Algorithms for Design and Interrogation of Functionally Graded Material Solids

by

Hongye Liu

B.E., University of Science and Technology of China, 1993

Submitted to the Department of Ocean Engineering

and

Department of Mechanical Engineering
in partial fulfillment of the requirements for the degrees of

Master of Science in Naval Architecture and Marine Engineering

and

Master of Science in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2000

Author . . . .
Department of Ocean Engineering
ary 25, 2000

Certified by . . . .
Nicholas M. Patrikalakis
[illegible] Kawasaki Professor of Engineering

Certified by . . . .
Emanuel M. Sachs
chanical Engineering

Accepted by . . . .
Nicholas M. Patrikalakis, Kawasaki Professor of Engineering
Chairman [illegible] Committee on Graduate Studies

Accepted by . . . .
Ain A. Sonin, Professor of Mechanical Engineering
Chairman, Departmental Committee on Graduate Studies

Algorithms for Design and Interrogation of Functionally Graded Material Solids

by

Hongye Liu

Submitted to the Department of Ocean Engineering
and
Department of Mechanical Engineering
on February 25, 2000, in partial fulfillment of the
requirements for the degrees of
Master of Science in Naval Architecture and Marine Engineering
and
Master of Science in Mechanical Engineering

Abstract

A Functionally Gradient Material (FGM) part is a 3D solid object that has varied local material composition that is defined by a specifically designed function. Recently, research has been performed at MIT in order to exploit the potential of creating FGM parts using a modern fabrication process, 3D Printing, that has the capability of controlling composition to the length scale of 100 μm. As part of the project of design automation of FGM parts, this thesis focuses on the issue of the development of efficient algorithms for design and composition interrogation. Starting with a finite element based 3D model, the design tool based on the distance function from the surface of the part and the design tool allowing the user to design within a .STL file require enhanced efficiency and so does the interrogation of the part. The approach for improving efficiency includes preprocessing the model with bucket sorting, digital distance transform of the buckets and an efficient point classification algorithm. Based on this approach, an efficient algorithm for distance function computation is developed for the design of FGM through distance to the surface of the part or distance to a .STL surface boundary. Also an efficient algorithm for composition evaluation at a point, along a ray or on a plane is developed. The theoretical time complexities of the developed algorithms are analyzed and experimental numerical results are provided.

Thesis Co-Supervisor: Nicholas M. Patrikalakis, Ph.D.
Title: Kawasaki Professor of Engineering
Department of Ocean Engineering

Thesis Co-Supervisor: Emanuel M. Sachs, Ph.D.
Title: Professor of Mechanical Engineering
Department of Mechanical Engineering

Dedication

This thesis is dedicated to my parents Guohua Liu and Tanghua Shen.

Acknowledgments

Foremost, my sincere gratitude goes to my advisor Professor N. M. Patrikalakis for his expert guidance and strong support throughout the entire process. I also would like to thank Professor E. M. Sachs for his insight and valuable advice. I am also grateful to Dr. W. Cho, who helped me during my advisor's absence and provided useful comments on my thesis. My special thanks to Mr. T. R. Jackson, who put the first footprint on the FGM field that I followed. He showed me both warmth and wit throughout the project and gave me help in many various ways, such as providing me with sample models for testing. I also want to thank Dr. T. Maekawa, who helped me start research at MIT and showed me encouragement through out the last two years. I also cherish the experience of working with all the members of the Design Laboratory: Mr. G. Shen and Mr. G. Yu who helped me a lot during my tenure here apart from talking Chinese to me; Mr. F. Baker, Design Laboratory manager, without whose technical support my research would not be possible; Ms. K. Gunst for improving the English composition in an earlier draft of this thesis and all the rest of the Design Laboratory fellows whose friendship made my life at MIT easier.

Financial support of this project was provided in part by the National Science Foundation (grant #DMI-9617750) and by the Office of Naval Research (grant #N00014-96-1-0857). CAD models for this thesis were generated with SolidWorks^{TM}, meshed with Algor^{TM}.

Contents

List of Tables

List of Figures

List of Symbols

Symbol	**Definition**
n	Number of buckets in the bucketing domain of the object
n_{bf}	Number of boundary facets of the object
n_t	Number of tetrahedra in the tetrahedral mesh of the object
n_v	Number of vertices in the tetrahedral mesh
n_x, n_y, n_z	Number of buckets along axes x, y, z respectively
l_x, l_y, l_z	Length of object domain along axes x, y, z respectively
l_b	Side length of one bucket
x^*, y^*, z^*	Coordinates of a point in the bucketing frame
B_{ijk}	Bucket that is indexed i, j, k along axes x, y, z respectively
$\delta i, \delta j, \delta k$	Absolute differences between two buckets in terms of the indices of the buckets
B_q	The bucket that contains the query point
$ListB_{nr}$	The list of nearest non-empty buckets relative to B_q
T	The total time cost for an algorithm
T_{pre}	The time cost for preprocessing step in an algorithm
T_i	The time cost for distance computation for the ith query point
E_f	Efficient enhancement factor of distance function algorithm compared to exhaustive searching method
PT	Experimental result of preprocessing time
u, v, w, η	Barycentric coordinates of a query point in a tetrahedron
δt	Resolution in parametric form
P_a	Integral volume ratio of material **a** in the object
v_m	Volume of tetrahedron indexed m in the tetrahedral mesh
b_{kl}	Structure constant of a homogeneous incidence structure from k dimension element to l dimension element.

Chapter 1

Introduction

1.1 Background

Solid Freeform Fabrication (SFF) technology, also called Rapid Prototyping is a modern Computer Aided Manufacturing technology through which prototypes, parts, and tools are built in an additive fashion directly from CAD models. Among various SFF processes, the 3D Printing at MIT, the Selective Laser Sintering at University of Texas and the Shape Deposition Manufacturing at Carnegie Mellon and Stanford Universities are among the most prominent. 3D Printing [28] is one of the SFF manufacturing processes in which a 3D structure is built layer by layer and completed near-pointwise. Compared with the other SFF manufacturing processes, 3D Printing not only possesses the advantage of producing new complex solids that traditional technologies such as subtractive machining, forming or casting can not make or make efficiently, but also has better flexibility in exercising control over composition. Via selective placement of different materials available to the machine, 3DP can achieve near-pointwise local composition control (LCC). A 3DP machine exercises LCC in a fashion similar to ink-jet color printer printing which is illustrated in Figure 1-1.

The LCC characteristics of 3DP manufacturing opens the door to the production of material with graded local composition that is called Functionally Graded Material (FGM). FGM has many possible applications such as structural property control, thermal property control, medicine delivery control, multicolor visualization, etc.

In current rapid prototyping practices, designers usually create a model using a tradi-

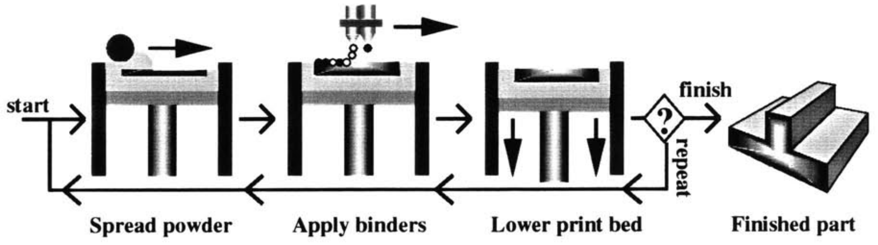


Figure 1-1: Functioning of LCC of 3D Printing, adapted from [16]

tional CAD system and obtain a tessellation of the boundary of the model in the form of a collection of triangles. The file format storing the model is an industry standard known as .STL [22]. The trimmed surface model is then processed into machine instructions to control the fabrication.

In order to fabricate an FGM part through LCC, it is necessary to build a CAD solid modeling system by capturing graded material composition and generating appropriate machine instructions. The traditional CAD systems do not facilitate such an implementation because the solid model they deal with is a digital representation of only the external geometry of a physical object and therefore do not permit easy LCC manufacturing. In order to go further to represent, design, and process models with graded material compositions, various research groups have investigated the extension methods concerning representing FGM from traditional solid representation. Among them T. R. Jackson *et al.* has proposed and developed a prototype FGM representation using an extension of cell-tuple data structure [5] with volumetric, FGM cells [16].

Regardless of what representation method is chosen, in order to represent graded material variation and plan the machine processing, the model is necessarily divided into sub-regions. At M.I.T. , a method for piping information from CAD system to the 3DP machine has been developed in order to produce an FGM part, see Figure 1-2.

1.2 Motivation and objectives

Although with the developed cell-tuple FGM modeling method, the users are able to capture their ideas as models with graded compositions and then convert these models into machine instructions for their fabrication, the modeling system also needs to be efficient in terms

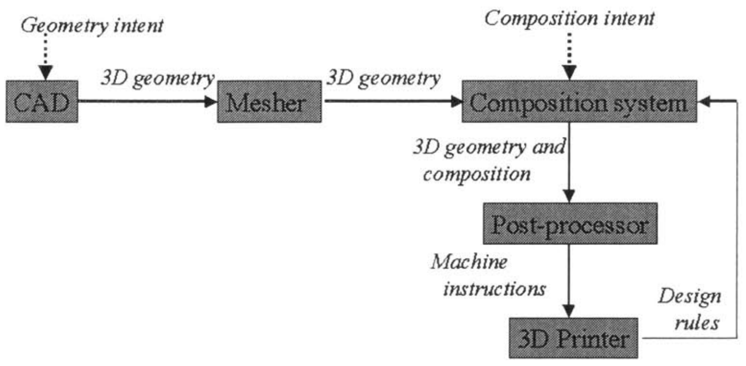


Figure 1-2: Information flow of FGM modeling system, adapted from [16]

of memory and speed of execution. For example, beginning with a model derived from triangulated, STL model, the expected FGM model with uniform mesh of tetrahedra is expected to be large. "In case of a cube subdivided into a structured mesh of tetrahedra, the relationship between the number of tetrahedra n_{tet} in the mesh and the number of boundary facets n_b in the STL model is: $n_{tet} = 5(n_b/12)^{3/2}$. For a small STL model of only 9408 facets, the corresponding FGM model of cube with uniform mesh would have 497225 cells of dimensions 0 to 3 (24389 vertices, 138852 edges, 224224 faces, 109760 tetrahedron regions) and requires a graph with 2690688 nodes to maintain the topology [16]." Estimating from this observation, the model from a large STL model will have prohibitive large size. Therefore, as stated before, for a large model it would be very useful to define a finite element mesh FGM cell for graded composition in order to reduce the memory cost for complete topology storage. Because it is desirable to achieve overall efficiency in terms of both memory and speed, we do not want to discard all the topology information that can help with faster query algorithms.

In addition, although the prototype system that Jackson *et al.*[15] developed based on cell-tuple struture provides a useful tool for designing compositions in terms of distance from a fixed feature in a straightforward manner, the efficiency of the distance function becomes an issue especially when designing from the boundary of a model. For example, the algorithm for assigning the control compositions must compute the minimum distance from each query point (corresponding to a control composition) to the boundary of the model. With the potential of a model having a large number of query points, the exhaustive searching through all the boundary facets may be prohibitively time consuming.

The CAD modeling system not only needs to provide design tools, but also needs to provide functionality for the user to evaluate the properties of the FGM model. The evaluation of composition of FGM will be most important for either the visualization or the post-processing of the FGM model. Query of the composition may be in the form of the query for a point, a ray, or a plane. Since the composition of the model is represented by finite number of control compositions, given for an arbitrary point inside the solid, it is necessary to evaluate the composition at that point using interpolation of the control compositions.

As part of the CAD system project of FGM for 3DP, this thesis work addresses the above efficiency problems and gives an effective solution.

1.3 Summary of methodologies

In order to model a general FGM cell with a finite element mesh efficiently, it is necessary to keep some of the topology information of the mesh. In the case that the model is converted from a tetrahedral mesh, the incidence relationship from a node to its incident tetrahedra is maintained, which helps to speed up the querying algorithms.

In this thesis work, an efficient distance function algorithm is developed based on the "Bucketing" algorithm and the digital distance transform of the buckets.

In our approach of the composition evaluation of an FGM model, an efficient point location algorithm is developed to identify the sub-region (tetrahedron) corresponding to the query point, and the control compositions of that tetrahedron are used to interpolate for the query point. An efficient point location algorithm is developed based also on the "Bucketing" method and the finite mesh structure with certain topology information and boundary facets. Once an efficient point location algorithm is available, the composition evaluation at a point is done via linear interpolation using barycentric Bernstein basis polynomials. Composition evaluation along a given ray or on a given plane can be easily developed by extending the composition evaluation algorithm at a point.

1.4 Thesis organization

Chapter 2 begins with a brief review of recent work on the representation and design of FGM objects.

As a memory saving choice for FGM object representation, a finite element based FGM model is described in chapter 3. In an attempt to facilitate efficient query algorithms, the finite element structure maintains certain topology information. In addition, the data structure and its relationship with a development environment of FGM modeling, design and processing is described. At the end of chapter 3, an algorithm for the extraction of the boundary facets is presented.

Chapter 4 contains the preprocessing methods of the finite element based FGM model that are essential in helping generate efficient queries used for both the efficient distance evaluation and efficient point location algorithms. The preprocessing of the model includes bucket sorting of the boundary facets and vertices, 3D digital distance transform, solid bucket identification and the point location algorithm. The point location algorithm is included here not only because it is closely related with the preprocessing method but because the algorithm itself is developed in order to enhance the efficiency of composition evaluation of the FGM model; therefore, the point location algorithm is also considered a preprocessing procedure.

In Chapter 5, the algorithm for design of FGM through the efficient distance function is presented. As the spirit of this algorithm, the efficient evaluation of the distance function to the surface of the model is analyzed and the experimental comparison with exhaustive searching method is given.

Chapter 6 provides the method for the evaluation of the composition of the FGM model, which is also useful in the rendering of the model. In addition, time complexity analysis of the method is presented.

The implementation of all the algorithms from Chapter 3 to Chapter 6 is given in Chapter 7 with numerical results on several sample models.

Chapter 8 concludes the thesis and provides potential directions for related future work.

Appendix A describes the implementation and integration of our algorithms into a software called FGMViewer along with a user's manual.

Appendix B provides pseudo-code for some of the algorithms developed in this work.

Chapter 2

Review

2.1 Representation of FGM objects

In order to achieve FGM object fabrication, researchers in SFF community have been investigating the method of representing FGM objects by extending existing CAD representation methods. The followings are some of the proposed methods. Among them, a cell-tuple structure based method has been developed sufficiently along with methods to transmit data to the $3DP^{TM}$ machine.

2.1.1 Decomposition based method

The traditional decomposition models represent objects by subdividing space into multiple sub-regions. This method is often used in finite element analysis, medical data rendering and so on by attaching physical properties to individual sub-regions. In order to represent FGM objects, the model can be enriched by attaching material information to each sub-region [25]. This method has the advantage in that there are a lot of volume graphics algorithms available, though the design and interrogation of FGM objects using this representation is cumbersome because it does not maintain topological information about the model. In addition, this method does not have the generality of describing free-form curves and surfaces; similarly, it is not easy to represent arbitrarily graded composition using this method. In terms of data exchange, methods compatible with the neutral standards such as IGES [12], STEP [13] need to be developed to exchange models based on the decomposition method. In the case of representing FGM with a large constant material composition area, using uniform decomposition (e.g. in a tetrahedral mesh) is not memory efficient compared to boundary representation and an abstract structure such as the cell-tuple representation.

2.1.2 Boundary representation based method

In current CAD systems, the boundary representation (B-rep) is most used because of its flexibility in modeling complex geometry precisely. Boundary representation describes solids in terms of their bounding entities such as shells, faces, loops, edges and vertices [2]. As described in Chapter 1, the traditional B-rep models do not contain material information explicitly. Based on one of the traditional B-rep models, the r-sets model, a heterogeneous solid model (r_m-set) is developed as a finite number of subdivisions with each subdivision being a material domain with a defined material variation function [19]. This approach is first proposed for representing composite models (material within each domain is constant) with the model constructed by using Boolean operators. Although in principle this method is able to represent graded composition by choosing varied material function for each domain, the transformation of the analytic composition variation information to specific process plans has not been presented.

2.1.3 Extended cell-tuple structure based method

In the traditional cell-tuple structure, a model M is represented in terms of a set of cells C where each cell c_k is a topological entity such as a vertex, edge, face or region. Here the edge and face can be arbitrary curved-entities and the region can be any valid manifold hemeomorphic to a topological open ball. All the component cells are connected through a graph T. Geometrically the model is determined by the geometric information associated with each cell (expect the region). A cell-tuple data structure can be constructed using data from a neutral standard format with or without approximation. Without approximation, the cell-tuple structure will be able to precisely describe the geometry of the original solid.

"To represent an FGM model within the cell-tuple structure, composition information as well as geometric information is also associated with each cell. The information begins with the concept of a material space M spanning the d_m primary materials available to an SFF machine capable of LCC. The composition of the model is represented as a vector valued function $\vec{m}(\vec{x})$ defined over the model's interior. Each component m_j of $\vec{m}$ represents the volume fraction of the corresponding material in the material system present at point $\vec{x}$ within the model" [16]. As a general abstract data structure, cell-tuple structure based FGM model has the possibility of describing graded material composition of arbitrary degrees. And because of the generality of its cells, one can even construct a cell of finite-element mesh, which is important for a large complicated model. In order to provide the capability of describing how the FGM composition varies within a solid, the FGM modeling has to decompose the interior of the solid into simpler sub-regions and each sub-region has the information about the composition variation in its domain. Theoretically, models can be arbitrarily subdivided into topologically simpler domains over which shape and composition functions can be more readily defined analytically. To simplify the procedure, the prototype FGM system at MIT begins with models subdivided into tetrahedral meshes, and the conversion from traditional solid models to FGM system is done by employing standard meshing algorithms.

2.2 Design of FGM

The design of FGM is another important phase in the whole FGM modeling process. In one of the definitions of "Solid Modeler", a solid modeling system is defined as a computer program that provides facilities for storing and manipulating data structures that represent the geometry of individual objects or assemblies [21].

Currently, there are two different categories of FGM design approaches. One is design in top-down fashion, in which the CAD model is decomposed into simpler geometry sub-domains, and then the designer designs graded composition over all the sub-domains. For example, the system developed at MIT provides composition functions, especially a graded composition in terms of volume fractions of the material over the domain of each sub-region. Over each cell's domain c_k, the shape and composition is formulated in terms of a set of control points and control compositions which are blended with the barycentric Bernstein polynomials [11]. The degrees of control points and control compositions are determined according to the degree of variation of the geometry and composition of cells. With each control composition of the model representing a degree of freedom, the design of the FGM parts becomes the procedure of assigning values to each and blending over the whole domain. The design tool that helps in design of control compositions in terms of distance functions to a selected feature is developed. The selected feature may be a fixed reference in the model space, such as a point, line or plane, or a feature of the model, such as a particular face or its entire boundary, or an independent boundary shell in .STL format. After a feature is selected, the designer specifies a variation for the FGM in terms of distance from the feature: $\vec{m}(\vec{x}^*) = \vec{m}(r(\vec{x}^*))$, where r is the distance of a query point x^* from the reference feature. Next, the design tool automatically visits and assigns the control compositions for each cell, and in this way defines the composition over the whole model domain. The other approach is to design FGM by composition using a library of predefined components. The composing of different components can be done by using operators specific to the chosen data structures. This approach has not yet been fully explored but preliminary results are reported by researchers at Stanford University and the University of Michigan [3][26].

Chapter 3

Finite element based FGM model

3.1 Introduction

In Chapter 2, we have reviewed three proposed approaches of representation of FGM objects and one can see each method has its own advantages and disadvantages. In terms of generality, flexibility and approximation precision, the B-Rep and Cell-Tuple approaches are obviously better choices. But once we have the objective to provide the whole pipeline including modeling, design of graded material composition and process planning into SFF fabrication processes that are capable of LCC, it is necessary to discretize the model either before the design phase, during this phase, before the process planning or embedding in the process planning phase. The choice of designing composition before subdivision will limit the design function in the scope of analytic function; subdividing before the interactive design of composition cannot take into account the user's design intentions, etc, therefore it is not ideal either; the ideal method therefore is to subdivide the model in the process of design according to user's design intention and design rules that are defined in an expert system.

Although the discretization is a very important issue in FGM modeling, it has not been studied in depth yet. For example, in order to achieve FGM production, the approach based on B-rep (extended r-set) method [20] [26] hasn't addressed automatic subdivision issues and has only achieved composite composition, which means piecewise constant material composition in each sub-region. In the case of graded composition, the proposed design method [18] is to attach an analytical function to each sub-region, which will leave further discretization to process planning, and also incur further problem of modeling designer's (users) intuitive design intention in terms of analytic function in each sub-region. In comparison, the approach based on cell-tuple structure [16] has simplified the subdivision procedure via subdividing the traditional CAD model in neutral standard format into meshes of simpler subdivisions using commercial software, as described in Chapter 1 and 2. This approach has a memory bottleneck in the case of large dense uniform finite element mesh input, because a cell-tuple structure maintains complete topology information between the topology entities including internal structures.

From the above analysis, we can see the modeling of FGM still has a lot of open questions and problems to solve, therefore it is difficult to draw a conclusion with an optimum solution. When the development of a cell-tuple structure based on FGM modeling met the obstacle of memory bottleneck for large mesh input, questions were raised such as "Which is the best choice; cell-tuple alone, finite element alone, or a mix of these two?". In this context, research was carried out on using a finite element based approach to represent FGM.

It was decided to develop algorithms based on a finite element approach because both the finite element based approach and the cell-tuple with finite element based approach involve a finite element subdivision and it turns out that a finite element based structure is easier for implementation. Under the assumption that we want to achieve efficiency based on a decomposition approach, it was also decided to keep some of the topology information of the model for better efficiency in query algorithms. Meanwhile, as an effort of developing a newer approach of the FGM system, an exploratory environment of modeling, designing, and post-processing of FGM has been developed and a pure finite element representation implemented. Therefore it is natural to place this thesis work into that environment. Figure 3-1 demonstrates the inheritance tree of the finite element mesh with some topology in this thesis from the implemented classes of that exploratory environment [14]. Following this introduction, the data structure used in this work will be described and also the algorithm of extraction of the boudary facets from the input finite element mesh is presented. Appendix A.1 also gives the extension of the computer-user interface developed in this work.

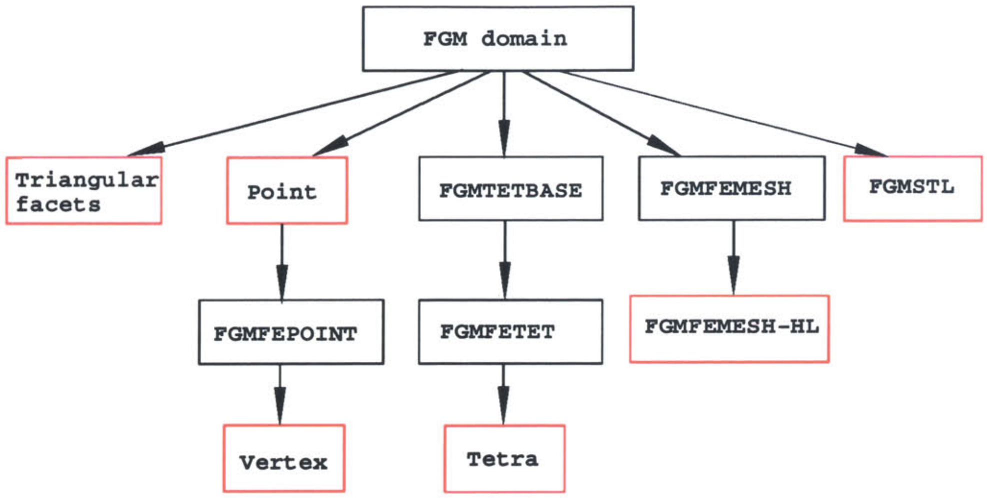
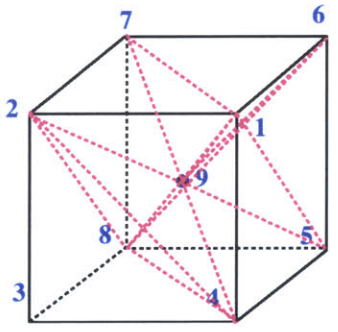


Figure 3-1: Inheritance tree of FGMviewer object classes



Tetrahedra of the model:	Vertices of the model:
1 = { 2 , 7, 8, 9 }	1∈ (2, 4, 6, 7, 8)
2 = { 1, 2, 7, 9 }	2∈ (1, 2, 3, 4, 5)
3 = { 2, 4, 8, 9 }	3∈ (5)
4 = { 1, 2, 4, 9 }	4∈ (3, 4, 5, 6, 9)
5 = { 2, 3, 4, 8 }	5∈ (6, 8, 9, 10)
6 = { 1, 4, 5, 9 }	6∈ (7, 8, 10, 11)
7 = { 1, 6, 7, 9 }	7∈ (1, 2, 7, 11)
8 = { 1, 5, 6, 9 }	8∈ (1, 3, 5, 9, 10, 11)
9 = { 4, 5, 8, 9 }	9∈ (1, 2, 3, 4, 6, 7, 8, 9, 10, 11)
10 = { 5, 6, 8, 9 }	
11 = { 6, 7, 8, 9 }	

Figure 3-2: Data Structure: Cube example

3.2 Data structure

The data structure of an efficient finite element based FGM model is presented here. Specifically, the data structure maintains an array of object "Vertex" pointers, an array of objects "Tetrahedron" pointers, a list of indices of the tetrahedra that are boundary tetrahedra and a bucketing system. The bucketing system has n number of buckets and several bucketing parameters, where n is the number of boundary facets. Each bucket is associated with a list of triangular boundary facets and a list of vertices. An object of "Vertex" is associated with the geometric position of the vertex, a queue of incident tetrahedra indices and the material composition vector at that vertex. A tetrahedron has an array of four vertices, and the status of each tetrahedron with respect to it being a boundary tetrahedron or not; in addition, if a tetrahedron is a boundary tetrahedron, the status of each face is stored according to the face being a boundary face. Figures 3-2 and 3-3 demonstrate the data structure for representing a model of a cube split into 11 tetrahedra.

3.3 Algorithm for the extraction of surface boundary

Because the finite element mesh does not have explicit surface boundary storage, it is necessary to extract the boundary information for future use. This is done in the process of initialization of the data structure. The initialization of the data structure and the extraction of boundary procedures are as follow:

Algorithm 1 initDataStr(FEMesh M); initialize the data structure from finite element mesh generated from *Algor*

```
1:  for each node $Nd \in M$ do
2:    construct new node $newNd \in newM$;
3:  for each tetrahedron $T \in M$ do
4:    initialize $newT \in newM$;
5:    set each face status as exterior;
6:    set four associated vertex pointers to newT;
7:    for each of the four vertices do
8:      add newT to the corresponding incidence list;
9:    for each face $F \in newT$ do
10:     check if $F$ is interior and set the status; ▷see Algorithm 2
11: initialize $BTL$;                ▷$BTL$ is the boundary tetrahedra list
12: for each $newT \in newM$ do
13:   if $newT$ is boundary tetrahedron then
14:     add $newT$ to $BTL$;
15:   else
16:     delete face information.
```

Based on the assumption that the finite element mesh is conforming, for an interior face, the two incident tetrahedra should appear in all the three parent tetrahedra lists according to the three vertices of the face. Therefore, the two incident tetrahedra should each appear 3 times in the combined list of the parent tetrahedra lists of the three vertices. Under the above assumption and observation, the algorithm for checking if a face of a tetrahedron is interior is done as follows:

Algorithm 2 isInterior(faceNo F, Tetrahedron T); check if F $\in$ T is interior face

```
1: if F has the status as interior then
2:   return;
3: else
4:   for each vertex ∈ F do
5:     check out the incidence tetrahedra list;
6:   if the three incidence tetrahedra lists overlap then
7:     put the three lists into one list;
8:     count the occurrence of each element of the list;
9:     if there exists $T_1 \neq$ T counted 3 times then
10:      set F as an interior facet in T and $T_1$;
11:    else
12:      status remain;
13:  else
14:    no, the face status remains unchanged.
```

Figure 3.3 shows how the algorithm of checking interior faces works.

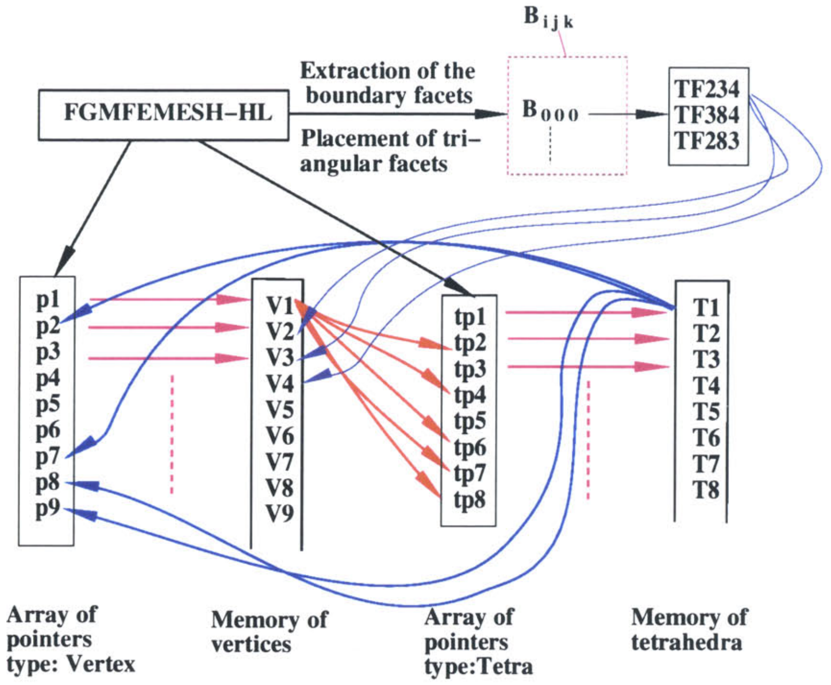
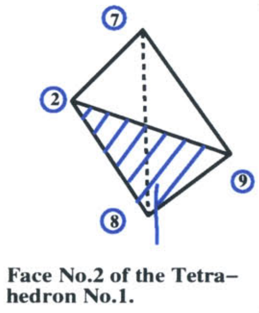


Figure 3-3: Data structure



Vertices	Parent TetraList	Integer Interval
2	(1, 2, 3, 4, 5)	[1, 5]
8	(1, 3, 5, 9, 10, 11)	[1, 11]
9	(1, 2, 3, 4, 6, 7, 8, 9, 10, 11)	[1, 11]

Do if the three integer intervals overlap:

a: Count the occurence of elements in the three TetraLists; Tetrahedron No. 1 and No. 3 counted 3 times.

b: Face No. 2 of Tetrahedron No. 1 is incident to both Tetrahedron No. 1 and No. 3, i.e. set face No. 2 as interior facet.

Figure 3-4: Check to see if a face of a tetrahedron is interior

Chapter 4

Preprocessing of finite element FGM model

4.1 Introduction

In order to improve the efficiency for computing minimum distance from a query point to the boundary of the model or a given .STL boundary, a preprocessing method using bucket sorting technique [8] and digital distance transform [4] is employed. After such preprocessing, computation of the Euclidean distance for a specific query point has to be done with respect to triangular facets in the buckets which have the nearest digital distance to the query point. Figure 4-1 graphically illustrates this preprocessing. Based on the bucketing processing, an efficient 'Point Location' algorithm is developed, which is very essential to the efficient evaluation of the composition of an FGM model. Because the 'Point Location' algorithm is very much related with the 'Bucketing' processing, it is also placed in this chapter together with the 'Bucketing' preprocessing.

4.2 Computation of the boundary facets of the model

After the initialization of the data structure, the boundary tetrahedra list is obtained as described in the previous chapter and each boundary tetrahedron has its boundary facets identified explicitly. Based on this boundary tetrahedra list, an array of triangular facets is

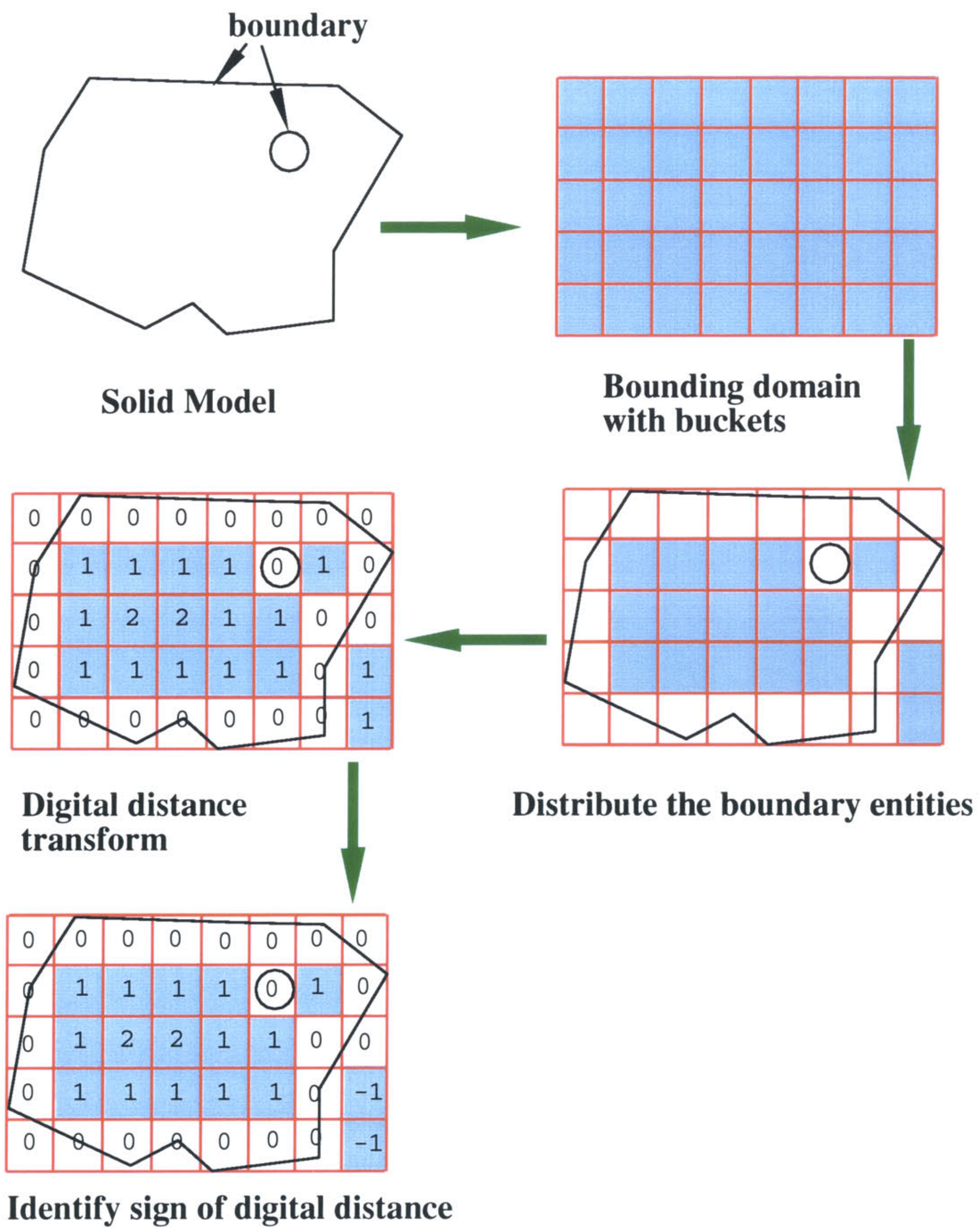

boundary
Solid Model
Bounding domain with buckets
Distribute the boundary entities
Digital distance transform
Identify sign of digital distance

Figure 4-1: Preprocessing

constructed with their normal vectors calculated.

Algorithm 3 getTriFacets(BndTetList BTL); construct the boundary facets list from boundary tetrahedra list

```
1: initialize $TriFacetList$;
2: for each boundary tetrahedron $BT \in BTL$ do
3:   for each F ($\in [0, 3]$) of T do
4:     if F is a boundary facet then
5:       initialize $Tri$;
6:       calTrifrmF($Tri$,F,T);
7:       add $Tri$ to $TriFacetList$;
```

Algorithm 4 calTrifrmF(Triangle Tri, faceNo F, Tetrahedron T) refer to Figure 4-2

```
1: set vertices of Tri;
2: identify vertex D;
3: if $\vec{CA} \times \vec{CB} \cdot \vec{CD} > 0$ then
4:   $$\vec{N} = -\frac{\vec{CA} \times \vec{CB}}{| \vec{CA} \times \vec{CB} |}$$
                                        ▷N is the normal vector of Tri
5: else
6:   $$\vec{N} = \frac{\vec{CA} \times \vec{CB}}{| \vec{CA} \times \vec{CB} |}$$
```

4.3 Construction of 3D bucketing system

Given n numbers in [0,1),$K_1, K_2, ...K_n$, the bucketing technique [8] in one dimension divides the interval [0,1) into n equal-sized subintervals, or buckets, and then distributes the n input numbers into the buckets. The analogous bucketing process in 3D [6][7] is essentially dividing the bounding box of the model into equal sized cubic sub-regions (buckets). Given the number of facets of a solid model (n), we build the bucket system such that $n_x \times n_y \times n_z = n$, where n_x, n_y and n_z are the number of buckets along the Cartesian coordinate axes x, y, z respectively. If we define l_x, l_y, l_z as the lengths of the object along x, y, z axes, in order to

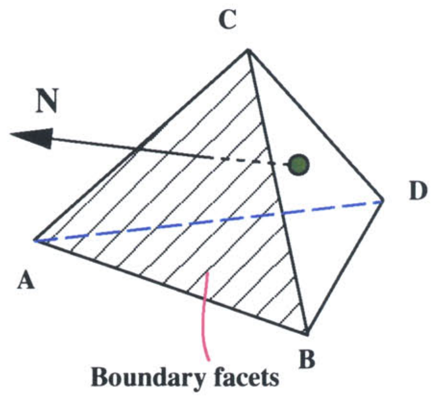


Figure 4-2: Calculating the normal vector of a boundary facet

build cubic buckets, n_x, n_y and n_z should obey the following formula

$$n_x = \lfloor \sqrt[3]{\frac{n \cdot l_x^2}{l_y \cdot l_z}} \rfloor; \qquad n_y = \lfloor \sqrt[3]{\frac{n \cdot l_y^2}{l_x \cdot l_z}} \rfloor; \qquad n_z = \lfloor \sqrt[3]{\frac{n \cdot l_z^2}{l_x \cdot l_y}} \rfloor$$

that relates them to the dimension lengths of the object, and therefore the side length of one bucket is

$$l_b = \sqrt[3]{\frac{l_x \cdot l_y \cdot l_z}{n}}.$$

Apparently l_x , l_y and l_z can be found by computing the minimum of x_i, minimum of y_i, minimum z_i, maximum of x_i, maximum of y_i, maximum of z_i. After we build the bucketing frame system that originates at $(x_{min}, y_{min}, z_{min})$, we need to transfer the old coordinate position of each vertex of the object to its new position in the new frame system. The transform formulae are as follows

$$x^* = \frac{x - x_{min}}{l_b}; \; y^* = \frac{y - y_{min}}{l_b}; \; z^* = \frac{z - z_{min}}{l_b}$$

In the bucketing system as in Figure 4-3, a bucket that is the i^{th} from left, the j^{th} from front and the k^{th} from the bottom is denoted B_{ijk}.

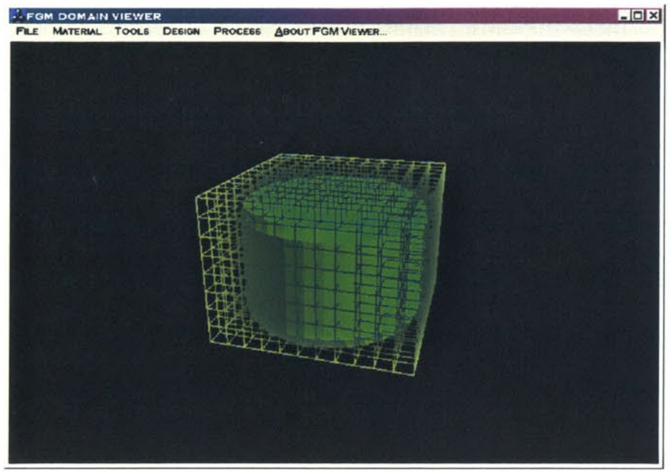


Figure 4-3: Bounding box with buckets

4.4 Placement of the triangular facets into buckets

The purpose of this procedure is to build the reference for each triangular facet on the object boundary to the buckets that intersect it. For one single facet, the intersection detection is divided into three phases: the first step is to find buckets that intersect the vertices of the facet; the second step is to find buckets that intersect the edges of the facet, when there is edge intersecting more than two buckets; the third step is to find buckets that pass through the facet without intersecting the edges of the facet. All three steps are illustrated in Figure 4-4.

STEP 1 *Find buckets that intersect the vertices of the facet*

This can be done easily by taking the floor (or integer part) of the bucketing coordinates of each vertex. The resulting integer coordinates are the bucket indices. $i = \lfloor x^* \rfloor, j = \lfloor y^* \rfloor, k = \lfloor z^* \rfloor$

STEP 2 *Find buckets that intersect the edges of the facet*

After the buckets that intersect the vertices of the facet are obtained, we can trace on each edge from one end point bucket to the new bucket that intersects the edge until the other end. This can be done by identifying the side of the six sides of the current bucket that is intersected by the edge in the tracing procedure. If the current bucket is intersected at right, the index i of the new bucket should be increased by

1, while decreased by 1 if intersected at left. Similar operations can be done if the current bucket is intersected at other sides.

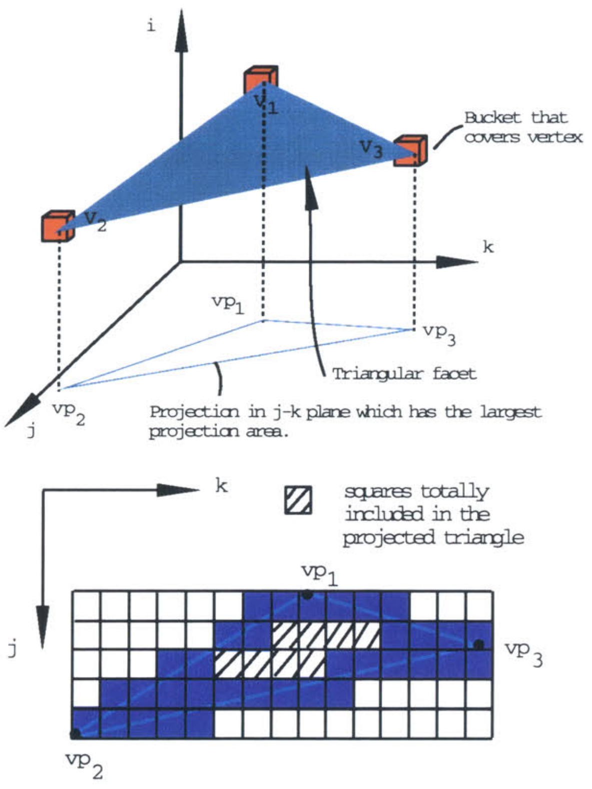

i
k
j
v_1
v_2
v_3
Bucket that covers vertex
vp_1
vp_2
vp_3
Triangular facet
Projection in j-k plane which has the largest projection area.
▨ squares totally included in the projected triangle

Figure 4-4: Distribute triangular facets into buckets

STEP 3 *Find buckets that pass through the facet but not intersect the edges of the triangular facet*

The detection work is divided into two parts. First, we find two of the three indices of each such bucket by projecting the triangular facet along its maximum normal direction, which means the direction that the normal vector has the maximum value of projection area. For example, if the normal vector to a facet is (n_x, n_y, n_z) and $|n_x| \geq |n_y|, |n_z|$, we project the triangular facet into $y - z$ plane, and then first find the index j, k of the bucket if square (j, k) is completely inside the projected triangle. Since we have found buckets that intersect the edges, we can just scan for each possible k to see if there are some j_0 that satisfies $j_l(k) < j_0 < j_u(k)$, where $j_l(\mathrm{k})$ means the index j of the projected square that is on the lower edge and has index k. If j_0

```
* * * * * * *        6 5 4 3 4 5 6
* * * * * * *        5 4 3 2 3 4 5
* * * * * * *        4 3 2 1 2 3 4
* * * 0 1 2 3        3 2 1 0 1 2 3
* * * 1 2 3 4        4 3 2 1 2 3 4
* * * 2 3 4 5        5 4 3 2 3 4 5
* * * 3 4 5 6        6 5 4 3 4 5 6

After forward pass   After backward pass
```

Figure 4-5: Example of chamfer distance transform: 0 is the feature pixel

```
* * * * * * *        3 3 3 3 3 3 3
* * * * * * *        3 2 2 2 2 2 3
* * * * * * *        3 2 1 1 1 2 3
* * * 0 1 2 3        3 2 1 0 1 2 3
* * 1 1 1 2 3        3 2 1 1 1 2 3
* 2 2 2 2 2 3        3 2 2 2 2 2 3
3 3 3 3 3 3 3        3 3 3 3 3 3 3

After forward pass   After backward pass
```

Figure 4-6: Example of chessboard distance transform: 0 is the feature pixel

exists, then we find two of the indices of the bucket (see Figure 4-4). The next step is to find the last index, which we can calculate by computing intersections of four lines with the facet. These four lines are $\{(j = j_0) \cap (k = k_0)\}$, $\{(j = j_0 + 1) \cap (k = k_0)\}$, $\{(j = j_0) \cap (k = k_0 + 1)\}$, $\{(j = j_0 + 1) \cap (k = k_0 + 1)\}$. Finally, for each bucket that intersects the boundary, we obtain a list of triangular facets which are contained in the bucket.

4.5 3D digital distance transform

A distance transform (DT) in 2D is an operation that converts a binary picture, consisting of feature and nonfeature elements, to a picture where each element has a value that approximates the distance to the nearest feature element [4]. Borgefors [4] has extensively studied digital distance transforms in arbitrary dimension for different families of digital distances. Among different families of digital distances, the most popular one is the city block/chessboard distance family. The algorithm for this family of distance transform is given via examples in Figures 4-5 and 4-6.

In our algorithm we use 3D chessboard distance transform to compute for each bucket the corresponding chessboard distance to the boundary buckets. Chessboard distance is defined such that for a pair of buckets A(i_1,j_1,k_1) and B(i_2,j_2,k_2), it is equal to $max(\delta i, \delta j, \delta k)$, where $\delta i = |i_1 - i_2|$ and $\delta j, \delta k$ are similarly defined. We can see graphically the difference between Chessboard distance and Euclidean distance from the fact that buckets at equal chessboard distance from a bucket form a cubic shell while points at distance from a point form a sphere. Due to the symmetry of distance ($d(p,q) = d(q,p)$), we can conclude the nearest boundary buckets to a specific bucket are located in the cubic shell which has unit thickness and an offset equal to the digital distance. Although in our algorithm 3-D DT is

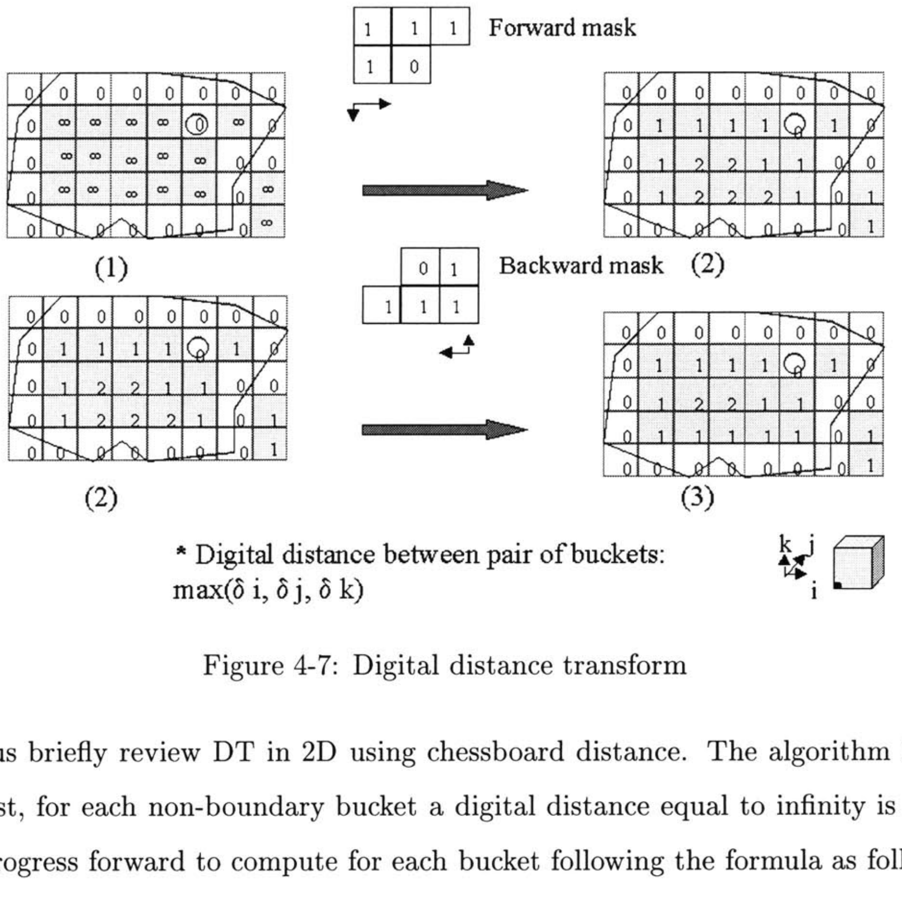


Figure 4-7: Digital distance transform

used, let us briefly review DT in 2D using chessboard distance. The algorithm has three steps. First, for each non-boundary bucket a digital distance equal to infinity is assigned. Second, progress forward to compute for each bucket following the formula as follows:

$$v_{i,j}^{new} = min(v_{i,j}^{curr}, v_{i-1,j}^{curr} + 1, v_{i-1,j-1}^{curr} + 1, v_{i,j-1}^{curr} + 1, v_{i+1,j-1}^{curr} + 1)$$

Third, progress backward for each bucket and compute following the formula:

$$v_{i,j}^{new} = min(v_{i,j}^{curr}, v_{i+1,j}^{curr} + 1, v_{i-1,j+1}^{curr} + 1, v_{i,j+1}^{curr} + 1, v_{i+1,j+1}^{curr} + 1)$$

Here, $v_{i,j}$ is the value of digital distance for a raster unit (i^{th}, j^{th}). Graphically speaking, the operation for step 2 and step 3 is positioning the corresponding mask with 0 value square covering the bucket, the new value for the bucket is the minimum of the five sums of pair of mask value and current bucket value (Figure 4-7). Similar scheme can be applied to 3D cases, the masks would be like in Figure 4-8 that is composed of two planar masks in two planes. Considering that one 3D voxel has 26 neighbors, in each path of the transform, the masks will transform 13 neighbors which is consistent with the 13 masks apart from the feature voxel. The algorithm is also adapted for the buckets that are on the border of the bounding box. For those buckets, the number of operators in mask should be reduced accordingly.

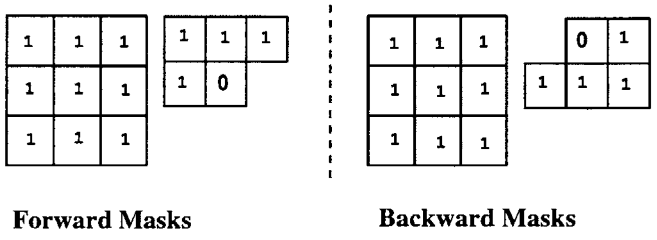


Figure 4-8: Chessboard DT Mask in 3D

4.6 Identification of solid buckets

From the previous work on the distance transform, the algorithm has identified the boundary buckets and non-boundary buckets. Here for the purpose of efficiently searching the sub-region location of a query point with Cartesian coordinates, the method involves processing of all the non-boundary buckets in order to identify all the buckets that are inside the solid. For convenience we call such buckets 'solid buckets'. Figure 4-9 illustrates the algorithm.

The method for identifying the solid buckets involves assigning a sign to the digital distance value of each bucket. Solid buckets are non-boundary buckets, therefore, as long as one point inside a bucket is inside the solid, then the bucket is a solid bucket. So the algorithm is to use the center point of a non-boundary bucket as seed, check if this seed is inside the solid. If the seed is inside the solid then the corresponding bucket is a solid bucket, otherwise, the bucket is outside the solid. Therefore, the problem essentially

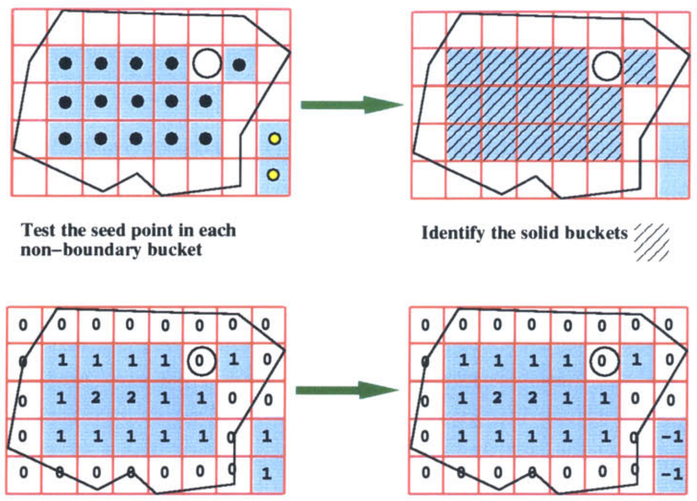

Test the seed point in each non−boundary bucket
Identify the solid buckets

Figure 4-9: Identify the signs of non-boundary buckets

becomes checking if a point is inside the solid, which will be explained in detail later together with the point location algorithm.

4.7 Bucketing vertices

After the bucketing processing of boundary facets, all the vertices of finite element mesh are also classified with respect to buckets, which will help the point classification queries of the model. Figure 4-10 illustrates the method, in which the white vertices are representative vertices. The procedure is simply taking the integer parts of each vertex's bucket coordinates, and insert the vertex into the list of vertices of the corresponding bucket. For each bucket, the vertex that has the minimum number of parent tetrahedra is considered as the representative vertex of that bucket.

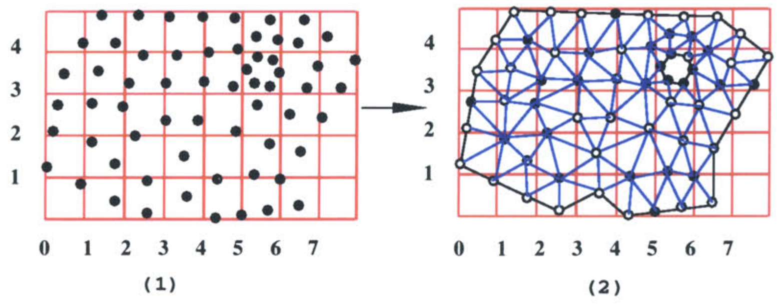


Figure 4-10: Bucketing vertices based on the original bucket system

4.8 Point location algorithm

Point location [24] is one of the classical problems in computational geometry and has been extensively studied. One of the optimal algorithms is "Bucketing", see Asano et al.[1]. This method can even achieve constant time for a uniformly distributed mesh and is chosen here. "Bucketing" method is chosen also for the purpose of distance function evaluation. Here in our application we define the problem as follows:

Given a query point Q,

If Q is not in the body, return not in;

If inside the body, return the tetrahedron that contains it.

4.8.1 Point membership classification (PMC)

Point Membership Classification involves testing if a given query point Q is contained inside the body (or on its boundary) or not. The most popular algorithm of PMC with respect to solids represented by their boundary is ray-casting. In our method, the steps are as follows: Shooting a ray from Q to one of the six coordinate directions, using the stored digital distance value to find the *first* boundary bucket in that direction, start from this bucket to find the first boundary triangular facet the ray intersects. In this work, the shooting direction is chosen such that the bounding box is nearest in that direction. By checking the inner product of ray vector and the oriented normal vector of the triangular facet, one can determine if a point is inside the solid; if the inner product is positive then the point is interior otherwise exterior. Here the method for detecting the first boundary triangular facet the ray intersects is to compare the parameter values of the intersection points. The procedure is illustrated in Figure 4-11, where the thick-lined facet is the first boundary facet that the ray intersects. If the intersection points happen to coincide, then the triangular facet that has the smaller minimum distance to the query point will be chosen. If the facets involved have the same minimum distance to the query point, then the boundary triangular facet that has the bigger inner product magnitude with the ray is chosen. Figure 4-12 demonstrates two special cases of identifying the closest triangular facets relative to the shooting ray, where d_a is the minimum distance to facet **a** and d_b is the minimum distance

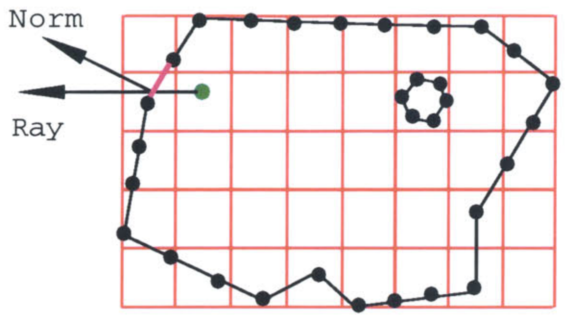
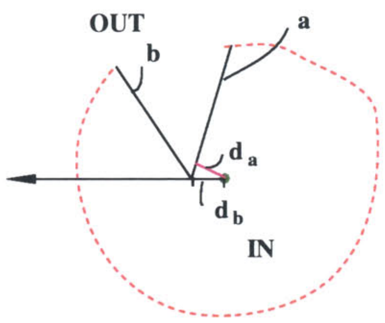


Figure 4-11: Check if a given point is inside the body

to facet **b**.



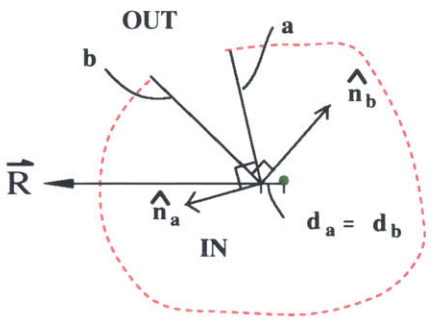


Figure 4-12: Intersection points coincide: $d_a < d_b$, choose facet **a**

Figure 4-13: Intersection points coincide: $d_a = d_b$, $|\hat{n_a} \cdot \vec{R}| > |\hat{n_b} \cdot \vec{R}|$, choose facet **a**.

One other special case is that the intersection point has parameter 0, which means the query point is on the boundary, hence we can use the corresponding boundary facet to identify the tetrahedron it is located in.

4.8.2 Identification of the object tetrahedron

Once the algorithm concludes a query point is contained in the body, the identification becomes finding the tetrahedron that contains the query point. Using an exhaustive searching method (checking every mesh element in the mesh) will be prohibitively slow in case of a large FEM model and large number of query points, therefore development of an efficient method is necessary. Here an algorithm relying on the bucketing of FEM vertices and the topology information is developed. The idea of locating the object tetrahedron is to jump onto a vertex that is near to the query point with the help of bucket sorting of vertices and then trace the straightline segment between the starting vertex and the query point to search and check each tetrahedron that intersects the line segment. The idea is illustrated in Figure 4-14.

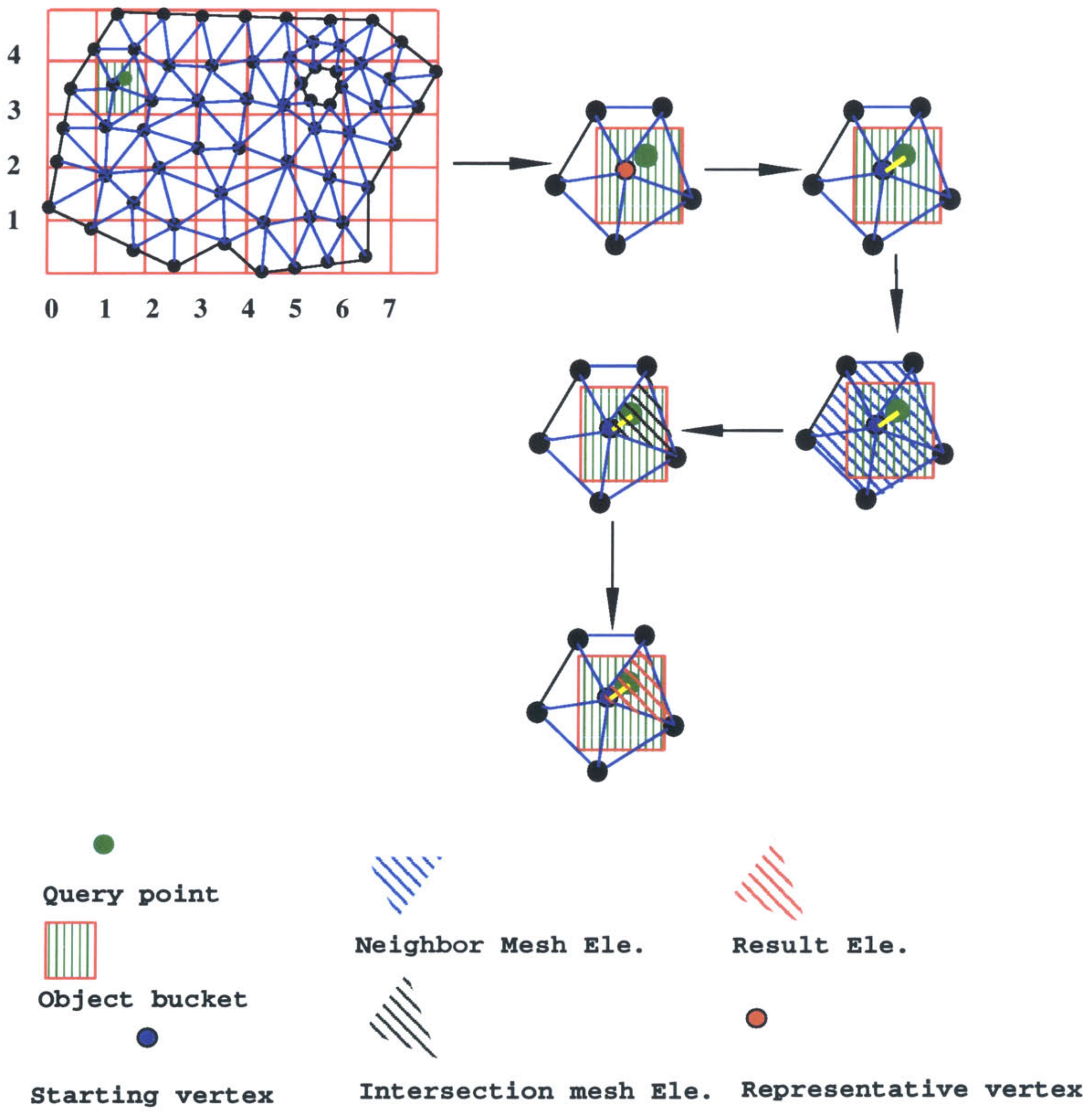


Figure 4-14: Idea of locating the tetrahedron containing a given point

The following pseudo-code describes the algorithm.

In order to make the algorithm easier to understand, several cases of the input that are handled are described in the following and illustrated in Figures 4-15 to 4-17.

Algorithm 5 pointLocus(Point Q); find the tetrahedron that contains Q

```
Require: The query point is correctly classified as interior point.
 1: StartVert ⇐ findStartVertex(Q);        ▷Q is the query point ;StartVert is the
    starting vertex
 2: while StartVert ≠ NULL do
 3:   StartTet ⇐ findStartTetra(Q, StartVert);        ▷StartTet is the starting tetra
 4:   define Ray myRay;
 5:   myRay ⇐ (StartVert → Q);
 6:   define point R;
 7:   if StartTet = NULL then
 8:     StartTet ⇐ newStartTetra(Q,R);
 9:     myRay ⇐ (R → Q);
10:     modeFlag = true;        ▷Starting with a nearest boundary tetrahedron
11:   TrialTet ⇐ StartTet;
12:   while TrialTet does not contain Q do
13:     define newStartVert;
14:     TrialTet ⇐ findNextTetra(TrialTet, myRay, newStartVert);
15:     if TrialTet = NULL then
16:       if newStartVert ≠ NULL then
17:         StartVert ⇐ newStartVert;
18:         break;
19:       else
20:         StartVert ⇐ NULL;
21:       TrialTet ⇐ newStartTetra(Q,R);
22:       if modeFlag = true then
23:         exit;        ▷Q is not interior point
24:     else
25:       StartVert ⇐ NULL;
26:   objTet ⇐ TrialTet;
27: return objTet;
```

Case 1 This is the situation that the query point is connected with the starting vertex by a solid straight line segment; the algorithm for this situation is as simple as checking out all the tetrahedra along the segment by querying the neighboring tetrahedron along the line segment one by one until the object found. Here the solid straight line segment means every abstract point on the straight line segment is occupied by the solid. Figure 4-15 demonstrates this aspect of the algorithm.

Case 2 This is the situation that the straightline segment between the query point and the starting vertex is not solid, in other words, there are points on this line segment that are not contained in the solid. The algorithm for this situation is a modification of that for **Case 1** in that, at some step, when the algorithm for **Case 1** can not find the neighboring tetrahedron along the line segment, an alternative algorithm of finding the nearest boundary facet the ray intersects is employed. Starting with the found triangular facet, the tracing of tetrahedron along the line segment until location is resumed. Figure 4-16 demonstrates this aspect of the algorithm.

Special case This is the situation that when the straight line segment between the query point and the starting vertex intersects a vertex of the mesh. To deal with this situation, the algorithm has to be modified to update the starting vertex with the newly encountered vertex. The case of the line segment intersects an edge of the mesh is dealt with by updating the starting vertex with the nearest vertex on that edge.

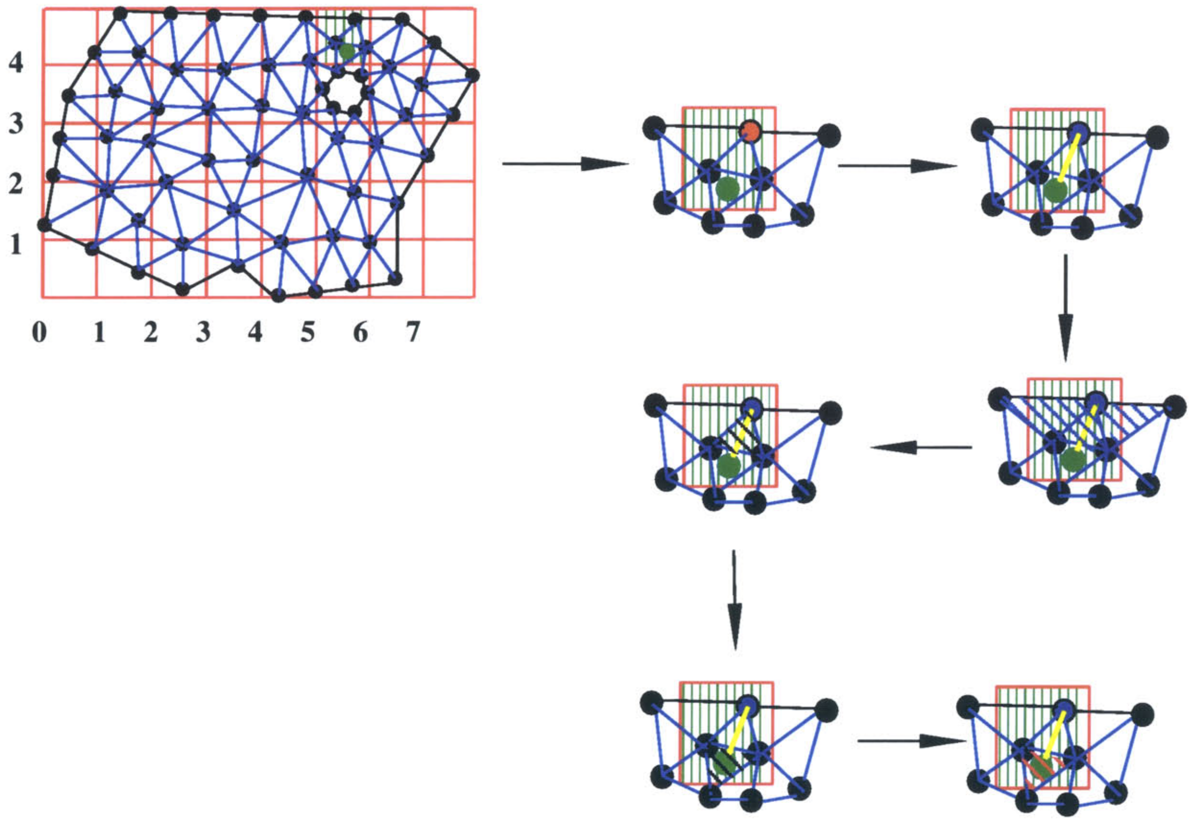


Figure 4-15: Case One: query point is connected with the starting vertex by a solid straight line segment

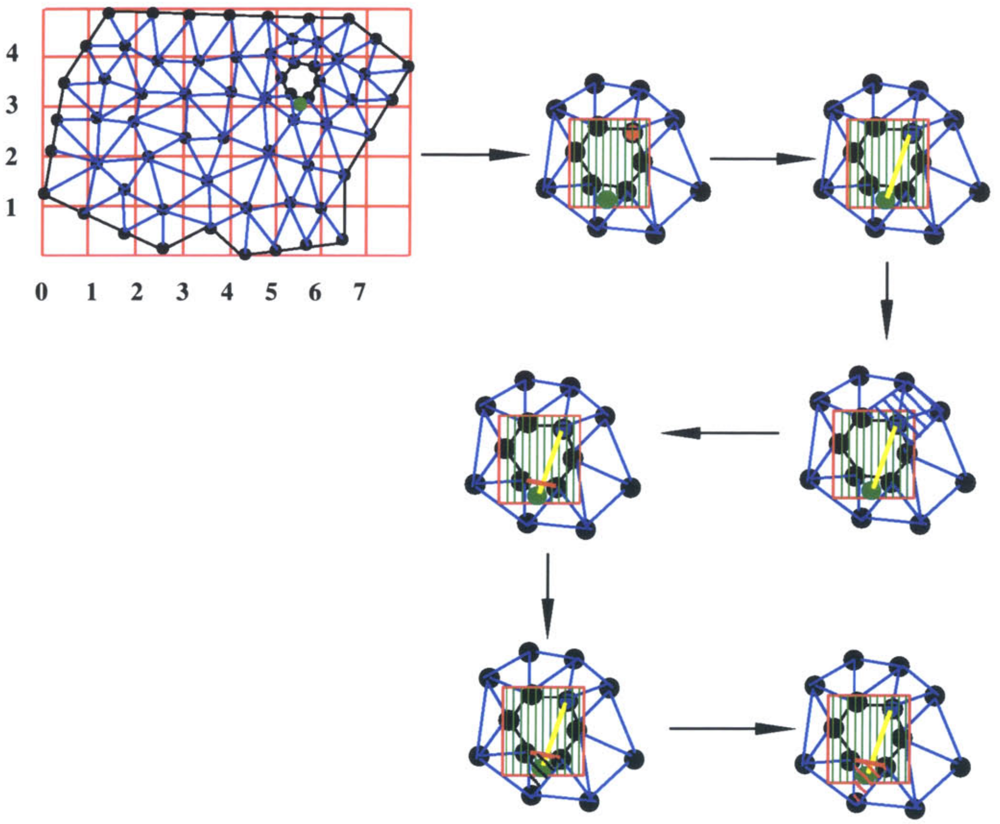


Figure 4-16: Case Two: Straight line segment between the query point and the starting vertex is not solid

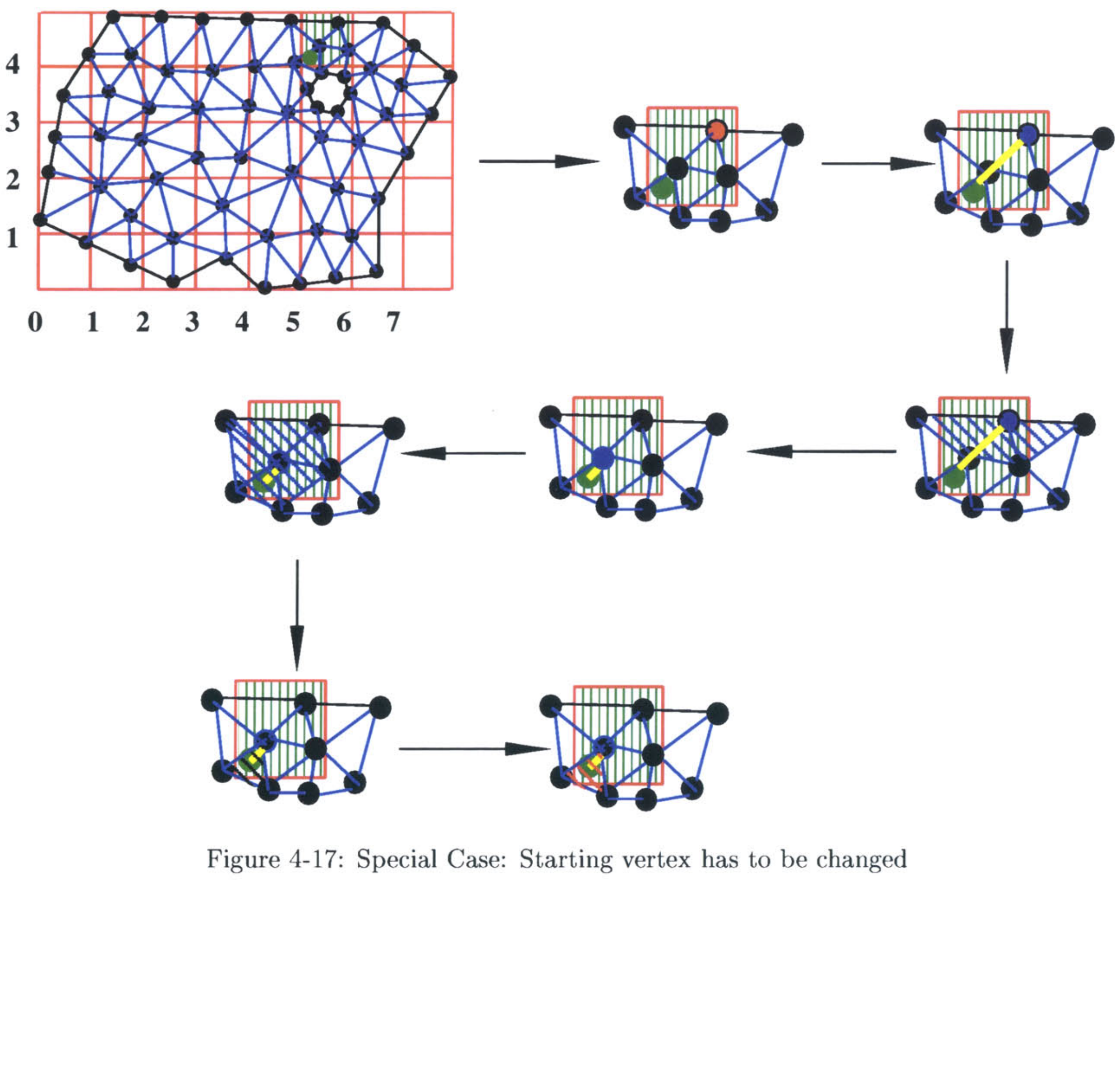


Figure 4-17: Special Case: Starting vertex has to be changed

Chapter 5

Design of FGM solid using efficient distance function algorithm

5.1 Introduction

Based on the developed FGM modeling system, the design of graded material composition can be done by assigning composition value on each control composition vertex inside the model. Nevertheless, this approach is ineffective because a model may have millions of cells and millions of control compositions to assign. Furthermore, the approach of directly assigning is not intuitive for users to implement their design ideas. Therefore, an efficient design tool needs to be developed. One of the methods of designing FGM is to assign com-

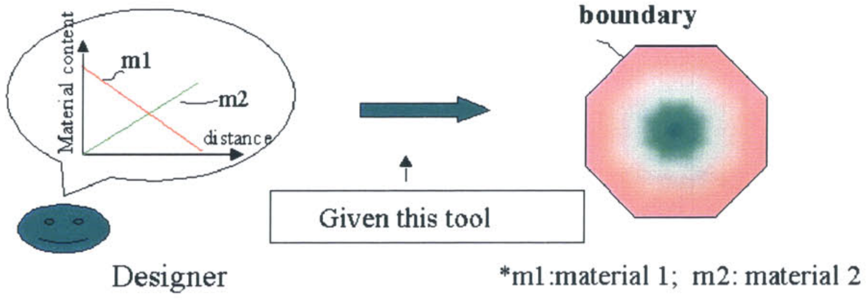


Figure 5-1: Idea of design FGM from the boundary of object

position to the control compositions according to its minimum distance to the boundary of the object. For a single point inside the FGM body, the minimum distance to the boundary surface is the minimum of the minimum distances from the point to every boundary triangular facet. As we can see immediately, the wanted minimum distance can be computed repeatedly over each boundary facets, but since the represented real part model usually has a large number of boundary facets, the direct computational method will be too costly in time. Therefore an efficient algorithm is necessary for the design of FGM through distance function. Figure 5-1 demonstrates how a user can design an FGM through distance function to the boundary automatically. Simply the user only needs to choose the materials variation in terms of distance from the boundary. Here the variation of materials has to satisfy certain design rules for matetial system [16].

The method of designing FGM as to the distance from the boundary of model can be extended to designing FGM as to the distance from arbitrary .STL shell, which is in the form of a bunch of triangular facets. Because of the similar reason as previously described, minimum distance from a control composition, which is outside the shell to the shell needs to be efficiently calculated.

Another design scheme is to allow users to design only the control compositions that are contained in a given .STL shell, and the composition functions can be one of the distance functions to different fixed features. It is necessary to efficiently identify those control compositions that are inside the .STL among the whole set of control compositions. Because of the large number of control compositions, the efficiency is also important here.

5.2 Algorithm for efficient distance function evaluation

5.2.1 Distance computation for a single query point inside bounding box

After the preprocessing described in Chapter 4, we can compute the exact minimum Euclidian distance for each query point (interior) by computing minimum distances to those facets that are located in the nearest buckets. Figure 5-2 illustrates the procedure in steps.

STEP 1 *Find the nearest buckets ListBnr.*

Suppose the bucket the query point lies in is B_q, the nearest non-empty buckets relative to B_q make up a list of buckets called $ListB_{nr}$. All $B_i \in ListB_{nr}$ have the

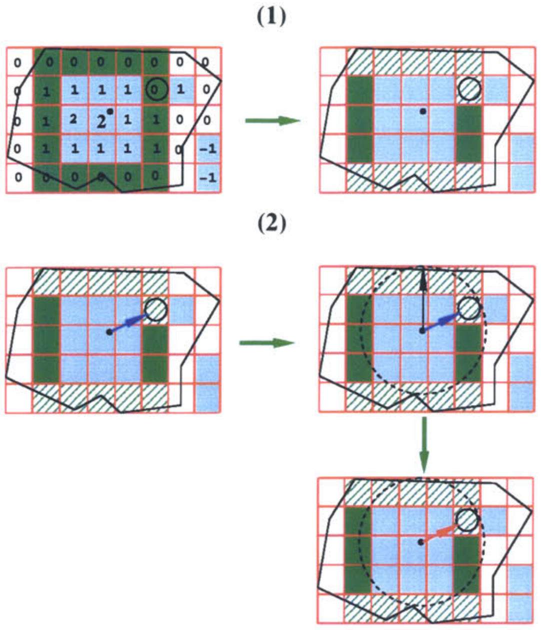


Figure 5-2: Compute exact Euclidean minimum distance for point inside bounding box

chessboard distance from B_q of the value that is stored in B_q. $ListB_{nr}$ can be easily constructed by checking each bucket in the cubic shell that is offset at that value.

STEP 2 *Compute the distances between the point and the facets in the nearest buckets.*

After the nearest buckets are found, the exact minimum distance to the boundary of the object can be calculated by computing the minimum distance from the query point to every facet inside those buckets. The next step is to check and repeat the procedure to guarantee the correct minimum distance is obtained because we were using the biggest empty cube in probing instead of the sphere to find the minimum distance. The minimum distance from one point to a triangular facet is either normal distance or normal distance plus the minimum distance from the projected foot to the edges of the triangle when the foot is outside the triangle (Figure 5-3).

5.2.2 Distance computation for a single query point outside bounding box

When we extend the design method via distance to the model boundary to the design method via distance to an arbitrary .STL boundary, an efficient evaluation of the distance

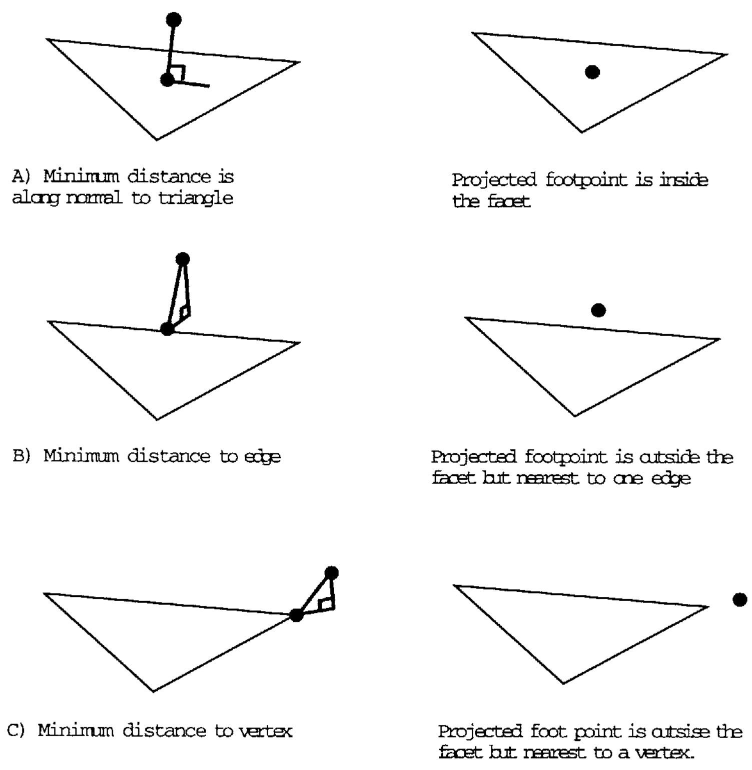

A) Minimum distance is along normal to triangle
Projected footpoint is inside the facet
B) Minimum distance to edge
Projected footpoint is outside the facet but nearest to one edge
C) Minimum distance to vertex
Projected foot point is outsise the facet but nearest to a vertex.

Figure 5-3: Euclidean minimum distance between a point and a triangular facet

from a query point outside the .STL boundary bounding box needs to be developed. For a query point outside the bounding box of the given .STL boundary surface, there is no query bucket existing, therefore, we don't have the information available about the nearest boundary buckets in terms of minimum digital distance. The method for this situation is to calculate the worst estimated buckets that might have the minimum digital distance from the virtual query bucket, then to search the minimum distance progressively further from that shell of buckets. In the following Figures 5-4 and 5-5, examples of two cases for 2D problem are given while a similar method is used in 3D. In Figure 5-4, the closest point from the query point to the bounding box is on an edge of the bounding box, while in Figure 5-5, the closest point from the query point to the bounding box is a vertex of the bounding box.

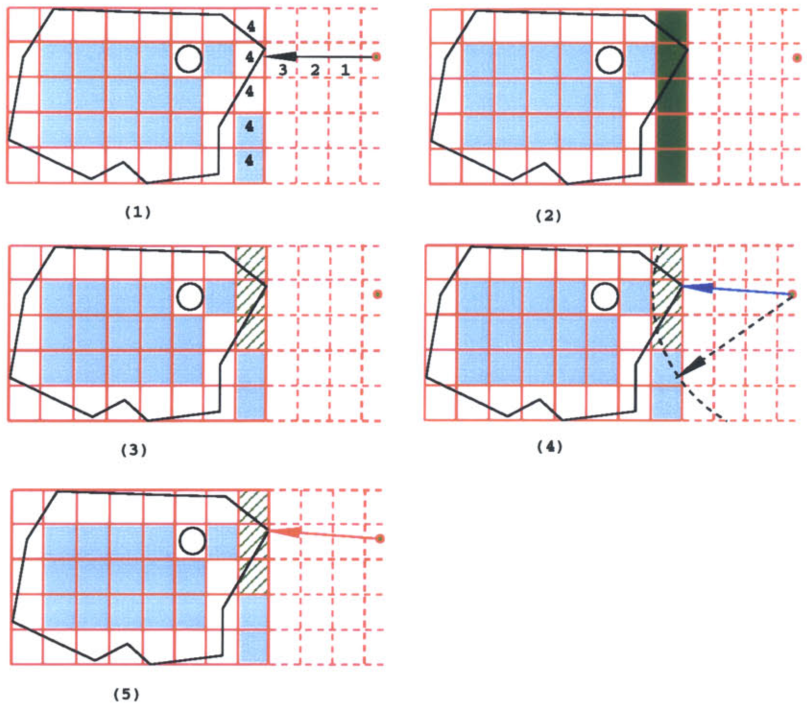


Figure 5-4: Example A of distance function for points outside the bounding box of the model

5.2.3 Computation of the list of query points

For sequence of query points, we can just simply repeat the steps stated in previous section, but actually we can check if a new query point lies in the same bucket as the one before, so time is saved in finding $ListB_{nr}$.

5.3 Design of FGM solids within given .STL boundaries

Designing FGM within .STL boundary is done by assigning composition to the control compositions that are contained in the .STL boundary. As it is said in the introduction, efficient method is also necessary, the approach here is to efficiently identify the buckets that intersect the bounding box of the .STL boundary, and then test the vertices in each of such buckets to see if the vertices are contained in the .STL boundary using the efficient point classification algorithm stated in Section 4.8.1. Figure 5-6 illustrates the method.

5.4 Complexity analysis of distance function computation

It should be noted that the performance of the bucketing algorithm does not just depend on the number of facets on the boundary of models (n_{bf}) and the number of query points (m). Because different points require different computing time; i.e. the nearer a point is to the boundary, the fewer buckets need to be searched. In addition, the geometry of the boundary also influences the performance of the bucketing algorithm; the bigger amplitude the boundary oscillates to (relative to the dimension of boundary), the fewer searches are needed, because points are more locally confined. From these observations, we can deduce that a cube is one of the most costly object geometry incidences for distance computation, therefore we can use the cube to see the worst behavior of the algorithm. Here we only consider the average of (m) operations of minimum distance computation. Recall from previous parts of this thesis that query points correspond to control composition points within the model and maybe distributed throughout the model. Here bellow, we will use parameter n instead of n_{bf} because the number of buckets n is equal to n_{bf}.

For convenience of this complexity analysis, it is assumed that the query points result from uniform rectangular meshing. Consider a cube object as in (Figure 5-7), with the boundary meshed into n_{bf} triangular facets. Asumming there are m number of interior points uniformly distributed throughout the body, the total time cost T shall be

$$T = T_{pre} + \sum_{i=1}^{m} T_i \tag{5.1}$$

where T_{pre} is the time cost for the preprocessing and T_i is the time cost for the ith query point.

From the algorithm of the preprocessing, one can see $T_{pre} = T_{bk} + T_{dt}$, where T_{bk} denotes the time cost for bucketing and T_{dt} denotes the time cost for digital distance transform. It is obvious that T_{bk} is linear to the number of boundary facets ($T_{bk} = O(n)$), and T_{dt} is linear to the number of total buckets which is also n ($T_{dt} = O(n)$).

$$T_{pre} = O(n) \tag{5.2}$$

From the algorithm of the distance function, we can see query points that fall in the same bucket basically cost the same amount of time except for those points that lies along the diagonal lines and diagonal planes. Suppose the time cost for distance computation of query points that lie in the buckets that have chessboard distance k to the boundary buckets is $T_q(k)$, we can rewrite $\sum_{i=1}^{m} T_i$ as follows:

$$\sum_{i=1}^{m} T_i = \sum_{k=0}^{\lfloor n_s/2 \rfloor} T_q(k) \cdot N(k) \tag{5.3}$$

where $n_s = \sqrt[3]{n}$ and $N(k)$ is the number of query points whose buckets have chessboard distance value k to the boundary buckets. Since we sample query points uniformly, we can define the density of number of query points per bucket as ρ, and because we have in total n buckets and m query points, therefore $\rho = \frac{m}{n}$. Further, we define the number of buckets which have chessboard distance value k to the boundary buckets as $N_b(k)$. Graphically, we can see those buckets make up the cubic shell that is between the cube with side length $n_s - 2k$ and the cube with side length $n_s - 2k - 2$. Therefore, one can deduce that

$$N_b(k) = [(n_s - 2k)^3 - (n_s - 2k - 2)^3] \tag{5.4}$$

and then

$$N(k) = N_b(k) \cdot \rho = [(n_s - 2k)^3 - (n_s - 2k - 2)^3] \cdot \frac{m}{n} \tag{5.5}$$

From the algorithm for computation of distance for a single query point, one can deduce that the time cost $T_q(k)$ includes the time for searching for nearest boundary buckets in the layer of buckets that have chessboard distance to the query bucket B_q and the time cost in computing the exact Euclidean distances from the query point to all the boundary facets that are in the nearest boundary buckets. Here we denote the time for searching as T_{sr} and the time for computing exact distances as T_{cmp}. Therefore,

$$T_q(k) = T_{sr}(k) + T_{cmp}(k) \tag{5.6}$$

For convenience of analysis, here we define several characteristic numbers that are related.

We define $N_{tb}(k)$ as the number of buckets that need to be searched for a query point that has chessboard distance k to the boundary buckets. Buckets need to be searched are the cubic shell of buckets that has offset of k to B_q, therefore,

$$N_{tb}(k) = \begin{cases} (2k+1)^3 - (2k-1)^3 & \text{if } k > 0 \\ 1 & \text{if } k = 0 \end{cases} \tag{5.7}$$

We also define N_{bs} as the number of boundary squares on the six outer sides of the cubic shell.

$$N_{bs} \leq 6(2k+1)^2 \tag{5.8}$$

We use $N_{ft}(k)$ as the number of facets in the nearest buckets that have to be calculated for exact Euclidean distances.

$$N_{ft} = P_f \cdot N_{bs}(k) \tag{5.9}$$

where $P_f = \frac{\sqrt[3]{n}}{6}$ which is the number of facets per square on the boundary of the cubic object. After the above definitions, we can deduce that:

$$T_{sr}(k) = c_1 \cdot N_{tb}(k), \tag{5.10}$$

$$T_{cmp}(k) = c_2 \cdot N_{ft}(k), \tag{5.11}$$

where c_1 and c_2 are constants, therefore

$$T = O(n) + \sum_{k=0}^{\lfloor n_s/2 \rfloor} T_{sr}(k) \cdot N(k) + \sum_{k=0}^{\lfloor n_s/2 \rfloor} T_{cmp}(k) \cdot N(k) \tag{5.12}$$

After some algebra, we have:

$$\sum_{k=0}^{\lfloor n_s/2 \rfloor} T_{sr}(k) \cdot N(k) = \sum_{k=0}^{\lfloor n_s/2 \rfloor} 24c_1[k^4 \cdot \frac{24m}{n} + k^3 \cdot (\frac{24m}{n} - \frac{24m}{n_s^2}) + k^2 \cdot (\frac{6m}{n_s} - \frac{24m}{n_s^2} + \frac{8m}{n})] \tag{5.13}$$

$$\sum_{k=0}^{\lfloor n_s/2 \rfloor} T_{cmp}(k) \cdot N(k) \leq \sum_{k=0}^{\lfloor n_s/2 \rfloor} c_2[k^4 \cdot \frac{96m}{n_s^2} + k^3 \cdot (\frac{192m}{n_s^2} - \frac{96m}{n_s})$$

$$+k^2 \cdot (24m - \frac{144m}{n_s} - \frac{152m}{n_s^2}) + k \cdot (\frac{24m}{n_s^2} - \frac{24m}{n_s})$$
$$+6m - \frac{12m}{n_s} + \frac{8m}{n_s^2}] \tag{5.14}$$

Using the Euler summation equation [9]

$$\sum_{k=1}^{n} k^m = \frac{n^{m+1}}{m+1} + \frac{n^m}{2} + \frac{mn^{m-1}}{12} + O(n^{m-2}) \tag{5.15}$$

We can derive that

$$\sum_{k=0}^{\lfloor n_s/2 \rfloor} k^4 \simeq \frac{n_s^5}{160} \tag{5.16}$$

$$\sum_{k=0}^{\lfloor n_s/2 \rfloor} k^3 \simeq \frac{n_s^4}{64} \tag{5.17}$$

$$\sum_{k=0}^{\lfloor n_s/2 \rfloor} k^2 \simeq \frac{n_s^3}{24} \tag{5.18}$$

Therefore, T reduces to:

$$T = O(n) + O(\frac{3c_1}{5} mn^{\frac{2}{3}} + \frac{c_2}{10} mn) \tag{5.19}$$

5.5 Experimental results comparison with exhaustive searching method

The efficient distance function was implemented on a Pentium II PC with 450MHz processor and 128MB memory. The test runs are done on several models with uniform query points chosen in the bounding domain of each model. Models are listed in Table 5.1 with the corresponding factor of efficiency enhancement (E_f) and the preprocessing time (PT) for each model. From this table, we can see for all the models, the efficiency enhancement is better than the theoretical value from the previous worst case time complexity analysis. The visualized models and the corresponding curves of experimental results comparison with exhaustive searching method are also given in Figures 5-8 to 5-14. The factor of efficiency enhancement is obtained through comparing the slopes of the linear regression of running curves with both methods. From these experimental results, one can conclude that the bucketing method significantly improved the efficiency compared to the exhaustive searching method by a large factor.

	Model 'Pill'	*Model 'Propeller'*	*Model 'Bracket'*	*Model 'Sump'*
n_{bf}	9572	6210	2924	10296
E_f	15.79	24.42	13.9	44.82
$PT(sec)$	2.47	2.47	1.02	3.81

Table 5.1: Efficiency enhancement on the example models

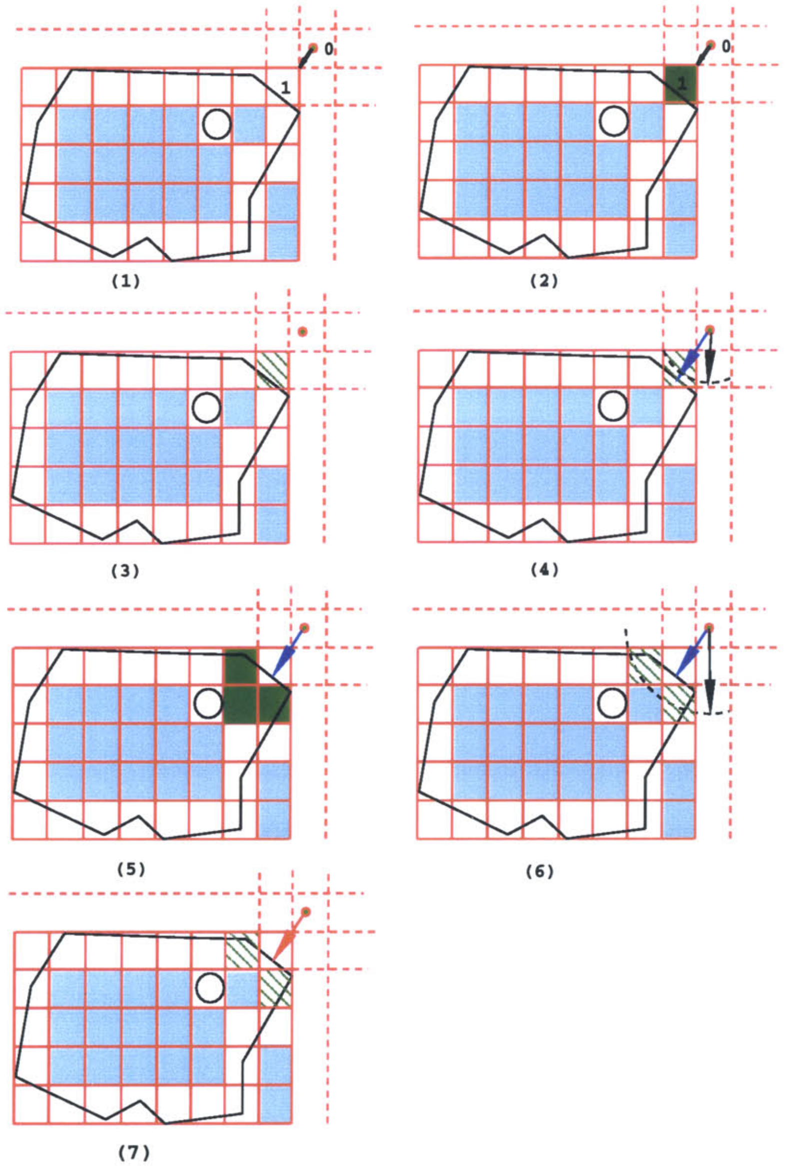


Figure 5-5: Example B of distance function for points outside the bounding box of the model

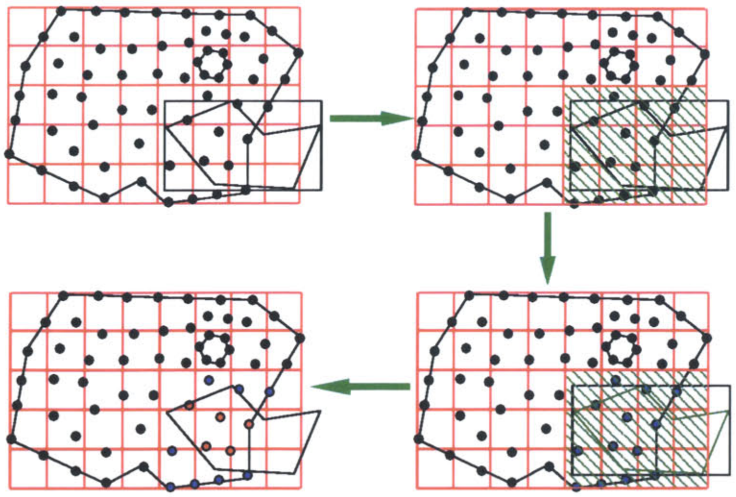


Figure 5-6: Identify the vertices that are contained in the given .STL boundary

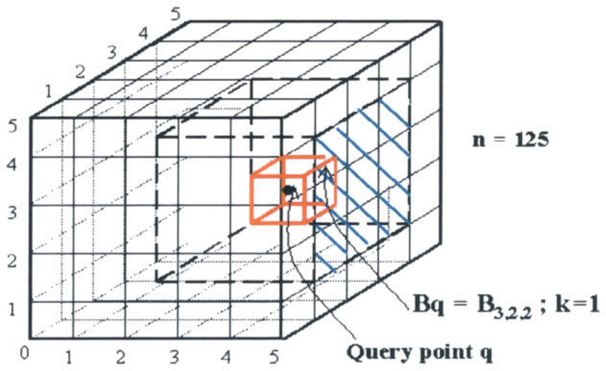


Figure 5-7: k is the chessboard distance of the query bucket to the boundary buckets; number of non-empty buckets that the boundary occupied is $6n^{2/3}$, where n is the number of total buckets; buckets in the shaded area are nearest non-empty buckets needed to be searched for the query point

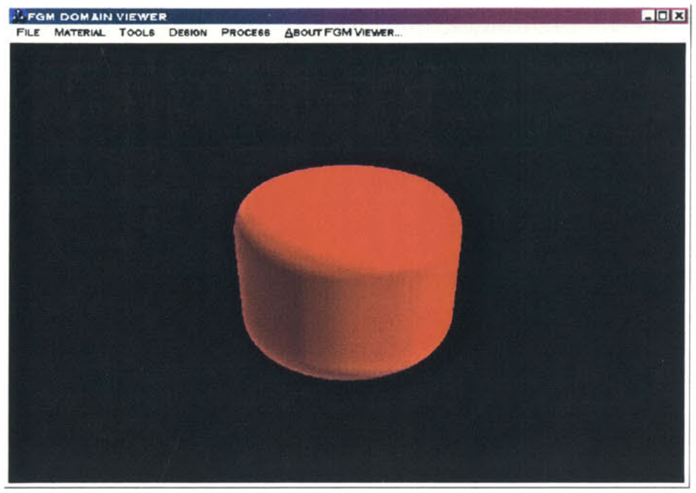


Figure 5-8: Model 'Pill'

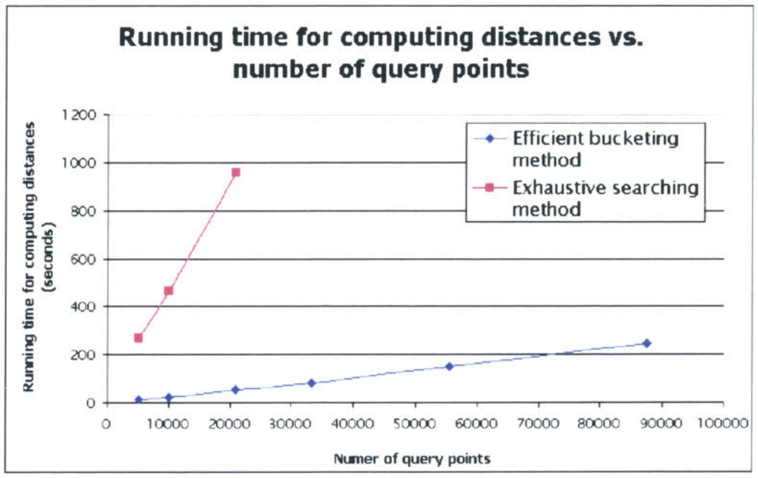


Figure 5-9: Experimental results on 'Pill'

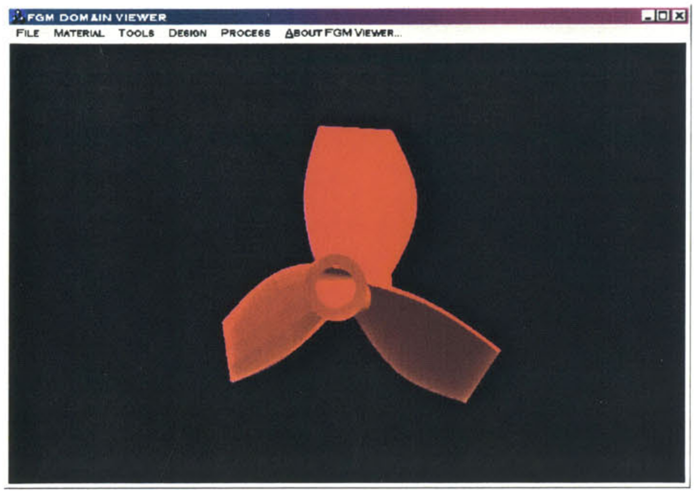


Figure 5-10: Model 'Propeller'

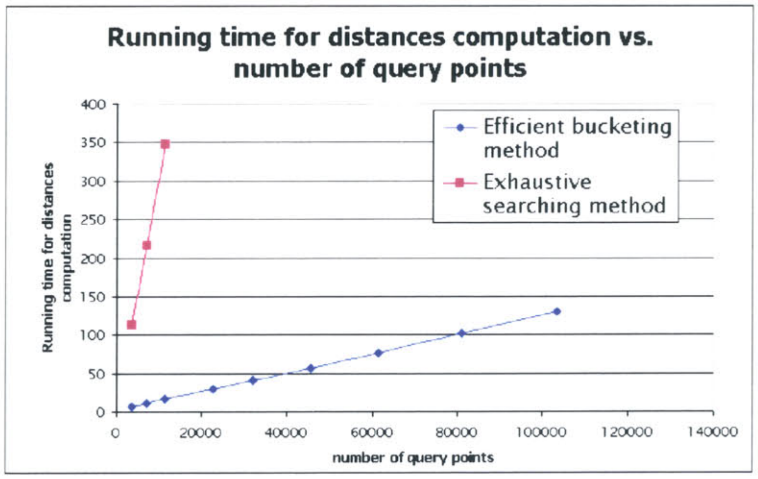


Figure 5-11: Experimental results on 'Propeller'

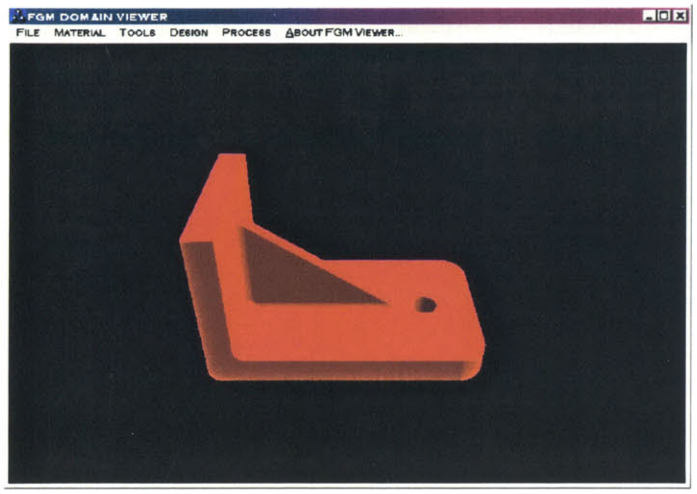


Figure 5-12: Model 'Bracket'

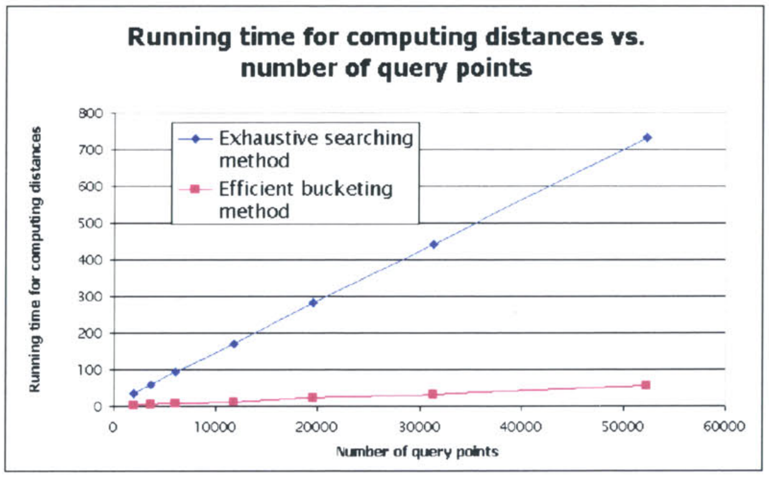


Figure 5-13: Experimental results on 'Bracket'

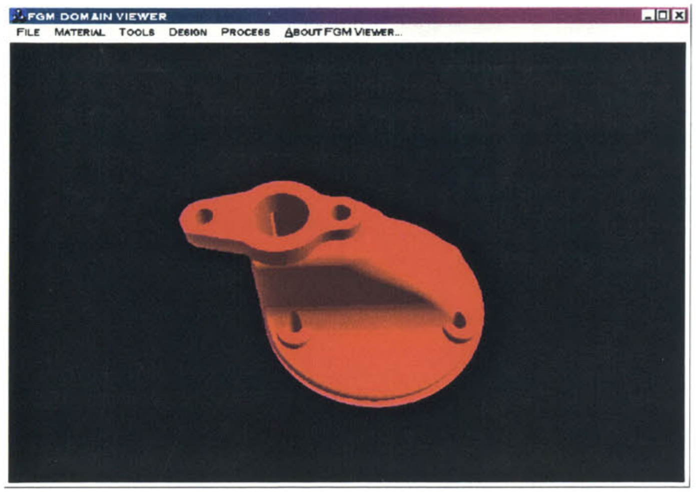


Figure 5-14: Model 'Sump'

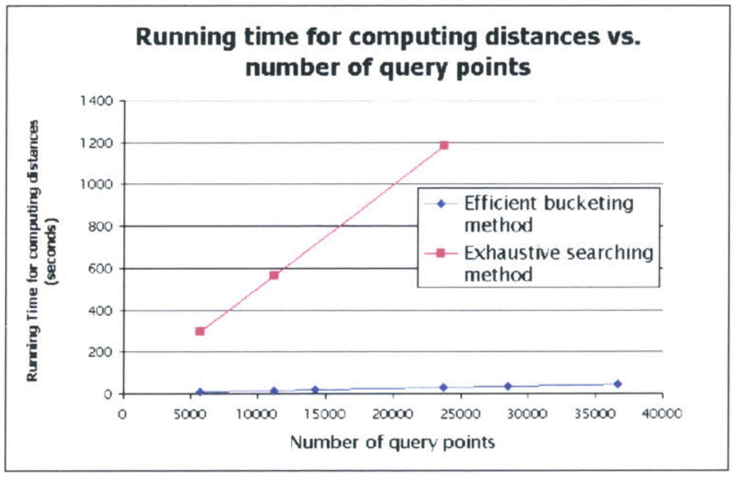


Figure 5-15: Experimental results on 'Sump'

Chapter 6

Efficient evaluation of composition

6.1 Composition evaluation at a point using barycentric coordinates

The composition evaluation at a query point is done by interpolating the composition values at the nodes of the object tetrahedron in which it is contained. In the case of tetrahdra mesh using linear interpolation, we have the following formula for the composition interpolation. Fig. 6-1 demonstrates the idea.

$$u = \frac{\text{Volume of Tetrahedron}(QV_1V_3V_4)}{\text{Volume of Tetrahedron}(V_1V_2V_3V_4)} \tag{6.1}$$

$$v = \frac{\text{Volume of Tetrahedron}(QV_1V_2V_4)}{\text{Volume of Tetrahedron}(V_1V_2V_3V_4)} \tag{6.2}$$

$$w = \frac{\text{Volume of Tetrahedron}(QV_1V_2V_3)}{\text{Volume of Tetrahedron}(V_1V_2V_3V_4)} \tag{6.3}$$

$$\eta = 1 - u - v - w \tag{6.4}$$

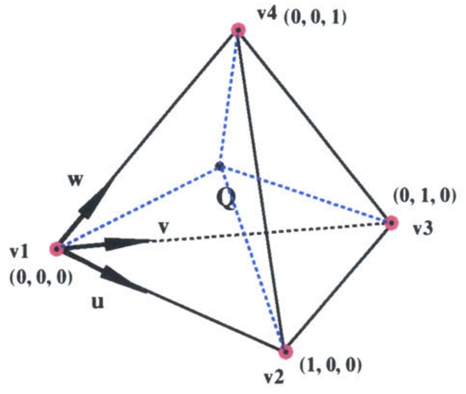
$$Compsition(Q) = \eta \cdot Comp(V_1) + u \cdot Comp(V_2) + v \cdot Comp(V_3) + w \cdot Comp(V_4) \tag{6.5}$$

STEP 1 *Determine if the query point is interior point:* Use the algorithm described in Chapter 4 to determine the status (Figure 4-11).

STEP 2 *Identify the object tetrahedron if the query point is interior point:* Use the algorithm described in Chapter 4 Figures 4-15 to 4-17.

STEP 3 *Calculate the barycentric coordinates:* Use the formula as above.

STEP 4 *Interpolate the composition*



Barycentric coordinates of a point in tetrahedron space :
$Q(u, v, w, \eta)$, $\eta = 1 - u - v - w$.

Figure 6-1: Evaluate composition at a given point

6.2 Composition evaluation along a given ray at a given resolution

Based on the above algorithm for point evaluation, the efficient ray-casting algorithm for the problem is developed to find the composition for all the points along a given ray at a given resolution. The ray is represented by a starting point and an ending point, while the ending point just guides the direction. The resolution is represented by an interval of parameter δt.

$$X(t) = (1-t) \cdot X_b + t \cdot X_e \tag{6.6}$$

STEP 1 Find the starting point that intersects the bounding box

STEP 2 Find the end point that intersects the bounding box

STEP 3 Evaluate the points at a given resolution using the parametric expression of straightline.

STEP 4 Evaluate the composition for each point: Apply the algorithm for composition evaluation at a point.

Figure 6-2 illustrates the algorithm of evaluation of composition along a ray.

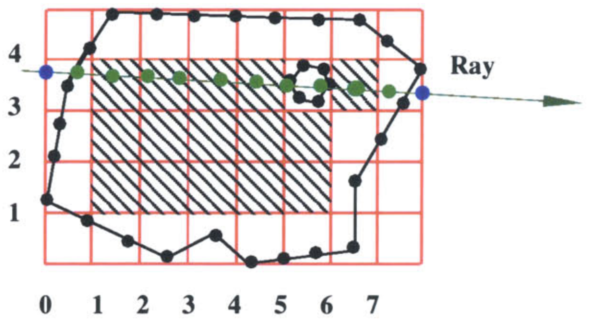


Figure 6-2: Evaluate composition along a given ray

6.3 Composition evaluation on a cutting plane

For the purpose of evaluating compositions of materials, the program will find the composition for the points on a given plane at a given resolution. Here the plane is given in general form

$$A \cdot x + B \cdot y + C \cdot z - D = 0 \tag{6.7}$$

The method is calculating the intersection points of the plane with the bounding box of the model, using the intersection points coordinates one can express the plane in parametric form as following:

$$X(u, v) = X_{b_1} + (X_{b_2} - X_{b_1})u + (X_{b_3} - X_{b_2})v \tag{6.8}$$

After the parametric expression of the plane is obtained, one can repeatedly call the ray casting method to evaluation compositions of the query points on the given plane.

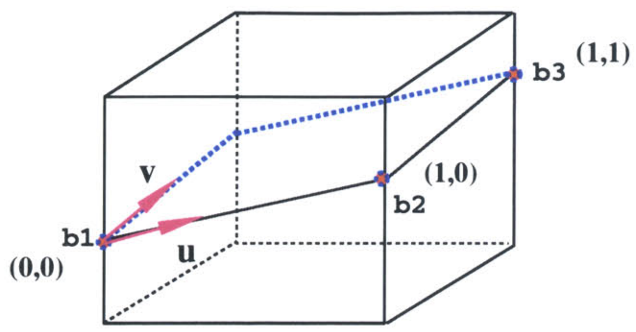


Figure 6-3: Parametric cutting plane

6.4 Volume integral of material

It is very necessary to evaluate the volume ratio of different materials of a FGM object after the design of composition on the control compositions inside the body. Here the control compositions are the interior vertices of the tetrahedra mesh. Given the fact that we have calculated the volume ratio of one specific material at all the control compositions, one can integrate the volume ratio with respect to the whole object for that material using barycentric coordinates.

To derive the formula of the integration of volume ratio of material **a** (noted P_a) within the whole body, it is convenient to use the natural coordinate system for each tetrahedron as demonstrated in Figure 6-4.

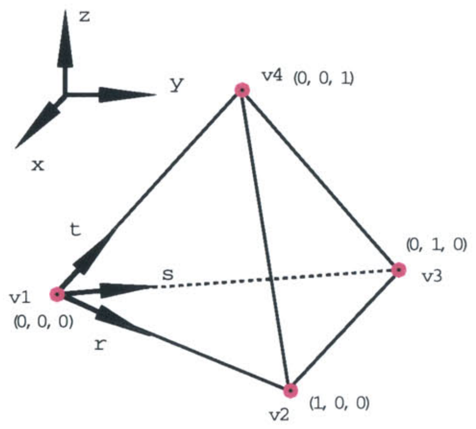


Figure 6-4: Tetrahedron natural coordinates

For a tetrahedron indexed m, we define that the volume ratio of material **a** at a point which has coordinate (r, s, t) as $C_{a,m}(r, s, t)$. Therefore, we can express the relation between

P_a and $C_{a,m}(r,s,t)$ in the following formula.

$$P_a = \frac{\sum_{m=1}^{n_t} \int_{v_m} C_{a,m}(r,s,t)\,dv_m}{\sum_{m=1}^{n_t} v_m} \tag{6.9}$$

For tetrahedron m, we define the volume ratio of material **a** at the four vertices as c_1, c_2, c_3 and c_4 respectively. Using the natural coordinate like in Figure 6-4, we have:

$$x = (1-r-s-t)\cdot x_1 + r\cdot x_2 + s\cdot x_3 + t\cdot x_4 \tag{6.10}$$

$$y = (1-r-s-t)\cdot y_1 + r\cdot y_2 + s\cdot y_3 + t\cdot y_4 \tag{6.11}$$

$$z = (1-r-s-t)\cdot z_1 + r\cdot z_2 + s\cdot z_3 + t\cdot z_4 \tag{6.12}$$

and similarly we have for the composition interpolation as:

$$C_{a,m}(r,s,t) = (1-r-s-t)\cdot c_1 + r\cdot c_2 + s\cdot c_3 + t\cdot c_4 \tag{6.13}$$

Therefore, the Jacobian matrix is:

$$J = \begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} & \frac{\partial z}{\partial r} \\ \frac{\partial x}{\partial s} & \frac{\partial y}{\partial s} & \frac{\partial z}{\partial s} \\ \frac{\partial x}{\partial t} & \frac{\partial y}{\partial t} & \frac{\partial z}{\partial t} \end{bmatrix} = \begin{bmatrix} x_2-x_1 & y_2-y_1 & z_2-z_1 \\ x_3-x_1 & y_3-y_1 & z_3-z_1 \\ x_4-x_1 & y_4-y_1 & z_4-z_1 \end{bmatrix} \tag{6.14}$$

and then,

$$det\mathbf{J} = 6\cdot v_m \tag{6.15}$$

Hence we can derive that,

$$\begin{aligned} \int C_{a,m}(r,s,t)\,dv_m &= \int C_{a,m}(r,s,t)\cdot detJ\,drdsdt \\ &= det\mathbf{J} \int_0^1 dr \int_0^{1-r} ds \int_0^{1-r-s} C_{a,m}(r,s,t)dt \end{aligned} \tag{6.16}$$

if

$$\begin{cases} r+s+t=u \\ s+t=uv \\ t=uvw \end{cases} \tag{6.17}$$

then

$$\begin{cases} r=u(1-v) \\ s=uv(1-w) \\ t=uvw \end{cases} \tag{6.18}$$

therefore the Jacobian matrix J' from (r,s,t) to (u,v,w) will have a determinant

$$det\mathbf{J}' = u^2 v \tag{6.19}$$

hence,

$$\begin{aligned} \int C_{a,m}(r,s,t)dv_m &= det\mathbf{J}\int_0^1 du\int_0^1 dv\int_0^1 \{c_1+(c_2-c_1)u \\ &\quad + (c_3-c_2)uv + (c_4-c_3)uvw\}\cdot det\mathbf{J}'dw \\ &= det\mathbf{J}\int_0^1 du\int_0^1 dv\int_0^1 \{c_1+(c_2-c_1)u \\ &\quad + (c_3-c_2)uv + (c_4-c_3)uvw\}\cdot u^2vdw \\ &= \frac{1}{24}det\mathbf{J}\cdot(c_1+c_2+c_3+c_4) \\ &= \frac{v_m}{4}(c_1+c_2+c_3+c_4) \end{aligned} \tag{6.20}$$

Plug the above equation into (1), we derive that,

$$P_a = \frac{1}{4}\frac{\sum_{m=1}^{n_t} v_m\cdot(c_1+c_2+c_3+c_4)}{\sum_{m=1}^{n_t} v_m} \tag{6.21}$$

6.5 Time analysis

6.5.1 Theoretical background

For a 3D or higher degree space object represented by incidence structure, if the structure is homogeneous, there is a structure constant b_{kl} which is the number of pointers from any k-

dimension element to l-dimension elements. Suppose a_k is the total number of k-dimension elements, from the Matching Theorem ("Principle of Double Counting") [30], the following formula gives the relation between a_k and b_{kl}:

$$a_k \cdot b_{kl} = a_l \cdot b_{lk} \qquad 0 \leq k, l \leq n \tag{6.22}$$

where n is the dimension of the space.

In our case, 3D objects are subdivided into tetrahedra, and then elements in the incidence structure are vertices, edges, faces and tetrahedra. For convenience, *we suppose the FEM data is near homogeneous.* Our interest is to find the relation between the number of tetrahedra and the number of vertices. It is obvious that each tetrahedron points to 4 vertices, which means $b_{30} = 4$. Therefore by the matching formula we have:

$$b_{03} = \frac{a_3 \cdot b_{30}}{a_0} = \frac{4n_t}{n_v} \tag{6.23}$$

where n_t is the total number of tetrahedra, n_v is the total number of vertices.

6.5.2 Time cost for the extraction of boundary

Because the extraction of boundary is done in the initialization of data structure, first we need to analyze the time cost for reading FEM data into my data structure, time cost for that part is apparently $O(n_v + 4n_t)$, where n_v is the total number of vertices, n_t is the total number of tetrahedra. And also after we extract the boundary, we need to delete the interior faces and put the boundary facets into a list, which costs $O(4n_t)$.

Using the algorithm described before, the time cost for extracting boundary would be:

$$T = 4\sum_{i=1}^{n_t} T_{cf}, \tag{6.24}$$

where T_{cf} is the time cost of checking one face of a tetrahedron if it is interior. Given a triangular face $F_{\mathbf{abc}}$, we define that the number of incident tetrahedra to vertex **a** is Pt_a, and similarly Pt_b and Pt_c for vertex **b** and **c**. The time cost T_{cf} is virtually the sum of

Pt_a, Pt_b and Pt_c, that is:

$$T_{cf} = O(Pt_a + Pt_b + Pt_c) \tag{6.25}$$

From the background theory we know on average one vertex has $\frac{4n_t}{n_v}$ number of incident tetrahedra, therefore,

$$T_{cf} = 12 \cdot O(\frac{n_t}{n_v}) \tag{6.26}$$

In summary, the total cost for extracting boundary will be:

$$T = O(48 \cdot \frac{n_t}{n_v} \cdot n_t) \tag{6.27}$$

and the total cost for initialization of the data structure will be:

$$T_i = O(c_1 \cdot n_v + c_2 \cdot n_t + c_3 \cdot \frac{n_t^2}{n_v}) \tag{6.28}$$

6.5.3 Time cost of point location algorithm

From the algorithm described in the previous text, we can see the time cost of point location for a single query point is:

$$T = T_{in} + T_{mch}, \tag{6.29}$$

where T_{in} is the time spent on checking if the query point is inside the body, here this time will be linear to the number of boundary facets in one bucket, which is about $n_{bf}^{\frac{1}{3}}/6$ for a cube, where n_{bf} is the total number of boundary facets. Hence,

$$T_{in} = O(n_{bf}^{\frac{1}{3}}) \tag{6.30}$$

T_{mch} here is the time spent on marching all the tetrahedra from a starting vertex to the query point. If we define the number of tetrahedra we checked as N_{tc}, the number of faces we checked to find the neighbor tetrahedron along the line as N_{fc} and the time for each neighboring check as T_{cnb}, we have the following formula.

$$T_{mch} = O(N_{tc}) + O(N_{fc} \cdot T_{cnb}) \tag{6.31}$$

We use a modified version of the algorithm used for checking interior faces to check the neighboring tetrahedron, hence,

$$T_{cnb} = 12 \cdot O(\frac{n_t}{n_v}) \tag{6.32}$$

Since for each tetrahedron visited, we need to check $0 \sim 1$ face for neighboring, therefore,

$$T_{mch} = O(N_{tc}) + O(N_{tc} \cdot \frac{n_t}{n_v}) \tag{6.33}$$

From all the above, we can deduce that:

$$T = O(c_1 \cdot n_{bf}^{\frac{1}{3}} + c_2 \cdot N_{tc} + c_3 \cdot N_{tc} \cdot \frac{n_t}{n_v}) \tag{6.34}$$

It will be interesting if we can give an expectation of the value of $\frac{n_t}{n_v}$. We know for a Delaunay mesh, the worst case would be quadratic, and in problems of practical relevance, this degree is expected to be constant [23].

6.5.4 Time cost of ray casting algorithm

In order to analyze the time cost of ray casting, we define several parameters as follows: δt as the resolution which is a number between $0 - 1$; n_r as the number of query points inside the bounding box along the ray; N_{br} as the number of boundary buckets that intersect the ray; n_{bq} as the number of query points in boundary buckets along the ray; ρ as the number of query points per bucket that intersects the ray. From our ray casting algorithm described before, we know that the total time of the algorithm T should be equal to the time spent on checking each point for interior (T_{ckin}) status plus the time spent on locating the sub-regions for points identified as interior (T_{lcpo}). Therefore, we have:

$$T = T_{ckin} + T_{lcpo} \tag{6.35}$$

$$T_{ckin} = n_{bq} \cdot O(n_{bf}^{\frac{1}{3}}) + (n_r - n_{bq}) \cdot O(1) \tag{6.36}$$

$$T_{ckin} = n_{bq} \cdot O(n_{bf}^{\frac{1}{3}}) \tag{6.37}$$

$$n_{bq} = N_{br} \cdot \rho \tag{6.38}$$

$$n_r \leq 1 + \frac{1}{\delta t} \tag{6.39}$$

Suppose the bounding box is a cube, that is $n_x = n_y = n_z$, then the shooting ray can at least intersects $\sqrt[3]{n_{bf}}$ buckets. Hence

$$\rho \leq \frac{n_r}{\sqrt[3]{n_{bf}}} \leq \frac{1}{\delta t \cdot \sqrt[3]{n_{bf}}} \tag{6.40}$$

Therefore,

$$n_{bq} \leq \frac{(1 + \delta t) \cdot N_{br}}{\sqrt{3} \cdot \delta t \cdot \sqrt[3]{n_{bf}}}; \qquad 0 \leq \delta t \leq 1 \tag{6.41}$$

Hence

$$T_{ckin} = O(\frac{N_{br}}{\delta t}) \tag{6.42}$$

As to the time of locating positions, supposing that all the query points are inside the body, we know that:

$$T_{lcpo} = n_r \cdot T_{mch} \tag{6.43}$$

where T_{mch} is defined as in the last section. Therefore,

$$T_{lcpo} = O(\frac{N_{tc}}{\delta t} \cdot [1 + \frac{n_t}{n_v}]) \tag{6.44}$$

$$T = O(\frac{N_{br}}{\delta t} + \frac{N_{tc}}{\delta t} \cdot [1 + \frac{n_t}{n_v}]) \tag{6.45}$$

Chapter 7

Implementation and numerical results

7.1 Implementation

The algorithms developed in this thesis are implemented in Microsoft Visual C++ [10] on a Intel Pentium II CPU 450M with SRAM 128MHz. The algorithms include all the described algorithms in Chapters 3 – 6. As described already in Chapter 3, this thesis work is integrated with an existing environment of modeling, designing, and postprocessing of FGM. The approach is using object oriented programming techniques such as inheritance of classes and virtual functions. For details of the mentioned environment, refer to Jackson [14]. For the newly developed functionalities, the relevant user interfaces are extended using also OOP and Microsoft Foundation Classes programming [17][10]. The extension will be given in Appendix A.1.

7.2 Numerical results

Results on three models are given here, a model named 'Pill' (Figure 7-1) which has 1329 vertices, 6091 tetrahedra, a model named 'Bracket' (Figure 7-2) which has 2356 vertices, 8503 tetrahedra and a model named 'Widget' (Figure 7-3) which has 18683 vertices, 83827 tetrahedra.

	Model 'Pill'	*Model 'Brack'*	*Model 'Widget'*
n_{bf}	788	2884	11038
n_v	1329	2356	18683
n_t	6091	8503	83827

Table 7.1: Parameters of the FEM example models

Time (sec)		*Model'Pill'*	*Model 'Brack'*	*Model 'Widget'*
IT		*10.88*	*16.26*	*382.01*
PT	*BT*	*0.11*	*0.22*	*0.88*
	DT	*0*	*0.11*	*0.39*
	ST	*0.38*	*1.21*	*4.61*
DBT		*6.43*	*6.26*	*31.75*
PCT $\delta t = 0.01$		*7*	*7*	*6.43*

Table 7.2: Performance of program on the examples

The Table 7.2 gives the experimental running results of the above three models, where IT is the time spent on the initialization of data structure and extraction of the boundary tetrahedra, BP is the time spent on creating the bucketing system, DT is the time spent on digital distance transformation, ST is the time spent on identifying the signs of buckets, DBT is the time spent on the designing composition as to the distance from boundary and PCT represents the time spent on the evaluation of composition on one slice.

7.2.1 Design from boundary

In Figures 7-4 to 7-10 all the composition designs are done via distance function from boundary. For model "Pill" a planar cutting as in Figure 7-5 costs about 7 secs.

For a model in Figure 7-2, which has 2356 vertices, 8503 tetrahedra, a planar cutting as in Figure 7-8 costs about 7 secs. There are about 10000 query points on each plane.

For a model in Figure 7-3, which has 18683 vertices, 83827 tetrahedra, one slice costs 6.43 secs. There are about 10000 query points on the plane.

7.2.2 Design from .STL file boundary

Figures 7-11 to 7-12 demonstrate the example of designing from STL file. When designing according to the distance from .STL file, distance calculation also involves nodes that are

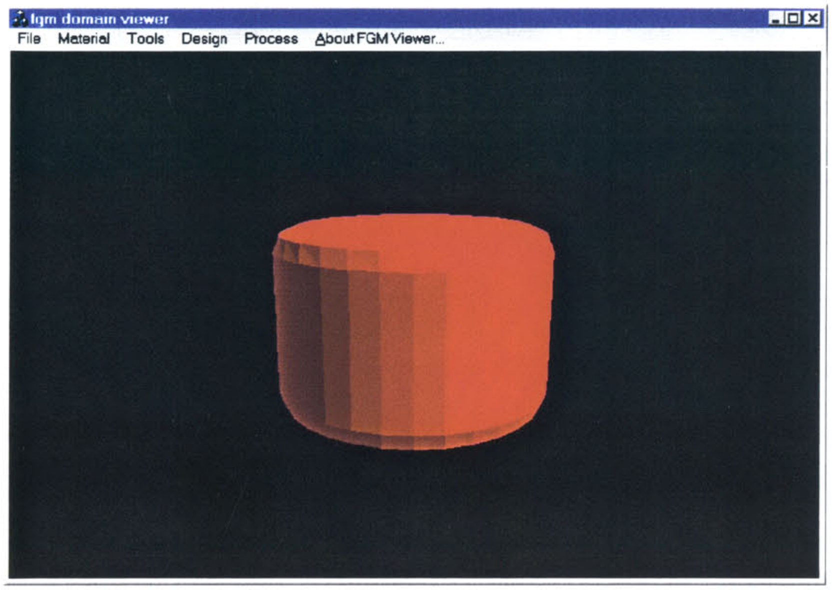


Figure 7-1: Pill

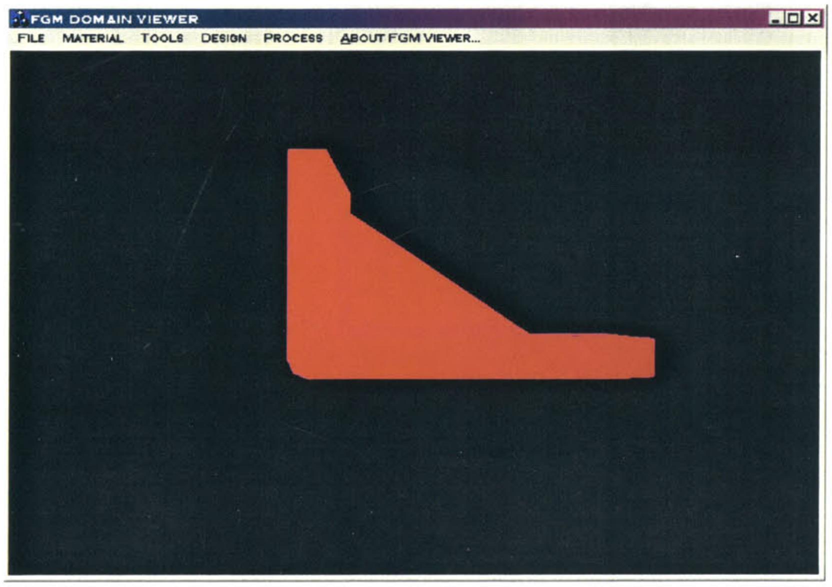


Figure 7-2: Bracket with a hole

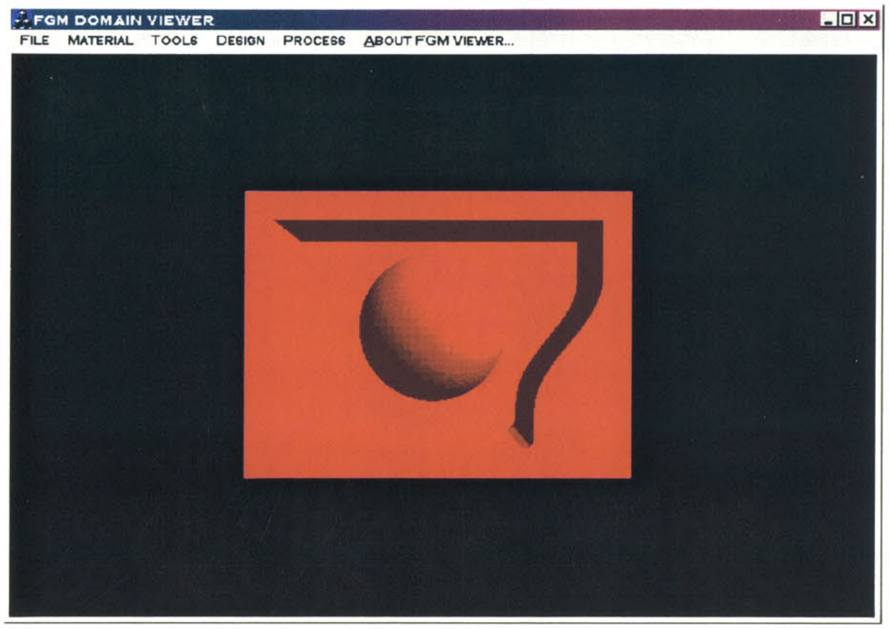


Figure 7-3: Widget

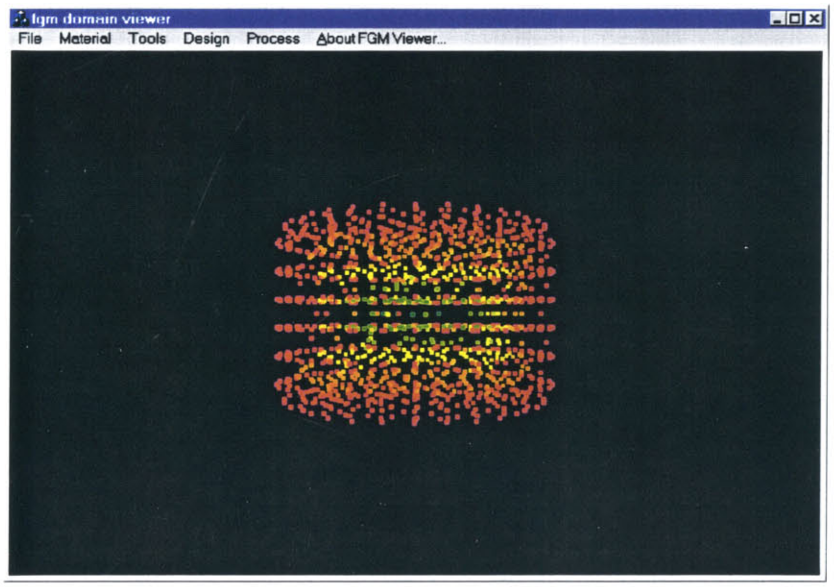


Figure 7-4: Pill composition data

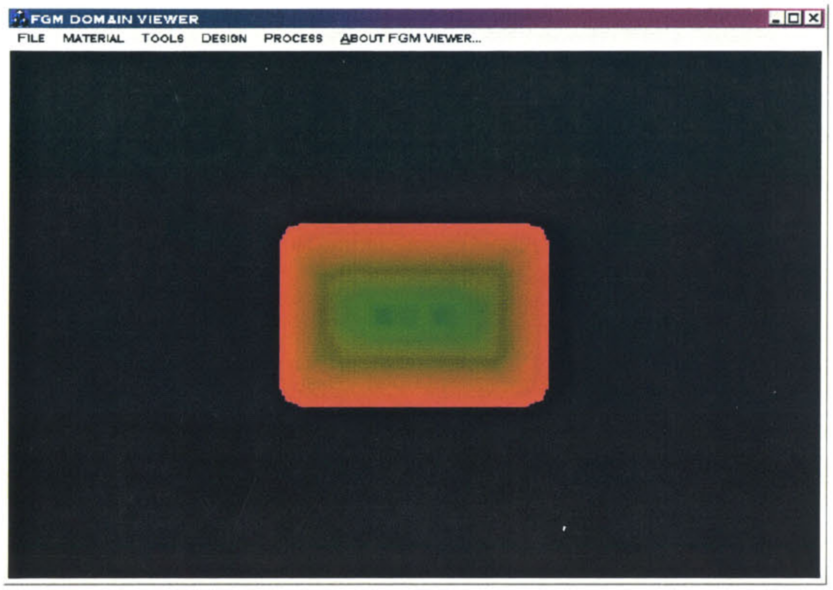


Figure 7-5: Pill slice at $z = 0$

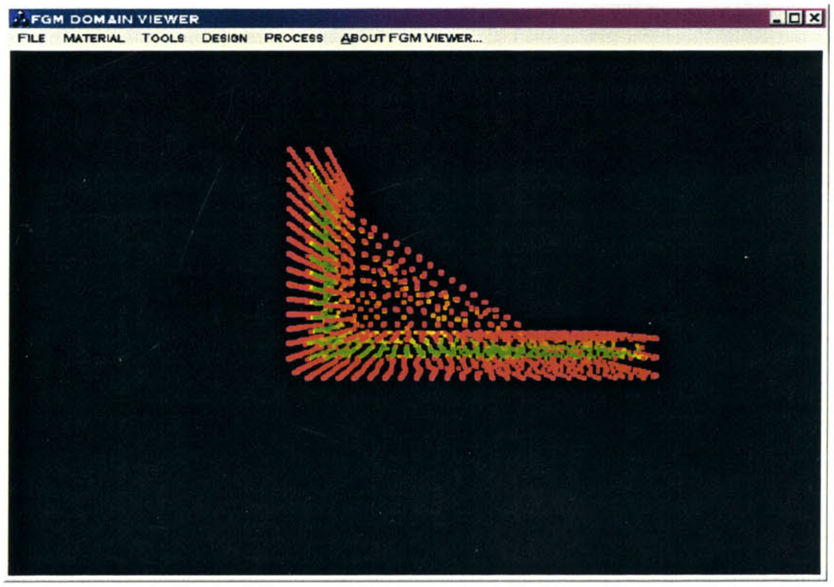


Figure 7-6: Bracket's composition data

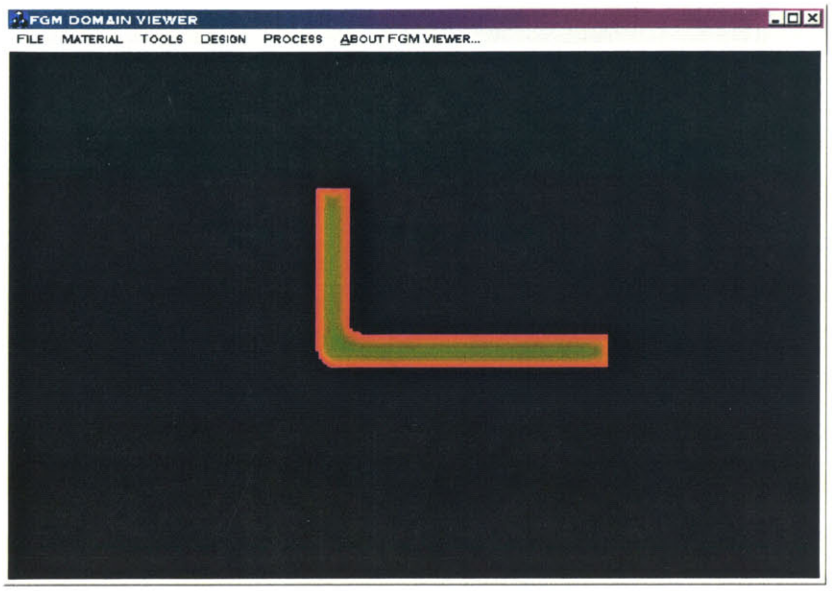


Figure 7-7: Bracket slice at $z = 1$

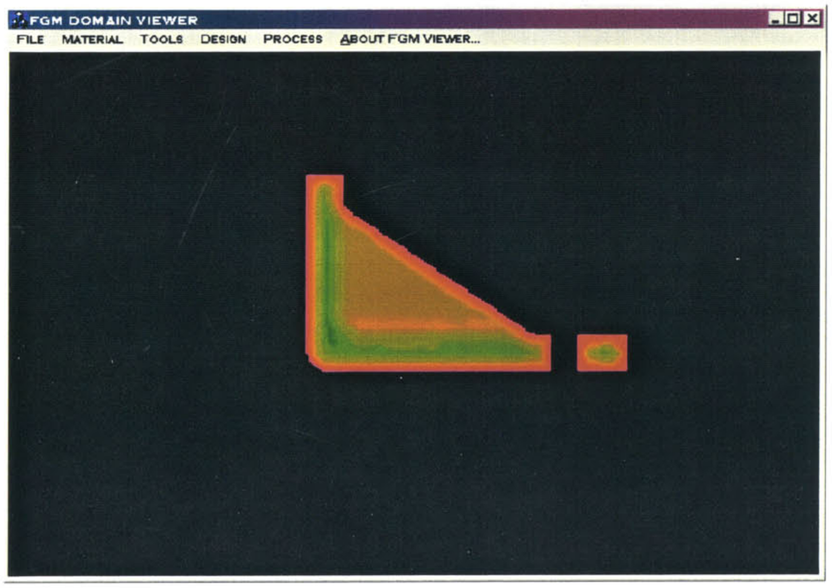


Figure 7-8: Bracket slice at $z = 3$

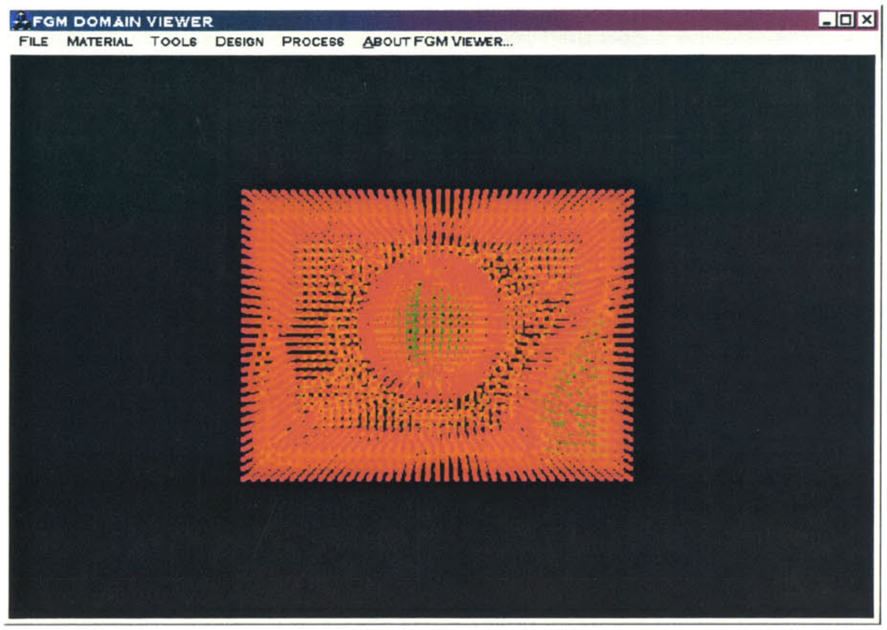


Figure 7-9: Widget composition

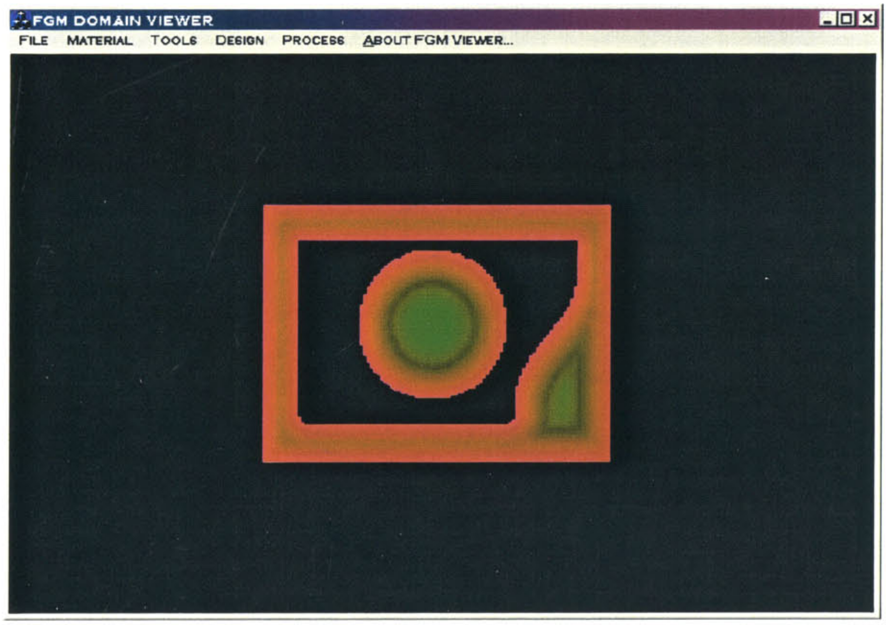


Figure 7-10: Widget slice

outside of the STL boundary.

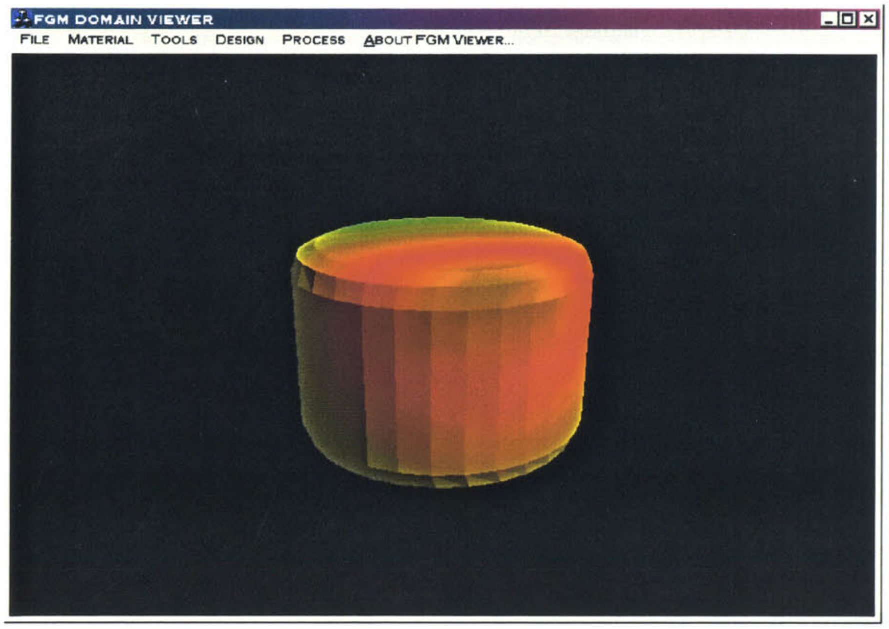


Figure 7-11: Design from a stl file, range (0, 90)

7.2.3 Design within .STL boundary

Figure 7-13 to 7-18 demonstrate the design method within a STL boundary, which means user can select only nodes inside an input STL boundary to design the compositions based on various distance functions.

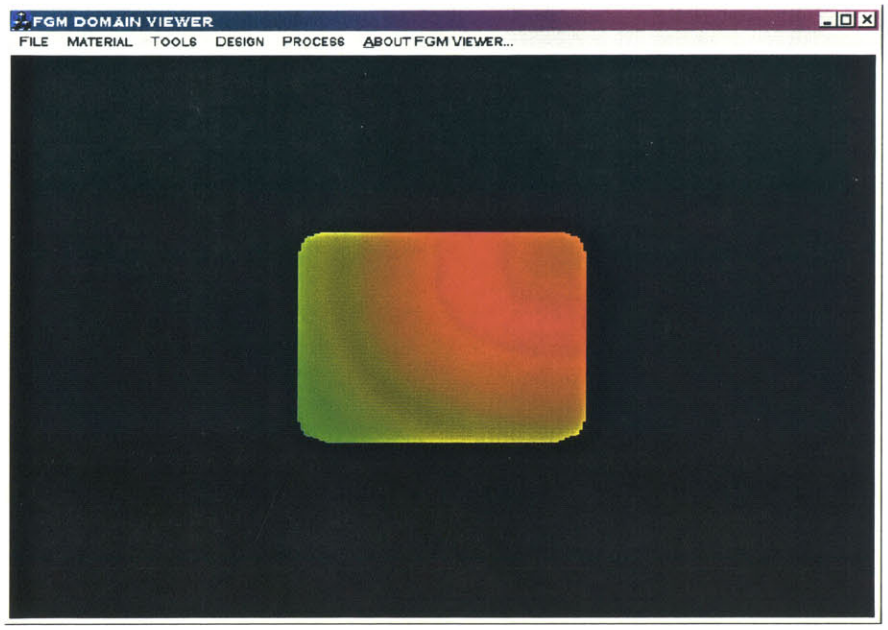


Figure 7-12: Slice at $Z = 20$

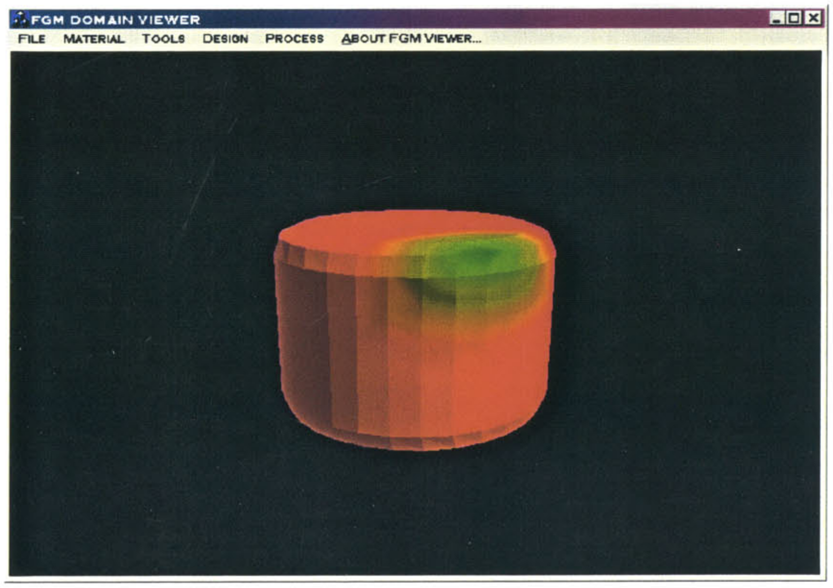


Figure 7-13: Design within STL boundary according to the distance to the STL mesh

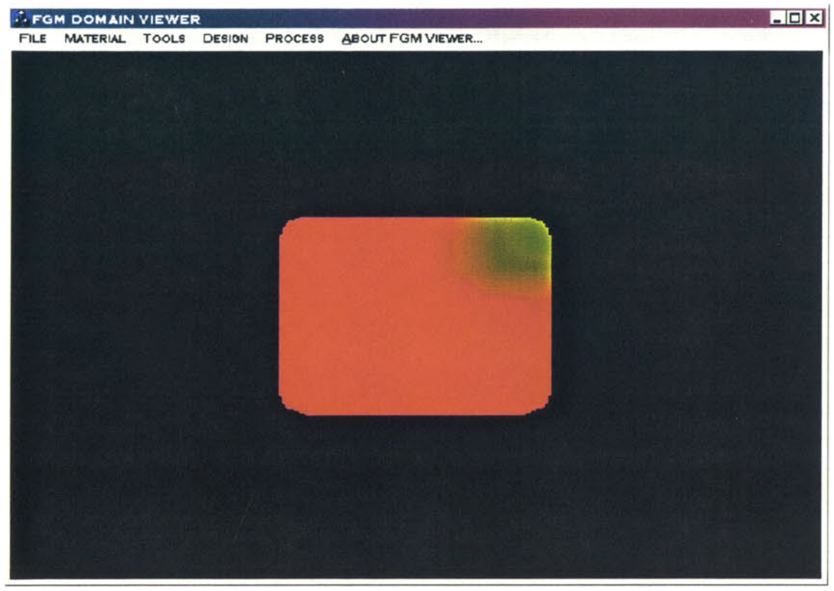


Figure 7-14: Slice of above at Z=20

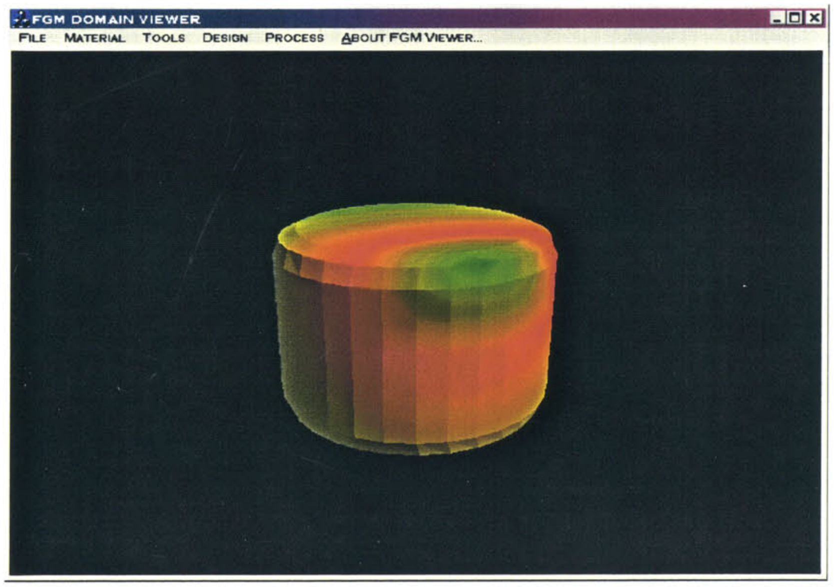


Figure 7-15: Design first through distance to STL mesh (0-90) then design within that STL boundary through distance to that STL mesh (0-40)

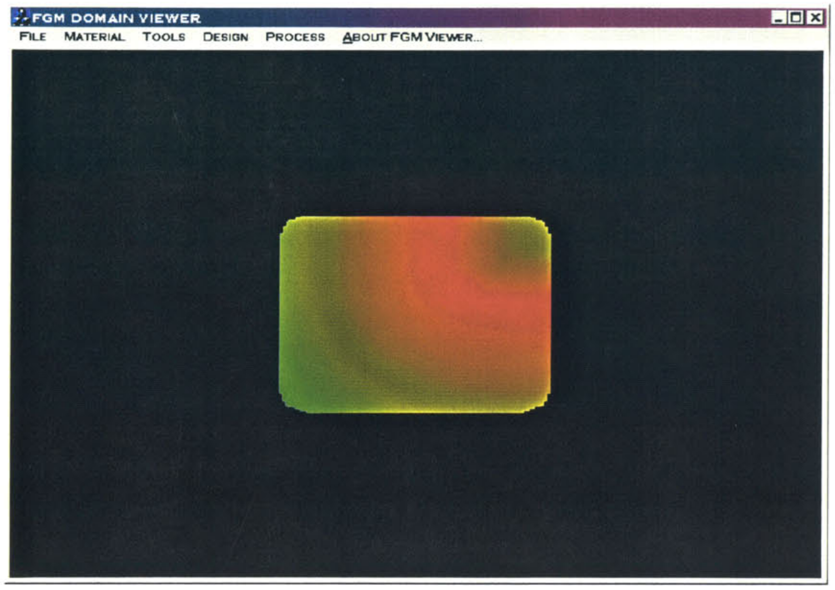


Figure 7-16: Design first through distance to STL mesh (0-90) then design within that STL boundary through distance to that STL mesh (0-40)

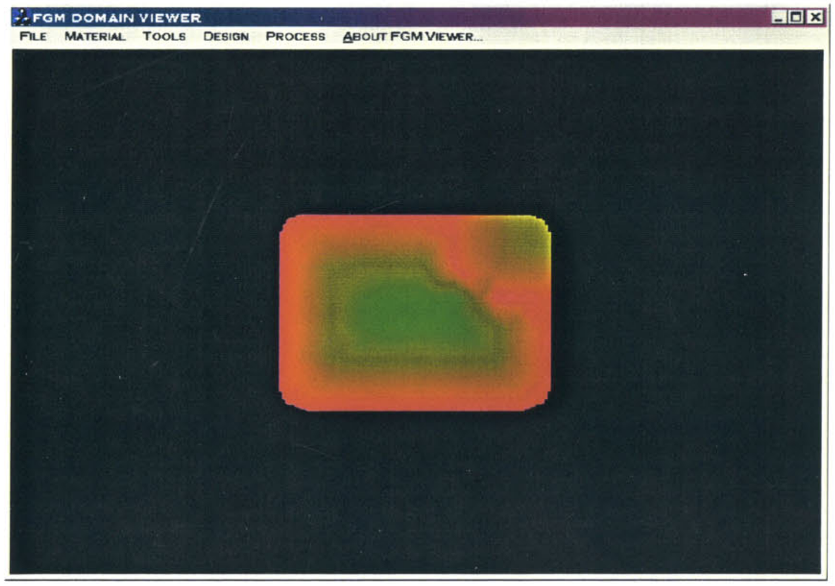


Figure 7-17: Slice (Z=20): Design first through distance to boundary (0-90) then design within a STL boundary through distance to that STL mesh (0-40)

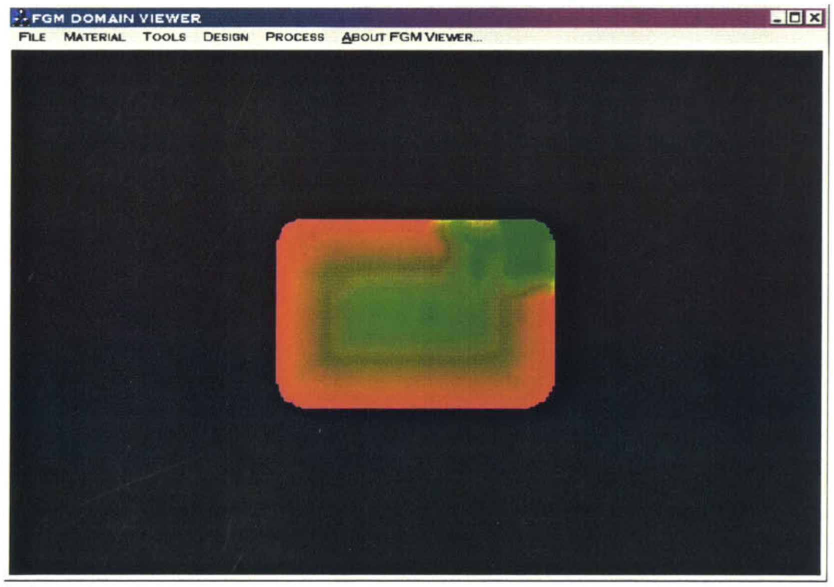


Figure 7-18: Slice (Z=10): Design first through distance to boundary(0-90) then design within a STL boundary a constant material

Chapter 8

Conclusions and recommendations

8.1 Conclusions

As part of the larger project of "Modeling and Designing Functionally Graded Material Components for Fabrication with Local Composition Control", this thesis addresses the issue of the development of efficient algorithms for design and composition interrogation of FGM solids. In order to represent graded material composition, a model has to be divided into sub-regions. Enhancement of the efficiency in designing FGM becomes inevitable when the representation of the FGM model has a large number of sub-regions. Similarly the algorithm for composition evaluation of an FGM has to be efficient as well. Starting with a finite-element with topology model, efficient algorithms for design and composition evaluation have been developed through the methods of bucket sorting, distance transform, and point classification, etc. The analysis on the developed algorithms and experimental results demonstrated these algorithms are effective. The developed algorithms have also been integrated into an existing FGM modeling, designing and post-processing program through software development techniques, which provides the pathway for the designer to design and model FGM and then see an FGM part fabricated through LCC, especially through 3D Printing process.

8.2 Recommendations

There are many potential directions for future work on the topic of FGM design. First, more composition design functions need to be developed, for example, the composition can be designed as a function of the minimum distance function and the corresponding boundary facet on which the projection of the query point happens. In order to enable the users to design the compositions as piecewise polynomial or rational forms functions of the minimum distance function, the maximum of the minimum distance values for all the points inside a solid needs to be calculated. Another prospective plan is bringing the design tool to a higher level to enable more design methods to capture users' real world design intentions that are often physical, descriptive or even an esthetic. Design in different dimensions may allow more varieties of FGM parts, i.e. in 2D plane domain, 2.5D domain or 3D domain or even design on the 3D surface of a model then offset both geometry and materials based on design intention. For the sake of effective design and redesign, good visualization is also necessary. The visualization methods can be isosurface extraction, cuberilles and color-coded planar sections. In order to produce FGM successfully through SFF processes, it is also necessary for the design system to be adjusted according to design rules that come from the process limits, etc. A dithering algorithm plays an important role as an interface between the ideal CAD model and its 3DP machine instructions. The current 2D dithering algorithm may extended to volume dithering algorithm that can minimize the low frequency textures. In order to evaluate the functionality of an FGM part, it is necessary to analyze the physical properties as functions of material composition. The design system may provide the basic information, such as maximum material gradient, minimum material gradient, iso-surfaces etc. or more advanced algorithms for analyzing physical properties directly. When we think about the nature around us, we can see the world is full of natural FGM, therefore, it is also possible that the design of FGM be assisted by reverse engineering to take a model from reality. Apart from the application directly in SFF fabrication, the research on design of FGM can be also oriented to model exchange, distribution through Internet etc. Design of FGM is closely related with the representation scheme of FGM, therefore the development of a general, efficient and flexible modeling method is very important. For example, general adaptive subdivision of a solid model could reduce the model size while keep the accuracy of the model, the subdivision of the models can be either structural or non-structural and subjected to different applications. Similar to the fact that complex mechanical parts are often an assembly of simpler parts, FGM can also be designed using a library of FGM components, with each component representing a subdivided element.

Appendix A

Development on FGMViewer

A.1 Extension to FGMViewer system

A.1.1 Introduction

As mentioned in the thesis, the algorithms developed in this thesis have all been integrated with an existing FGM domain viewer that has a pure finite element implementation. Using object oriented programming methods, the efficient finite element mesh based model is developed from the pure FEM mesh, the data structure is already described in Chapter 3. In this appendix, the extension of the interface and the major classes are described. In order to learn about the whole FGM viewer system please refer to [14].

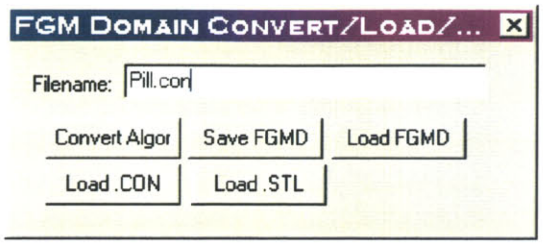
A.1.2 Menu extension

A.1.2.1 File

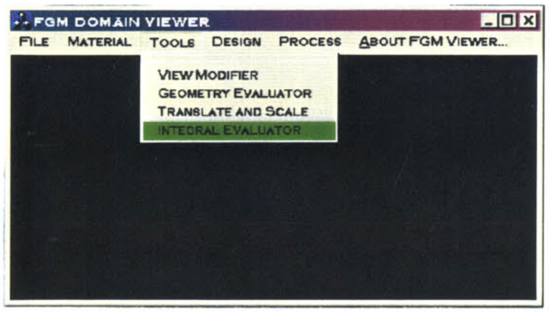
The file access window *Convert/Load/save* has been extended to load a tetrahedral mesh into the efficient FEM representation, and also an input of .STL (ascii format) file is allowed to view geometry represented in triangular facets.

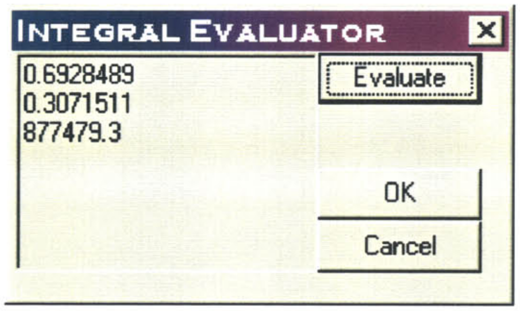
A.1.2.2 Utilities

Integral Evaluator Apart from the tools in the existing FGM domain viewer, an item for evaluation of the integral composition volume ratios and volume of the object is



added. Once the button *evaluate* is pressed, the program will give the volume ratio of each material and the volume of the object in a matrix form.





```
0.6928489
0.3071511
877479.3
```

Geometry evaluator The *Geometry evaluator* menu is connected to efficient method.

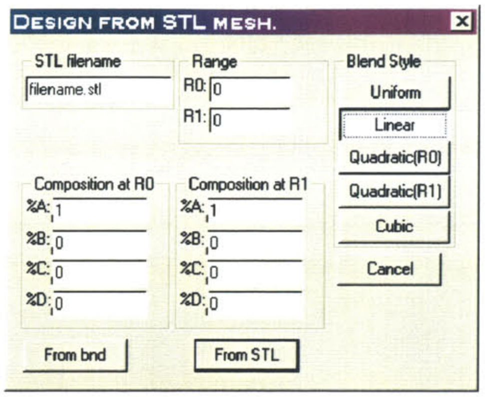
Translate and scale Extend the translation and scale to the efficient FEM structure including bucketing system.

View Modifier The *Geometry* and *Composition* rendering are done by only rendering the geometry (or composition) of the boundary triangular facets. The rendering of *slices (material)* are realized by ray tracing method.

A.1.2.3 Design

From STL shell This window is extended to incorporate design through distance function to the boundary of the solid except for a given .STL shell.

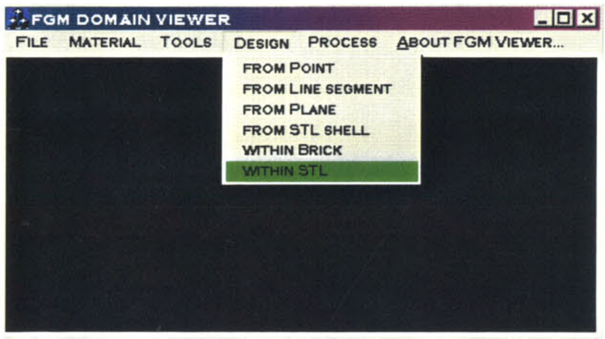
Within STL This window is added to allow user to design only the control compositions that are in a given .STL region through distance functions.

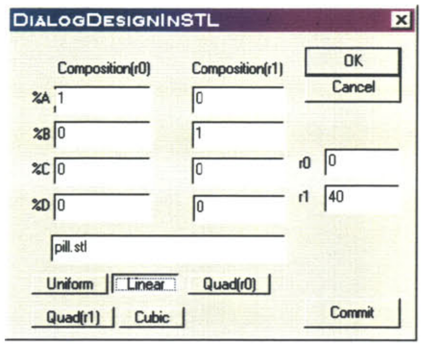


STL filename: filename.stl

Range: R0: 0 R1: 0

Composition at R0		Composition at R1	
%A:	1	%A:	1
%B:	0	%B:	0
%C:	0	%C:	0
%D:	0	%D:	0





	Composition(r0)	Composition(r1)
%A	1	0
%B	0	1
%C	0	0
%D	0	0

r0: 0
r1: 40

pill.stl

A.1.3 Major extension in classes

A.1.3.1 FGMFEMesh_hl

The class FGMFEMesh_hl has already been described in Chapter 3. Being derived from class FGMFEMesh by inheritance, this class shares all the data members of FGMFEMesh. Apart from the common data of FEM mesh, this class maintains the list of tetrahedra that contain boundary facets and a bucket sorting system. Because this class is a descendent of class FGMDomain, it inherited all the methods from class FGMDomain and FGMFEMesh. In addition, it possesses its own methods that are implementation of the algorithms described in this thesis.

A.1.3.2 BucketSys

The class "BucketSys" provides the bucket sorting methods and other algorithms that are described in this thesis.

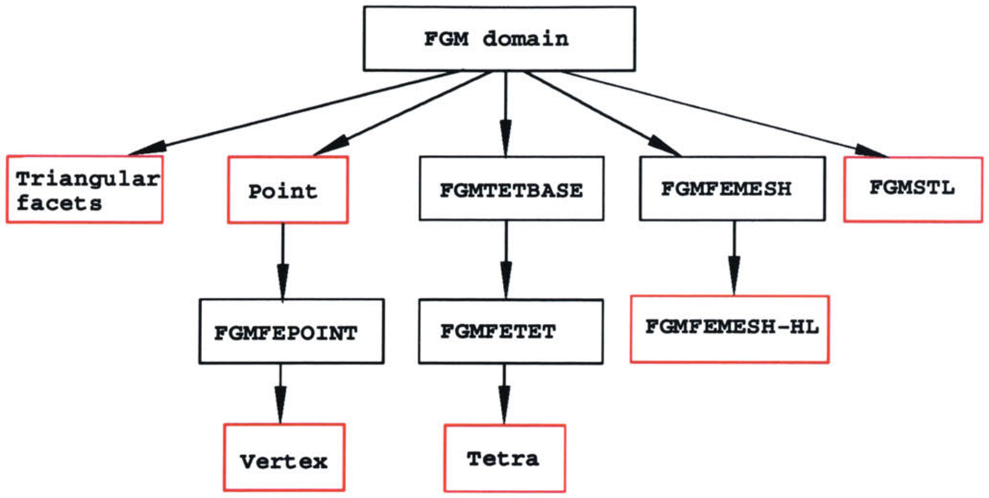


Figure A-1: Inheritance tree of FGMviewer object classes

A.1.3.3 Composition design function classes

mfunctionSTL "mfunctionSTL" class is extended to have distance either to an input .STL boundary or the boundary of the solid itself.

mfunctionSTLRegion "mfunctionSTLRegion" class is added to allow applying various distance functions to certain part of the solid that is defined by an input .STL boundary.

FGMFEMesh_hl:FGMFEMesh

BoundTetraList: The list of tetrahedra that contain boundary facets.

MyBuckets: The bucket sorting system of FGMFEMesh.

---

initModel(datastream): initialization of the model

invokeBkt(double, double ,double, double, double, double): invoke bucket system and preprocess the model by bucket sorting boundary facets and digital transform,etc.

setVertsToBuck(): bucket sorting vertices.

getU(datacontainer & X): get barycentric coordinates of query point

getTris(Triangle tris, int n):get boundary facets list

pointLocus(Point & Q): efficient point location searching method

faceNotBound(int fi, int tind);

nbrOfTetAtFace(int Tind, int find);

get_bndTetra(fgmfepoint vp1, fgmfepoint vp2, fgmfepoint vp3);

TetraOfBdFt(Triangle& nearWall);

rayTracer(Point& Bp, Point& Ep, double rslu);

planeCut(Point& p1, Point& p2, Point& p3, double rslu);

clusterPlPlane(Point& p1, Point& p2, Point& p3, double d, double rslu);

. . .

Tetra:FGMFETET

---

stat[4]: status of four faces

Vertex:FGMFEPoint

PrTetras: back indices to parent tetrahedra.

BucketSys

---

n_t: total number of buckets in the system

n_x, n_y, n_z: number of buckets along 3 coordinate axes

xmin, ymin, zmin, xmax, ymax, zmax: extreme coordinates of the bucket system

lb: side length of bucket in Cartesian coordinates.

BucketCon BA; pointer to a three dimensional pointer array.

BucketQueue CurrentQueue: queue of buckets stored for operation.

---

readSTL(char infile): initialization with .STL boundary

prePro(Triangle tris, int n): preprocessing with a list of triangular facets

PutInBuckSys(Triangle tris): distribute a list of triangular facets into buckets

DistTrans(): distance transform of buckets with respect to the input trimmed boundary

setBktsSign(): calculate the sign of buckets with respect to the boundary

computeR(Point Q): distance function from a query point to the boundary

PointIn(Point& p): test if query point is contained in the boundary

scale(scale factor)

translate($\Delta x, \Delta y, \Delta z$)

. . .

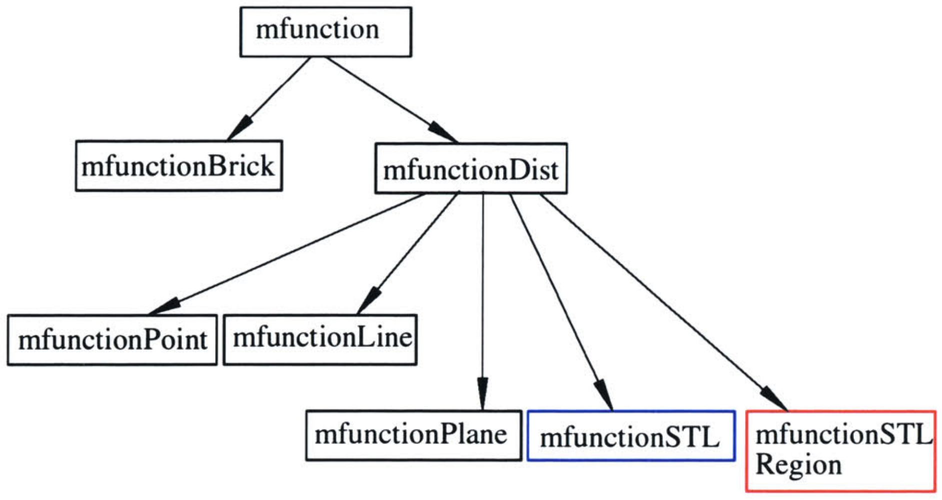


Figure A-2: Inheritance tree of mfunction classes

mfunctionSTLRegion:mfunctionDist

BucketSys bucketsort: Bucket sort of facets

---

evaluatem(Point Q): evaluate composition only if Q is inside the boundary

A.2 Example of the use of FGMViewer

STEP 1 Start the **FGM Domain Viewer** by double clicking its desktop icon and click **OK** to close the welcome **About FGM Domain Viewer** dialog.

STEP 2 Choose the **Convert/Load/Save** menu item from the **File** menu.

STEP 3 Enter the .con file name in the edit box of **FGM Domain Convert/Load/Save** dialog; here it is M7030.con and click the **Load CON** button.

STEP 4 Choose the **View Modifier** menu item from the Utilities menu and the **FGM View Modifier** dialog will appear.

STEP 5 Click the **Geometry** button to see the geometric boundary of the part and click the **Scale** arrow to enlarge the image.

STEP 6 Choose the **Translate and Scale** menu item from the **Utilities** menu.

STEP 7 Click the **Position at (0,0,0)** button and then click the **Apply** button.

STEP 8 Click the **Geometry** button twice and check the two lines of coordinates on the left bottom corner of the FGM View Modifier dialog to see if the first line is(0,0,0); if not, repeat STEP 7 and STEP 8 until it is (0,0,0).

STEP 9 Choose the **Create material system** menu item from the **Material** menu.

STEP 10 Enter the name and representing color of each material and click the **Add material** button; here we have two materials binder (A, represented in red) and steel (B, represented in green).

STEP 11 Click the **Done** button after you defined all material. (you can change the color of the materials using the dialog **Adjust Material Colors**).

STEP 12 Choose the **within Brick** menu item from the **Design** menu.

STEP 13 Enter a box that is large enough to include the part. Check the coordinate on the left bottom corner of the FGM View Modifier dialog. This is the bounding box of the part. Enter a coordinate with larger values for all three axis. Here the value is (150, 110, 50).

STEP 14 Enter a composition value that will be assigned to the whole area inside the above box. Here for example, we choose 0.85 for B, 0.15 for A, which means 85% B, 15% A.

STEP 15 Click the **Commit** button and then click the **OK** button in the **Elapse time reporting** dialog.

STEP 16 Choose the **from STL shell** menu item from the **Design** menu.

STEP 17 Define the range of the region that this operation will influence; here the range is from the boundary (0mm) to 5mm inside the part.

STEP 18 Enter the composition value at the boundary of the region; here the composition is 0% steel, 100% binder on the boundary and 15 % binder, and 85% steel 5 mm inside the part.

STEP 19 Define how the composition vector changes from the values on the boundary to the values at 5mm inside the part; click the **Linear** button to give a linear interpolation.

STEP 20 Click the **From bnd** button to perform the design.

STEP 21 View the designed product; choose the menu item **View Modifier** from the Utilities menu.

STEP 22 Click the **Composition data** button to see the composition at control points. You can select the proper view to see clear by selecting angles, scale. When you choose the **Cube scale** with a value less than 1, the viewer displays the exploded view in cuberille. You can also click the **Composition** button to see the composition on the boundary.

STEP 23 Click the **Composition data** button again to clear the view.

STEP 24 View the composition on a cutting plane; enter the values that define the cutting plane in the edit boxes A,B,C,D, if you want to cut a cluster of parallel planes, enter the number of planes and the step distance too.

STEP 25 Click the **Go** button and then click the **Slice (material)** button. If you want to clear the view, click the **Clear rendered slices** button.

STEP 26 Evaluate composition at a point you are interested; select the **Geometry evaluator** menu item from the **Utilities** menu.

STEP 27 In the appeared dialog, enter the coordinates of the query point and click the **Evaluate composition** button, the composition vector will appear in the upper right read-only text box, then close the dialog.

STEP 28 Evaluate the volume percentage of each material and the volume of the part; select the **Integral evaluator** menu item from Utilities menu.

STEP 29 In the popped dialog, click the **Evaluate** button, the volume percentages of materials will appear in vector format, and the last value is the volume of the part. Then click the **Cancel** button to exit.

STEP 30 Choose the **Translate and Scale** menu item from the **Utilities** menu.

STEP 31 Enter 1000 in the **Scale** edit box and click the Apply button. This step changes the unit from mm to micron. It is the preparation for the slicing.

STEP 32 Choose the **Slice to file** menu item from the **Process** menu.

STEP 33 Enter the layer thickness in the corresponding edit box (in micron) ; here it is 170 micron.

STEP 34 Enter the name of the file that you want for the generated material slice (.HIN file); here it is Midget.HIN.

STEP 35 Click the **Process FGM object into layers** button and click the **OK** buttons in the two **elapse time reporting** dialog.

STEP 36 Redesign the FGM; select the **From STL shell** menu item from the **Design** menu, enter the .STL file you want to design from in the upper left edit box. Here it is Midget.STL.

STEP 37 Do the same as in STEP 17-19.

STEP 38 Click the **From STL** button to perform the design.

STEP 39 Repeat the STEP 21-25 to view the redesigned part.

STEP 40 Redesign the FGM; select the **Within STL** menu item from the **Design** menu.

STEP 41 Enter the .STL file that you want to design within.

STEP 42 Enter the range distance value from the .STL file.

STEP 43 Enter the composition vector values at two ends of the range.

STEP 44 Click the **Linear** button to perform linear interpolation between the range.

STEP 45 Click the **Commit** button to perform design and then click the **Cancel** button to cancel the dialog.

STEP 46 Repeat the STEP 21-25 to view the redesigned part.

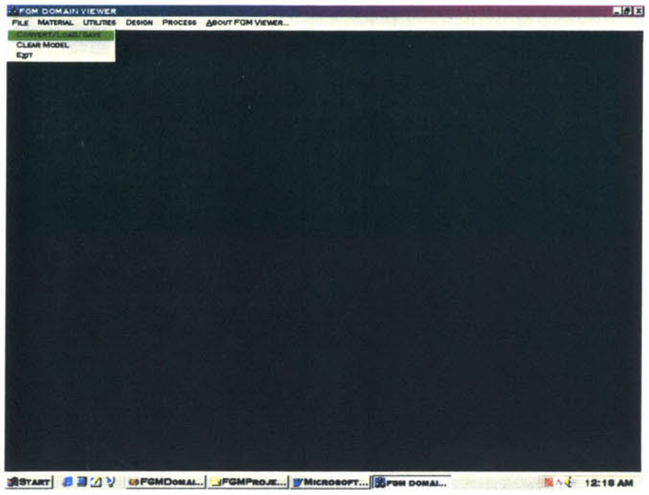
STEP 47 Select the **Clear model** menu item from the **File** menu to cancel the current model.

STEP 48 Load another model; choose the **Convert/Load/Save** menu item from the **File** menu.

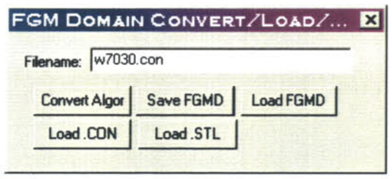
STEP 49 Enter the .stl file name in the edit box of **FGM Domain Convert/Load/Save** dialog; here it is Midget.stl and click the **Load STL** button.

STEP 50 Select the **View modifier** menu item from the **Utilities menu** to view this STL model.

STEP 51 Select the **Exit** menu item from the **File** menu.

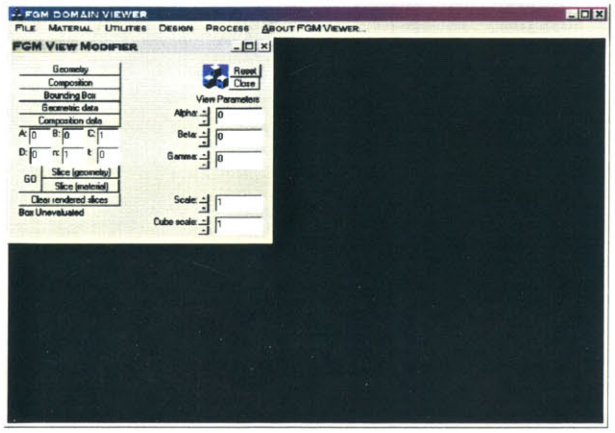


STEP 2



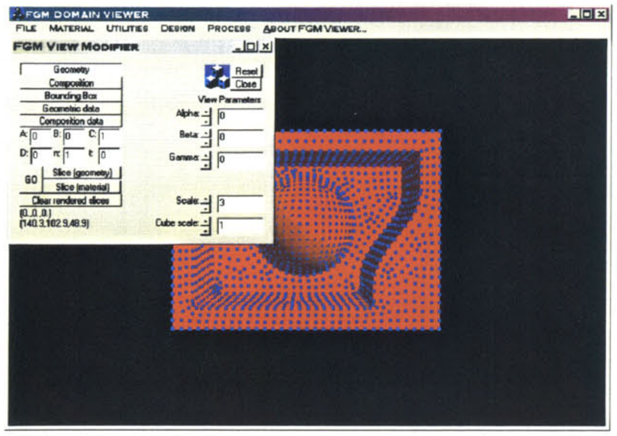
Filename: w7030.con

STEP 3



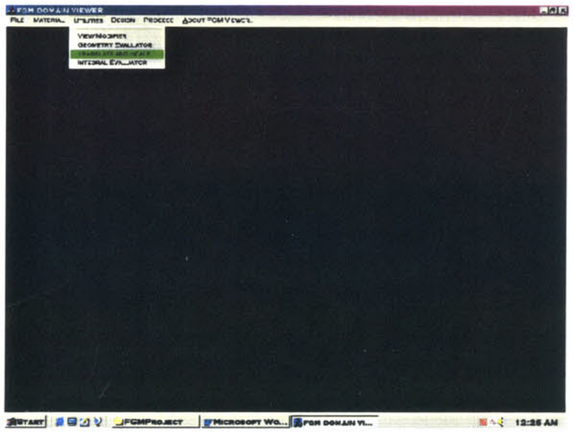
A: 0 B: 0 C: 1
D: 0 n: 1 t: 0
Box Unevaluated
Alpha: 0
Beta: 0
Gamma: 0
Scale: 1
Cube scale: 1

STEP 4

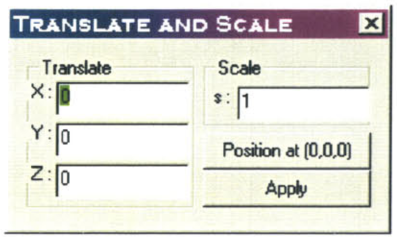


A: 0 B: 0 C: 1
D: 0 r: 1 t: 0
(0.,0.,0.)
(140.3,102.9,48.9)
Alpha: 0
Beta: 0
Gamma: 0
Scale: 3
Cube scale: 1

STEP 5

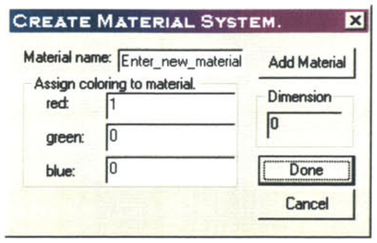


STEP 6



X: 0
Y: 0
Z: 0
s: 1

STEP 7


CREATE MATERIAL SYSTEM.

Material name: Enter_new_material

Assign coloring to material.

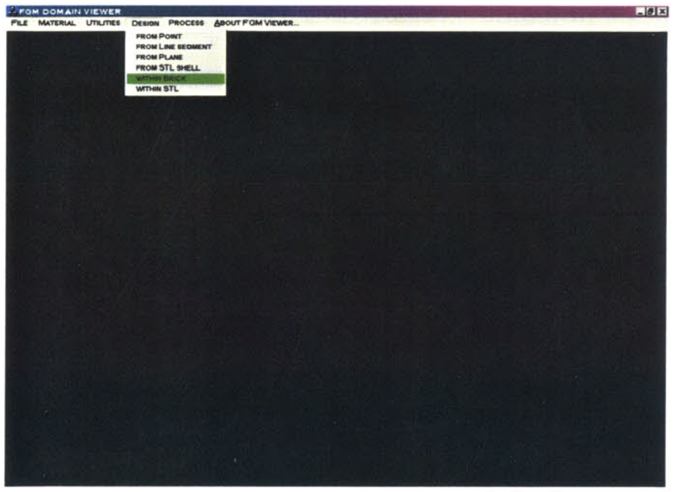
red: 1

green: 0

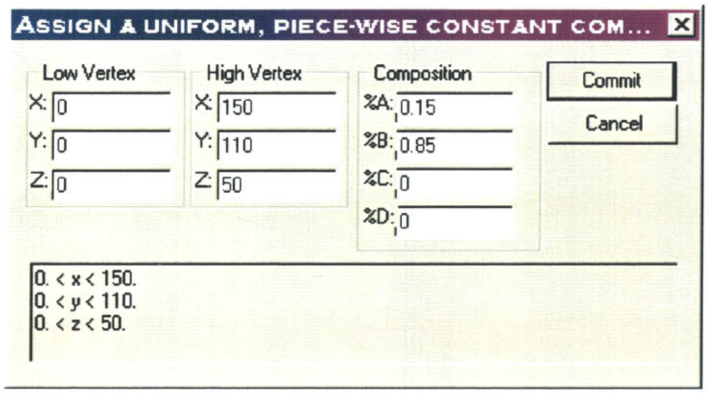
blue: 0

Dimension: 0

STEP 10



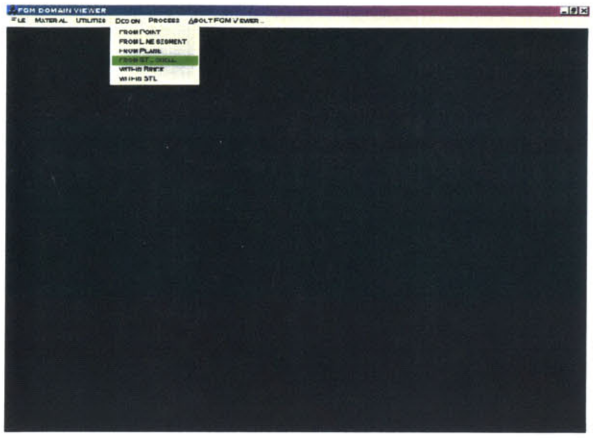
STEP 12


ASSIGN A UNIFORM, PIECE-WISE CONSTANT COM...

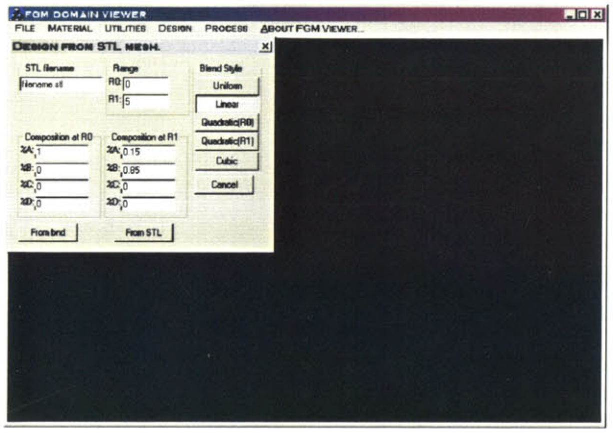
Low Vertex	High Vertex	Composition
X: 0	X: 150	%A: 0.15
Y: 0	Y: 110	%B: 0.85
Z: 0	Z: 50	%C: 0
		%D: 0

```
0. < x < 150.
0. < y < 110.
0. < z < 50.
```

STEP 13



STEP 16



DESIGN FROM STL MESH.

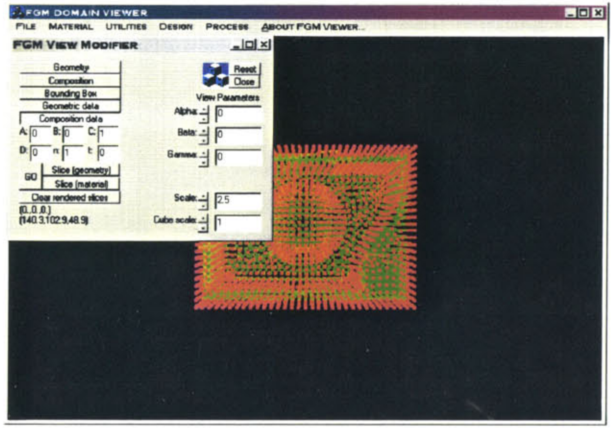
STL filename: filename.stl

Range
R0: 0
R1: 5

Composition at R0
%A: 1
%B: 0
%C: 0
%D: 0

Composition at R1
%A: 0.15
%B: 0.85
%C: 0
%D: 0

STEP 17



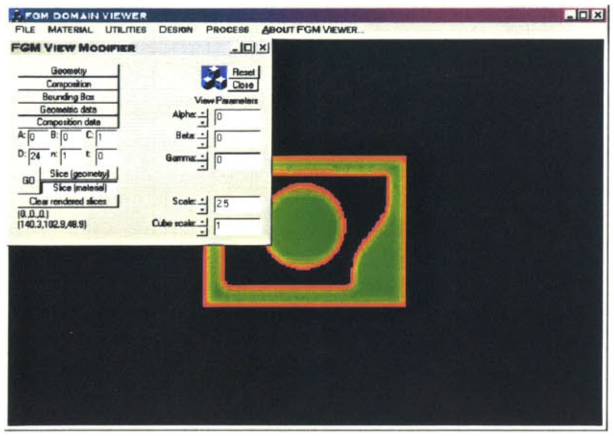
FGM VIEW MODIFIER

A: 0 B: 0 C: 1
D: 0 n: 1 t: 0

(0.,0.,0.)
(140.3,102.9,48.9)

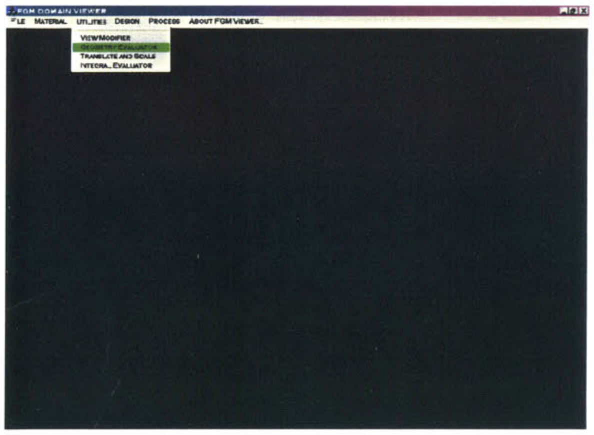
View Parameters
Alpha: 0
Beta: 0
Gamma: 0
Scale: 2.5
Cube scale: 1

STEP 21

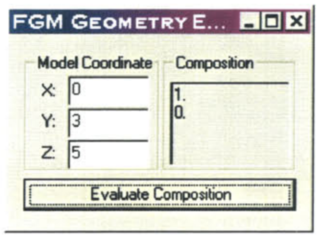


A: 0 B: 0 C: 1
D: 24 n: 1 t: 0
(0.,0.,0.)
(140.3,102.9,48.9)
Alpha: 0
Beta: 0
Gamma: 0
Scale: 2.5
Cube scale: 1

STEP 24

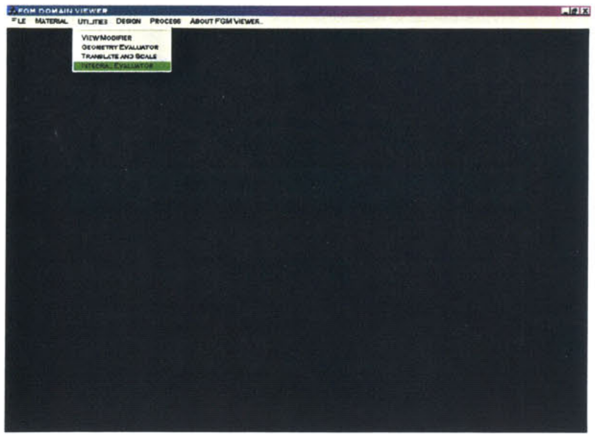


STEP 26

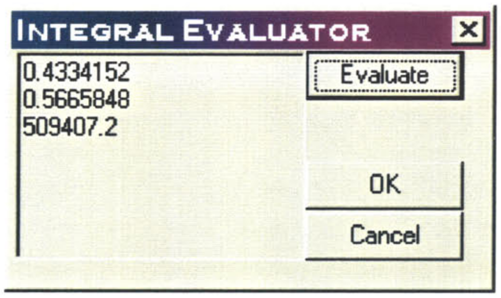


Model Coordinate
X: 0
Y: 3
Z: 5
Composition
1.
0.

STEP 27

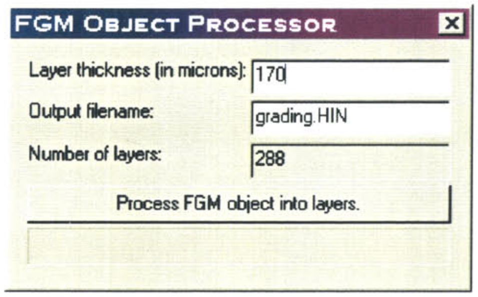


STEP 28



```
0.4334152
0.5665848
509407.2
```

STEP 29



Layer thickness (in microns): 170

Output filename: grading.HIN

Number of layers: 288

STEP 32

Appendix B

Geometric Algorithms

B.1 Algorithm for testing if a point is contained in a tetrahedron

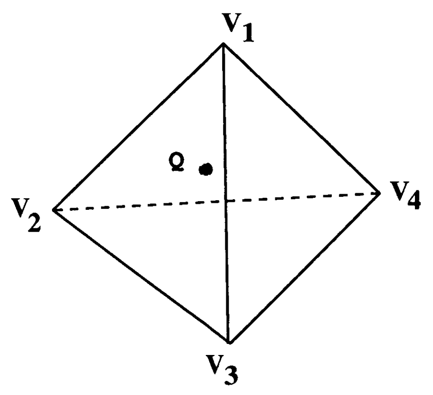


Figure B-1: Test if a given point is contained in a tetrahedron

Algorithm 6 isQinTet(Point Q, Tetrahedron $V_1V_2V_3V_4$); (Figure B.1)

```
1: calculate the mixed product $(\overrightarrow{V_3V_1} \times \overrightarrow{V_3V_2}) \cdot \overrightarrow{V_3Q}$;
2: calculate the mixed product $(\overrightarrow{V_4V_3} \times \overrightarrow{V_4V_2}) \cdot \overrightarrow{V_4Q}$;
3: calculate the mixed product $(\overrightarrow{V_1V_4} \times \overrightarrow{V_1V_2}) \cdot \overrightarrow{V_1Q}$;
4: calculate the mixed product $(\overrightarrow{V_1V_3} \times \overrightarrow{V_1V_4}) \cdot \overrightarrow{V_1Q}$;
5: if all four mixed products are positive or all four mixed products are negative then
6:   return true;
7: else
8:   if any of these mixed products is zero then
9:     if query point is on the corresponding face then
10:      return true;
11:    else
12:      return false;
13:  else
14:    return false;
```

B.2 Algorithm for testing if a point is contained in a triangle

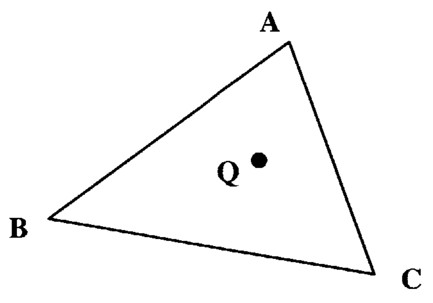


Figure B-2: Test if a given point is contained in a triangular facet

Algorithm 7 isQinTri(Point Q, Triangle ABC); (Figure B.2)

```
Require: The given point is on the plane of the triangular facet
 1: $\vec{v_1} \Leftarrow \vec{QA} \times \vec{QB}$;
 2: $\vec{v_2} \Leftarrow \vec{QB} \times \vec{QC}$;
 3: $\vec{v_3} \Leftarrow \vec{QC} \times \vec{QA}$;
 4: if $\vec{v_1} \cdot \vec{v_2} > 0$ and $\vec{v_2} \cdot \vec{v_3} > 0$ and $\vec{v_1} \cdot \vec{v_3} > 0$ then
 5:   return true;
 6: else
 7:   if any of these inner products is zero then
 8:     if query point is on the corresponding edge then
 9:       return true;
10:     else
11:       return false;
12:   else
13:     return false;
```

B.3 Algorithm for testing if a line segment intersects a triangular facet

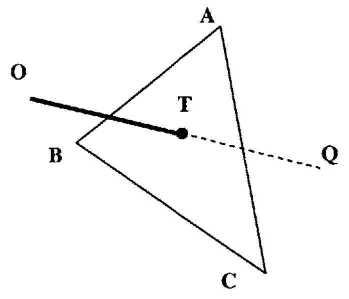


Figure B-3: Test if a line segment intersects a triangular facet

Algorithm 8 isOQinscTri(Point O, Point Q, Triangle ABC); (Figure B.3)

```
1: calculate the mixed product $(\vec{CA} \times \vec{CB}) \cdot \vec{CO}$;
2: calculate the mixed product $(\vec{CA} \times \vec{CB}) \cdot \vec{CQ}$;
3: **if** the two mixed products have different signs **then**
4:   calculate the intersection T of $\bar{OQ}$ and Triangle ABC;
5:   **if** T $\in$ ABC is true **then**
6:     return true;
7:   **else**
8:     return false;
9: **else**
10:   **if** only one of these mixed products is zero **then**
11:     **if** the corresponding point is $\in$ ABC **then**
12:       return true;
13:     **else**
14:       return false;
15:   **else**
16:     **if** both mixed products equall zero **then**
17:       **for** each edge of ABC **do**
18:         **if** it intersects $\bar{OQ}$ **then**
19:           return true;
20: return false;
```

Bibliography

[1] T. Asano, M. Edahiro, H. Imai, M. Iri, and K. Murota. Practical use of bucketing techniques in computational geometry. In G. T. Toussaint, editor, *Computational Geometry*, pages 153–195. 1985.

[2] L. Bardis and N. M. Patrikalakis. Topological structures for generalized boundary representations. MITSG 94-22, MIT Sea Grant College Program, Cambridge, Massachusetts, September 1994.

[3] M. Binnard. *Design by composition for rapid prototyping.* PhD thesis, Mechanical Engineering, Stanford University, San Jose, CA, February 1999.

[4] G. Borgefors. Distance transformations in arbitrary dimensions. *Computer Vision, Graphics, and Image Processing*, 27:321–345, 1984.

[5] E. Brisson. Representing geometric structures in d dimensions: Topology and order. *Discrete and Computational Geometry*, 9:387–426, 1993.

[6] W. Cho, T. Maekawa, and N. M. Patrikalakis. Topologically reliable approximation of composite Bézier curves. *Computer Aided Geometric Design*, 13(6):497–520, August 1996.

[7] W. Cho, T. Maekawa, N. M. Patrikalakis, and J. Peraire. Topologically reliable approximation of trimmed polynomial surface patches. *Graphical Models and Image Processing*, 61(2):84–109, March 1999.

[8] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990.

[9] D. H. Greene. *Mathematics for the Analysis of Algorithms.* Birkhäuser, 1990.

[10] S. Holzner. *Microsoft Visual C++ 5.* SYBEX Inc., 1997.

[11] J. Hoschek and D. Lasser. *Fundamentals of Computer Aided Geometric Design.* A. K. Peters, Wellesley, MA, 1993. Translated by L. L. Schumaker.

[12] IGES/PDES Organization, U.S. Product Data Association, Fairfax, VA. *Digital Representation for Communication of Product Definition Data, US PRO/IPO-100, Initial Graphics Exchange Specification (IGES) 5.2*, November 1993.

[13] American National Standards Institute. *Product Data Exchange Using STEP (PDES) Part 42, Integrated generic resources: geometric and topological representation.* 1995.

[14] T. R. Jackson. *Analysis of Functionally Graded Material Object Representation Methods.* PhD thesis, M.I.T., Cambridge, MA, January 2000.

[15] T. R. Jackson. FGM object modeler. Design Laboratory Memorandum 2000–2, MIT, Department of Ocean Engineering, Cambridge, MA, February 2000.

[16] T. R. Jackson, H. Liu, N. M. Patrikalakis, E. M. Sachs, and M. J. Cima. Modeling and designing functionally graded material components for fabrication with local composition control. *Materials and Design*, 20(2/3):63–75, June 1999.

[17] R. Johnsonbaugh and M. Kalin. *Object-Oriented Programming in C++.* Prentice Hall, 1995.

[18] V. Kumar, D. Burns, D. Dutta, and C. Hoffmann. A framework for object modeling. *Computer Aided Design*, 31(9):541–556, August 1999.

[19] V. Kumar and D. Dutta. An approach to modeling multi-material objects. In C. Hoffman and W. Bronsvort, editors, *Fourth Symposium on Solid Modeling and Applications, Atlanta, Georgia, May 14-16, 1997*, pages 336–353, New York, 1997. ACM SIGGRAPH.

[20] V. Kumar, P. Kulkarni, and D. Dutta. Adaptive slicing of heterogeneous solid models for layered manufacturing. Technical Report UM-MEAM-98-02, University of Michigan, Ann Arbor, MI, January 1998.

[21] M. Mäntylä. *An Introduction to Solid Modeling.* Computer Science Press, Rockville, Maryland, 1988.

[22] A. Marsan, V. Kumar, D. Dutta, and M. Pratt. An assessment of data requirements and data transfer formats for layered manufacturing. Technical Report NISTIR 6216, U.S. Department of Commerce, Geithersburg, Maryland, 1999.

[23] E. P. Mücke, I. Saias, and B. Zhu. Fast randomized point location without preprocessing in two- and three-dimensional delaunay triangulations. *Computational Geometry*, 12:63–83, 1999.

[24] J. O'Rourke. *Computational Geometry in C.* Cambridge University Press, 1993.

[25] J. Pegna and A. Safi. Cad modeling of multi-modal structures for free-form fabrication. In *Presentation at Solid Freeform Fabrication Symposium*, Austin, Texas, August 1998.

[26] S. Rajagopalan, R. Goldman, K. Shin, V. Kumar, M. Cutkosky, and D. Dutta. Design, processing and freeform-fabrication of heterogeneous objects. Technical Report UM-MEAM-99-10, University of Michigan, Department of Mechanical Engineering, Ann Arbor, MI, 1999.

[27] D. F. Rogers. *Procedural Elements for Computer Graphics.* McGraw-Hill, 1985.

[28] E. Sachs, J. Haggerty, M. Cima, and P. Williams. Three-dimensional printing techniques. U.S. Patent No. 5,204,055, April 20 1993.

[29] E. M. Sachs, N. M. Patrikalakis, D. Boning, M. J. Cima, T. R. Jackson, and R. Resnick. The distributed design and fabrication of metal parts and tooling by 3d printing. In *Proceedings of the 1998 NSF Design and Manufacturing Grantees Conference, Cintermex Conference Center, Monterrey, Mexico, January 1998*, pages 35–36. Arlington, VA: NSF, 1998.

[30] K. Voss. *Discrete Images, Objects, and Functions in Z_n*. Springer-Verlag, 1993.

[31] M. Woo, J. Neider, and T. Davis. *OpenGL Programming Guide, second edition.* Addison-Wesley Developers Press, 1997.

[32] H. Wu, E. M. Sachs, N. M. Patrikalakis, D. Brancazio, J. Serdy, T. R. Jackson, W. Cho, H. Liu, M. Cima, and R. Resnick. Distributed design and fabrication of parts with local composition control. In *2000 NSF Design and Manufacturing Grantees Conference*, Vancouver, BC, Canada, January 2000. http://deslab.mit.edu/3dp/3dppapers.html, http://www.engr.washington.edu/ uw-epp/nsf/.