call get_primary_input

      open(unit=9,file='rcs.dat',status='unknown',form='formatted')

      do 10 freq = start_freq,end_freq,dfreq
         kwave = 2*pi*freq/c

         if (comp_forw_scat.eq.1) then
            call compute_forward_fields
         else if (comp_forw_scat.eq.2) then
            call compute_back_fields
         end if

         call write_out_rcs_surface_data
         call calc_rcs

 10   continue

      END

c************************************************************************
c GET_PRIMARY_INPUT gets info from user about geomfile name, incident
c    wave direction and frequency, paraxial direction, and desired out-
c    put information.
c************************************************************************

      SUBROUTINE get_primary_input

      implicit none
      include 'common.f'

      character*72 ifname
      integer image_test, nang
      real freq
      integer num_freqs

      write(6,'(''*Enter geometry file name: '', $)')
      read(5,*) geomfile

      write(6, '(''*Store image? (1 for Y, !=1 for N) '', $)')
      read(5,*) image_test
      store_image = (image_test.eq.1)

      if (store_image) then
         write(6,'(''*Movie imgfile name: '', $)')
         read(5,*) ifname
c**** Open up file 'ifname' for image storage
         open(unit=15,file=ifname,status='unknown',form='formatted')
         write(6,*) 'Erho=1, Ephi=2, Ez=3'
         write(6,'(''*Enter fieldtype ID: '', $)')
         read(5,*) imagefieldtype
      end if

      write(6,'(''*Enter incident angle in degree: '', $)')
      read(5,*) inc_ang

 31   if (inc_ang.gt.360) then
         inc_ang = inc_ang-360
         goto 31
      else if (inc_ang.lt.0) then
         inc_ang = inc_ang+360
         goto 31
      else
         inc_ang = inc_ang/180*pi
      end if

      write(6,'(''*Enter lowest frequency of interest: '',$)')
      read(5,*) start_freq
      write(6,'(''*Enter highest frequency of interest: '',$)')
      read(5,*) end_freq
      kwave = 2*pi*end_freq/c

      dfreq = start_freq
      if (abs(end_freq-start_freq).gt.tole) then
         write(6,'(''*Enter the number of frequencies: '',$)')
         read(5,*) num_freqs

         dfreq = (end_freq-start_freq)/(num_freqs-1.0)
      end if

c      write(6,'(''*Enter frequency for simulation: '', $)')
c      read(5,*) freq
c      kwave = 2*pi*freq/c

      polarization = HORZ
c      polarization = VERT

 33   write(6,'(''*Select polarization (1=HORZ,2=VERT): '',$)')
      read(5,*) polarization
      if (polarization.eq.1) then
         polarization = HORZ
         Ehg = 1
         Evg = 0
      else if (polarization.eq.2) then
         polarization = VERT
         Ehg = 0
         Evg = 1.0
      else
         goto 33
      end if

      write(6,*) '1. Forward only'
      write(6,*) '2. Backward only'
      write(6,*) '3. Forward, then Backward'
 34   write(6,'(''*Select paraxial direction(s): '',$)')
      read(5,*) comp_forw_scat
      if (comp_forw_scat.lt.1.OR.comp_forw_scat.gt.3) goto 34

      call setup_geometry

      if (abs(sin(inc_ang)).lt.1d-6) then
         smode = 1
         emode = 1
      else
         write(6,*) 'k*rho_max = ', heightkwave
         write(6,*) 'k*rho_max*sin(theta_i) = ', heightkwave*
     1        sin(inc_ang)
         write(6,'(''*Enter start mode: '', $)')
         read(5,*) smode
         write(6,'(''*Enter end mode: '', $)')
         read(5,*) emode
      end if

      write(6,*) 'Bistatic RCS angles (in degrees)'
      write(6,'(''*Enter initial and final phi: '',$,$)')
      read(5,*) low_phi,high_phi

      if (abs(low_phi-high_phi).lt.tole) then
```

```
         dphi = high_phi-low_phi+1.0
      else
         write(6,'(''*Enter number of angles: '',$)')
         read(5,*) nang
         dphi = (high_phi-low_phi)/ real(nang-1.0)
      end if

      write(6,'(''*Enter initial and final theta: '',$,$)')
      read(5,*) low_theta,high_theta

      if (abs(low_theta-high_theta).lt.tole) then
         dtheta = high_theta-low_theta+1.0
      else
         write(6,'(''*Enter number of angles: '',$)')
         read(5,*) nang
         dtheta = (high_theta-low_theta)/ real(nang-1.0)
      end if


      RETURN
      END

c**********************************************************************
c COMPUTE_FORWARD_FIELDS controls the computation of the forward
c    scattered fields. Uses subroutines SETUP_LIN_SYS and LINBCGSTAB
c**********************************************************************

      SUBROUTINE compute_forward_fields

      implicit none
      include 'common.f'

      integer nrange_step, mode, k, iter, step
      double precision err
      complex*16 ef2(NMAX)

      call memory_check
      call write_out_all_parms

      write(6,*) 'Beginning Forward range stepping'
c      write(6,'(''*Enter step to record linsys at: '',$)')
c      read(5,*) step
      step = 196

      do 15 mode = smode, emode
         write(6,*) 'Initializing fields for mode =', mode
         call init_fields

      open(unit=18,file='iter.dat',status='unknown',form='formatted')
      do 20 nrange_step = 1,max_range_step
         write(6,*) mode,nrange_step
         call setup_lin_sys(nrange_step,mode)

         if (nrange_step.eq.step) call write_out_lin_sys
         if (store_image) call write_out_image(15)

         call linbcgstab(3*max_k+6,efields,ef2,1,1.d-15,max(6*max_k,
     1        9000),iter,err)
         write(18,*) mode,nrange_step,iter,err

         do 25 k=1,3*max_k+6
            efields(k) = ef2(k)
 25      continue

         call store_rcs_comp(nrange_step, mode)
 20   continue

      call calc_h_rcs_comp(mode)

 15   continue

      close(unit=18)

      if (store_image) then
         close(unit=15)
      end if

      RETURN
      END

c**********************************************************************
c COMPUTE_BACK_FIELDS controls the computation of the backward
c    scattered fields. Uses subroutines SETUP_LIN_SYS and LINBCGSTAB
c**********************************************************************

      SUBROUTINE compute_back_fields

      implicit none
      include 'common.f'

      integer nrange_step, mode, k, iter, step
      double precision err
      complex*16 ef2(NMAX)

      call memory_check
      call write_out_all_parms

      write(6,*) 'Beginning Backward range stepping'
c      write(6,'(''*Enter step to record linsys at: '',$)')
c      read(5,*) step
      step = 196

      do 15 mode = smode, emode
         write(6,*) 'Initializing fields for mode = ', mode
         call init_fields

      open(unit=18,file='iter.dat',status='unknown',form='formatted')
      do 20 nrange_step = max_range_step,1,-1
         write(6,*) mode,nrange_step
         call setup_lin_sys(nrange_step,mode)

         if (nrange_step.eq.step) call write_out_lin_sys
         if (store_image) call write_out_image(15)

         call linbcgstab(3*max_k+6,efields,ef2,1,1.d-15,max(6*max_k,
     1        9000),iter,err)
         write(18,*) mode,nrange_step,iter,err

         do 25 k=1,3*max_k+6
            efields(k) = ef2(k)
 25      continue

         call store_rcs_comp(nrange_step, mode)
 20   continue

      call calc_h_rcs_comp(mode)

 15   continue

      close(unit=18)

      if (store_image) then
         close(unit=15)
      end if

      RETURN
      END


c**********************************************************************
c SETUP_LIN_SYS setups the wave equation linear system that enables
c    the fields at range step 'nrange_step' to be solved for. It uses
c    data stored in the geometry description variables to implement
c    correct boundary conditions which serve to source the propagating
c    fields.
c**********************************************************************

      SUBROUTINE setup_lin_sys(nrange_step,mode)

      implicit none
      include 'common.f'

      integer rowr,rowp,rowz,colr,colp,colz,rowbcp,rowbcrz,rowdiv,
     1     rowdiv2,rowbcrz2
      integer nrange_step, k, point_type(0:(MAX_R_CELLS+1)), pml_start
      complex*16 tikdi, mtik, ptil, qptil, sptil, temp
      complex*16 Erhoinc, Ephiinc, Ezinc, nz, nr
      double precision ekwave
      integer mode,pmlt,offdiagind

c**** calculate/set PML parameters

      pml_start = max_k-PML_DEPTH+1
      pmlt = PML_DEPTH

c**** Size of matrix is (3*max_k+6)x(3*max_k+6)

c      tikdi = 1.0/(2.0*uniti*kwave*delta)
      tikdi = (1.0/(2.0*uniti*kwave*delr))*(delz/delr)

c      mtik = uniti/2./kwave

      if (comp_forw_scat.eq.1) then
         mtik = uniti/2./kwave
         ekwave = kwave
      else if (comp_forw_scat.eq.2) then
         mtik = -uniti/2./kwave
         ekwave = -kwave
      else
         print *,'bad paraxial direction'
         pause
      end if
```

```
c**** Note that all fields calculated with the PWE method are scattered
c**** fields. Therefore to satisfy B.C. we use the fact that
c**** Etot = Escat + Einc = 0
c**** along the boundary for a PEC, therefore Escat = -Einc.

c**** Classify all points on current range step as either a
c**** Boundary (1) or a free space point (0).
      do 10 k=0,max_k+1
         point_type(k) = 0
 10   continue

c**** If point_type.ne.0 then it labels the point as some HAT type.
      do 20 k=range_index(nrange_step,1),range_index(nrange_step,2)
         if (abs(normsinf(k,1)).lt.900.0) then
            point_type(normsloc(k,2)-1) = k
         else
            print *,'ignoring in corner', nrange_step,normsloc(k,2)
         end if
 20   continue

      offdiagind = 3*max_k+6+2

c**** Initialize matrix
      call wtmatrix(nrange_step,0,0,(0.d0,0.d0),offdiagind)

c**** The order that wtmatrix is called matters. Each row equation
c**** must be entered in order. In addition, the diagonal element
c**** of each row equation be must placed in the sparse matrix
c**** structure before the off diagonal terms for that row.

c*************************************************************************
c*****************************ERHO TERMS**********************************
c*************************************************************************

c**** Treat lower boundary conditions: essentially same as other free
c**** space equations except use 1st order representation of 2nd
c**** derivative in rho. assume position k=0.5

c      temp = (1.d0,0.d0)
c      call wtmatrix(nrange_step,rowr(0),colr(0),temp,offdiagind)
c      Efields(rowr(0)) = (0.d0,0.d0)

c**** Diagonal term
      temp = 1.+tikdi*(ONE-ONE/(0.5+ZERO)-(mode*mode+ONE)/
     1     ((0.5+ZERO)**2))
      call wtmatrix(nrange_step,rowr(0),colr(0),temp,offdiagind)
c**** supersuperdiagonal portion
      temp = tikdi
      call wtmatrix(nrange_step,rowr(0),colr(2),temp,offdiagind)
c**** Superdiagonal portion
      temp = tikdi*(-2*ONE + ONE/(0.5+ZERO))
      call wtmatrix(nrange_step,rowr(0),colr(1),temp,offdiagind)
c***** Super-Super-superdiagonal portion (Ephi term)
      temp = -tikdi*polarization*2.0*ONE*mode/((0.5+ZERO)**2)
      call wtmatrix(nrange_step,rowr(0),colp(0),temp,offdiagind)


c**** Treat all other Erho fields
      do 30 k=1,max_k
         if (point_type(k).eq.0) then

c******* Calculate PML parameters ptil, sptil, qptil

            if (k.ge.pml_start) then
               ptil = k+(alpha+beta*uniti*eta/kwave)*
     1              ((k-pml_start)**3.0)/(3.0*pmlt*pmlt)
               sptil = 1.0+alpha*((k-pml_start+0.0)/pmlt)**2.0+beta*
     1              (uniti*eta/kwave)*((k-pml_start+0.0)/pmlt)**2.0
               qptil = -((2.0*(k-pml_start+0.0)/(pmlt*pmlt*1.0))*(alpha+
     1              beta*uniti*eta/kwave))/(sptil*sptil)
            else
               ptil = k+0.0
               sptil = (1.0,0.0)
               qptil = 0.0
            end if

c******* Use free space equations
c********** Diagonal term
            temp = 1.+tikdi*(-2.0/(sptil*sptil)-(1.0/sptil)*
     1           ((1.0/ptil)+qptil)-(mode*mode+1.0)/(ptil*ptil))
            call wtmatrix(nrange_step,rowr(k),colr(k),temp,offdiagind)
c********** Subdiagonal portion
            temp = tikdi*1.0/(sptil**2.0)
            call wtmatrix(nrange_step,rowr(k),colr(k-1),temp,offdiagind)
c********** Superdiagonal portion
            temp = (tikdi/sptil)*(1.0/sptil+(1.0/ptil+qptil))
            call wtmatrix(nrange_step,rowr(k),colr(k+1),temp,offdiagind)
c********** Super-superdiagonal portion (Ephi term)
            temp = -tikdi*polarization*2.0*mode/(ptil*ptil)
            call wtmatrix(nrange_step,rowr(k),colp(k),temp,offdiagind)
         else
c******* Use boundary point condition equations.

c********** Enforce n x (Ei+psi) = 0, ie. tangential component is zero

c********** Ez and Erho components
            nz = -(1.d0,0.d0)*cos(normsinf(point_type(k),1))
            nr = (1.d0,0.d0)*sin(normsinf(point_type(k),1))
            if (abs(nz).lt.1.e-6) nz = (0.d0,0.d0)
            if (abs(nr).lt.1.e-6) nr = (0.d0,0.d0)

            if (abs(nz).lt.1.e-6) then
               print *,'n=',nrange_step, ' k=',k,' debug=',colr(k),
     1              ' nz = ',nz,' nr= ',nr
c************** Place equations so that diagonal term is not zero.

c************** Enforce divergence free condition
               temp = -1.0/delr
               call wtmatrix(nrange_step,rowdiv2(k),colr(k),temp,
     1              offdiagind)

               temp = uniti*ekwave-(mtik/(delr*delr))*(1./(k+0.0)-1.0
     1              +(mode*mode*1.0)/(k*k*1.0))
               call wtmatrix(nrange_step,rowdiv2(k),colz(k),temp,
     1              offdiagind)

               temp = ((k+1.0)/(k+0.0))/delr
               call wtmatrix(nrange_step,rowdiv2(k),colr(k+1),temp,
     1              offdiagind)

               temp = polarization*mode*ONE/(k*delr)
               call wtmatrix(nrange_step,rowdiv2(k),colp(k),temp,
     1              offdiagind)

               temp = (mtik/(delr*delr))*( (1./(k+0.0))-2.0 )
               call wtmatrix(nrange_step,rowdiv2(k),colz(k+1),temp,
     1              offdiagind)

               temp = (mtik/(delr*delr))
               call wtmatrix(nrange_step,rowdiv2(k),colz(k+2),temp,
     1              offdiagind)
               Efields(rowdiv2(k)) = (0.d0,0.d0)
            else
c               print *,'n=',nrange_step, ' k=',k,' debug=',colr(k),
c     1              ' nz = ',nz,' nr= ',nr
               call wtmatrix(nrange_step,rowbcrz(k),colr(k),-nz,
     1              offdiagind)
               call wtmatrix(nrange_step,rowbcrz(k),colz(k),nr,
     1              offdiagind)
               Efields(rowbcrz(k)) = nz*Erhoinc(nrange_step,k,mode)-
     1              nr*Ezinc(nrange_step,k,mode)
            end if

         end if

 30   continue

c**** Treat upper boundary condition

      temp = (1.d0,0.d0)
      call wtmatrix(nrange_step,rowr(max_k+1),colr(max_k+1),temp,
     1     offdiagind)

      Efields(rowr(max_k+1)) = (0.d0,0.d0)


c*************************************************************************
c*****************************EPHI TERMS**********************************
c*************************************************************************

c**** Treat on-axis
c      temp = (1.d0,0.d0)
c      call wtmatrix(nrange_step,rowp(0),colp(0),temp,offdiagind)
c      Efields(rowp(0)) = (0.d0,0.d0)

c**** Diagonal term
      temp = 1.+tikdi*(ONE-ONE/(0.5+ZERO)-(mode*mode+ONE)/
     1     ((0.5+ZERO)**2))
      call wtmatrix(nrange_step,rowp(0),colp(0),temp,offdiagind)
c**** supersuperdiagonal portion
      temp = tikdi
      call wtmatrix(nrange_step,rowp(0),colp(2),temp,offdiagind)
c**** Superdiagonal portion
      temp = tikdi*(-2*ONE + ONE/(0.5+ZERO))
      call wtmatrix(nrange_step,rowp(0),colp(1),temp,offdiagind)
c***** Super-Super-superdiagonal portion (Erho term)
      temp = tikdi*polarization*2.0*ONE*mode/((0.5+ZERO)**2)
      call wtmatrix(nrange_step,rowp(0),colr(0),temp,offdiagind)


c**** Treat all other Erho fields
      do 40 k=1,max_k
```

```
        if (point_type(k).eq.0) then

c******* Calculate PML parameters ptil, sptil, qptil

            if (k.ge.pml_start) then
               ptil = k+(alpha+beta*uniti*eta/kwave)*
     1              ((k-pml_start)**3.0)/(3.0*pmlt*pmlt)
               sptil = 1.0+alpha*((k-pml_start+0.0)/pmlt)**2.0+beta*
     1              (uniti*eta/kwave)*((k-pml_start+0.0)/pmlt)**2.0
               qptil = -((2.0*(k-pml_start+0.0)/(pmlt*pmlt*1.0))*(alpha+
     1              beta*uniti*eta/kwave))/(sptil*sptil)
            else
               ptil = k+0.0
               sptil = (1.0,0.0)
               qptil = 0.0
            end if

c******* Use free space equations
c********** Diagonal term
            temp = 1.+tikdi*(-2.0/(sptil*sptil)-(1.0/sptil)*
     1           ((1.0/ptil)+qptil)-(mode*mode+1.0)/(ptil*ptil))
            call wtmatrix(nrange_step,rowp(k),colp(k),temp,offdiagind)
c********** Subdiagonal portion
            temp = tikdi*1.0/(sptil**2.0)
            call wtmatrix(nrange_step,rowp(k),colp(k-1),temp,offdiagind)
c********** Superdiagonal portion
            temp = (tikdi/sptil)*(1.0/sptil+(1.0/ptil+qptil))
            call wtmatrix(nrange_step,rowp(k),colp(k+1),temp,offdiagind)
c********** Sub-subdiagonal portion (Erho term)
            temp = tikdi*polarization*2.0*mode/(ptil*ptil)
            call wtmatrix(nrange_step,rowp(k),colr(k),temp,offdiagind)
         else
c********** Use boundary point condition equations.
c********** Enforce n x (Ei+psi) = 0, ie. tangential component is zero

            temp = (1.d0,0.d0)
            call wtmatrix(nrange_step,rowbcp(k),colp(k),temp,offdiagind)
            Efields(rowbcp(k)) = -Ephiinc(nrange_step,k,mode)

         end if
 40   continue

c**** Treat upper boundary condition

      temp = (1.d0,0.d0)
      call wtmatrix(nrange_step,rowp(max_k+1),colp(max_k+1),temp,
     1     offdiagind)
      Efields(rowp(max_k+1)) = (0.d0,0.d0)

c*********************************************************************
c*****************************EZ TERMS********************************
c*********************************************************************

c**** Treat on-axis
c       temp = (1.d0,0.d0)
c       call wtmatrix(nrange_step,rowz(0),colz(0),temp,offdiagind)
c       Efields(rowz(0)) = (0.d0,0.d0)

c**** Diagonal term
      temp = 1.+tikdi*(ONE-ONE/(0.5+ZERO)-(mode*mode+ZERO)/
     1     ((0.5+ZERO)**2))
      call wtmatrix(nrange_step,rowz(0),colz(0),temp,offdiagind)
c**** supersuperdiagonal portion
      temp = tikdi
      call wtmatrix(nrange_step,rowz(0),colz(2),temp,offdiagind)
c**** Superdiagonal portion
      temp = tikdi*(-2*ONE + ONE/(0.5+ZERO))
      call wtmatrix(nrange_step,rowz(0),colz(1),temp,offdiagind)


c**** Treat all other Erho fields
      do 50 k=1,max_k
         if (point_type(k).eq.0) then

c******* Calculate PML parameters ptil, sptil, qptil

            if (k.ge.pml_start) then
               ptil = k+(alpha+beta*uniti*eta/kwave)*
     1              ((k-pml_start)**3.0)/(3.0*pmlt*pmlt)
               sptil = 1.0+alpha*((k-pml_start+0.0)/pmlt)**2.0+beta*
     1              (uniti*eta/kwave)*((k-pml_start+0.0)/pmlt)**2.0
               qptil = -((2.0*(k-pml_start+0.0)/(pmlt*pmlt*1.0))*(alpha+
     1              beta*uniti*eta/kwave))/(sptil*sptil)
            else
               ptil = k+0.0
               sptil = (1.0,0.0)
               qptil = 0.0
            end if

c******* Use free space equations
c********** Diagonal term
            temp = 1.+tikdi*(-2.0/(sptil*sptil)-(1.0/sptil)*
     1           ((1.0/ptil)+qptil)-(mode*mode+ZERO)/(ptil*ptil))
            call wtmatrix(nrange_step,rowz(k),colz(k),temp,offdiagind)
c********** Subdiagonal portion
            temp = tikdi*1.0/(sptil**2.0)
            call wtmatrix(nrange_step,rowz(k),colz(k-1),temp,offdiagind)
c********** Superdiagonal portion
            temp = (tikdi/sptil)*(1.0/sptil+(1.0/ptil+qptil))
            call wtmatrix(nrange_step,rowz(k),colz(k+1),temp,offdiagind)

         else
c********** Use boundary point condition equations.
c********** Enforce n x (Ei+psi) = 0, ie. tangential component is zero
c********** Ez and Erho components
            nz = -(1.d0,0.d0)*cos(normsinf(point_type(k),1))
            nr = (1.d0,0.d0)*sin(normsinf(point_type(k),1))
            if (abs(nz).lt.1.e-6) nz = (0.d0,0.d0)
            if (abs(nr).lt.1.e-6) nr = (0.d0,0.d0)

            if (abs(nz).lt.1.e-6) then
c************* Place equations so that diagonal term is not zero.
c              print *,'n=',nrange_step, ' k=',k,' debug=',colr(k),
c     1              ' nz = ',nz,' nr= ',nr
               call wtmatrix(nrange_step,rowbcrz2(k),colz(k),nr,
     1              offdiagind)
               call wtmatrix(nrange_step,rowbcrz2(k),colr(k),-nz,
     1              offdiagind)
               Efields(rowbcrz2(k)) = nz*Erhoinc(nrange_step,k,mode)-
     1              nr*Ezinc(nrange_step,k,mode)

            else
c************* Enforce divergence free condition
               temp = uniti*ekwave-(mtik/(delr*delr))*(1./(k+0.0)-1.0
     1              +(mode*mode*1.0)/(k*k*1.0))
               call wtmatrix(nrange_step,rowdiv(k),colz(k),temp,
     1              offdiagind)

               temp = -1.0/delr
               call wtmatrix(nrange_step,rowdiv(k),colr(k),temp,
     1              offdiagind)

               temp = ((k+1.0)/(k+0.0))/delr
               call wtmatrix(nrange_step,rowdiv(k),colr(k+1),temp,
     1              offdiagind)

               temp = mode*polarization*ONE/(k*delr)
               call wtmatrix(nrange_step,rowdiv(k),colp(k),temp,
     1              offdiagind)

               temp = (mtik/(delr*delr))*( (1./(k+0.0))-2.0 )
               call wtmatrix(nrange_step,rowdiv(k),colz(k+1),temp,
     1              offdiagind)

               temp = (mtik/(delr*delr))
               call wtmatrix(nrange_step,rowdiv(k),colz(k+2),temp,
     1              offdiagind)
               Efields(rowdiv(k)) = (0.d0,0.d0)
            end if

         end if

 50   continue

c**** Treat upper boundary condition

      temp = (1.d0,0.d0)
      call wtmatrix(nrange_step,rowz(max_k+1),colz(max_k+1),temp,
     1     offdiagind)
      Efields(rowz(max_k+1)) = (0.d0,0.d0)

c**** finish up sparse matrix structure
      temp = (0.d0,0.d0)
      call wtmatrix(nrange_step,1,0,temp,offdiagind)

      if (offdiagind-1.gt.NNZ) then
         print *,'offdiagind = ',offdiagind
         print *,'NNZ = ', NNZ
         pause 'error, NNZ less than number of nonzeros'
      end if

      RETURN
      END


c*********************************************************************
c Matrix Row and Column function:
c*********************************************************************

      INTEGER FUNCTION rowr(k)

      implicit none
      integer max_k, obj_k
      common /matfcn/ max_k, obj_k
```

```
      integer k

c      rowr = 3*k+1
      rowr = k+1

      RETURN
      END

      INTEGER FUNCTION rowp(k)

      implicit none
      integer max_k, obj_k
      common /matfcn/ max_k, obj_k

      integer k

c      rowp = 3*k+2
      rowp = k+1+max_k+2

      RETURN
      END

      INTEGER FUNCTION rowz(k)

      implicit none
      integer max_k, obj_k
      common /matfcn/ max_k, obj_k

      integer k

c      rowz = 3*k+3
      rowz = k+1+2*max_k+4

      RETURN
      END

      INTEGER FUNCTION colr(k)

      implicit none
      integer max_k, obj_k
      common /matfcn/ max_k, obj_k

      integer k

c      colr = 3*k+1
      colr = k+1

      RETURN
      END

      INTEGER FUNCTION colp(k)

      implicit none
      integer max_k, obj_k
      common /matfcn/ max_k, obj_k

      integer k

c      colp = 3*k+2
      colp = k+1+max_k+2

      RETURN
      END

      INTEGER FUNCTION colz(k)

      implicit none
      integer max_k, obj_k
      common /matfcn/ max_k, obj_k

      integer k

c      colz = 3*k+3
      colz = k+1+2*max_k+4

      RETURN
      END

      INTEGER FUNCTION rowbcp(k)

      implicit none
      integer max_k, obj_k
      common /matfcn/ max_k, obj_k

      integer k

c      rowbcp = 3*k+2
      rowbcp = k+1+max_k+2

      RETURN
      END

      INTEGER FUNCTION rowbcrz(k)

      implicit none
      integer max_k, obj_k
      common /matfcn/ max_k, obj_k

      integer k

c      rowbcrz = 3*k+1
      rowbcrz = k+1

      RETURN
      END

      INTEGER FUNCTION rowdiv(k)

      implicit none
      integer max_k, obj_k
      common /matfcn/ max_k, obj_k

      integer k

c      rowdiv = 3*k+3
      rowdiv = k+1+2*max_k+4

      RETURN
      END

      INTEGER FUNCTION rowbcrz2(k)

      implicit none
      integer max_k, obj_k, rowdiv
      common /matfcn/ max_k, obj_k

      integer k

      rowbcrz2 = rowdiv(k)

      RETURN
      END

      INTEGER FUNCTION rowdiv2(k)

      implicit none
      integer max_k, obj_k, rowbcrz
      common /matfcn/ max_k, obj_k

      integer k

      rowdiv2 = rowbcrz(k)

      RETURN
      END

c**********************************************************************
c WTMATRIX writes the value temp to a given matrix structure that is
c    hard-coded in this subroutine. Currently, subroutine write to a
c    sparse matrix structure of the type specified the Numerical Recipes,
c    Section
c**********************************************************************

      SUBROUTINE wtmatrix(nrange_step,i,j,temp,offdiagind)

      implicit none
      include 'common.f'

      integer i,j,offdiagind,m,n,nrange_step
      complex*16 temp
c      integer cmi

      if (i.eq.-1) print *,'dummy statement to avoid inlining'

      if (i.eq.0.AND.j.eq.0) then

c******* Initialize Matrix
         do 10 m = 1,NMAX
            do 20 n = 1,NMAX
c               Awave(m,n) = (0.0,0.0)
c               Bwave(cmi(m,n)) = (0.0,0.0)
 20         continue
 10      continue

      else if (i.eq.1.AND.j.eq.0) then
c******* Complete sparse matrix structure
         ija(3*max_k+7) = offdiagind

      else if (i.gt.0.AND.j.gt.0) then

         if (i.eq.j) then
            sa(i) = temp
            ija(i) = offdiagind
```

```
         else
            sa(offdiagind) = temp
            ija(offdiagind) = j
            offdiagind = offdiagind + 1
         end if

c         Awave(i,j) = temp
c         Bwave(cmi(i,j)) = temp

      else
         pause 'bad index into matrix'
      end if

      RETURN
      END

c*******************************************************************************
c INCIDENT Wave functions defined for given incident angle and given
c    paraxial direction
c*******************************************************************************

      COMPLEX*16 FUNCTION Erhoinc(nrange_step,k,mode)

      implicit none
      include 'common.f'

      integer nrange_step,k,mode
      double precision rho, zz, kps, sintole
      parameter(sintole=-1d-6)
      real besselj

      rho = delr*k
      zz = delz*nrange_step
      kps = kwave*rho*sin(inc_ang)
c      print *,'kps=',kps,k,rho,sin(inc_ang)

      if (abs(sin(inc_ang)).lt.sintole.AND.mode.eq.1) then
c         Erhoinc = AMP*(Ehg*cos(inc_ang)+Evg*1.0)
         if (comp_forw_scat.eq.2) Erhoinc = Erhoinc*
     1        exp(2*uniti*kwave*zz)
      else
c         print *,'didnt make it in', nrange_step,k,mode
         if (mode.eq.0) then
            if (polarization.eq.HORZ) then
               Erhoinc = Ehg*cos(inc_ang)*exp(uniti*1.5*pi)*
     1              besselj(real(kps),1)
            else if (polarization.eq.VERT) then
               Erhoinc = 0.0
            else
               print *,'bad polarization'
               pause
            end if
         else
            if (polarization.eq.HORZ) then
               Erhoinc = cos(inc_ang)*(exp(uniti*(mode+1)*1.5*pi)*
     1              besselj(real(kps),mode+1)+exp(uniti*(mode-1)*1.5*pi)
     2              *besselj(real(kps),mode-1))
            else if (polarization.eq.VERT) then
               Erhoinc = exp(uniti*(mode-1)*1.5*pi)*besselj(real(kps),
     1              mode-1)-exp(uniti*(mode+1)*1.5*pi)*besselj(
     2              real(kps),mode+1)
            else
               print *,'bad polarization'
               pause
            end if
c            print *,'besselj(kps,2) = ',kps,real(kps),
c     1           besselj(real(kps),2),
c     1           besselj(real(kps),0)
         end if
         if (comp_forw_scat.eq.2) then
            Erhoinc = Erhoinc*exp(-uniti*kwave*zz*(cos(inc_ang)-1))
         else
            Erhoinc = Erhoinc*exp(-uniti*kwave*zz*(cos(inc_ang)+1))
         end if
      end if
c      print *,'Erhoinc=', nrange_step,rho,Erhoinc

      RETURN
      END

      COMPLEX*16 FUNCTION Ephiinc(nrange_step,k,mode)

      implicit none
      include 'common.f'

      integer nrange_step,k,mode
      double precision rho, zz, kps, sintole
      parameter(sintole=-1d-6)
      real besselj

      rho = delr*k
      zz = delz*nrange_step
      kps = kwave*rho*sin(inc_ang)

      if (abs(sin(inc_ang)).lt.sintole.AND.mode.eq.1) then
c         Ephiinc = AMP*(-Ehg*cos(inc_ang)+Evg*1.0)
         if (comp_forw_scat.eq.2) Ephiinc = Ephiinc*
     1        exp(2*uniti*kwave*zz)

      else
c         print *,'didnt make it in', nrange_step,k,mode
         if (mode.eq.0) then
            if (polarization.eq.HORZ) then
               Ephiinc = ZERO
            else if (polarization.eq.VERT) then
               Ephiinc = exp(uniti*1.5*pi)*besselj(real(kps),1)
            else
               print *,'bad polarization'
               pause
            end if
         else
            if (polarization.eq.HORZ) then
               Ephiinc = -cos(inc_ang)*(exp(uniti*(mode-1)*1.5*pi)*
     1              besselj(real(kps),mode-1)-exp(uniti*(mode+1)*1.5*pi)
     2              *besselj(real(kps),mode+1))
            else if (polarization.eq.VERT) then
               Ephiinc = exp(uniti*(mode+1)*1.5*pi)*besselj(real(kps),
     1              mode+1)+exp(uniti*(mode-1)*1.5*pi)*besselj(
     2              real(kps),mode-1)
            else
               print *,'bad polarization'
               pause
            end if
         end if
         if (comp_forw_scat.eq.2) then
            Ephiinc = Ephiinc*exp(-uniti*kwave*zz*(cos(inc_ang)-1))
         else
            Ephiinc = Ephiinc*exp(-uniti*kwave*zz*(cos(inc_ang)+1))
         end if
      end if
c      print *,'Ephiinc=', nrange_step,rho,Ephiinc

      RETURN
      END

      COMPLEX*16 FUNCTION Ezinc(nrange_step,k,mode)

      implicit none
      include 'common.f'

      integer nrange_step,k,mode
      double precision rho, zz, sintole, kps
      parameter(sintole=-1.d-6)
      real besselj

      rho = delr*k
      zz = delz*nrange_step
      kps = kwave*rho*sin(inc_ang)

      if (abs(sin(inc_ang)).lt.sintole.AND.mode.eq.1) then
c         Ezinc = ZERO
      else
c         print *,'didnt make it in', nrange_step,k,mode
         if (mode.eq.0) then
            if (polarization.eq.HORZ) then
               Ezinc = -sin(inc_ang)*besselj(real(kps),0)
            else if (polarization.eq.VERT) then
               Ezinc = ZERO
            else
               print *,'bad polarization'
               pause
            end if
         else
            if (polarization.eq.HORZ) then
               Ezinc = -2*ONE*sin(inc_ang)*exp(uniti*mode*1.5*pi)*
     1              besselj(real(kps),mode)
            else if (polarization.eq.VERT) then
               Ezinc = ZERO
            else
               print *,'bad polarization'
               pause
            end if
         end if
         if (comp_forw_scat.eq.2) then
            Ezinc = Ezinc*exp(-uniti*kwave*zz*(cos(inc_ang)-1))
         else
            Ezinc = Ezinc*exp(-uniti*kwave*zz*(cos(inc_ang)+1))
         end if
      end if
c      print *,'Ezinc=', mode,nrange_step,rho,Ezinc

      RETURN
      END
```

```
c**************************************************************************
c CMI: returns single index for represent 2 indices in 2 dim structure
c**************************************************************************

      INTEGER FUNCTION cmi(i,j)

      implicit none
      include 'common.f'

      integer i,j

      cmi = (i-1)*(3*max_k+6)+j

      RETURN
      END

c**************************************************************************
c SOLVE_TRI_LIN_SYS takes tridiagonal linear system stored in special
c    matrix data structure A, and gives b' = inv(A)*b. Note that the
c    solution vector is stored in the same array as the RHS vector b.
c**************************************************************************

      SUBROUTINE solve_tri_lin_sys(Adiag, Asup, Asub, b, N)

      implicit none
      include 'common.f'

      double complex Adiag(1:MAX_R_CELLS), Asup(1:(MAX_R_CELLS-1)),
     1     Asub(2:MAX_R_CELLS), b(1:MAX_R_CELLS), pivot
      integer N, i

c***  Note the definition of the subdiagonal defined with indices from
c***  2...N

c***  Routine below works by using a simple Gaussian elmination
c***  technique for the specific case of a tridiagonal system.

c***  Forward Elmination.
      do 10 i = 2, N
         pivot = -Asub(i)/Adiag(i-1)
         Adiag(i) = Adiag(i) + pivot*Asup(i-1)
         b(i) = b(i) + pivot*b(i-1)
 10   continue

c**** Backward Substitution
      b(N) = b(N)/Adiag(N)
      do 20 i = N-1,1,-1
         b(i) = (b(i)-Asup(i)*b(i+1))/Adiag(i)
 20   continue

      RETURN
      END

c**************************************************************************
c STORE_RCS_COMP saves the efield components on the Huygens' surface as
c    well as those efields needed to calculate the hfield components on
c    the Huygens' surface.
c**************************************************************************

      SUBROUTINE store_rcs_comp(nrange_step, mode)

      implicit none
      include 'common.f'

      integer nrange_step, mode, i, k, phishift, zshift
      integer colr,colp,colz

      phishift=max_k+2
      zshift=2*max_k+4

c**** the RCS points compromise a rectangle box

      if (nrange_step.eq.x1) then
         do 10 i=y1,y2
            k=i
            rcserho(mode,k) = Efields(colr(i-1))
            rcsephi(mode,k) = Efields(colp(i-1))
            rcsez(mode,k) = Efields(colz(i-1))
 10      continue
      else if (nrange_step.eq.x2) then
         k=y2-y1+x2-x1
         do 20 i=y2,y1,-1
            k=k+1
            rcserho(mode,k) = Efields(colr(i-1))
            rcsephi(mode,k) = Efields(colp(i-1))
            rcsez(mode,k) = Efields(colz(i-1))
 20      continue
      else if (nrange_step.gt.x1.AND.nrange_step.lt.x2) then
         k=y2-y1+nrange_step-x1+1
         rcserho(mode,k) = Efields(colr(y2-1))
         rcsephi(mode,k) = Efields(colp(y2-1))
         rcsez(mode,k) = Efields(colz(y2-1))
      end if

c**** now deal with the points to the left and right that enable H calc.

      if (nrange_step.eq.(x1-1)) then
         k=0
         do 30 i=y1,y2
            k=k+1
            rcsEphi_rl(k) = Efields(colp(i-1))
            rcsErho_rl(k) = Efields(colr(i-1))
 30      continue
      else if (nrange_step.eq.(x1+1)) then
         k=y2
         do 40 i=y1,y2
            k=k+1
            rcsEphi_rl(k) = Efields(colp(i-1))
            rcsErho_rl(k) = Efields(colr(i-1))
 40      continue
      else if (nrange_step.eq.(x2-1)) then
         k=2*y2
         do 50 i=y1,y2
            k=k+1
            rcsEphi_rl(k) = Efields(colp(i-1))
            rcsErho_rl(k) = Efields(colr(i-1))
 50      continue
      else if (nrange_step.eq.(x2+1)) then
         k=3*y2
         do 60 i=y1,y2
            k=k+1
            rcsEphi_rl(k) = Efields(colp(i-1))
            rcsErho_rl(k) = Efields(colr(i-1))
 60      continue
      end if

c**** now deal with the points on the top and bottom that enable H calc.

      if (nrange_step.ge.x1.AND.nrange_step.le.x2) then
         k = 2*(nrange_step-x1+1)-1
         rcsEphi_ab(k) = Efields(colp(y2+1-1))
         rcsEz_ab(k) = Efields(colz(y2+1-1))
         rcsEphi_ab(k+1) = Efields(colp(y2-1-1))
         rcsEz_ab(k+1) = Efields(colz(y2-1-1))
      end if

      RETURN
      END


c**************************************************************************
c CALC_H_RCS_COMP calculates the H fields based on E fields calculated
c    using the parabolic wave equation.
c**************************************************************************

      SUBROUTINE calc_h_rcs_comp(mode)

      implicit none
      include 'common.f'

      integer i, k
      real forwback
      integer mode

      write(6,*) 'Calculating H fields from E fields...'

c**** Use differential form of Maxwell's modal equations to calculate H
c**** Note, H fields calculated here are actually eta H. Also, note that
c**** kwave*eta = omega mu (OR abb. as kn=wu).

c**** In the following calculations, the index 'i' always refers to the
c**** physical locations (i.e. i=0, implies on axis).

c**** Forward and backward waves have exp(+ikz) and exp(-ikz) terms, so
c**** equations below change in sign for certain terms (where d/dz were
c**** computed)

      if (comp_forw_scat.eq.1) then
         forwback = 1.0
      else
         forwback = -1.0
      end if

c**************************************************************************
c**************************************************************************
c**** calculate all the Hrho fields
c**** -iwu Hrho(k) = (d/dz)ephi(k) + polarization*(m/rho)ez(k)

c**** by symmetry
      rcsHrho(mode,1) = (0.0,0.0)
      rcsHrho(mode,2*y2+x2-x1-1) = (0.0,0.0)

c**** left hand side
      k=1
```

```
      do 10 i = 1,y2-1
         k=i+1
         rcsHrho(mode,k) = ((0.5/delz)*(rcsEphi_rl(i+y2+1)-
     1        rcsEphi_rl(i+1))+polarization*mode*ONE/((i-0.0)*delr)*
     2        rcsEz(mode,k)+forwback*uniti*kwave*rcsEphi(mode,k))*
     3        (uniti/kwave)
 10   continue

c**** right hand side
      k=y2-y1+x2-x1
      do 20 i = y2-1,y1,-1
         k=k+1
         rcsHrho(mode,k) = ((0.5/delz)*(rcsEphi_rl(i+3*y2+1)-
     1        rcsEphi_rl(i+2*y2+1))+polarization*mode*ONE
     2        /((i-0.0)*delr)*rcsEz(mode,k)+forwback*uniti*kwave*
     3        rcsEphi(mode,k))*(uniti/kwave)
 20   continue

c**** middle points
      k=y2
      do 30 i=x1+1,x2-1
         k=k+1
         rcsHrho(mode,k) = ((0.5/delz)*(rcsEphi(mode,k+1)-
     1        rcsEphi(mode,k-1))+polarization*mode*ONE/((y2-1.0)*delr)*
     2        rcsEz(mode,k)+forwback*uniti*kwave*rcsEphi(mode,k))*
     3        (uniti/kwave)
 30   continue

c***********************************************************************
c***********************************************************************
c**** calculate all the Hphi fields
c**** -iwu hphi(k) = (d/drho)ez(k) - (d/dz)erho(k)

c**** by symmetry
      rcsHphi(mode,1) = (0.0,0.0)
      rcsHphi(mode,2*y2+x2-x1-1) = (0.0,0.0)

c**** Left hand side points
      k=1
      do 40 i=1,y2-2
         k=k+1
c******* Note: now, k=i+1
         rcsHphi(mode,k) = ((0.5/delr)*(rcsEz(mode,k+1)-
     1        rcsEz(mode,k-1))-(0.5/delz)*(rcsErho_rl(i+y2+1)-
     2        rcsErho_rl(i+1))-forwback*uniti*kwave*rcsErho(mode,k))*
     3        (uniti/kwave)
 40   continue

c**** Top left corner
      k=k+1
      rcsHphi(mode,k) = ((0.5/delr)*(rcsEz_ab(1)-rcsEz_ab(2))-
     1     (0.5/delz)*(rcsErho_rl(2*y2)-rcsErho_rl(y2))-forwback*uniti*
     2     kwave*rcsErho(mode,k))*(uniti/kwave)

c**** Top Right corner
      k=y2+x2-x1
      rcsHphi(mode,k) = ((0.5/delr)*(rcsEz_ab(2*(x2-x1)+1)-
     1     rcsEz(mode,k+1))-(0.5/delz)*(rcsErho_rl(4*y2)-
     2     rcsErho_rl(3*y2))-forwback*uniti*kwave*rcsErho(mode,k))*
     3     (uniti/kwave)

c**** Right hand side points
      k=y2+x2-x1
      do 50 i=y2-2,1,-1
         k=k+1
         rcsHphi(mode,k) = ((0.5/delr)*(rcsEz(mode,k-1)-
     1        rcsEz(mode,k+1))-(0.5/delz)*(rcsErho_rl(i+3*y2+1)-
     2        rcsErho_rl(i+2*y2+1))-forwback*uniti*kwave*
     3        rcsErho(mode,k))*(uniti/kwave)
 50   continue

c**** Top middle points.

      k=y2
      do 60 i=x1+1,x2-1
         k=k+1
         rcsEphi(mode,k) = ((0.5/delr)*(rcsEz_ab(2*(i-x1)+1)-
     1        rcsEz_ab(2*(i-x1)+2))-(0.5/delz)*(rcsErho(mode,k+1)-
     2        rcsErho(mode,k-1))-forwback*uniti*kwave*rcsErho(mode,k))*
     3        (uniti/kwave)
 60   continue

c***********************************************************************
c***********************************************************************
c**** calculate all Hz points
c**** -iwu hz(k) = (-1/rho)(d/drho)(rho ephi(k)) - pol (m/rho) erho(k)

c**** by symmetry
      rcsHz(mode,1) = (0.0,0.0)
      rcsHz(mode,2*y2+x2-x1-1) = (0.0,0.0)

c**** left hand side
      k=1
      do 70 i = 1,y2-2
         k=k+1
         rcsHz(mode,k) = ((-0.5/((i-0.)*delr))*((i+1.)*
     1        rcsEphi(mode,k+1)-(i-1.)*rcsEphi(mode,k-1))-polarization*
     2        (ONE*mode/(delr*(i-0.)))*rcsErho(mode,k))*(uniti/kwave)
 70   continue

c**** Top left corner
      k=y2
      rcsHz(mode,k) = ((-0.5/((y2-1.)*delr))*((y2+0.)*rcsEphi_ab(1)-
     1     (y2-2.)*rcsEphi_ab(2))-polarization*(ONE*mode/(delr*
     2     (y2-1.)))*rcsErho(mode,k))*(uniti/kwave)

c**** Top Right corner
      k=y2+x2-x1
      rcsHz(mode,k) = ((-0.5/((y2-1.)*delr))*((y2+0.)*
     1     rcsEphi_ab(2*(x2-x1)+1)-(y2-2.)*rcsEphi(mode,k+1))-
     2     polarization*(ONE*mode/(delr*(y2-1.)))*rcsErho(mode,k))*
     3     (uniti/kwave)

c**** right hand side
      k=y2+(x2-x1)
      do 80 i = y2-2,1,-1
         k=k+1
         rcsHz(mode,k) = ((-0.5/((i-0.)*delr))*((i+1.)*rcsEphi(mode,k-1)
     1        -(i-1.)*rcsEphi(mode,k+1))-polarization*(ONE*mode/(delr*
     2        (i-0.)))*rcsErho(mode,k))*(uniti/kwave)
 80   continue

c**** Top middle points
      k=y2
      do 90 i=x1+1,x2-1
         k=k+1
         rcsHz(mode,k) = ((-0.5/((y2-1.)*delr))*((y2+0.)*
     1        rcsEphi_ab(2*(i-x1)+1)-(y2-2.)*rcsEphi_ab(2*(i-x1)+2))-
     2        polarization*(ONE*mode/(delr*(y2-1.)))*rcsErho(mode,k))*
     3        (uniti/kwave)
 90   continue


      RETURN
      END

c***********************************************************************
c READ_IN_RCS_SURFACE_DATA reads in the field values over a Huygens'
c    surface that was previously calculated that can be used to
c    calculate bistatic RCS values.
c***********************************************************************

      SUBROUTINE read_in_rcs_surface_data

      implicit none
      include 'common.f'

      integer k, mode
      double precision tempr, tempi

      mode = 1
c**** Write out RCS surface data
      open(unit=10,file='erfreq.dat',status='unknown',form='formatted')
      open(unit=11,file='hrfreq.dat',status='unknown',form='formatted')
      do 60 k=1,2*(y2-y1)+(x2-x1)+1
         read(10,*) tempr, tempi
         rcserho(mode,k) = tempr+uniti*tempi
         read(11,*) tempr, tempi
         rcshrho(mode,k) = tempr+uniti*tempi
 60   continue
      close(unit=10)
      close(unit=11)

      open(unit=10,file='epfreq.dat',status='unknown',form='formatted')
      open(unit=11,file='hpfreq.dat',status='unknown',form='formatted')
      do 70 k=1,2*(y2-y1)+(x2-x1)+1
         read(10,*) tempr, tempi
         rcsephi(mode,k) = tempr+uniti*tempi
         read(11,*) tempr, tempi
         rcshphi(mode,k) = tempr+uniti*tempi
 70   continue
      close(unit=10)
      close(unit=11)

      open(unit=10,file='ezfreq.dat',status='unknown',form='formatted')
      open(unit=11,file='hzfreq.dat',status='unknown',form='formatted')
      do 80 k=1,2*(y2-y1)+(x2-x1)+1
         read(10,*) tempr, tempi
         rcsez(mode,k) = tempr+uniti*tempi
         read(11,*) tempr, tempi
         rcshz(mode,k) = tempr+uniti*tempi
 80   continue
      close(unit=10)
      close(unit=11)
```

```
      RETURN
      END

c*************************************************************************
c WRITE_OUT_RCS_SURFACE_DATA writes out the field values over the
c    Huygens' surface that can be later used to calculate bistatic
c    RCS values.
c*************************************************************************

      SUBROUTINE write_out_rcs_surface_data

      implicit none
      include 'common.f'

      integer k, mode

      write(6,*) 'Writing out rcs surface data...'

c**** Write out RCS surface data
      open(unit=10,file='erfreq.dat',status='unknown',form='formatted')
      open(unit=11,file='hrfreq.dat',status='unknown',form='formatted')
      do 55 mode = smode, emode
      do 60 k=1,2*(y2-y1)+(x2-x1)+1
         write(10,99) real(rcserho(mode,k)), imag(rcserho(mode,k))
         write(11,99) real(rcshrho(mode,k)), imag(rcshrho(mode,k))
 60   continue
 55   continue
      close(unit=10)
      close(unit=11)

      open(unit=10,file='epfreq.dat',status='unknown',form='formatted')
      open(unit=11,file='hpfreq.dat',status='unknown',form='formatted')
      do 65 mode = smode, emode
      do 70 k=1,2*(y2-y1)+(x2-x1)+1
         write(10,99) real(rcsephi(mode,k)), imag(rcsephi(mode,k))
         write(11,99) real(rcshphi(mode,k)), imag(rcshphi(mode,k))
 70   continue
 65   continue
      close(unit=10)
      close(unit=11)

      open(unit=10,file='ezfreq.dat',status='unknown',form='formatted')
      open(unit=11,file='hzfreq.dat',status='unknown',form='formatted')
      do 75 mode = smode, emode
      do 80 k=1,2*(y2-y1)+(x2-x1)+1
         write(10,99) real(rcsez(mode,k)), imag(rcsez(mode,k))
         write(11,99) real(rcshz(mode,k)), imag(rcshz(mode,k))
 80   continue
 75   continue
      close(unit=10)
      close(unit=11)

 99   format(2X,E21.13,5X,E21.13)

      RETURN
      END

c*************************************************************************
c CALC_RCS calculates the RCS based on the near field data computed
c    by the PWE method.
c*************************************************************************

      SUBROUTINE calc_rcs

      implicit none
      include 'common.f'

      integer k, PDIV
      double complex phase
      double precision kps_tole

      complex*16 uErho,uEphi,uEz,uHrho,uHphi,uHz
      complex*16 vErho,vEphi,vEz,vHrho,vHphi,vHz
      complex*16 Escat_theta_A, Escat_phi_A, Escat_theta_B,
     1     Escat_phi_B, Escat_theta_C, Escat_phi_C
      complex*16 I1, I3, I5, c1, c2, c3, c4, c5, RCSc, At, Ap, A

      real cz, rho, besselj, phase_z, kps
      double precision RCS, sint, cost, sinp, cosp, theta, phi,
     1     sinmp, cosmp
      integer mode

      parameter(PDIV=20,kps_tole=1.0e-9)

      write(6,*) 'Calculating RCS...'

c**** Add in phase components.

c**** Right hand side
      if (comp_forw_scat.eq.1) then
         phase = exp(uniti*kwave*x1*delz)
      else
         phase = exp(-uniti*kwave*x1*delz)
      end if
      do 11 mode = smode,emode
      do 10 k=1,y2
         rcsErho(mode,k) = phase*rcsErho(mode,k)
         rcsEphi(mode,k) = phase*rcsEphi(mode,k)
         rcsEz(mode,k) = phase*rcsEz(mode,k)
         rcsHrho(mode,k) = phase*rcsHrho(mode,k)
         rcsHphi(mode,k) = phase*rcsHphi(mode,k)
         rcsHz(mode,k) = phase*rcsHz(mode,k)
 10   continue
 11   continue

c**** Left hand side
      if (comp_forw_scat.eq.1) then
         phase = exp(uniti*kwave*x2*delz)
      else
         phase = exp(-uniti*kwave*x2*delz)
      end if
      do 21 mode = smode,emode
      do 20 k=y2+x2-x1,2*y2+x2-x1-1
         rcsErho(mode,k) = phase*rcsErho(mode,k)
         rcsEphi(mode,k) = phase*rcsEphi(mode,k)
         rcsEz(mode,k) = phase*rcsEz(mode,k)
         rcsHrho(mode,k) = phase*rcsHrho(mode,k)
         rcsHphi(mode,k) = phase*rcsHphi(mode,k)
         rcsHz(mode,k) = phase*rcsHz(mode,k)
 20   continue
 21   continue

c**** Middle points
      do 31 mode= smode,emode
      do 30 k=y2+1,y2+x2-x1-1
         if (comp_forw_scat.eq.1) then
            phase = exp(uniti*kwave*(k-y2+x1)*delz)
         else
            phase = exp(-uniti*kwave*(k-y2+x1)*delz)
         end if
         rcsErho(mode,k) = phase*rcsErho(mode,k)
         rcsEphi(mode,k) = phase*rcsEphi(mode,k)
         rcsEz(mode,k) = phase*rcsEz(mode,k)
         rcsHrho(mode,k) = phase*rcsHrho(mode,k)
         rcsHphi(mode,k) = phase*rcsHphi(mode,k)
         rcsHz(mode,k) = phase*rcsHz(mode,k)
 30   continue
 31   continue

c**** Integrate over Huygens' surface

      do 40 phi = low_phi,high_phi,dphi
         sinp = sin(pi*(phi/180.))
         cosp = cos(pi*(phi/180.))

         do 50 theta = low_theta,high_theta,dtheta
            sint = sin(pi*(theta/180.))
            cost = cos(pi*(theta/180.))

            Escat_theta_A = 0.0
            Escat_phi_A = 0.0
            Escat_theta_B = 0.0
            Escat_phi_B = 0.0
            Escat_theta_C = 0.0
            Escat_phi_C = 0.0

            do 55 mode=smode,emode
               sinmp = sin(mode*pi*(phi/180.))
               cosmp = cos(mode*pi*(phi/180.))

C********** Three different integrals to evaluate
               c3 = 2*pi*exp(uniti*mode*1.5*pi)
               c4 = 2*pi*exp(uniti*(mode+1)*1.5*pi)

C*****************Integral A: z1 --> z2 -center integral at r0
               rho = (y2-1.0)*delr
               kps = kwave * rho * sint

               if (abs(kps).lt.kps_tole) then
                  if (mode.eq.1) then
                     I1=0.0
                     I3=pi
                     I5=pi
                  else
                     I1=0.0
                     I3=0.0
                     I5=0.0
                  end if
                  if (mode.eq.0) then
                     I1=2*pi
                  end if
               else
                  c2 = 2.0*pi*uniti*mode/kps
```

```
                c5 = c2*exp(uniti*mode*1.5*pi)
                I1 = c3*besselj(kps,int(mode))
                I3 = c4*besselj(kps,int(mode)+1)+c5*besselj(kps,
     1               int(mode))
                I5 = c5*besselj(kps,int(mode))
             end if

             do 60 k = y2, y2+x2-x1
                if (polarization.eq.HORZ) then
                   uEphi = rcsEphi(mode,k)*sinmp
                   vEphi = rcsEphi(mode,k)*cosmp
                   uErho = rcsErho(mode,k)*cosmp
                   vErho = -rcsErho(mode,k)*sinmp
                   uEz = rcsEz(mode,k)*cosmp
                   vEz = -rcsEz(mode,k)*sinmp
                   uHphi = rcsHphi(mode,k)*cosmp
                   vHphi = -rcsHphi(mode,k)*sinmp
                   uHrho = rcsHrho(mode,k)*sinmp
                   vHrho = rcsHrho(mode,k)*cosmp
                   uHz = rcsHz(mode,k)*sinmp
                   vHz = rcsHz(mode,k)*cosmp
                else
                   uEphi = rcsEphi(mode,k)*cosmp
                   vEphi = -rcsEphi(mode,k)*sinmp
                   uErho = rcsErho(mode,k)*sinmp
                   vErho = rcsErho(mode,k)*cosmp
                   uEz = rcsEz(mode,k)*sinmp
                   vEz = rcsEz(mode,k)*cosmp
                   uHphi = rcsHphi(mode,k)*sinmp
                   vHphi = rcsHphi(mode,k)*cosmp
                   uHrho = rcsHrho(mode,k)*cosmp
                   vHrho = -rcsHrho(mode,k)*sinmp
                   uHz = rcsHz(mode,k)*cosmp
                   vHz = -rcsHz(mode,k)*sinmp
                end if

                do 65 phase_z = 0,PDIV-1
                   cz = (x1+k-y2)*delz+phase_z*delz/PDIV
                   c1 = exp(-uniti*kwave*cz*cost)

                   Escat_theta_A = (delz/PDIV)*rho*c1*(-sint*uHphi*I1
     1                 +uEz*I3+cost*vHz*I5)+Escat_theta_A

                   Escat_phi_A = (delz/PDIV)*rho*c1*(-uHz*I3-sint*
     1                 uEphi*I1+cost*vEz*I5)+Escat_phi_A
 65             continue
 60          continue

C*************************************************************

C********** Integral B: 0 --> r0 -left integral at z1

             cz = x1*delz
             c1 = exp(-uniti*kwave*cz*cost)

             do 70 k = 1,y2

                if (polarization.eq.HORZ) then
                   uEphi = rcsEphi(mode,k)*sinmp
                   vEphi = rcsEphi(mode,k)*cosmp
                   uErho = rcsErho(mode,k)*cosmp
                   vErho = -rcsErho(mode,k)*sinmp
                   uEz = rcsEz(mode,k)*cosmp
                   vEz = -rcsEz(mode,k)*sinmp
                   uHphi = rcsHphi(mode,k)*cosmp
                   vHphi = -rcsHphi(mode,k)*sinmp
                   uHrho = rcsHrho(mode,k)*sinmp
                   vHrho = rcsHrho(mode,k)*cosmp
                   uHz = rcsHz(mode,k)*sinmp
                   vHz = rcsHz(mode,k)*cosmp
                else
                   uEphi = rcsEphi(mode,k)*cosmp
                   vEphi = -rcsEphi(mode,k)*sinmp
                   uErho = rcsErho(mode,k)*sinmp
                   vErho = rcsErho(mode,k)*cosmp
                   uEz = rcsEz(mode,k)*sinmp
                   vEz = rcsEz(mode,k)*cosmp
                   uHphi = rcsHphi(mode,k)*sinmp
                   vHphi = rcsHphi(mode,k)*cosmp
                   uHrho = rcsHrho(mode,k)*cosmp
                   vHrho = -rcsHrho(mode,k)*sinmp
                   uHz = rcsHz(mode,k)*cosmp
                   vHz = -rcsHz(mode,k)*sinmp
                end if

                rho = (k-1)*delr
                kps = kwave * rho * sint

                if (abs(kps).lt.kps_tole) then
                   if (mode.eq.1) then
                      I1=0.0
                      I3=pi
                      I5=pi
                   else
                      I1=0.0
                      I3=0.0
                      I5=0.0
                   end if
                   if (mode.eq.0) then
                      I1=2*pi
                   end if
                else
                   c2 = 2.0*pi*uniti*mode/kps
                   c5 = c2*exp(uniti*mode*1.5*pi)
                   I1 = c3*besselj(kps,int(mode))
                   I3 = c4*besselj(kps,int(mode)+1)+c5*besselj(kps,
     1                 int(mode))
                   I5 = c5*besselj(kps,int(mode))
                end if

                Escat_theta_B = -delr*rho*c1*(-cost*uHphi*I3-uErho*I3-
     1               cost*vHrho*I5+vEphi*I5)+Escat_theta_B

                Escat_phi_B = -delr*rho*c1*((uHrho-cost*uEphi)*I3+
     1               (-vHphi-cost*vErho)*I5)+Escat_phi_B

 70          continue
C*************************************************************

C********** Integral C: 0 --> r0 -right integral at z2

             cz = x2*delz
             c1 = exp(-uniti*kwave*cz*cost)

             do 80 k=y2+x2-x1,2*y2+x2-x1-1

                if (polarization.eq.HORZ) then
                   uEphi = rcsEphi(mode,k)*sinmp
                   vEphi = rcsEphi(mode,k)*cosmp
                   uErho = rcsErho(mode,k)*cosmp
                   vErho = -rcsErho(mode,k)*sinmp
                   uEz = rcsEz(mode,k)*cosmp
                   vEz = -rcsEz(mode,k)*sinmp
                   uHphi = rcsHphi(mode,k)*cosmp
                   vHphi = -rcsHphi(mode,k)*sinmp
                   uHrho = rcsHrho(mode,k)*sinmp
                   vHrho = rcsHrho(mode,k)*cosmp
                   uHz = rcsHz(mode,k)*sinmp
                   vHz = rcsHz(mode,k)*cosmp
                else
                   uEphi = rcsEphi(mode,k)*cosmp
                   vEphi = -rcsEphi(mode,k)*sinmp
                   uErho = rcsErho(mode,k)*sinmp
                   vErho = rcsErho(mode,k)*cosmp
                   uEz = rcsEz(mode,k)*sinmp
                   vEz = rcsEz(mode,k)*cosmp
                   uHphi = rcsHphi(mode,k)*sinmp
                   vHphi = rcsHphi(mode,k)*cosmp
                   uHrho = rcsHrho(mode,k)*cosmp
                   vHrho = -rcsHrho(mode,k)*sinmp
                   uHz = rcsHz(mode,k)*cosmp
                   vHz = -rcsHz(mode,k)*sinmp
                end if

                rho = (2*y2+x2-x1-1-k)*delr
                kps = kwave * rho * sint

                if (abs(kps).lt.kps_tole) then
                   if (mode.eq.1) then
                      I1=0.0
                      I3=pi
                      I5=pi
                   else
                      I1=0.0
                      I3=0.0
                      I5=0.0
                   end if
                   if (mode.eq.0) then
                      I1=2*pi
                   end if
                else
                   c2 = 2.0*pi*uniti*mode/kps
                   c5 = c2*exp(uniti*mode*1.5*pi)
                   I1 = c3*besselj(kps,int(mode))
                   I3 = c4*besselj(kps,int(mode)+1)+c5*besselj(kps,
     1                 int(mode))
                   I5 = c5*besselj(kps,int(mode))
                end if

                Escat_theta_C = delr*rho*c1*(-cost*uHphi*I3-uErho*I3-
     1               cost*vHrho*I5+vEphi*I5)+Escat_theta_C

                Escat_phi_C = delr*rho*c1*((uHrho-cost*uEphi)*I3+
     3               (-vHphi-cost*vErho)*I5)+Escat_phi_C
```

```
 80            continue

c******* MODE loop******
 55         continue
c************************

           At = Escat_theta_A + Escat_theta_B + Escat_theta_C
           Ap = Escat_phi_A + Escat_phi_B + Escat_phi_C

           if (polarization.eq.HORZ) then
              A = At*(cost*cost*cosp+sint*sint)-Ap*cost*sinp
           else
              A = At*cost*sinp+Ap*cosp
           end if

           RCSc = ((kwave**2)*(A**2))/(4.0*pi*(AMP**2))
           RCS = ((kwave**2)*((abs(A))**2))/(4.0*pi*(AMP**2))

c           write(6,*) phi, theta, 10*LOG10(RCS)
           write(9,99) kwave,phi,theta,10*LOG10(RCS),abs(RCSc),
     1          atan2(imag(RCSc),real(RCSc))


 50      continue
 40   continue

 99   format(F12.7,1X,F6.1,1X,F6.1,1X,F10.5,1X,E19.12,1X,F17.10)

      RETURN
      END

c*********************************************************************
c SETUP_GEOMETRY setups all the parameters needed to run the simulation
c    including the target meshing algorithm which determines the surface
c    normals.
c*********************************************************************

      SUBROUTINE setup_geometry

      implicit none
      include 'common.f'

      integer index, round, xspacing, count

      real max_x_node, max_y_node, min_x_node, min_y_node,
     1     slope, xnodes(1:MAX_NODES), ynodes(1:MAX_NODES), sgn,
     1     length, height, inttol, sgnx, sgny, slope2, cx, cy, err

      parameter(inttol=1e-5)

      parameter(xspacing = 10)
c      parameter(yspacing = 35)

      write(6,*) 'Setting up geometry...'

      write(6,'(''*Enter yspacing: '', $)')
      read(5,*) yspacing

      write(6,'(''*Enter PML_DEPTH: '', $)')
      read(5,*) PML_DEPTH

c      write(6,'(''*Enter alpha: '', $)')
c      read(5,*) alpha

c      write(6,'(''*Enter beta: '', $)')
c      read(5,*) beta

      yspacing = yspacing+PML_DEPTH
      errorcount = 0

c**** Read geometry file in.

      open(unit=10,file=geomfile,status='unknown',form='formatted')

      read(10,*) delz
      read(10,*) delr

      if (PML_DEPTH.gt.0) then
         alpha = 0.075/(PML_DEPTH*delr)
         beta = alpha
      else
         alpha = 0.0
         beta = 0.0
      end if

c      delz = delta
c      delr = delta

      read(10,*) total_nodes
      if (total_nodes.gt.MAX_NODES) then
         errorcount = errorcount+1
         errors(errorcount) = NODE_ERROR
         call memory_check
      end if

      do 10 index=1,total_nodes
         read(10,*) xnodes(index), ynodes(index)
c         print *,xnodes(index), ynodes(index)
 10   continue
      close(unit=10)

C**** Scale, position, and round object

      max_x_node = delz*round(real(xnodes(1)/delz))
      max_y_node = ynodes(1)
      min_x_node = delz*round(real(xnodes(1)/delz))
      min_y_node = ynodes(1)

      do 20 index=1,total_nodes
         xnodes(index) = delz*round(real(xnodes(index)/delz))
         if (xnodes(index).gt.max_x_node) max_x_node=xnodes(index)
         if (xnodes(index).lt.min_x_node) min_x_node=xnodes(index)
         ynodes(index) = ynodes(index)
         if (ynodes(index).gt.max_y_node) max_y_node=ynodes(index)
         if (ynodes(index).lt.min_y_node) min_y_node=ynodes(index)
 20   continue

      print *,'max_x_node=',max_x_node
      print *,'min_x_node=',min_x_node
      length = int((max_x_node - min_x_node)/delz)+1
      height = int((max_y_node - min_y_node)/delr)
      heightkwave = height*delr*kwave
      obj_k = height+3
      print *,'length = ',length
      print *,'height = ',height
      max_range_step = round(2.0*xspacing + length)
      max_k = round(yspacing + height)

      if (max_range_step.gt.MAX_Z_CELLS) then
         errorcount = errorcount + 1
         errors(errorcount) = MAX_Z_ERROR
      end if

      if ((3*max_k+6).gt.NMAX) then
         errorcount = errorcount+1
         errors(errorcount) = MAX_R_ERROR
      end if

      do 30 index=1,total_nodes
         xnodes(index) = xnodes(index) - min_x_node + xspacing*delz
         ynodes(index) = ynodes(index) - min_y_node + delr
 30   continue

c**** Estimate total number of staircase nodes needed.

      count = 0
      do 40 index=1,total_nodes-1
         if (abs(xnodes(index+1)-xnodes(index)).gt.inttol) then
            count = count + abs(xnodes(index+1)-xnodes(index))/delz
         else
            count = count + abs(ynodes(index+1)-ynodes(index))/delr
         end if
 40   continue
      count = count+1
c      print *,'count = ',count

      if (count.gt.MAX_STAIR_NODES) then
         stair_node_count = count
         errorcount = errorcount+1
         errors(errorcount) = MAX_STAIR_ERROR
      end if

c**** define RCS box for calculating far-fields, only need top and
c**** right side (forward scattering case)
c      x1=xspacing-3
      x1 = 2
      y1=1
c      x2=max_range_step+3-xspacing
      x2=max_range_step-2
      y2=max_k-yspacing+4
      if ((2*(y2-y1)+(x2-x1)+1).gt.MAX_RCS_NODES) then
         errorcount = errorcount+1
         errors(errorcount) = MAX_RCS_ERROR
      end if
      call memory_check

c**** Begin geometry mesher routine.

      count = 0
      do 50 index=1,total_nodes-1
c         print *,abs(xnodes(index+1)-xnodes(index)),inttol
         print *,xnodes(index)/delz,ynodes(index)/delr
         if (abs(xnodes(index+1)-xnodes(index)).gt.inttol) then
```

```
            sgnx = sgn(xnodes(index+1)-xnodes(index))
            do 60 cx = xnodes(index),xnodes(index+1)-sgnx*(delz-
     1           delz/10),sgnx*delz
c               if (index.eq.2) then
c                  print *,'TEST:',index,cx/delz
c                  pause
c               end if
                if (abs(cx-xnodes(index)).lt.inttol.AND.index.gt.1) then
                   print *,'debug xnodes',round(real(cx/delz)),
     1                round(real(cy/delr))
                   if (abs(xnodes(index)-xnodes(index+1)).lt.inttol.AND.
     1                (xnodes(index)-xnodes(index-1)).gt.0.0.AND.
     2                abs(ynodes(index)-ynodes(index-1)).lt.inttol.AND.
     3                (ynodes(index+1)-ynodes(index)).gt.0.0) then
                      count = count + 1
                      normsloc(count,1) = round(real(cx/delz))
                      normsloc(count,2) = round(real(cy/delr))
c******************** AN INHAT T1
                      normsinf(count,1) = 999.0
                      normsinf(count,2) = 999.0
                   else if (abs(xnodes(index)-xnodes(index-1)).lt.inttol
     1                   .AND.(xnodes(index+1)-xnodes(index)).gt.0.0.
     2                   AND.abs(ynodes(index)-ynodes(index+1)).lt.
     3                   inttol.AND.(ynodes(index-1)-ynodes(index)).
     4                   gt.0.0) then
                      count = count + 1
                      normsloc(count,1) = round(real(cx/delz))
                      normsloc(count,2) = round(real(cy/delr))
c******************** AN INHAT T2
                      normsinf(count,1) = -999.0
                      normsinf(count,2) = -999.0
                   else if (abs(xnodes(index)-xnodes(index-1)).lt.inttol
     1                   .AND.(xnodes(index+1)-xnodes(index)).gt.0.0.
     2                   AND.abs(ynodes(index)-ynodes(index+1)).lt.
     3                   inttol.AND.(ynodes(index)-ynodes(index-1)).
     4                   gt.0.0) then
                      count = count + 1
                      normsloc(count,1) = round(real(cx/delz))
                      normsloc(count,2) = round(real(cy/delr))
c******************** A CORHAT T1
c                      normsinf(count,1) = -500.0
c                      normsinf(count,2) = -500.0
                      normsinf(count,1) = -0.4867680792
                      normsinf(count,2) = 0.0
                   else if (abs(xnodes(index)-xnodes(index+1)).lt.inttol
     1                   .AND.(xnodes(index)-xnodes(index-1)).gt.0.0.
     2                   AND.abs(ynodes(index)-ynodes(index-1)).lt.
     3                   inttol.AND.(ynodes(index)-ynodes(index+1)).
     4                   gt.0.0) then
                      count = count + 1
                      normsloc(count,1) = round(real(cx/delz))
                      normsloc(count,2) = round(real(cy/delr))
c******************** A CORHAT T2
c                      normsinf(count,1) = 500.0
c                      normsinf(count,2) = 500.0
                      normsinf(count,1) = 3.6283607328
                      normsinf(count,2) = 0
                   else
                      slope = (ynodes(index+1)-ynodes(index))/(
     1                     xnodes(index+1)-xnodes(index))
                      cy = slope*(cx-xnodes(index+1))+ynodes(index+1)
                      count = count + 1
                      err = cy-delr*round(real(cy/delr))
                      if (abs(err).lt.inttol) err = 0.0
                      normsloc(count,1) = round(real(cx/delz))
                      normsloc(count,2) = round(real(cy/delr))
                      if (abs(cx-xnodes(index)).gt.inttol) then
                         normsinf(count,1) = pi/2-atan(slope)
                      else
                         if (index.gt.1) then
                            slope2 = (ynodes(index)-ynodes(index-1))/
     1                           (xnodes(index)-xnodes(index-1))
                            normsinf(count,1) = pi/2-atan((slope+slope2)
     1                           /2.0)
                         else
                            normsinf(count,1) = 0.0
                         end if
                      end if
                      normsinf(count,2) = err/delr
                   end if
                else
                   slope = (ynodes(index+1)-ynodes(index))/(
     1                  xnodes(index+1)-xnodes(index))
                   cy = slope*(cx-xnodes(index+1))+ynodes(index+1)
                   count = count + 1
                   err = cy-delr*round(real(cy/delr))
                   if (abs(err).lt.inttol) err = 0.0
                   normsloc(count,1) = round(real(cx/delz))
                   normsloc(count,2) = round(real(cy/delr))
                   if (abs(cx-xnodes(index)).gt.inttol) then
                      normsinf(count,1) = pi/2-atan(slope)
                   else
                      if (index.gt.1) then
                         slope2 = (ynodes(index)-ynodes(index-1))/
     1                        (xnodes(index)-xnodes(index-1))
                         normsinf(count,1) = pi/2-atan((slope+slope2)
     1                        /2.0)
                      else
                         normsinf(count,1) = 0.0
                      end if
                   end if
                   normsinf(count,2) = err/delr
                end if
 60          continue
          else
             sgny = sgn(ynodes(index+1)-ynodes(index))
c             print *,'sgny=',sgny
             do 70 cy = ynodes(index),ynodes(index+1)-sgny*(delr-
     1           delr/10),(sgny*delr)
                if (abs(cy-ynodes(index)).lt.inttol.AND.index.gt.1) then
                   print *,'debug ynodes',round(real(cx/delz)),
     1                round(real(cy/delr))
                   if (abs(xnodes(index)-xnodes(index+1)).lt.inttol.AND.
     1                (xnodes(index)-xnodes(index-1)).gt.0.0.AND.
     2                abs(ynodes(index)-ynodes(index-1)).lt.inttol.AND.
     3                (ynodes(index+1)-ynodes(index)).gt.0.0) then
                      count = count + 1
                      normsloc(count,1) = round(real(cx/delz))
                      normsloc(count,2) = round(real(cy/delr))
c******************** AN INHAT T1
                      normsinf(count,1) = 999.0
                      normsinf(count,2) = 999.0
                   else if (abs(xnodes(index)-xnodes(index-1)).lt.inttol
     1                   .AND.(xnodes(index+1)-xnodes(index)).gt.0.0.
     2                   AND.abs(ynodes(index)-ynodes(index+1)).lt.
     3                   inttol.AND.(ynodes(index-1)-ynodes(index)).
     4                   gt.0.0) then
                      count = count + 1
                      normsloc(count,1) = round(real(cx/delz))
                      normsloc(count,2) = round(real(cy/delr))
c******************** AN INHAT T2
                      normsinf(count,1) = -999.0
                      normsinf(count,2) = -999.0
                   else if (abs(xnodes(index)-xnodes(index-1)).lt.inttol
     1                   .AND.(xnodes(index+1)-xnodes(index)).gt.0.0.
     2                   AND.abs(ynodes(index)-ynodes(index+1)).lt.
     3                   inttol.AND.(ynodes(index-1)-ynodes(index)).
     4                   gt.0.0) then
                      count = count + 1
                      normsloc(count,1) = round(real(cx/delz))
                      normsloc(count,2) = round(real(cy/delr))
c******************** A CORHAT T1
c                      normsinf(count,1) = -500.0
c                      normsinf(count,2) = -500.0
                      normsinf(count,1) = -0.4867680792
                      normsinf(count,2) = 0
                   else if (abs(xnodes(index)-xnodes(index+1)).lt.inttol
     1                   .AND.(xnodes(index)-xnodes(index-1)).gt.0.0.
     2                   AND.abs(ynodes(index)-ynodes(index-1)).lt.
     3                   inttol.AND.(ynodes(index)-ynodes(index+1)).
     4                   gt.0.0) then
                      count = count + 1
                      normsloc(count,1) = round(real(cx/delz))
                      normsloc(count,2) = round(real(cy/delr))
c******************** A CORHAT T2
c                      normsinf(count,1) = 500.0
c                      normsinf(count,2) = 500.0
                      normsinf(count,1) = 3.6283607328
                      normsinf(count,2) = 0
                   end if
                else
                   cx = xnodes(index)
                   count = count + 1
                   normsloc(count,1) = round(real(cx/delz))
                   normsloc(count,2) = round(real(cy/delr))
                   normsinf(count,1) = 0.0
                   normsinf(count,2) = 0.0
                end if
 70          continue
          end if
 50    continue

c**** end point
       count = count + 1
       normsloc(count,1) = round(real(xnodes(total_nodes)/delz))
       normsloc(count,2) = round(real(ynodes(total_nodes)/delr))
       normsinf(count,1) = 0.0
       normsinf(count,2) = 0.0

       stair_node_count = count
       print *,'after count=',count

       goto 900
       stair_node_count = 211
```

```
      open(unit=10,file='norms2.dat',status='unknown',form='formatted')
      do 1001 index=1,stair_node_count
c         print *,'debug read', index
         read(10,*) normsloc(index,1), normsloc(index,2), count
c         print *,'debug read A', index
         if (count.eq.RHOHAT) normsinf(index,1) = pi/2
         if (count.eq.PMZHAT) normsinf(index,1) = 0.0
         if (count.eq.CORHAT) normsinf(index,1) = 0.0
         if (count.eq.INHAT) normsinf(index,1) = 999.0
         normsinf(index,2) = 0.0
 1001 continue
      close(unit=10)


 900   stair_node_count = stair_node_count
c**** Write out staircase model.
      open(unit=10,file='norms.dat',status='unknown',
     1     form='formatted')

      do 1000 index=1,stair_node_count
         write(10,*) normsloc(index,1), normsloc(index,2),
     1        round(real(100.*normsinf(index,1)*180/pi))*1.0/100.0,
     2        round(real(1000.*normsinf(index,2)))*1.0/1000.0
 1000 continue
      close(unit=10)

c**** Now arrange staircase data so that it is easy to manipulate

      call sort_staircase_data(stair_node_count)

      RETURN
      END

c*************************************************************************
c SORT_STAIRCASE_DATA sorts the stair_zero array by the range index so
c    that calc_inc_field subroutine can quickly find the cells that are
c    on the target surface at a given range step.
c*************************************************************************

      SUBROUTINE sort_staircase_data(N)

      implicit none
      include 'common.f'

      integer k,i,N,oldri,newri

c**** Sort staircase points by range index

      do 10 k = N-1,1,-1
         do 20 i = 1,k
            if (normsloc(i,1).gt.normsloc(i+1,1)) then
               call swap_stair(i,i+1)
            end if
 20      continue
 10   continue

c**** Figure out starting and ending indices of each range step

      do 30 k = 1, max_range_step
         range_index(k,1) = 1
         range_index(k,2) = 0
 30   continue

      oldri = 1
      do 40 k = 1, N
         newri = normsloc(k,1)
         if (newri.ne.oldri) then
            range_index(oldri,2) = k-1
            range_index(newri,1) = k
            oldri = newri
         end if
 40   continue
      range_index(normsloc(N,1),2) = N

      open(unit=10,file='range.dat',status='unknown',form='formatted')
      do 50 k = 1, max_range_step
         write(10,*) range_index(k,1), range_index(k,2)
 50   continue
      close(unit=10)

      END

c**** routine for swapping two elements in the stair_zero array
      SUBROUTINE swap_stair(ind1,ind2)

      implicit none
      include 'common.f'

      integer ind1, ind2, temp1, temp2
      real temp3, temp4

      temp1 = normsloc(ind1,1)
      temp2 = normsloc(ind1,2)
      temp3 = normsinf(ind1,1)
      temp4 = normsinf(ind1,2)
      normsloc(ind1,1) = normsloc(ind2,1)
      normsloc(ind1,2) = normsloc(ind2,2)
      normsinf(ind1,1) = normsinf(ind2,1)
      normsinf(ind1,2) = normsinf(ind2,2)

      normsloc(ind2,1) = temp1
      normsloc(ind2,2) = temp2
      normsinf(ind2,1) = temp3
      normsinf(ind2,2) = temp4

      RETURN
      END

c*************************************************************************
c INIT_FIELDS
c*************************************************************************

      SUBROUTINE init_fields

      implicit none
      include 'common.f'

      integer k

      do 10 k = 1, NMAX
         Efields(k) = (0.0,0.0)
 10   continue

      RETURN
      END

c*************************************************************************
c REAL FUNCTION DIST_TO_LINE returns the perpendicular distance from a
c point in space (x,y) to a line that is of the form Ax+By=C
c*************************************************************************

      REAL FUNCTION dist_to_line(A,B,C,x,y)

      implicit none
      real A,B,C,x,y

      dist_to_line = abs((A*x+B*y-C)/sqrt(A**2.0 + B**2.0))

      RETURN
      END


c*************************************************************************
c LOGICAL FUNCTION INSIDE_PEC(xlist,ylist,x,y) returns TRUE if the point
c   (x,y) is inside the polygon described by the points in x,y lists
c*************************************************************************

      LOGICAL FUNCTION inside_pec(px,py)

      implicit none
      include 'common.f'

      integer count
      real xlist(1:MAX_STAIR_NODES),ylist(1:MAX_STAIR_NODES),xp1,
     1     yp1,px,py,slope

      integer maxx, minx, maxy, miny, index, above, below, connect

c**** define xlist,ylist,count

      count = stair_node_count
      do 5 index=1,count
         xlist(count) = normsloc(index,1)+0.0
         ylist(count) = normsloc(index,2)+normsinf(index,2)
 5    continue

c**** if polygon not closed, close it.
      if (xlist(1).ne.xlist(count).OR.ylist(1).ne.ylist(count)) then
         connect = 0
      else
         connect = -1
      end if

      maxx = xlist(1)
      minx = xlist(1)
      maxy = ylist(1)
      miny = ylist(1)

      do 10 index=1,count
         if (xlist(index).gt.maxx) maxx=xlist(index)
         if (ylist(index).gt.maxy) maxy=ylist(index)
         if (xlist(index).lt.minx) minx=xlist(index)
         if (ylist(index).lt.miny) miny=ylist(index)
 10   continue
```

```
      if (px.gt.maxx.OR.px.lt.minx.OR.py.gt.maxy.OR.py.lt.miny) then
         inside_pec = .FALSE.
      else

c**** count the intersections

         above = 0
         below = 0

         do 20 index = 1,count+connect
            if (index.eq.count) then
               xp1 = xlist(1)
               yp1 = ylist(1)
            else
               xp1 = xlist(index+1)
               yp1 = ylist(index+1)
            end if

            if ( (abs(px-xlist(index)).lt.tole).AND.(abs(px-
     1           xlist(index+1)).lt.tole)) then
               if (((py.le.ylist(index)).AND.(py.ge.ylist(index+1))).
     1              OR.((py.ge.ylist(index)).AND.(py.le.ylist(index
     2              +1))) ) then
                  inside_pec = .FALSE.
                  RETURN
               end if
            end if

            if ( ((abs(px-xlist(index)).lt.tole).AND.(abs(py-
     1           ylist(index)).lt.tole)).OR.((abs(px-xlist(index+1)).
     2           lt.tole).AND.(abs(py-ylist(index+1)).lt.tole))) then
               inside_pec = .FALSE.
               RETURN
            end if

            if ((px.le.xlist(index).AND.px.ge.xp1).OR.
     1           (px.ge.xlist(index).AND.px.le.xp1)) then
               slope = (ylist(index)-yp1)/(xlist(index)-xp1)
               if (abs((slope*(px-xlist(index))+ylist(index))-py).
     1              lt.tole) then
                  inside_pec = .FALSE.
                  RETURN
               else if ((slope*(px-xlist(index))+ylist(index)).gt.
     1                 py) then
                  above = above+1
               else
                  below = below+1
               end if
            end if
 20      continue

         if (mod(above,2).eq.1.AND.mod(below,2).eq.1) then
            inside_pec = .TRUE.
         else
            inside_pec = .FALSE.
         end if
      end if

      RETURN
      END


c*********************************************************************
c REAL FUNCTION SGN returns the sign -1,0,1 of the arguement
c*********************************************************************

      REAL FUNCTION sgn(x)

      real x, stol

      stol = 1e-7

      if (x.gt.stol) then
         sgn = 1.0
      else if (x.lt.-stol) then
         sgn = -1.0
      else
         sgn = 0.0
      end if

      RETURN
      END


c*********************************************************************
c INTEGER FUNCTION ROUND returns the integer nearest in absolute
c   distance to the real arguement
c*********************************************************************

      INTEGER FUNCTION round(x)

      real x, fpart
      integer ipart

      ipart = int(x)
      fpart = x-aint(x)

      if (fpart.gt.0.5) then
         round = ipart+1
      else if (fpart.gt.0.0) then
         round = ipart
      else if (fpart.ge.-0.5) then
         round = ipart
      else
         round = ipart-1
      end if

      RETURN
      END

c*********************************************************************
c MEMORY_CHECK checks if enough memory has been allocated and reports
c   all errors stored in error buffer
c*********************************************************************

      SUBROUTINE memory_check

      implicit none
      include 'common.f'

      integer i, id, MAX_RCS_RL_ERROR, MAX_RCS_AB_ERROR

      parameter(MAX_RCS_RL_ERROR=6, MAX_RCS_AB_ERROR=7)

      if ((4*y2).gt.MAX_RCS_RL_NODES) then
         errorcount = errorcount+1
         errors(errorcount) = MAX_RCS_RL_ERROR
      end if

      if ((2*(x2-x1)+2).gt.MAX_RCS_AB_NODES) then
         errorcount = errorcount+1
         errors(errorcount) = MAX_RCS_AB_ERROR
      end if

      if (errorcount.gt.0) then
         write(6,*) '*********************************************'
         write(6,*) 'Insufficient memory to begin simulation. The'
         write(6,*) 'following parameter(s) in the common.f file'
         write(6,*) 'need to be adjusted:'

         do 10 i=1,errorcount
            id = errors(i)
            write(6,*)
            if (id.eq.NODE_ERROR) then
               write(6,*) 'Set MAX_NODES to at least',total_nodes
            else if (id.eq.MAX_Z_ERROR) then
               write(6,*) 'Set MAX_Z_CELLS to at least',max_range_step
            else if (id.eq.MAX_R_ERROR) then
               write(6,*) 'Set MAX_R_CELLS to at least',max_k
            else if (id.eq.MAX_STAIR_ERROR) then
               write(6,*) 'Set MAX_STAIR_NODES to at least',
     1              stair_node_count
            else if (id.eq.MAX_RCS_ERROR) then
               write(6,*) 'Set MAX_RCS_NODES to at least',
     1              (x2-x1)+2*(y2-y1)+1
            else if (id.eq.MAX_RCS_AB_ERROR) then
               write(6,*) 'Set MAX_RCS_AB_NODES to at least',
     1              2*(x2-x1)+2
            else if (id.eq.MAX_RCS_RL_ERROR) then
               write(6,*) 'Set MAX_RCS_RL_NODES to at least',
     1              4*y2
            end if

 10      continue
         write(6,*) '*********************************************'
         stop

      end if

      RETURN
      END

c*********************************************************************
c WRITE_OUT_ALL_PARMS outputs to a file all important parameters used
c   in running the simulation.
c*********************************************************************

      SUBROUTINE write_out_all_parms

      implicit none
      include 'common.f'

      integer index, round
```

```
      real x(1:MAX_STAIR_NODES), y(1:MAX_STAIR_NODES)

      do 10 index = 1,stair_node_count
         x(index) = real(normsloc(index,1))
         y(index) = real(normsloc(index,2))
 10   continue

      open(unit=10,file='pwe.out',status='unknown',form='formatted')

      write(10,*) 'max_range_step = ', max_range_step,';'
      write(10,*) 'max_k = ', max_k,';'
      write(10,*) 'delz = ', delz,';'
      write(10,*) 'delr = ', delr,';'
      write(10,*) 'inc_ang = ', inc_ang*180/pi,';'
      write(10,'('' freq (GHz) = '', F6.3, '';'')') kwave*c/2.0/pi/1e9
      write(10,*) 'kwave = ', kwave
      write(10,*)
      write(10,*) 'delz = lambda/',round(real(2*pi/kwave/delz))
      write(10,*) 'delr = lambda/',round(real(2*pi/kwave/delr))
      write(10,*) 'stair_node_count = ', stair_node_count
      write(10,*) 'running modes, ', smode,' to ',emode
      write(10,*) 'PML_DEPTH = ', PML_DEPTH
      write(10,*) 'yspacing = ', yspacing
      write(10,*) 'alpha = ', alpha
      write(10,*) 'beta = ', beta
      write(10,*) 'x1,y1,x2,y2= ',x1,y1,x2,y2
      write(10,*) 'rcs_nodes = ', 2*(y2-y1)+(x2-x1)+1
      if (comp_forw_scat.eq.1) then
         write(10,*) 'Paraxial Direction = forward'
      else if (comp_forw_scat.eq.2) then
         write(10,*) 'Paraxial Direction = backward'
      else
         write(10,*) 'Paraxial Direction = forw/back'
      end if
      if (polarization.eq.HORZ) then
         write(10,*) 'HH RCS calculated'
      else
         write(10,*) 'VV RCS calculated'
      end if
      write(10,*) 'Geomfile used was', geomfile
c      write(10,*) 'low_freq (GHz) = ', freqlist(1,1)/1.0e9
c      write(10,*) 'high_freq (GHz) = ', freqlist(num_freqs,1)/1.0e9
c      write(10,*) 'num_freqs = ', maxf-minf+1

      write(10,*)
      write(10,*) 'Optimal memory settings...'
      write(10,*)
      write(10,*) 'MAX_NODES =', total_nodes
      write(10,*) 'MAX_Z_CELLS = ', max_range_step
      write(10,*) 'MAX_R_CELLS = ', max_k
      write(10,*) 'MAX_STAIR_NODES = ', stair_node_count
      write(10,*) 'MAX_RCS_NODES = ', (x2-x1)+2*(y2-y1)+1
      write(10,*) 'MAX_RCS_AB_NODES = ', 2*(x2-x1)+2
      write(10,*) 'MAX_RCS_RL_NODES = ', 4*y2
      write(10,*) 'NMAX=3*MAX_R_CELLS+6 = ',3*max_k+6

c      call plotb(x,y,stair_node_count,51,41,10)

      close(unit=10)

      RETURN
      END

c**********************************************************************
c WRITE_OUT_LIN_SYS writes out the linear that represents the
c     propagation at the current range step.
c**********************************************************************

      SUBROUTINE write_out_lin_sys

      implicit none
      include 'common.f'

      integer k

      open(unit=12,file='linsys.dat',status='unknown',form='formatted')
      open(unit=13,file='efields.dat',status='unknown',form='formatted')
      open(unit=14,file='sa.dat',status='unknown',form='formatted')
      open(unit=15,file='ija.dat',status='unknown',form='formatted')

c      do 10 k=1,3*max_k+6
c         do 15 j=1,3*max_k+6
c            write(12,99) real(Awave(k,j)),imag(Awave(k,j))
c 15      continue
c 10   continue

      do 20 k=1,3*max_k+6
         write(13,99) real(Efields(k)),imag(Efields(k))
 20   continue

      print *,ija(3*max_k+7)-1
      do 30 k=1,ija(3*max_k+7)-1
         write(14,99) real(sa(k)),imag(sa(k))
         write(15,*) ija(k)
 30   continue

 99   format(2X,E21.13,5X,E21.13)

      close(unit=13)
      close(unit=12)
      close(unit=14)
      close(unit=15)

      RETURN
      END

c**********************************************************************
c WRITE_OUT_IMAGE writes out the values of the chosen efield component
c     so that a PWE "image" can be constructed.
c**********************************************************************

      SUBROUTINE write_out_image(FN)

      implicit none
      include 'common.f'

      integer FN, k

c**** imagefieldtype = 1 --> Record Erho field
c**** imagefieldtype = 2 --> Record Ephi field
c**** imagefieldtype = 3 --> Record Ez field

      if (imagefieldtype.eq.1) then
         do 10 k = 1,max_k+2
            write(FN,99) real(efields(k)), imag(efields(k))
 10      continue
      else if (imagefieldtype.eq.2) then
         do 20 k = max_k+3, 2*max_k+4
            write(FN,99) real(efields(k)),imag(efields(k))
 20      continue
      else if (imagefieldtype.eq.3) then
         do 30 k = 2*max_k+5,3*max_k+6
            write(FN,99) real(efields(k)),imag(efields(k))
 30      continue
      end if

 99   format(2X,E21.13,5X,E21.13)

      RETURN
      END


c**********************************************************************
c PLOTB takes a set of data points (X,Y) and makes and ascii plot out
c   of them
c**********************************************************************

      SUBROUTINE PLOTB(X,Y,N,NC,NR,FID)
C
C WRITTEN 2/14/74   BY J. M. PUTNAM       DEPT 220     X23877
C
C THIS ROUTINE PRODUCES A LINEAR XY PLOT.
C    N IS THE NUMBER OF POINTS TO BE PLOTTED.
C    NR IS THE NUMBER OF ROWS TO BE USED FOR THE Y-AXIS.
C    NC IS THE NUMBER OF COLUMNS TO BE USED FOR THE X-AXIS.
C       NOTE, NC-1 MUST BE DIVISIBLE BY 10 AND LESS THAN 102.
Cf
      REAL X(161),Y(161),HEAD(10)
      INTEGER  LINE(101),BLANK,STAR,FID
      DATA BLANK,STAR /1H ,1H*/
      N10=(NC-1)/10
      WRITE(FID,500)
 500  FORMAT(///,17H1BODY COORDINATES)
      WRITE(FID,504)
      XMIN=X(1)
      XMAX=X(1)
      YMIN=Y(1)
      YMAX=Y(1)
      DO 6 I=1,N
      IF(X(I).LT.XMIN) XMIN=X(I)
      IF(X(I).GT.XMAX) XMAX=X(I)
      IF(Y(I).LT.YMIN) YMIN=Y(I)
      IF(Y(I).GT.YMAX) YMAX=Y(I)
 6    CONTINUE
      DEL=XMAX-XMIN
      IF(YMAX-YMIN.GT.DEL) DEL=YMAX-YMIN
      XMAX=XMIN+DEL
      YMAX=YMIN+DEL
      DO 5 I =1,N10
      Z=I
    5 HEAD(I)=(XMAX-XMIN)*Z/N10+XMIN
      DY=(YMAX-YMIN)/(NR-1)
      Z=YMAX+DY
      YL=Z-DY/2.
```

```
      DO 7 J=1,NR
      DO 8 K=1,NC
 8    LINE(K)=BLANK
      Z=Z-DY
      YU=YL
      YL=Z-DY/2.
      DO 9 I=1,N
      IF(Y(I).GE.YU) GO TO 9
      IF(Y(I).LT.YL) GO TO 9
      K=(X(I)-XMIN)/(XMAX-XMIN)*(NC-1)+1.5
      IF(K.GT.NC) K=NC
      LINE(K)=STAR
 9    CONTINUE
      WRITE(FID,508) Z,(LINE(K),K=1,NC)
 7    CONTINUE
      WRITE(FID,504)
      WRITE(FID,3002)
      WRITE(FID,507) XMIN,(HEAD(I),I=1,N10)
C     *************************
      RETURN
  504 FORMAT ( 1X, 14(1H-), 1H., 10(5H----.), 1H- )
  507 FORMAT(10X,11(F10.4))
  508 FORMAT (1X, F12.4,1X, 1HI, 51A1, 1HI )
 3002 FORMAT(4X,7HRB / ZB,4X,1HI,5(9X,1HI))

      END
```

The following, **bicgstab.f**, contains the subroutines used to implement the stablized biconjugate gradient method, an iterative technique used to solve sparse matrix equations.

```
      SUBROUTINE linbcgstab(n,b,x,itol,tol,itmax,iter,err)

      implicit none
      include 'common.f'

      INTEGER iter,itmax,itol,n
      complex*16 b(NMAX),x(NMAX)
      DOUBLE PRECISION err,tol,EPSV
      parameter (EPSV=1.d-14)

C     U    USES atimes,asolve,snrm
      INTEGER j
      DOUBLE PRECISION bnrm,dxnrm,xnrm,zminrm,znrm,snrm
      complex*16 alpha1,beta1,rho,rho1,omega,rtv,tt
      complex*16 p(NMAX),r(NMAX),rr(NMAX),xhalf(NMAX),v(NMAX),s(NMAX)
      complex*16 t(NMAX),z(NMAX),xmin(NMAX)

      double precision minerr
      integer iterminerr

      iter=0

c**** intitial guess is efields
      do 2 j=1,n
         x(j) = b(j)
 2    continue

      call atimes(n,x,r,0)
      do 11 j=1,n
         r(j)=b(j)-r(j)
         rr(j)=r(j)
 11   continue
C     call atimes(n,r,rr,0)
      znrm=1.0
      rho=(1.,0.)
      omega=(1.,0.)
      if(itol.eq.1) then
         bnrm=snrm(n,b,itol)
c****** If right hand side all zeros, don't iterate, return all zeros
         if (bnrm.lt.tol) then
            do 5 j=1,n
               x(j) = 0.0
 5          continue
            write(*,*) 'finished, no iterations, RHS all zeros'
            return
         end if
      else if (itol.eq.2) then
         call asolve(n,b,z,0)
         bnrm=snrm(n,z,itol)
c****** If right hand side all zeros, don't iterate, return all zeros
         if (bnrm.lt.tol) then
            do 6 j=1,n
               x(j) = 0.0
 6          continue
            write(*,*) 'finished, no iterations, RHS all zeros'
            return
         end if
      else if (itol.eq.3.or.itol.eq.4) then
         call asolve(n,b,z,0)
         bnrm=snrm(n,z,itol)
c****** If right hand side all zeros, don't iterate, return all zeros
         if (bnrm.lt.tol) then
            do 7 j=1,n
               x(j) = 0.0
 7          continue
            write(*,*) 'finished, no iterations, RHS all zeros'
            return
         end if
         call asolve(n,r,z,0)
         znrm=snrm(n,z,itol)
      else
         pause 'illegal itol in linbcg'
      endif

 100  if (iter.le.itmax) then
         iter=iter+1
c         write(*,*) iter,err
         rho1=rho
         zminrm=znrm
         rho=(0.,0.)
         do 12 j=1,n
            rho=rho+r(j)*conjg(rr(j))
 12      continue
         if(iter.eq.1) then
            do 13 j=1,n
               p(j)=r(j)
 13         continue
         else
            beta1=(rho/rho1)*(alpha1/omega)
            do 14 j=1,n
               p(j)=beta1*(p(j)-v(j)*omega)+r(j)
 14         continue
         endif
         call atimes(n,p,v,0)
         rtv=(0.,0.)
         do 15 j=1,n
            rtv=rtv+v(j)*conjg(rr(j))
 15      continue
         alpha1=rho/rtv
         do 16 j=1,n
            xhalf(j)=x(j)+alpha1*p(j)
            s(j)=r(j)-alpha1*v(j)
 16      continue
         call atimes(n,s,t,0)
         tt=(0.,0.)
         do 17 j=1,n
            tt=tt+conjg(t(j))*t(j)
 17      continue
         omega=(0.,0.)
         do 18 j=1,n
            omega=omega+conjg(t(j))*s(j)
 18      continue
         omega=omega/tt
         do 19 j=1,n
            x(j)=x(j)+alpha1*p(j)+omega*s(j)
            r(j)=s(j)-omega*t(j)
 19      continue

         if(itol.eq.1.or.itol.eq.2)then
            znrm=1.d0
            err=snrm(n,r,itol)/bnrm
         else if(itol.eq.3.or.itol.eq.4)then
            znrm=snrm(n,z,itol)
            if(abs(zminrm-znrm).gt.EPSV*znrm) then
               dxnrm=abs(alpha1)*snrm(n,p,itol)
               err=znrm/abs(zminrm-znrm)*dxnrm
            else
               err=znrm/bnrm
               goto 100
            endif
            xnrm=snrm(n,x,itol)
            if(err.le.0.5d0*xnrm) then
               err=err/xnrm
            else
               err=znrm/bnrm
               goto 100
            endif
         endif
         if (iter.eq.1) then
            minerr = err
            iterminerr = iter
            do 201 j=1,n
               xmin(j) = x(j)
 201        continue
         else
            if (err.lt.minerr) then
               minerr = err
               iterminerr = iter
               do 202 j=1,n
```

```
                xmin(j) = x(j)
 202          continue
            end if
          end if

c          write (*,*) ' iter=',iter,' err=',err
          if(err.gt.tol) goto 100
        endif

        write (*,*) ' iter=',iter,' err=',err
c**** if solution diverged then return xmin out of all iterations
        if (err.gt.tol) then
          write(*,*) ' iterminerr=',iterminerr,' minerr=',minerr
          do 203 j=1,n
            x(j) = xmin(j)
 203      continue
          err = minerr
          iter = iterminerr
        end if
        return
        END

c**** computes the 1-vector normal.
        FUNCTION snrm(n,sx,itol)

        implicit none
        include 'common.f'

        INTEGER n,itol,i,isamax
        COMPLEX*16 sx(n)
        DOUBLE PRECISION snrm

        if (itol.le.3)then
          snrm=0.
          do 11 i=1,n
            snrm=snrm+abs(sx(i))**2
11        continue
          snrm=sqrt(snrm)
        else
          isamax=1
          do 12 i=1,n
            if(abs(sx(i)).gt.abs(sx(isamax))) isamax=i
12        continue
          snrm=abs(sx(isamax))
        endif
        return
        END
C  (C) Copr. 1986-92 Numerical Recipes Software ]2+r9,6)!.

c**** Computes the product of the sparse matrix structure (ija,sa)
c**** with the vector x and stores the result in b.
        SUBROUTINE atimes(n,x,b,itrnsp)

        implicit none
        include 'common.f'

        INTEGER n,itrnsp,i,j,k
        COMPLEX*16 x(n),b(n)

        if (itrnsp.eq.0) then
c****** calculate b=A*x
          if (ija(1).ne.n+2) pause 'mismatched vector & matrix in sprsax'
          do 12 i=1,n
            b(i)=sa(i)*x(i)
            do 11 k=ija(i),ija(i+1)-1
              b(i)=b(i)+sa(k)*x(ija(k))
 11         continue
 12       continue
        else
c****** calculate b=At*x where At is conjugate tranpose of A
          if (ija(1).ne.n+2) pause 'mismatched vector & matrix in sprstx'
          do 111 i=1,n
            b(i)=sa(i)*x(i)
 111      continue
          do 113 i=1,n
            do 112 k=ija(i),ija(i+1)-1
              j=ija(k)
              b(j)=b(j)+conjg(sa(k))*x(i)
 112        continue
 113      continue

        endif

        return
        END

c**** set pre-conditioner equal to identity matrix
        SUBROUTINE asolve(n,b,x,itrnsp)

        implicit none
        include 'common.f'

        INTEGER n,itrnsp,i
        COMPLEX*16 x(n),b(n)

        do 11 i=1,n
c           x(i)=b(i)/sa(i)
           x(i)=b(i)
11      continue
        return
        END
C  (C) Copr. 1986-92 Numerical Recipes Software ]2+r9,6)!.
```

The following, **common.f**, is used to create the common blocks that are included in most of the subroutines used by the BOR PWE program.

```
c**** common.include file.  Contains all global variable declarations.

c**** Variables that can be changed to allocate memory

      integer MAX_R_CELLS, MAX_Z_CELLS, MAX_NODES, MAX_STAIR_NODES,
     1     MAX_RCS_NODES, NMAX, MAX_RCS_AB_NODES, MAX_RCS_RL_NODES,
     2     MAX_NORM_NODES, PML_DEPTH, NNZ, MINMODE, MAXMODE

      parameter(MAX_Z_CELLS=2420)
      parameter(MAX_R_CELLS=600)
      parameter(MAX_NODES=201)
      parameter(MAX_STAIR_NODES=2641)
      parameter(MAX_NORM_NODES=2*MAX_STAIR_NODES+1)
      parameter(MAX_RCS_NODES=2653)
      parameter(NMAX=3*MAX_R_CELLS+6)
      parameter(MINMODE=0)
      parameter(MAXMODE=14)

c**** maximum number of nonzeros. computed as follows:
c****   tridiagonal terms: 3*(3*MAX_R_CELLS+6)-2
c****   super-super terms: 2*(2*MAX_R_CELLS+4)
c**** max z-coupled terms: 3*MAX_R_CELLS
c****                total: 16*MAX_R_CELLS+24
      parameter(NNZ=16*MAX_R_CELLS+24)

      parameter(MAX_RCS_RL_NODES=496)
      parameter(MAX_RCS_AB_NODES=4814)
c      parameter(PML_DEPTH=8)

c***********************************************************************
c**********************STATIC VARIABLE DECLATIONS BELOW*****************
c***********************************************************************

c**** Important contants
      double precision c, mu, eps, eta
      complex uniti

      parameter(c=2.9979247917E8, mu=1.25663706144E-6,
     1     eps=8.8541874E-12, eta=376.73032,uniti=(0.0,1.0))

      double precision tole, pi, ONE, ZERO
      parameter(tole = 1.0e-13, pi=3.14159265358979)
      parameter(ZERO=0.d0, ONE=1.d0)

c**** sa,ija -- sparse matrix structure for wave matrix
c**** Efields -- vector of Erho,Ephi,Ez fields along at current range

c      complex*16 Awave(1:NMAX,1:NMAX)
c      complex*16 Bwave(NMAX*NMAX)
      complex*16 Efields(1:NMAX)

      complex*16 sa(1:NNZ)
      integer ija(1:NNZ)

c**** rcsErho,rcsEphi,rcsEz -- vectors contains fields along a surface
c****   that encloses the scatterer.

      complex*16 rcsErho(MINMODE:MAXMODE,1:MAX_RCS_NODES),
     1     rcsEphi(MINMODE:MAXMODE,1:MAX_RCS_NODES),
     2     rcsEz(MINMODE:MAXMODE,1:MAX_RCS_NODES)
      complex*16 rcsHrho(MINMODE:MAXMODE,1:MAX_RCS_NODES),
     4     rcsHphi(MINMODE:MAXMODE,1:MAX_RCS_NODES),
     5     rcsHz(MINMODE:MAXMODE,1:MAX_RCS_NODES),
     6     rcsEphi_ab(1:MAX_RCS_AB_NODES),
     7     rcsEphi_rl(1:MAX_RCS_RL_NODES),
     8     rcsEz_ab(1:MAX_RCS_AB_NODES),
     9     rcsErho_rl(1:MAX_RCS_RL_NODES)

c**** Geometry description and incident wave description variables
      character*72 geomfile
      complex*16 AMP
      parameter(AMP = (1.d0,0.d0))
      real start_freq, end_freq, dfreq
      double precision delz, delr
```

```
      double precision delta, kwave, inc_ang, heightkwave
      integer max_k, max_range_step, polarization, HORZ, VERT,
     1     RHOHAT, PMZHAT, CORHAT, INHAT, obj_k, Ehg, Evg,
     2     smode, emode

      parameter(HORZ=+1, VERT=-1, RHOHAT=1, PMZHAT=2, CORHAT=3, INHAT=4)

c**** Geometry data points
      integer total_nodes, stair_node_count, x1,x2,y1,y2,
     2     normsloc(1:MAX_STAIR_NODES,1:2),
     3     range_index(1:MAX_Z_CELLS,1:2)

c**** contains info about angle of norm and length correction.
      real normsinf(1:MAX_STAIR_NODES,1:2)

c**** Error buffer and error types
      integer errorcount, errors(10),
     1     NODE_ERROR, MAX_Z_ERROR, MAX_R_ERROR, MAX_STAIR_ERROR,
     2     MAX_RCS_ERROR

      parameter(NODE_ERROR=1, MAX_Z_ERROR=2, MAX_R_ERROR=3,
     1     MAX_STAIR_ERROR=4, MAX_RCS_ERROR=5)

c**** Determines if image is to be stored.
      logical store_image
      integer imagefieldtype, comp_forw_scat, yspacing

c**** Bistatic calculation information
      double precision low_phi, high_phi, dphi, low_theta, high_theta,
     1     dtheta

c**** PML parameters
      double precision alpha, beta

c**** Common Blocks

      common /matfcn/ max_k, obj_k

      common delta, kwave, max_range_step
      common /newstf/ heightkwave, smode, emode

      common /incan/ inc_ang, store_image, comp_forw_scat, delz, delr,
     1     Ehg, Evg

      common /spar/ sa, ija

      common /bist/ low_phi, high_phi, dphi, low_theta, high_theta,
     1     dtheta
c       common /Aw/ Awave
c       common /Bw/ Bwave
      common /erho/ rcserho, rcsephi, rcsez, Efields, imagefieldtype

      common /hrho/ rcshrho, rcshphi, rcshz, rcsEphi_ab, rcsEphi_rl,
     1     rcsErho_rl, rcsEz_ab, PML_DEPTH, yspacing

      common /pml/ alpha, beta

c**** Common Geometry Block Variables
      common total_nodes, stair_node_count, errorcount, normsloc,
     1     errors, x1,x2,y1,y2, range_index, polarization, geomfile,
     2     normsinf


      common start_freq, end_freq, dfreq
```

Bibliography

[1] Mogens G. Andreasen. Scattering from bodies of revolution. *IEEE Trans. Antennas Propagat.*, 13:303–310, March 1965.

[2] Amalia E. Barrios. A terrain parabolic equation model for propagation in the troposphere. *IEEE Trans. Antennas Propagat.*, 42(1):90–98, January 1994.

[3] Jean-Pierre Berenger. A perfectly matched layer for the absorption of electromagnetic waves. *Journal of Computational Physics*, 114:185–200, 1994.

[4] P-P. Borsboom and M.F. Levy. Scattering with parabolic equation methods: application to RCS computation. In *IEE Colloquium on Common Modelling Techniques for Electromagnetic Wave and Acoustic Wave Propagation*, pages 5/1–5/4, London, UK, 1996.

[5] P-P. Borsboom and Hyaric A. Zebic-Le. RCS predications using wide-angle PE codes. In *IEE Tenth International Conference on Antennas and Propagation*, volume 2, pages 191–4, London, UK, 1997.

[6] Charles L. Britt. Solution of electromagnetic scattering problems using time domain techniques. *IEEE Trans. Antennas Propagat.*, 37(9):1181–1191, September 1989.

[7] Yinchao Chen, Raj Mittra, and Paul Harms. Finite-difference time-domain algorithm for solving Maxwell's equations in rotationally symmetric geometries. *IEEE Trans. Microwave Theory Tech.*, 44(6):832–839, June 1996.

[8] W. C. Chew, J. M. Jin, and E. Michielssen. Complex coordinate stretching as a generalized absorbing boundary condition. *Microwave Opt. Technol. Lett.*, 15(6):363–369, August 1997.

[9] Weng Cho Chew and William H. Weedon. A 3D perfectly matched medium from modified Maxwell's equations with stretched coordinates. *Microwave Opt. Technol. Lett.*, 7(13):599–604, September 1994.

[10] David B. Davidson and Richard W. Ziolkowski. Body-of-revolution finite-difference time-domain modeling of space-time focusing by a three-dimensional lens. *Journal of the Optical Society of America: A*, 11(4):1471–1490, April 1994.

[11] G. Daniel Dockery and James R. Kuttler. An improved impedance-boundary algorithm for Fourier split-step solutions of the parabolic wave equation. *IEEE Trans. Antennas Propagat.*, 44(12):1592–1598, December 1996.

[12] Paul DuChateau and David Zachmann. *Applied Partial Differential Equations*, chapter Finite-Difference Methods for Parabolic Equations. Harper and Row, Publishers, Inc., New York, 1989.

[13] E.P. Ekelman and Gary A. Thiele. A hybrid technique for combining the moment method treatment of wire antennas with the GTD for curved surfaces. *IEEE Trans. Antennas Propagat.*, 28:831–839, 1980.

[14] Bjorn Engquist and Andrew Majda. Absorbing boundary conditions for the numerical simulation of waves. *Mathematics of Computation*, 31(139):629–651, July 1977.

[15] M. Fusco. FDTD algorithm in curvilinear coordinates. *IEEE Trans. Antennas Propagat.*, AP-42(1):76–89, 1990.

[16] Roger F. Harrington. Matrix methods for field problems. *Proceedings of the IEEE*, 55(2):136–149, February 1967.

[17] Roger F. Harrington. *Field computation by moment methods.* Macmillan, New York, NY, 1968.

[18] Roger F. Harrington and J.R. Mautz. Radiation and scattering from bodies of revolution. *Applied Scientific Research*, 20:405–435, June 1969.

[19] L.W. Henderson and Gary A. Thiele. A hybrid technique MM-GTD technique for treatment of wire antennas near a curved surface. *Radio Science*, 16:1125–1130, 1981.

[20] R. Holland. Finite-difference solution of Maxwell's equations in generalized nonorthogonal coordinates. *IEEE Trans. Nuclear Science*, NS-30(6):4589–4591, 1983.

[21] R. Holland. A finite-difference time-domain EMP code in 3D spherical coordinates. *IEEE Trans. Nuclear Science*, NS-30(6):4592–4595, 1983.

[22] R. Holland. Finite difference time domain (FDTD) analysis of magnetic diffusion. *IEEE Trans. Electromagn. Compat.*, 36:32–39, Feb 1994.

[23] J. W. Crispin Jr. and K. M. Siegel, editors. *Methods of Radar Cross-Section Analysis.* Academic Press Inc., New York, NY, 1968.

[24] Thomas Jurgens. A broadband absorbing boundary condition for the FDTD modeling of circular waveguides. In *IEEE MTT-S International Microwave Symposium Digest*, volume 1, pages 35–37, New York, NY, USA, 1995.

[25] Thomas G. Jurgens and Allen Taflove. Three-dimensional contour FDTD modeling of scattering from single and multiple bodies. *IEEE Trans. Antennas Propagat.*, 41(12):1703–1708, December 1993.

[26] Thomas G. Jurgens, Allen Taflove, Korada Umashankar, and Thomas G. Moore. Finite-difference time-domain modeling of curved surfaces. *IEEE Trans. Antennas Propagat.*, 40(4):357–366, April 1992.

[27] Daniel S. Katz, Eric T. Thiele, and Allen Taflove. Validation and extension to three dimensions of the Berenger PML absorbing boundary condition for FD-TD meshes. *IEEE Microwave Guided Wave Lett.*, 4(8):268–270, August 1994.

[28] Robert E. Kell. On the derivation of bistatic RCS from monostatic measurements. *Proceedings of the IEEE*, 53:983–988, August 1965.

[29] J.B. Keller. Diffraction by an aperture. *J. Appl. Phys.*, 28(4):426–444, April 1957.

[30] J.B Keller. Geometrical theory of diffraction. *J. Opt. Soc. Amer.*, 52(2):116–130, February 1962.

[31] E. F. Knott, J. F. Schaeffer, and M. T. Tuley. *Radar Cross Section: Its Prediction, Measurement, and Reduction.* Artech House, Inc., Dedham, MA, 1985.

[32] J. A. Kong. *Electromagnetic Wave Theory.* John Wiley and Sons, New York, 1986.

[33] James R. Kuttler and G. Daniel Dockery. Theoretical description of the parabolic approximation/Fourier split-step method of representing electromagnetic propagation in the troposphere. *Radio Science*, 26(2):381–393, March-April 1991.

[34] M. F. Levy and A. A. Zaporozhets. Parabolic equation techniques for scattering. *Wave Motion*, 31:147–156, 2000.

[35] M.F. Levy. Transparent boundary conditions for parabolic equation solutions of radiowave propagation problems. *IEEE Trans. Antennas Propagat.*, 45(1):1–7, January 1997.

[36] M.F. Levy and P-P. Borsboom. Radar cross-section computations using the parabolic equation method. *Electron. Lett.*, 32(13):1234–1236, June 1996.

[37] M.F. Levy, P-P. Borsboom, and A.A. Zaporozhets. RCS computation with paraxial methods. In *IEEE Antennas and Propagation Society International Symposium Digest*, volume 2, pages 1152–1155, 1997.

[38] Mireille F. Levy and Andrew A. Zaporozhets. Target scattering calculations with the parabolic equation method. *J. Acoust. Soc. Am.*, 103(2):735–741, February 1998.

[39] Kevin Li. *Finite Difference-Time Domain Analysis of Electromagnetic Interference and Radiation Problems.* PhD thesis, M.I.T., 1995.

[40] Louis N. Medgyesi-Mitschang and Dau-Sing Wang. Hybrid solutions for scattering from perfectly conducting bodies of revolution. *IEEE Trans. Antennas Propagat.*, 31(4):570–583, July 1983.

[41] D. E. Merewether and R. Fisher. Finite difference solution of Maxwell's equation for EMP applications. Technical Report EMA-79-R-4, Defense Nuclear Agency, April 1980.

[42] E. A. Navarro, C. Wu, P. Y. Chung, and J. Litva. Application of PML super-absorbing boundary condition to non-orthogonal FDTD method. *IEEE Microwave Guided Wave Lett.*, 30(20):1654–1655, September 1994.

[43] Dennis W. Prather and Shouyuan Shi. Formulation and application of the finite-difference time-domain method for the analysis of axially symmetric diffractive optical elements. *Journal of the Optical Society of America: A*, 16(5):1131–1142, May 1999.

[44] Christopher E. Reuter, Rose M. Joseph, Eric T. Thiele, Daniel S. Katz, and Allen Taflove. Ultrawideband absorbing boundary condition for termination of waveguiding structures in FD-TD simulations. *IEEE Microwave Guided Wave Lett.*, 4(10):344–346, October 1994.

[45] J. A. Roden and S. D. Gedney. Efficient implementation of the uniaxial-based PML media in three-dimensional nonorthogonal coordinates with the use of the FDTD technique. *Microwave Opt. Technol. Lett.*, 14(2):71–75, February 1997.

[46] Frank J. Ryan. Analysis of electromagnetic propagation over variable terrain using the parabolic wave equation. Technical Report 1453, Ocean and Atmospheric Sciences Division, Naval Ocean Systems Center, San Diego, CA, October 1991.

[47] Gregory W. Saewert and Thomas G. Jurgens. Xwake 1.1: a new impedance and wake field software package. In *Proceedings of the 1995 Particle Accelerator Conference*, volume 4, pages 2303–5, New York, NY, USA, 1995.

[48] J.N. Sahalos and Gary A. Thiele. On the application of the GTD-MM technique and its limitation. *IEEE Trans. Antennas Propagat.*, 29:780–786, 1981.

[49] Allen Taflove. Review of the formulation and applications of the finite-difference time-domain method for numerical modeling of electromagnetic wave interactions with arbitrary structures. *Wave Motion*, 10:547–582, 1988.

[50] Allen Taflove. *The Finite-Difference Time-Domain Method.* Artech House, 1995.

[51] Allen Taflove and Korada Umashankar. Radar cross section of general three-dimensional scatterers. *IEEE Trans. Electromagn. Compat.*, 25(4):433–440, November 1986.

[52] Allen Taflove and Korada Umashankar. Review of FD-TD numerical modeling of electromagnetic wave scattering and radar cross section. *Proceedings of the IEEE*, 77(5):682–99, May 1989.

[53] Allen Taflove, K.R. Umashankar, and Thomas G. Jurgens. Validation of FD-TD modeling of the radar cross section of three-dimensional structures spanning up to nine wavelengths. *IEEE Trans. Antennas Propagat.*, 33(6):662–666, June 1985.

[54] F. L. Teixeira and W. C. Chew. PML-FDTD in cylindrical and spherical grids. *IEEE Microwave and Guided Wave Letters*, 7(9):285–287, September 1997.

[55] Gary A. Thiele. Overview of selected hybrid methods in radiating system analysis. *Proceedings of the IEEE*, 80(1):66–77, January 1992.

[56] Gary A. Thiele and T.H. Newhouse. A hybrid technique for combining moment methods with a geometrical theory of diffraction. *IEEE Trans. Antennas Propagat.*, 23:62–69, 1975.

[57] P. la. Ufimtsev. Approximate computation of the diffraction of plane electromagnetic waves at certain metal bodies: Pt. I diffraction patterns at a wedge and a ribbon. *Zh. Tekhn. Fiz.*, 27(8):1708–1718, 1957.

[58] K. S. Yee. Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media. *IEEE Trans. Antennas Propagat.*, AP-14(3):302–307, May 1966.

[59] Andrew A. Zaporozhets and Mireille F. Levy. Bistatic RCS calculations with the vector parabolic equation method. *IEEE Trans. Antennas Propagat.*, 47(11):1688–1696, November 1999.