# The Integration of Fluid Dynamics with a Discrete-Element Modelling System: Algorithms, Implementation, and Applications

by

Justin T. Klosek

B.S.E., Princeton University (1995)

Submitted to the Department of Civil and Environmental Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Civil and Environmental Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1997





Author .....
Department of Civil and Environmental Engineering
January 17, 1997

Certified by .....
John R. Williams
Associate Professor
Department of Civil and Environmental Engineering
Thesis Supervisor

Accepted by .....
Joseph M. Sussman
Chairman, Departmental Committee on Graduate Studies

# The Integration of Fluid Dynamics with a Discrete-Element Modelling System: Algorithms, Implementation, and Applications

by

Justin T. Klosek

Submitted to the Department of Civil and Environmental Engineering
on January 17, 1997, in partial fulfillment of the
requirements for the degree of
Master of Science in Civil and Environmental Engineering

## Abstract

The Discrete Element Method (DEM) is a numerical technique for analyzing the dynamic behavior of multibody, discontinous systems. The method solves the dynamic equilibrium equations for each body, subject to body and boundary interaction forces.

An overview of discrete-element methods and previous work using DEM is presented. Contact models and contact detection methods are discussed. Newtonian physics and time-integration techniques are detailed.

A discrete-element simulation system is extended to incorporate fluid flow and the interaction between the fluid and particle systems. Fluid flow is modeled using a finite-element Darcy flow, porous media "smeared" approach. The discrete-element/finite-element computational coupling is discussed and the integration of fluid effects into the discrete-element simulation are detailed. Techniques to calculate the fluid forces on an arbitrarily-shaped discrete body are developed. Explicit integration techniques are incorporated into the formulation to simulate dynamic changes in the fluid system.

The computational techniques presented here are validated against known analytical solutions to straightforward physical problems. Particle buoyancy is checked for various cases of $\rho_{particle}$ versus $\rho_{fluid}$. The dynamic solution for a fluid pressure wave is checked against the known analytical solution with excellent results. Directions for future work and ideas for potential applications of the methods are discussed.

Thesis Supervisor: John R. Williams
Title: Associate Professor
Department of Civil and Environmental Engineering

# Contents

# List of Figures

# Chapter 1

# Introduction

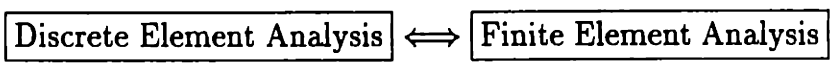
The field of computational mechanics has exploded in the past two decades with the advent of personal computers, providing fast, inexpensive computing power to researchers and practicioners alike. The development of matrix-based computational tools such as the finite-element method have led to the widespread use and acceptance of the method in computational mechanics, especially those problems which fall into the realm of civil and mechanical engineering.

However, even though the use and acceptance of the finite-element method is widespread, it does have limitations in terms of the type of problems it can be used to solve. The finite-element method tends to fail for discontinous problems, i.e. those problems involving granular materials, multi-body problems, or problems where the contact forces among bodies are significant. To this end, *discrete* element methods have been developed by Williams et. al. (1985) and others. The work has been expanded greatly in the last few years, especially due to the work of O'Connor (1996) and Rege (1996). In their work they have shown that the discrete-element method is a valid computational tool for multi-body mechanics problems and problems involving granular materials.

However, although the discrete-element method as presented by O'Connor and Rege has been proven to be a valid computational tool, it is still not robust in the sense that there are many problems which it cannot handle. The discrete-element method has been used to model a soil sample undergoing compression, as in Rege (1996) and Williams and Rege (1996), but these computations assume a dry soil sample—the discrete-element method as used in these publications does not account for any fluid which may be physically present in the sample.

This thesis presents one method of incorporating fluid-force loading into a discrete-element system. Using the finite-element method to calculate fluid pressures (assuming a continuous fluid region), the work here can be summarized as starting from


and expanding it to




Of course, while the overall concept is simple, the details in the coupling above (represented so simply by the arrow) are somewhat involved.

The next three chapters develop and explain the coupling between discrete-element analysis and finite-element fluid analysis, showing each step taken in the process to integrate the two systems. Chapter 2 presents an overview of discrete-element methods. Basic principles of discrete-element analysis are discussed. Previous work

is investigated and summarized. The MIMES discrete-element system is introduced.

Chapter 3 details the coupling of the two systems, including the finite-element formulation for the fluid elements, the mesh generation, and the solution of the finite-element system. Methods are developed to determine the fluid-loading forces on discrete-element particles and Finally the coupling involved within the code is discussed. The finite-element/discrete-element coupling requires the solution of some range-searching and geometric problems and efficient methods to solve these problems are developed.

Chapter 4 presents some examples of coupled fluid-force/discrete-element computations. Three test problems are developed to show that the expected, analytical result is returned by the coupled discrete-element/finite-element computational environment.

Chapter 5 concludes the work with a summary of this presentation and a look at the "knowledge added" here. Conclusions and future work directions are presented. Appendicies detail critical parts of the code, the fluid-loading commands added to the discrete-element system, and the commands for the finite-element solver.

# Chapter 2

# Discrete-Element Methods and the MIMES Discrete-Element System

This chapter presents the mathematical basis of the discrete-element method and summarizes previous work on the discrete-element method. The MIMES discrete-element system, selected as the base discrete-element package, is introduced.

## 2.1 The Discrete-Element Method: Basics

Perhaps Rege (1996) summarizes the Discrete-Element Method most concisely:

> Discrete Element Methods are a class of numerical techniques developed specifically to model systems that are inherently discrete in nature. The key feature of these methods is that they model systems as a collection of discrete objects. The objects interact through interparticle forces. These forces might arise from physical contacts with other particles or potential fields generated by the chemistry of the particles. The particle motion is based on physical laws such as Newtonian mechanics. (Rege, 1996, p. 20)

Essentially, computational tools based on the discrete-element method do not require the assumption of continuity implicit in finite-element solutions. In terms of finite

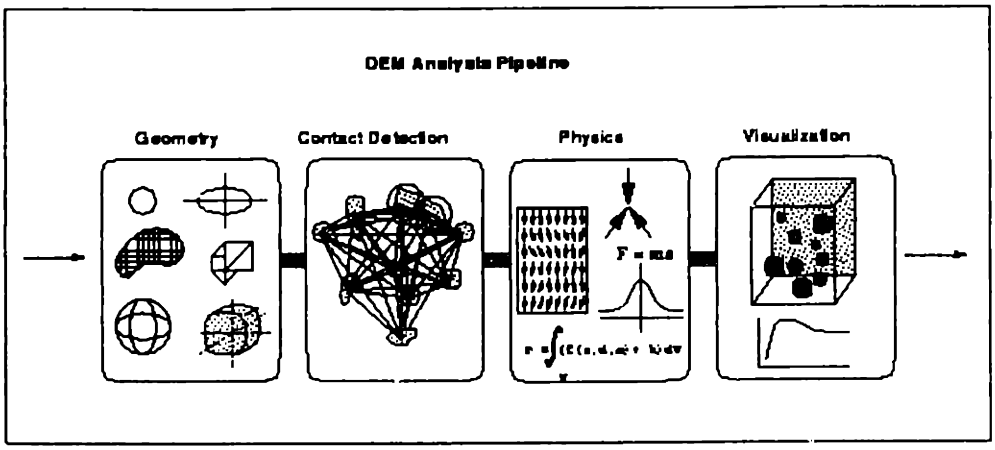



Figure 2-1: The Discrete Element Analysis Pipeline

elements, it is interesting to note, however, that Pande, Beer, and Williams (1989) show that "the underlying basis [for the discrete-element method] can be cast into a familiar Finite Element form. Indeed, a standard Finite Element code can be embedded in a Discrete Element system ... so that continuum deformations are handled using Finite Elements, while the contact and body interactions are handled using Discrete Element techniques." (Pande, Beer, and Williams, 1989, p. 222)

Figure 2-1[1] depicts the "pipeline" of a discrete-element analysis. A *system generator module* is employed to generate particle geometries and assemblages of particles. These particles are analyzed first in the *contact detection module* to determine particle-particle interactions, which is the heart of the discrete-element method. The contact forces are resolved in the *dynamics equation module*, which contains the mathematical basis behind the discrete-element method. Forces are calculated and integrated; the resultants are used to determine the motion and position of the objects. Finally, the *visualization module* presents results of the discrete-element run

[1]This figure from Rege (1996), p. 20.

for analysis and interpretation.

### 2.1.1 System Generator Module

In the *system generator module*, particle geometries and assemblages of the particles are defined. Boundary conditions for the analysis are also defined in this module.

To represent the particle geometry, two basic methods can be employed—an implicit functional representation and a boundary representation. In an implicit *functional* representation, the geometry of the particle is defined by a mathematical function. For instance, a solid circular particle with center $(h, k)$ and radius $r$ could be defined as

$$(x-h)^2 + (y-k)^2 \leq r. \tag{2.1}$$

Routines to check if a point $(x, y)$ is inside the particle are simple to implement: if the point satisfies the condition (2.1), it is "inside" the particle. Conversely, if the condition is not satisfied, the point is "outside" the particle.

In a *boundary* representation, particle geometries are described by a series of connected line segments. For instance, the disc detailed above in (2.1) could be represented as a sixteen-sided polygon, with points

$$\begin{gathered} P_1(h+r, k+r); \\ P_2(h + r\cos\tfrac{2\pi}{16}, k + r\sin\tfrac{2\pi}{16}); \\ P_3(h + r\cos\tfrac{4\pi}{16}, k + r\sin\tfrac{4\pi}{16}) \ldots . \end{gathered} \tag{2.2}$$

Line segments $P_1P_2$, $P_2P_3$, ..., complete the boundary representation of the particle. A boundary representation such as that in (2.2) allows for more general particle geometries than do simple functional representations. The computation time required for each of the representations differs as well; this is discussed below in Section 2.1.2.

Many different particle shapes have been used in discrete-element analyses, with varying success in their prediction of the actual, known physical behavior of the systems in question. Rege (1996) presents an excellent survey of the different particle shapes used in previous discrete-element implementations. Discs have been shown to be unsuitable for discrete-element simulations of some granular materials because

> ...the shape is not very representative of real materials. Discs are known to show excessive rolling and this is manifested in lower than realistic strengths exhibited in discrete-element simulations. Ellipses and superquadrics[2] overcome this shortcoming ... (Rege, 1996, p. 22)

It is also necessary to distribute the particles, once generated, into some accurate representation of a physical system. "It is difficult to generate a random distribution of particles of arbitrary shapes with a specific void ratio" (Rege, 1996, p. 21). The following methods are suggested for the generation of computed random samples:

1. Allow a set of particles to fall, under the influence of gravity, into a container which a size and shape representative of the problem of interest.

2. For a sparse distribution of particles, perform an isotropic compression simulation to create a closely-packed specimen. This also allows for generation of

[2]The general equation of a two-dimensional superquadric function is given as $F(x,y) = \left|\frac{x}{a}\right|^{n_1} + \left|\frac{y}{b}\right|^{n_2} - 1 = 0$. Superquadrics can represent circles ($a = b = 1$, $n_1 = n_2 = 2$), ellipses ($a \neq b$, $n_1 = n_2 = 2$), and various other shapes by varying the axis lengths $a$ and $b$, and exponents $n_1$ and $n_2$.

specimens subjected to different confining stresses. (Rege, 1996)

"The filling process is extremely time-consuming, often requiring more effort than the DEM experiment itself" (O'Connor, 1996, p. 171). That work presents another method, based on a fast packing algorithm, which may aid certain problems of particle distribution.

One concern with discrete-element analysis is the number of particles that make up the system under consideration. Rege (1996) gives an example:

> ...if are modeling a cubic meter of sand, let us say the volume of a single particle is 1 $mm^3$ and the void ratio is about 0.5. We would then need about $5 \times 10^8$ particles to generate an accurate model. The computational time required to simulate a system of $N$ particles is said to be $O(N^2)$ and can at best be reduced to $O(N \log N)$. ...this puts a severe constraint on the maximum number of particles that can be used to model a particular system. (Rege, 1996, p. 21)

O'Connor gives $10^{-5}$ seconds as "a conservative time to use for each *operation*" (O'Connor, 1996, p. 29). An $O(N^2)$ algorithm would then use $2.5 \times 10^{12}$ seconds of computation time for our $5 \times 10^8$-particle system above, or over 79,000 *years* of dedicated computer time. O'Connor's time measure of a unit operation is too conservative a measure for the current breed of fast processors, but it does give an idea of the magnitude of the computations required for discrete-element analysis. This quick "back-of-the-envelope" calculation expounds two salient points:

1. The complete natural system cannot currently be replicated in a discrete-element analysis. Instead, a representative problem or scale model needs to be selected for analysis. A statistically-sufficient number of particles can be used to study the particulate-level mechanisms; an examination of these mechanisms can be

used to gain insight into the macroscopic behavior of the system.

2. The algorithm by which discrete-element analysis proceeds needs to be improved from $O(N^2)$ to make computation feasible given a finite amount of computing time. Through use of an efficient algorithm or some system-wide assumptions this can be accomplished.

The idealization of a physical system must be taclked on a case-by-case basis, using engineering judgement. The second point relies mainly on the development of an excellent contact-detection model, which is described below.

### 2.1.2 Contact Detection Module

Contact detection, or the process of identifying surface-to-surface contact, is the heart of the discrete-element method. One naive method for contact detection is to simply check each of the $N$ particles in the system for intersection with every other particle in the system. This gives an $O(N^2)$ algorithm, which was shown above to give computation times that are inpracticable. The algorithm for contact detection needs to be improved.

One such improvement considers the process to be composed of two distinct phases. The first phase is the sorting of the elements ("spatial sorting"); the second phase is the determination of which particles are in contact with the others ("contact resolution"). O'Connor describes spatial sorting:

> Spatial sorting addresses the problem of deciding which pairs of objects should be considered for detailed contact resolution. It seeks to avoid conducting an exhaustive set of checks by spatially ordering the relative

> positions of the objects. The ordered set of objects is then searched for candidate pairs of objects on which to perform the second phase, contact resolution, a detailed check for contact. (O'Connor, 1996, p. 47)

O'Connor (1996) examines several different sorting methods as candidates for a discrete-element system and selects a spatial heapsort, which sorts an unordered list of $N$ objects in $O(N \log N)$ time. The candidate pairs for a detailed contact check are then selected from a neighbor list, which is generated (using the sorted list of objects) by determining which of these objects are within a certain distance of the object's centroid.

The mathematics of a contact check or *contact resolution* depend on the geometric representation used for the particles. For a functional representation such as that in (2.1), it is necessary to determine if two functions intersect. This is not particularly difficult to compute if all of the objects are discs of varying radii; however, determining the intersection of general curves becomes more computationally costly.

With a boundary representation of objects, it is simple to determine if two line segments intersection, and, if they do, what the point of intersection is. Unfortunately, polygon-intersection algorithms are typically $O(N^2)$ computations, which can prove costly for many-sided objects. However, it is possible to drastically reduce the number of computations by employing smart boundary-sampling schemes and an efficient scheme to determine the overlap region between between objects.

One method which efficiently handles boundary-sampling and overlap-region determination is presented by O'Connor (1996) as the Discrete Function Representation (DFR). "The scheme is both general, in its ability to handle a wide variety of shapes,

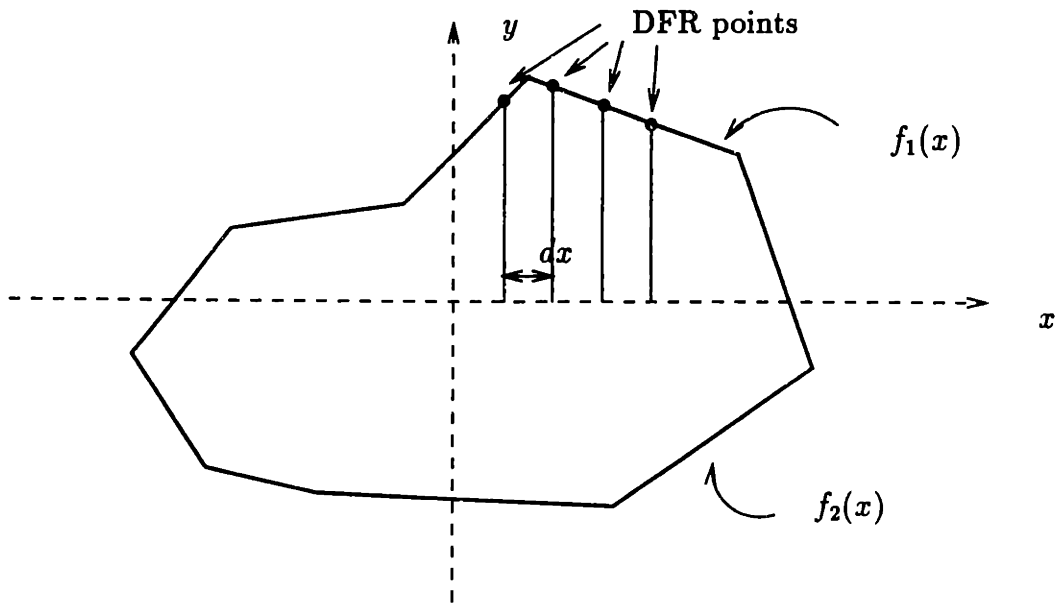



Figure 2-2: The Discrete Function Representation

and provides a structure with which contact resolution can be performed demonstrably faster than [many other techniques]." (O'Connor, 1996, p. 64)

O'Connor (1996) presents both two- and three-dimensional variations of the DFR scheme. Here, only the two-dimensional version (2D-DFR) will be considered. In the 2D-DFR representation, an object's boundary is considered to be represented by two single-valued functions $y = f_1(x)$ and $y = f_2(x)$, where the domains of the functions are separated by some dividing axis. The boundary of the polygon is then sampled at a fixed interval $dx$; the ordered set of points which is developed from sampled can be searched very quickly. A linear interpolation between boundary points is assumed[3].

This is easily visualized in Figure 2-2. The function $f_1(x)$ describes the points above the $x$ axis; the function $f_2(x)$ describes the points below the $x$ axis. Points

[3]The DFR scheme does not require linear interpolation between points. O'Connor (1996) suggests that a spline could also be used if necessary. However, for the applications considered here, linear interpolation is used and is generally sufficient.

for $f_1(x)$ are sampled at a fixed interval $dx$ and the values of $f_1$ at those sampling locations are indicated. Figure 2-2 does indicate a downfall of the DFR. Because a linear interpolation between points is assumed, it is possible that the DFR scheme will not account for a sharp corner in a polygon. This is allieviated by making an intelligent selection of polygon shape or by decreasing $dx$, thereby increasing the number of sample points.

O'Connor (1996) details the contact algorithm used for the DFR. The details of the contact algorithm are somewhat beyond the scope of this work, but the algorithm has been shown to work in $O(N)$ time. The speed-up generated by the DFR scheme allows us to consider more computationally complex problems or discrete-element simulations with many more particles.

### 2.1.3 Dynamics Equation Module

Rigid-body motion for a discrete element can be described using Newton's Second Law:

$$m\ddot{u} + c\dot{u} + ku = f, \tag{2.3}$$

where $m$ is the mass of the object, $c$ the damping, $k$ the stiffness, $f$ the applied force, $u$ the displacement. Superscript dots over $u$ refer to differentiation with respect to time. (2.3) can be written in a more general matrix form to handle more general two-dimensional and three-dimensional motion:

$$[M]\{\ddot{u}\} + [C]\{\dot{u}\} + [K]\{u\} = \{f\}. \tag{2.4}$$

Several integration schemes to evaluate (2.4) exist. Bathe (1996) covers several integration schemes, but for discrete-element analysis it is simplest to use an explicit integration scheme. Using an explicit scheme requires that the timestep $\Delta t$ selected for the simulation is less than the critical timestep $\Delta t_{cr}$ for the objects in the simulation; however, the explicit scheme requires no matrix inversion, which is computationally expensive but required for other, implicit, schemes.

Rege (1996) presents one explicit time-integration scheme which is suitable for discrete-element analysis. If the total force on a body of mass $m$ and moment-of-inertia $I$ is $F$ (a sum of the body forces, constraint forces, contact forces, fluid forces, etc.) and the total moment experienced by the body is $M$ (with contributions from the same sources), it is possible to determine the new displacement $u$ and rotation $\theta$ of the body. The scheme, step-by-step, is

$$\begin{aligned}
\ddot{u} &= \frac{F}{m}; \\
\ddot{\theta} &= \frac{M}{I}; \\
a &= 1 - \alpha\frac{dt}{2}; \\
b &= (1 + \alpha\frac{dt}{2})^{-1}; \\
\dot{u}_{t+1} &= (a\dot{u}_t + \ddot{u}\ dt)b; \\
\dot{\theta}_{t+1} &= (a\dot{\theta}_t + \ddot{\theta})b; \\
u_{t+1} &= u_t + \dot{u}_{t+1}dt; \\
\theta_{t+1} &= \theta_t + \dot{\theta}_{t+1}dt
\end{aligned} \tag{2.5}$$

(Rege, 1996, p. 30). In (2.5), $\alpha$ is an integration parameter and the superscript dots represent derivatives in time.

The DFR scheme described in the previous section allows for a straightforward calculation of the contact forces and moments. Several contact models exist; one such model is a "soft contact" model which allows deformation in the vicinity of the contact. Objects are given both normal and shear stiffnesses, and "the deformation is modeled as the compression of the spring between objects in contact." (Rege, 1996, p. 25)

### 2.1.4 Visualization Module

It is of course necesssary to view the results of the discrete-element simulation, preferably in a manner that permits quick-glance understanding and simulation-error detection. Many systems exist to display graphics on standard workstations; one powerful system is the *OpenGL* set of routines available on Silicon Graphics workstations. Originally based on the gl graphics library which was specific to Silicon Graphics' workstations, *OpenGL* is "open" and available across many platforms. The library is optimized to run on workstations with special graphics boards (such as those which made Silicon Graphics famous), but the graphics calculations can be performed in software for those workstations not so equipped.

The use of the *OpenGL* library allows the simulation to graph its progress as the calculations are made; animations of the particles can be captured for observance of the motion of the particles and any geometric formations which are created during the simulation.

The data collected from the discrete-element simulation, i.e. the contact forces, contact normals, particle velocities, etc., can be saved in a database through the simulation and processed later. Many database "post-processors" exist; some of the more powerful and useful packages are *MATLAB* and *AVSexpress*. Using these tools, data can be collated, manipulated, and interpreted to form a better understanding of the system under consideration.

## 2.2 State-of-the-Art in Discrete-Element Codes

Many different discrete-element codes exist, and they can be broken down into categories by the type of elements available for use in the simulation. Figure 2-3[4] depicts several discrete-element codes and classifies them as incorporating deformable vs. rigid particles and simple shapes vs. complex shapes.

From the figure, it is clear that Rege (1996) has developed the only system which permits deformable, arbitrarily-shaped particles. This work is the MIMES system, which is described below.

## 2.3 The MIMES Discrete-Element System

The MIMES Discrete-Element System is a computational materials laboratory based on the discrete-element method. It was developed by Rege (1996) with significant con-

[4]This figure, and the discussion developed from it, is taken from Rege, Nabha, "Computational Modeling of Granular Materials," Thesis defense, Department of Civil Engineering, Masschusetts Institute of Technology, Cambridge, MA, 15 February 1996.

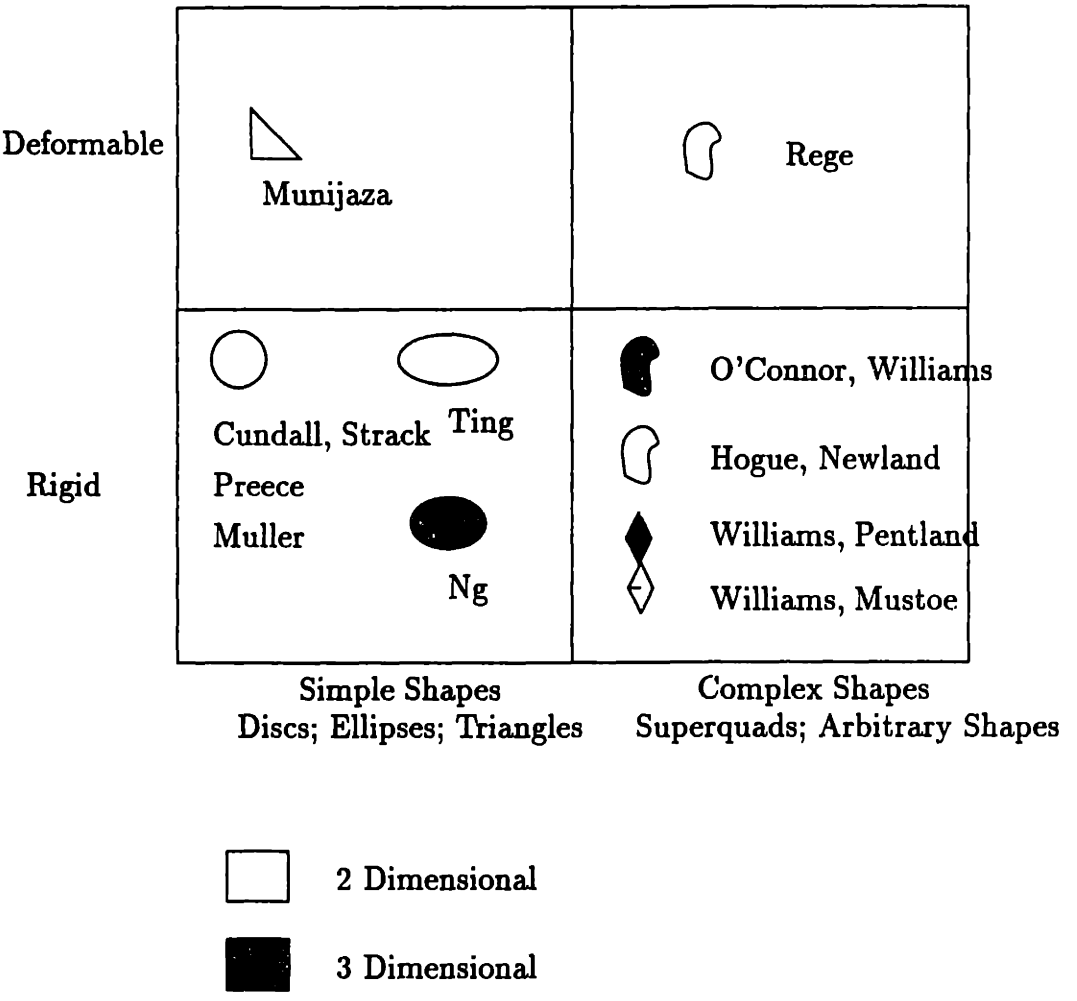



Figure 2-3: DEM State-of-the-Art

tributions from O'Connor (1996) and Williams[5]. The package, which is an acronym for Modeling Interacting Multibody Engineering Systems, is written in an object-oriented manner using C++. A *Tcl/Tk* wrapper is used to simplify interaction among the user, the discrete-element engine, and the window manager; the *Tcl/Tk* wrapper also provides a shell scripting language for the creation of complex systems.

At this writing, MIMES is implemented to solve only two-dimensional problems. However, its capabilities are robust. The system supports disc, line, quadrilateral, and superquadric shapes for basic particle geometries. Although these are the basic shapes, the quadrilaterals and superquadric shapes are represented internally as $n$-sided polygons so the system has the capability to handle any arbitrarily-shaped particle. Futhermore, MIMES allows for rigid-body motion *or* deformable particles. Particle deformations can be determined using either finite-element methods or a boundary-element method. The system has been compiled for parallel systems as well, allowing for marked increases in computational speed on parallel systems. From Figure (2-3) it is clear that MIMES is one of the most robust 2-dimensional discrete-element systems available.

A complete discussion of the C++ `class` structures used in MIMES is given in Rege (1996). That work also discusses the MIMES shell language, which is simply the *Tcl/Tk* scripting language augmented with some additional commands for discrete-element simulations.

MIMES has been proven to give some good results as a computational mechanics

[5]Associate Professor, Department of Civil and Environmental Engineering, Massachusetts Institute of Technology, 77 Massachusetts Avenue, rm 1-250, Cambridge, Massachusetts, 02139.

laboratory (e.g. Williams and Rege, 1996). Its object-oriented design and *Tcl/Tk* wrapper allow robust modifications to be made with relative ease. Furthermore, the complete source code for the system was readily available. For these reasons, the MIMES system was selected as a base discrete-element modeling system to extend with fluid-dynamics capabilities.

The following chapter explains the methods selected for fluid-dynamics computation and its implementation in MIMES.

# Chapter 3

# Fluid Forces in the MIMES Systems

This chapter discusses the addition of fluid loading into the MIMES system. Methods of fluid-flow computation are discussed and the "information pathway" between the discrete-element and finite-element systems is detailed. The sections of this chapter follow the flow of information between the systems to describe the implementation of the system and the choices and assumptions made for the coding.

## 3.1 Methods of Fluid-Flow Computation

There are several methods which can be used to numerically compute fluid flow in a domain. The most comprehensive theoretical model for fluid flow are the four Navier-Stokes equations which govern the motion of a viscous, incompressible fluid in laminar flow with heat transfer. The governing equations can be given as a momentum equation

$$\rho(\frac{\partial v_i}{\partial t} + v_{i,j}v_j) = \tau_{ij,j} + f_i^B, \tag{3.1}$$

a constitutive equation

$$\rho\tau_{ij} = -p\delta_{ij} + 2\mu e_{ij}, \tag{3.2}$$

a continuity equation

$$v_{i,i} = 0, \tag{3.3}$$

and a heat transfer equation

$$\rho c_p[\frac{\partial\theta}{\partial t} + \theta_{,i}v_i] = (k\theta_{,i})_{,i} + q^B. \tag{3.4}$$

(Bathe, 1996, p. 675) A description of the notation and boundary conditions for a full Navier-Stokes model are beyond the scope of this discussion, but Bathe describes the computational difficulties inherent to a Navier-Stokes formulation:

> Inherent nonlinearities are due to the convective terms in [(3.1)] and [(3.4)] and the radiation boundary conditions ... Additional nonlinearities arise if the viscosity coefficient [$\mu$] depends on the temperature or on the velocity strain [$e_{ij}$], if the specific heat $c_p$, conductivity $k$, and convection and radiation coefficients depend on temperature, and of course if turbulence descriptions are included .... (p. 676)

(3.1) through (3.4) can be rewritten to include nondimensionalized variables. The equations then include the Reynolds number Re and the Peclet number Pe. As one of the numbers increases (they are inversely proportional), the equations become increasingly nonlinear and the general solution of the equations becomes more difficult.

There are many techniques available to solve the Navier-Stokes equations for fluid flow. Of course the equations could be solved analytically, using standard differential-equation techniques. However, the number of solutions which can be reasonably han-

dled with analytic solution is small–the geometries available for solution are generally very simple.

Finite-element methods can also be used to discretize and solve the differential equations for fluid flow. These methods are well accepted and have been proven to converge to anm exact solution, given an element mesh which has been refined enough. The solution method also allows for arbitrary geometries. However, the meshing involved for solution can be very time-consuming and the solution phase can be very slow, especially as nonlinear terms grow in the formulation. Overall, though, finite element methods are very flexible and relatively straightforward to implement, and this solution technique is selected here for the coupling of discrete-element analysis with fluid flow.

Fortunately, the Navier-Stokes equations can often be simplified by making some engineering assumptions about the problem. Often convective terms and dissipation terms can be ignored, allowing for simpler solutions. (Kardestuncer, 1987). Foster and Metaxas (1996) use a relaxation of the Navier-Stokes equations to model splashing, voriticity, waves, and buoyancy in fluid-structure systems. Their discretization is a "particle-in-cell" method (PIC), which uses a finite-difference scheme is used for solution. Solutions using this method appear to be reasonable, at least on a macroscopic level; however, the orientation of their research is more for computer animation of fluid flows (especially splashing). Their work is not geared toward computational mechanics but rather the use of computers and computer graphics to simulate physical phenomena. In such a work, relaxation of some of the governing equations is not a major issue!

A further relaxation of the Navier-Stokes laws leads to a Darcy-flow or "seepage" approach. In the Darcy-flow approach, an approach commonly used for porous media, the governing law is Darcy's law, which gives the flow $q$ through the medium in terms of the gradient of the potential $\phi$:

$$\begin{aligned} q_x &= -k_x \frac{\partial \phi}{\partial x}; \\ q_y &= -k_y \frac{\partial \phi}{\partial y}; \\ q_z &= -k_z \frac{\partial \phi}{\partial z}, \end{aligned}$$

where $k_x$, $k_y$, and $k_z$ are the medium's permeability in the $x$-, $y$-, and $z$-directions, respectively. Continuity of flow requires that

$$\frac{\partial}{\partial x}(k_x \frac{\partial \phi}{\partial x}) + \frac{\partial}{\partial y}(k_y \frac{\partial \phi}{\partial y}) + \frac{\partial}{\partial z}(k_z \frac{\partial \phi}{\partial z}) = -q^B, \tag{3.5}$$

where $q^B$ is the flow per unit volume. (3.5) can be seen as one version of (3.3), so the Darcy-flow model is simply a Navier-Stokes formulation with many of the conditions relaxed. Typically Darcy flow is valid for laminar flow of water through void space in soils, and this formulation is used in many civil engineering applications.

## 3.2 Selection of the Finite Elements

When flow through porous media is discretized as a "seepage" or Darcy-flow problem, a finite area is assigned a given permeability $k$ and the flow is calculated using (3.5)

above. This approach results in a "smeared", homogeneous (per element) formulation. Torczyinski (1996) uses Darcy-flow elements for use in some porous-media problems and reports that the Darcy-flow formulation gives results in good agreement with an analytical solution for some test problems.

Given a Darcy-flow solution method, the next step is the selection of the shape of the finite elements used to discretize the problem. Triangular and square elements are popular; the square elements can be generalized to handle an arbitrary quadrilateral boundary through the use of an isoparametric formulation. Here we select a triangular element as it is possible to triangulate any given polygon shape in the plane. Also, triangular elements simplify adaptive-meshing algorithms (see Section 3.5.2 below) which would be further complicated by the use of square or isoparametric quadrilateral elements.

For a triangular element geometry, a finite element can easily be formulated to handle linear, quadratic, or even cubic interpolation between the nodes of the system. Here we select linear finite-element interpolation. Linear finite elements allow for fast computation (no explicit integration over the element is necessary) and provide answers sufficient for the problems at hand. It is a simple matter to change a bilinear triangular element into a quadratic element if the bilinear elements are not able to capture some effects of the system under consideration.

Bilinear, triangular, Darcy-flow elements are selected for the fluid solution in the MIMES system. The sections that follow discuss the incorporation of fluids into MIMES and the mathematics behind the Darcy-flow finite-element discretization for the fluid-flow problem.

## 3.3 DEM/FEM Coupling Information Flow Path

There are several steps which are required to couple the fluid finite-element analysis with the existing discrete-element system. These are depicted graphically in Figure 3-1. The discrete-element world has a fluid domain defined within its simulation plane, and this fluid domain must be meshed into a set of triangles.

The discrete-element particle configuration is then used to determine the the equivalent permeability of a triangular finite element and then an element stiffness matrix is computed. All of the element stiffness matrices are assembled into a global stiffness matrix for the entire fluid domain, boundary conditions are applied, and the finite-element system is solved.

The finite-element solution is then used to derive information about fluid pressures and velocities within each finite element. This information is used to determine the fluid force acting on each discrete element in the system. Next, the cumulative external forces (fluid force, contact force, etc.) on each discrete element are used to update their positions and orientations in the system, yielding a new configuration for the particle assemblage. The particles have moved; therefore the effective porosity terms for each fluid flow element have also changed. For the next timestep, the calculation repeats by reevaluating the flow field, with element porosities as given by the new configuration of the particles, closing the loop on the fluid-flow/particle-motion coupling.

The following sections elaborate on the "stations" of the information flow path in the algorithm outlined here.

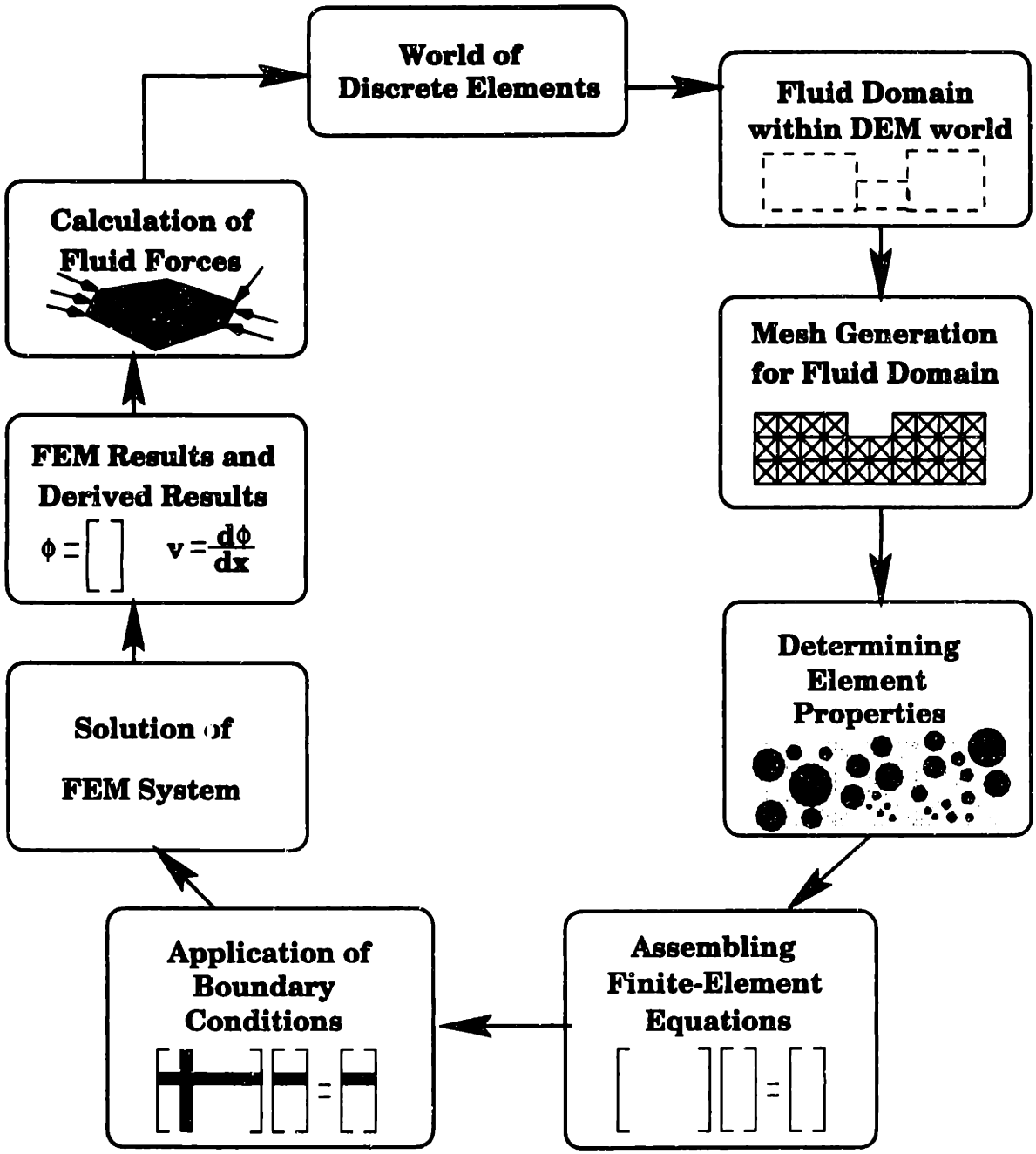



Figure 3-1: DEM/FEM Coupling Information Flow Path

## 3.4 The Fluid Domain

The fluid domain within the discrete-element world is defined by a set of *quadrilaterals* and *lines*. Quadrilaterals define the area(s) over which fluid computations should be made. The quadrilaterals can be of any size and orientation; we use a convention that their points are defined in a counterclockwise order. Several quadrilaterals can be defined to mesh together, allowing for complex areas to be described.

*Lines* define the boundary conditions for the domain bounded by the quadrilaterals. Given a line and prescribed pressure (or flow conditions), any finite-element nodes along that line are given those boundary conditions.

The use of arbitrary quadrilaterals and lines to define the fluid domain allows for much flexibility in the type of problem which can be considered.

## 3.5 Meshing the Fluid Domain into Finite Elements

The fluid domain is defined by the user in terms of a set of convex quadrilaterals. Breaking this overall domain down into finite elements requires two main steps:

1. mesh the quadrilateral region with triangles;

2. ensure nodal compatibility among the elements.

The following sections describe one method of handling these two steps.

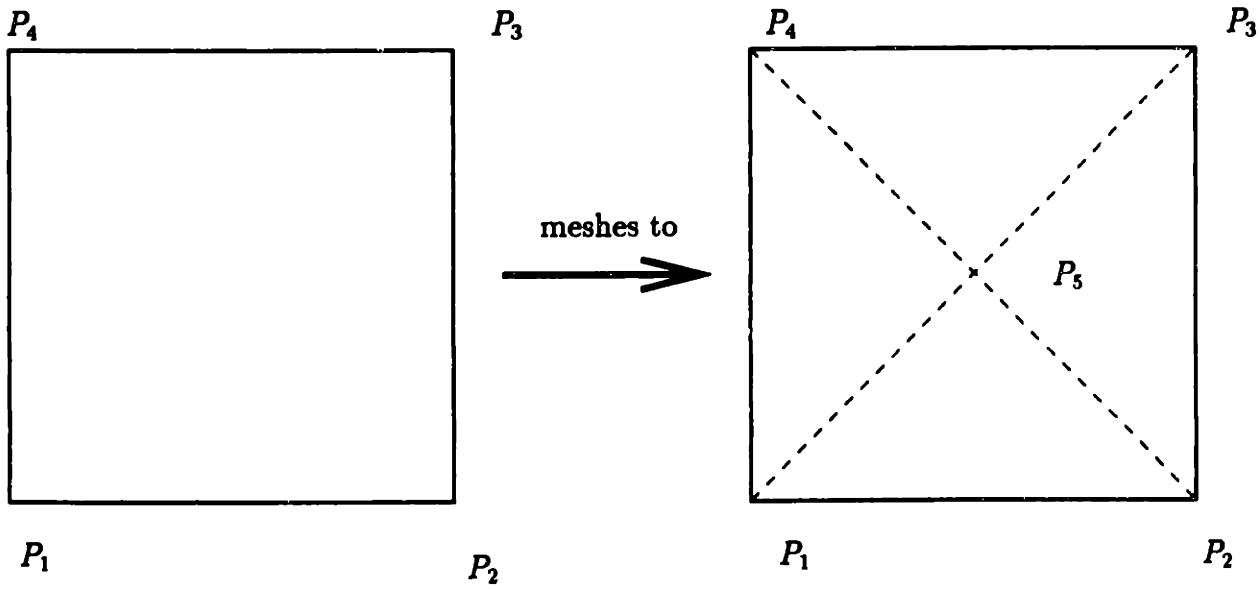



Figure 3-2: Meshing a Rectangle into Four Triangles

### 3.5.1 Meshing a Quadrilateral Region with Triangles

While it is a trivial problem to mesh a square into a given number of triangles, it is much more difficult to solve the general problems of meshing an arbitrary quadrilateral. One such solution is to mesh a square in local $(s,t)$ coordinates and then transform that solution into the global $(x,y)$ coordinates of the system such that it maps to the quadrilateral's geometry. This is the solution presented below.

#### Meshing a Square

Consider a square in $(s,t)$ space, bounded by the lines $s = \pm 1$ and $t = \pm 1$. To break each side parallel to the $s$ axis into $n_s$ sections and each side parallel to the $t$ axis into $n_t$ sections, it is simple to determine the cornerpoint coordinates $P_1(s_1,t_1)$ $\ldots P_4(s_4,t_4)$ of the rectangles $P_1P_2P_3P_4$ generated by this partitioning. If $n_s = n_t$, this procedure will create squares instead of rectangles.

Each rectangle can then be broken into triangles as desired. It is convenient to mesh the rectangle into four triangles as in Figure 3-2. The coordinates $P_5(s_5, t_5)$of the centerpoint node are simply the average of the $(s, t)$ coordinates for each cornerpoint of the rectangle:

$$P_5(s_5, t_5) = (\frac{1}{4}\sum_{i=1}^{4} s_i, \frac{1}{4}\sum_{i=1}^{4} t_i)$$

The rectangle can then be meshed into the four triangles $P_1P_2P_5$, $P_2P_3P_5$, $P_3P_4P_5$, and $P_4P_1P_5$, again as depicted in Figure 3-2. Note that it is important to keep the *counterclockwise* order of the points for the triangles, as the construction of the element stiffness matrix (discussed below) assumes that the nodal points are in counterclockwise order.

### Transforming from a Square to a Quadrilateral

Since our square is bounded by $s = \pm 1$ and $t = \pm 1$, we can think of our square in terms of an isoparametric finite element and use finite-element shape functions to transform from the local $(s, t)$ coordinates to the $(x, y)$ coordinates of the problem domain.

Since we will be using bilinear triangular finite elements, it follows that the shape functions used to transform our square into a quadrilateral should also be bilinear. Bathe (1996) gives the transformation as

$$\begin{aligned} x &= \sum_{i=1}^{4} h_i(s, t)x_i; \\ y &= \sum_{i=1}^{4} h_i(s, t)y_i, \end{aligned} \tag{3.6}$$

where $x_i$ and $y_i$ are the global coordinates for the cornerpoints of the fluid quadrilateral and $h_i$ are shape functions. Continuing the node-numbering system from Figure 3-2, the $h$ functions are:

$$\begin{aligned} h_1 &= \frac{1}{4}(1-s)(1-t); \\ h_2 &= \frac{1}{4}(1+s)(1-t); \\ h_3 &= \frac{1}{4}(1+s)(1+t); \\ h_4 &= \frac{1}{4}(1-s)(1+t), \end{aligned} \tag{3.7}$$

where the inputs $s$ and $t$ to the $h$ functions are the local coordinates of the point which is being transformed (Bathe, 1996).

It is simple, then, to map the triangulated square to our arbitrary quadrilateral: each node in local $(s,t)$ coordinates is passed through the transformations of (3.7) and then (3.6) and the output is the global coordinate $(x,y)$ for the node in our original quadrilateral.

### 3.5.2 Ensuring Nodal Compatibility Among the Elements

Finite-element analysis requires compatibility among the elements; that is,

> ...it is necessary that the coordinates and the displacements of the elements at the common face be the same. We note that for the elements considered here, the coordinates and displacements on an element face are determined only by nodes and nodal degrees of freedom on that face. Therefore, compatibility is satisfied if the elements have the same nodes on the common face and the coordinates and displacements on the common face are in each element defined by the same interpolation functions. (Bathe, 1996, p. 377)

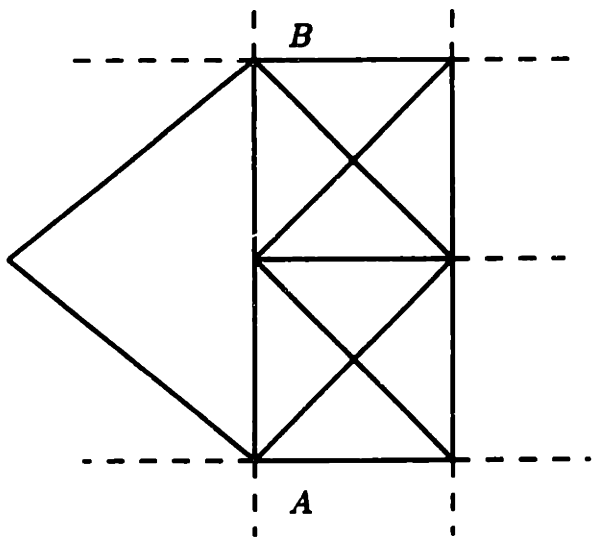



Figure 3-3: Incompatible Mesh Between Two Elements

This is one of the criteria for monotonic convergence of the finite-element method (Bathe, 1996).

Since our fluid-force formulation uses only one type of element, Bathe's condition that the displacements on the common face are defined by the same interpolation function is satisfied. However, it is possible that two adjacent elements do not have the same number of nodes on their common face, as depicted in Figure 3-3. Therefore, it is necessary to check the finite-element meshes, once generated, to determine if there is nodal incompatibility across the face of adjoining elements.

The test for incompatibility across adjoining elements can be broken down into two main parts:

1. searching the set of nodes for points along the face of the element;

2. remeshing any element which is incompatibile with other elements in the set.

The following sections describe a solution to this problem.

## Searching for Points Along a Line

The problem "given a set of points, determine which of these points are along a given line" is a special case of a range-search problem[1]. A naive algorithm would simply check each node in the world to see if fell on the face in question. With $N$ nodes in the world and at least $N$ faces to check, this first-cut algorithm would have running time $O(N^2)$ in the *best* case. Real problems will require systems with $N$ on the order of $10^3$ or $10^4$ nodes and the performance of this algorithm will become unacceptable. Fortunately, there are other ways to dramatically reduce search time.

The algorithm selected for use here is a two-dimensional (2D) tree based on that presented by Sedgewick (1992). Other methods for searching exist (and indeed a "grid based" search method is used later to determine the forces on a discrete element particle) but the 2D-tree method allows for points anywhere in the plane—here the algorithm is not limited to the regularity enforced by a grid-based search method.

The 2D-tree is similar to a binary tree in structure; however, the key for each node of the tree alternates between the $y$- and $x$-coordinates of the points in the tree. That is, the key for the root node is its $y$-coordinate; the key at the next level is the point's $x$-coordinate; then $y$, $x$, etc., until an external node is encountered. Traversing the tree is done just as with a binary tree (if the test key is lower than the node's key, go left; otherwise go right) and insertion is also performed in the same manner. Geometrically, this corresponds to cutting the plane of points in half; the next level of the tree cuts that subplane in half again, and so on. Sedgewick (1992) has some

[1]Range-searching, which has applications far beyond those of computational geometry or computational mechanics, is discussed in many sources, such as Laszlo (1996), Cormen et. al. (1990), Sedgewick (1992), and Preparata (1985).

excellent graphics which show how a 2D-tree breaks up the plane.

The implementation in the MIMES system is taken from the code in Sedgewick (1992), although it is modified somewhat to return a list of points that fall within the search rectangle and to correct some bugs. While this search method has yet to be completely analyzed, the construction of the tree from $N$ random points requires $2N \ln N$ comparisons (on average), and a search takes about "$R + \log N$ steps to find $R$ points in reasonable ranges in a region containing $N$ points" (Sedgewick, 1992, p. 384). This is of course an empirical result, but practical use of the algorithm shows improvement over an $N^2$ algorithm.

The 2D-tree is employed to search the set of nodes, using the endpoints of a test line as the cornerpoints for a search rectangle. Then it is a simple matter to determine if a returned point falls on the test line: Sedgewick (1992) gives such an algorithm. If points other than the endpoints on our test line are returned by the range-search, then the face of the element is incompatible with some other element and the triangle must be remeshed.

### Remeshing an Incompatible Element

Given a rectangular area meshed with triangular finite elements as in Figure 3-2, it is a relatively easy task to remesh the elements to ensure compatibility across the faces of the elements. In Figure 3-4, the triangular element is defined by the nodes $P_1$, $P_2$ and $P_3$. On the face $P_1P_2$, there are nodes $P_4$ and $P_5$ which were generated from another element. Element $P_1P_2P_3$ is remeshed into three new elements: $P_1P_4P_3$, $P_4P_5P_3$, and $P_5P_2P_3$. Note that the nodes of the triangular elements retain their *counterclockwise*

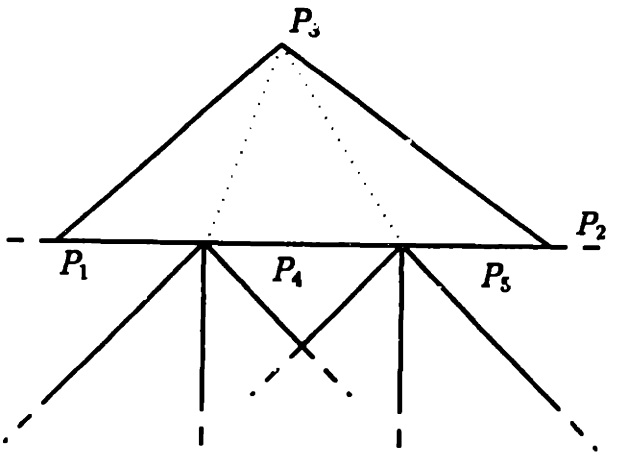



Figure 3-4: Remeshing a Triangular Element to Alleviate Incompability

order.

This algorithm is straightforward to implement, provided that the new nodes $P_4$ and $P_5$ are kept in order between $P_1$ and $P_2$. The range-searching algorithm returns points in no particular order as relates to this part of the problem, so care must be taken to ensure that the path $P_1P_4P_5P_2$ is uni-directional[2]. Once an incompatible element has been remeshed, all the compatibility requirements detailed above are satisfied.

Note carefully that this remeshing does not check for the stability of the triangular elements it creates. Typically, finite elements need to be "well-behaved" in the sense that the elements should not be too elongated or distorted. Some distortion is acceptable, but computational errors can occur when performing floating-point computations with oddly-shaped elements.

[2]There are several ways to accomplish this, such as sorting the new nodes by increasing distance from one endpoint of the test line.

## 3.6 Determining Effective Element Permeability

Darcy's Law states $v = ki$, where $v$ is the fluid velocity, $i$ is the fluid pressure head, and $k$ is the permeability of the media through which the fluid is flowing. But how exactly can $k$ be determined?

Unfortunately, $k$ is a "magic number" of sorts in soil mechanics. It can be determined in the laboratory by several different means, including a constant-head test and a falling-head test. (Das, 1990) However, an exact relationship between some physical properties of the porous medium and its permeability $k$ has not been given. Many empirical relationships exist, based on several experiments. For example, an

> ...equation [which] gives fairly good results in the estimation of the coefficient of permeability of sandy soils is based on the Kozeny-Carman equation. ...An application of the Kozeny-Carman equation ...can be written as

$$k = C_1 \frac{e^3}{1+e}, \tag{3.8}$$

(Das, 1990, p. 103), where $k$ is the coefficient of permeability of the medium, $C_1$ is a constant, and $e$ is the void ratio of the medium. Das (1990) gives several empirically-determined relationships for $C_1$ for various types of soils.

In the MIMES system it is straightforward to determine the area of a finite-element which is covered by discrete elements; in other words, a void ratio $e$ can be computed for a finite-element in our analysis. While it would be possible to determine the area of a finite element which is covered by discrete elements by determining intersection points and performing area subtractions, this algorithm is too exact and too involved to determine the void ratio $e$ which is used in an empirical formula anyway. Instead,

a *pixellation* technique is used where a finite element is covered by an imaginary grid. If the center of a grid square (or pixel) is inside the finite element, the area of the grid square is considered to be a component of the area of the finite element. Similarly, if the center of the pixel is inside a discrete element, the area of the pixel is considered to be a component of the solid area of the medium. The void ratio $e$ is then computed as

$$e = \frac{A_{triangle} - A_{solid}}{A_{solid}}. \tag{3.9}$$

As the size of a pixel diminishes, the pixellation method approaches an exact answer. This method, then, can be adjusted to provide a better answer for a better-defined problem; conversely, the resolution of the answer can be reduced to save computational time for a less-well-defined problem.

The void ratio $e$ is truly a three dimensional quantity:

$$e = \frac{V_{voids}}{V_{solids}},$$

but for a two-dimensional discretization it is unclear how to extrapolate the area of an arbitrarly 2D polygon to a volumetric quantity. For spherical particles it is possible to assume that all particles are held between two planes separated by a thickness $t$—an "antfarm" model—but this assumption is nonsensical for polygonal shapes.

It is necessary, then, to clamp the values calculated for $e$ from (3.9). It is possible to determine an $e$ of zero, if the finite-element is completely covered by discrete element, which is impossible for a porous medium (if $e = 0$, the medium is no longer porous

and that is a violation of an assumption for the problem). Similarly, it is possible to compute an undefined $e$ (where $A_{solid} = 0$), which would occur for problems where fluid stood alone, outside a porous medium. Again, this violates the assumption that the flow is through a porous medium, and so the values of $A_{solid}$ are clamped such that $0.01A_{triangle} \leq A_{solid} \leq 0.99A_{triangle}$ or $0.01 \leq e \leq 99$. Therefore a finite-element which has no discrete bodies inside it would have a void ratio of 99, which is about two orders of magnitude higher than that for any porous medium. This clamping, then, allows us to handle particle blockage ($e = 0.01$) and fluid pools ($e = 99$).

The $k$ for the element can then be determined using (3.8) or some other empirical relationship. However, permeabilities which are determined from these empirical relations are typical orders of magnitude too high when compared with the actual, experimentally-determined values for the soil[3]. Therefore, fluid-flow results dervied from (3.8) above may be very inaccurate.

Another method for selecting the finite-element $k$ is though the input of experimentally-determined permeabilities for the soil. Such data is widely available (see, for instance, Friedrich et. al., 1993), and values for two representative types of soils can be used to clamp the values of $k$. For instance, sandstone which is consolidated well within a formation may be known to have a permeability of $k_1$, while a sandstone which is more loosely consolidated may have a permeability of $k_2$. Therefore a problem which models fluid flow through a region of sandstone may be constrained such that

$$k_1 \leq k \leq k_2,$$

[3]Dale S. Preece, Sandia National Laboratories, private communication, September 1996.

where $k$ is interpolated between $k_1$ and $k_2$ using the relationship in (3.8), based on the calculated value of $e$.

This interpolation method is selected for the coupled fluid-flow/discrete-element system. For many problems considered here, there is adequate experimental data to select reasonable values of $k_1$ and $k_2$. If, at some later date, other empirical relations or interpolations are necessary for the simulation, it is a simple matter to update the code to reflect this new method.

## 3.7 Element Matricies for Darcy-Flow Finite Elements

The Darcy-flow finite element is used to solve the equation

$$D_x \frac{\partial^2 \phi}{\partial x^2} + D_y \frac{\partial^2 \phi}{\partial y^2} = 0, \tag{3.10}$$

where $D_x$ and $D_y$ are the permeabilities of the porous media in the $x$- and $y$-directions respectively (Segerlind, 1986). Here, the two-dimensional flow of the ideal fluid is formulated in terms of a potential function $\phi$, which is the only variable (or degree of freedom) in the problem.

Having selected a bilinear triangular Darcy-flow element to solve our fluid-flow problem, we now need to examine the admittance[4] matrix for such an element. Our inputs for the element are the $x$- and $y$-direction permeabilities $D_x$ and $D_y$ and the

[4]For a fluid-flow problem, the term "stiffness matrix" is probably inappropriate. Here we will use the term "admittance matrix" where "stiffness matrix" would be used for an analogous structural-mechanics problem.

coordinates of its three nodes $(X_i, Y_i)$, $(X_j, Y_j)$, and $(X_k, Y_k)$.

For our element, Segerlind (1984) gives the element admittance matrix $\mathbf{A}^{(e)}$ as

$$\mathbf{A}^{(e)} = \frac{D_x}{4A}\begin{bmatrix} b_i^2 & b_i b_j & b_i b_k \\ b_i b_j & b_j^2 & b_j b_k \\ b_i b_k & b_j b_k & b_k^2 \end{bmatrix} + \frac{D_y}{4A}\begin{bmatrix} c_i^2 & c_i c_j & c_i c_k \\ c_i c_j & c_j^2 & c_j c_k \\ c_i c_k & c_j c_k & c_k^2 \end{bmatrix}, \tag{3.11}$$

where $A$ is the area of the element and the constants $b_i$, $b_j$, $b_k$, $c_i$, $c_j$, and $c_k$ are given by

$$\begin{aligned} b_i &= Y_j - Y_k; \\ b_j &= Y_k - Y_i; \\ b_k &= Y_i - Y_j; \\ c_i &= X_k - X_j; \\ c_j &= X_i - X_k; \\ c_k &= X_j - X_i, \end{aligned} \tag{3.12}$$

This formulation requires that the nodes $i$, $j$, and $k$ are in counterclockwise order, which we ensured when generating the triangular mesh above.

Note that the only degree of freedom for this element is the pressure at the nodes; the units of the pressure are consistent with those selected for the permeability coefficients $D_x$ and $D_y$.

Since this element is triangular bilinear, no integration is required to determine

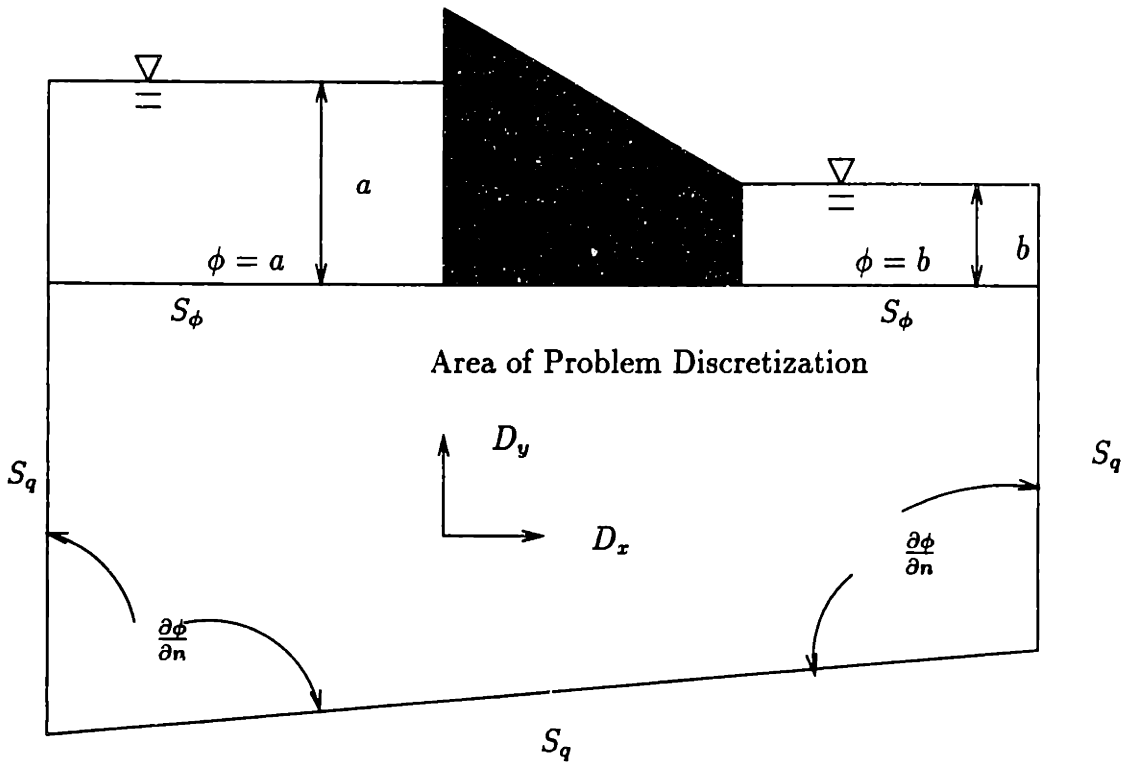



Figure 3-5: Boundary Conditions for Darcy-Flow Problem

$\mathbf{A}^{(e)}$ for the element; it can be determined directly from the geometry of the element and is given by (3.11) above. The elements can be assembled into a global admittance matrix $\mathbf{A}$ by standard means (Segerlind, 1986; Bathe, 1996).

## 3.8 Application of Boundary Conditions

Once all of the element matricies $\mathbf{A}^{(e)}$ have been assembled into a global admittance matrix $\mathbf{A}$, $\mathbf{A}$ needs to be modified to incorporate the boundary conditions of the system.

There are two types of boundary conditions for a Darcy-flow problem. The first

is a prescribed pressure $\phi$ at each node on the free surface $S_\phi$ of the problem domain,

$$\phi|_{S_\phi} = \phi^S, \tag{3.13}$$

and the second is a prescribed flow across a boundary $S_q$,

$$\left.\frac{\partial \phi}{\partial n}\right|_{S_q} = q^S. \tag{3.14}$$

(Bathe, 1996). Note that the surface $S = S_\phi \cup S_q$. These boundary conditions are depicted graphically in Figure 3-5[5]. The finite-element formulation has the automatic boundary condition $\frac{\partial \phi}{\partial n} = 0$, so if no boundary conditions in the form of (3.14) are explicitly applied, the boundaries default to being considered impermeable.

Application of the boundary conditions is straightforward, although some care must be taken to alter the flow vector $\mathbf{q}$ to accomodate prescribed pressure conditions. Consider the equation

$$\mathbf{A}\phi = \mathbf{q}$$

rewritten in the form

$$\begin{bmatrix} \mathbf{A}_{ff} & \mathbf{A}_{fp} \\ \mathbf{A}_{pf} & \mathbf{A}_{pp} \end{bmatrix} \begin{pmatrix} \phi_f \\ \phi_p \end{pmatrix} = \begin{pmatrix} \mathbf{q}_f \\ \mathbf{q}_p \end{pmatrix}, \tag{3.15}$$

where the subscript $f$ denotes a free degree-of-freedom and the subscript $p$ denotes a prescribed degree-of-freedom. Since some pressures $\phi_p$ may be constrained to non-zero

[5]This figure is adapted from Sedgewick (1984), p. 129.

values, (3.15) is altered to

$$\begin{bmatrix} \mathbf{A}_{ff} & 0 \\ 0 & \mathbf{I} \end{bmatrix} \begin{pmatrix} \phi_f \\ \phi_p \end{pmatrix} = \begin{pmatrix} \mathbf{q}_f - \mathbf{A}_{fp}\phi_p \\ \phi_p \end{pmatrix} \tag{3.16}$$

(Crisfield, 1991, p. 33). In practice, the sub-matrix $\mathbf{A}_{ff}$ need not be separated from the rest of the entries in the matrix, but care must be taken while "bookkeeping" for the problem.

## 3.9 Solution of the Finite-Element System

### 3.9.1 Static Solution

Finite-element problems are straightforward to solve in the static case. Once boundary conditions have been applied to the matrix $\mathbf{A}$ and the vector $\mathbf{q}$ as described above, we simply need to evaluate

$$\phi = \mathbf{A}^{-1}\mathbf{q}$$

to determine the pressure potentials $\phi$ for the system.

The computation of $\mathbf{A}^{-1}$ is a costly process; using a full Gaussian elimination scheme the computation cost is $O(N^3)$, where $\mathbf{A}$ is an $N \times N$ matrix. Some improvement can be obtained by using a Gaussian elimination scheme with back-substitution. Other schemes, which do not require the direct computation of $\mathbf{A}^{-1}$ abound, including the conjugate-gradient and Gauss-Seidel methods. Bathe (1996) and Press (1995) cover the subject of matrix solution methods at length.

### 3.9.2 Transient Solution

For some problems, transient effects in the pressure field need to be captured. For instance, an applied pressure sink will take some time for the forces to come under equilibrium. Using a static solution as above, equilibrium is achieved instantly; however, everyone who was ever pulled the plug on their drain (which is effectively applying a pressure sink) knows that it takes some time for equilibrium to be achieved. These effects are captured by the equation

$$D_x \frac{\partial^2 \phi}{\partial x^2} + D_y \frac{\partial^2 \phi}{\partial y^2} = \rho c \frac{\partial \phi}{\partial t}, \tag{3.17}$$

where $\rho$ is the density of the fluid under flow and $c$ is a "storativity" constant analagous to the damping constant for structural-mechanics problems. (3.17) is the transient version of (3.10).

(3.17) can be solved with a simple *explicit* time-stepping routine. We already use an explicit time-stepping to handle the motion of the discrete bodies so an explicit scheme for the finite-element component of the system is a natural extension.

Several transient-solution techniques are available, and many of them are well-documented in Bathe (1996). The method selected here is a simplified central-difference scheme, in which the pressures $\phi^{t+\Delta t}$ at time $t + \Delta t$ are determined explicitly based on the pressures already determined at the previous timestep $\phi^t$ and a residual $\mathbf{R}$.

An exact solution to the problem $\mathbf{A}\phi = \mathbf{q}$ is $\phi_{exact} = \mathbf{A}^{-1}\mathbf{q}$, which is exactly the static answer determined above. Assume instead that there is a "guess" answer $\phi^t$,

which differs from the exact solution $\phi_{exact}$. The residual $\mathbf{R}$ as

$$\mathbf{R} = \mathbf{A}\phi^t - \mathbf{q};$$

an exact answer $\phi_{exact}$ would give $\mathbf{A}\phi_{exact} = \mathbf{q}$ and the residual $\mathbf{R}$ would be zero. Using an explicit integration scheme, the pressures $\phi^{t+\Delta t}$ can be calculated as

$$\phi^{t+\Delta t} = \phi^t - (\mathbf{R}\Delta t).$$

After this calculation the system boundary conditions need to be imposed on $\phi^{t+\Delta t}$ to ensure that the physical constraints of the problem are met.

If the first "guess" answer $\phi^{t=0}$ is selected to be $\phi_{bc}$, a vector with zeroes everywhere except where boundary-condition pressures have been prescribed, the integration through time proceeds as:

$$\begin{aligned}
\phi^{t=0} &= \phi_{bc}; \\
\phi^{t=\Delta t} &= \phi^{t=0} - (\mathbf{R}^{t=\Delta t}\Delta t); \\
\phi^{t=2\Delta t} &= \phi^{t=\Delta t} - (\mathbf{R}^{t=2\Delta t}\Delta t); \\
\phi^{t=3\Delta t} &= \ldots
\end{aligned} \tag{3.18}$$

This integration captures the transient effects of the pressure field.

### 3.9.3 Mixed Transient and Static Solution

One final method of solution is a mixed transient/static solution. For this method, $\phi = \mathbf{A}^{-1}\mathbf{q}$ is evaluated for the *first time step only*, and this computed $\phi$ is used as $\phi^{t=0}$ in (3.18) above. This allows for analysis of problems which start with a static pressure field but whose physical characteristics (i.e. element permeabilities) change over the course of analysis. The static pressure field is set up to begin the analysis but the time-stepping scheme will capture pressure changes over the fluid field.

## 3.10 Results from the Finite-Element Solution

The only degree-of-freedom for the finite-element system considered here is the pressure of the fluid; therefore the result $\phi$ from the solution schemes above is a vector of the pressures at the nodes of the elements. The units of the pressure are consistent with the units of the permeabilities $D_x$ and $D_y$; typically the units are head, or length of fluid.

The velocity of the fluid in the system can be computed directly from Darcy's law:

$$\begin{aligned} v_x &= -D_x \frac{\partial \phi}{\partial x}; \\ v_y &= -D_y \frac{\partial \phi}{\partial y}. \end{aligned} \tag{3.19}$$

(Segerlind, 1984, p. 130) For coding purposes, the partial derivatives and fluid veloc-

ities in (3.19) are computed as

$$\begin{aligned} v_x(x,y) &= -D_x \frac{\phi(x+\Delta x, y) - \phi(x,y)}{\Delta x}; \\ v_y(x,y) &= -D_y \frac{\phi(x, y+\Delta y) - \phi(x,y)}{\Delta y}, \end{aligned} \tag{3.20}$$

where $\Delta x$ and $\Delta y$ are sufficiently small to capture any almost-instantaneous change in the $\phi(x,y)$ field. The fluid elements are linear; therefore any variation near $(x,y)$ will be linear so the selection of $\Delta x$ and $\Delta y$ can be arbitrary.

Equation (3.20) does require that $\phi(x,y)$ can be determined at any point $(x,y)$ in the finite-element domain. The finite-element solution gives answers only at the nodes, so the element's shape functions must be used to determine the value of $\phi$ inside an element.

Given an element with nodes $i$, $j$, and $k$, and pressures at the nodes $\phi_i$, $\phi_j$, and $\phi_k$, an equation for $\phi(x,y)$, (where $(x,y)$ is inside the element) can be written in terms of the shape functions $h_i$, $h_j$, and $h_k$:

$$\phi(x,y) = h_i\phi_i + h_j\phi_j + h_k\phi_k, \tag{3.21}$$

where the shape functions are

$$\begin{aligned} h_i &= \frac{1}{2A}[a_i + b_i x + c_i y]; \\ h_j &= \frac{1}{2A}[a_j + b_j x + c_j y]; \\ h_k &= \frac{1}{2A}[a_k + b_k x + c_k y]. \end{aligned} \tag{3.22}$$

In (3.22) above, $A$ is the area of the element and the constants $a_i$, $a_j$, and $a_k$ are given by

$$\begin{aligned} a_i &= X_j Y_k - X_k Y_j; \\ a_j &= X_k Y_i - X_i Y_k; \\ a_k &= X_i Y_j - X_j Y_i. \end{aligned} \tag{3.23}$$

The constants in (3.23) use the same notation as the constants $b_i$, $b_j$, $b_k$, $c_i$, $c_j$, $c_k$ in (3.12).

$x$ and $y$ are substituted in (3.22) and (3.21) to determine the value of $\phi$ at the point $(x, y)$. This calculation is used not only to determine the fluid velocity at a point in (3.20) but also to determine the fluid presure loading on particles as described below.

## 3.11 Fluid Forces on Arbitrarily-Shaped Particles

To add fluid-flow characteristics to the discrete-element modeling system, the forces that the discrete body will experience in a fluid environment need to be characterized. This section details the governing equations and techniques used to calculate the fluid forces acting on a body. The following section derives and checks the force relationships for a two-dimensional disk. The concepts developed for the two-dimensional disk are then extended to a general two-dimensional closed polygon.

### 3.11.1 Fluid-Force Equations

From the definition for pressure,

$$P = \frac{F}{A};$$

where $P$ is the pressure, $F$ is the applied force, and $A$ is the area over which the force is applied. This can be rewritten as

$$F = PA.$$

For a two-dimensional problem, only the horizontal and vertical forces acting on the body need to be determined:

$$\begin{aligned} F_x &= p_x A; \\ F_y &= p_y A. \end{aligned}$$

Given a closed two-dimensional object $C$ (as in Figure 3-6), it can be assigned an arbitrary thickness $t$ and then the total force acting on $C$ can be determined by summing up the pressures at all of the points around the object:

$$\begin{aligned} F_x &= t \oint_C p_x(x,y)ds; \\ F_y &= t \oint_C p_y(x,y)ds. \end{aligned} \tag{3.24}$$

In the equations above, $p_x$ denotes the horizontal component of the pressure vector at the point $(x,y)$ and $p_y$ denotes the vertical component of the pressure vector at

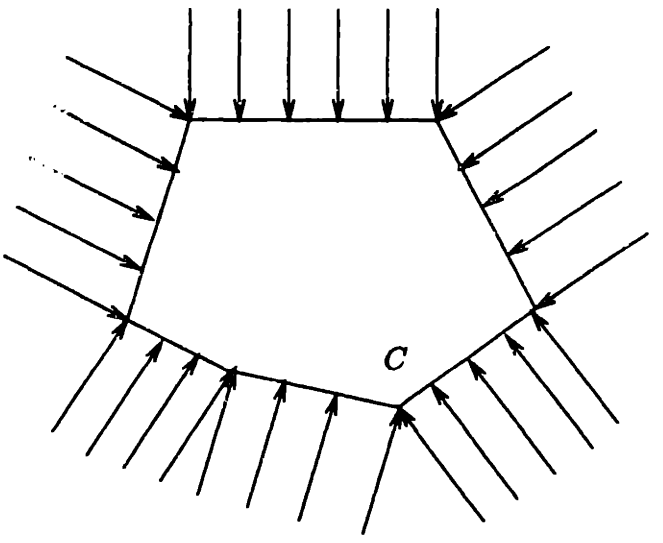



Figure 3-6: A Closed Polygon $C$ in a Pressure Field

the point $(x, y)$. Pressure forces always act normal to the surface of the object so it is necessary to make this distinction. The equations (3.24) are those which need to be evaluated to determine the total forces acting through the object's center of mass.

Similarly, the total moment acting at the object's center of mass can be determined by

$$M_z = \oint_C p_x(x,y)\Delta y ds + \oint_C p_y(x,y)\Delta x ds, \tag{3.25}$$

where $\Delta x$ is the distance along the $x$-axis between the point-of-action and the center of mass of the object and $\Delta y$ is the distance along the $y$-axis between the point-of-action and the center of mass of the object.

Evaluating (3.24) and (3.25) for each discrete element, the net fluid-pressure forces acting at the center of mass of each body are determined.

### 3.11.2 Symbolic Solution for a 2-D Disk

For most objects $C$ and most pressure fields $p(x, y)$ it would be difficult, if not impossible, to evaluate (3.24) and (3.25) symbolically. However, there are some cases which can be evaluated exactly. Select $C$ to be a circle and select a static column of fluid acted upon only by gravity, as in Figure 3-7. Then if the fluid has density $\rho_f$ and the pressure at the top of the circle is $p_0$, the pressure field $p(x, y)$ for any point $y = b$ below the top of the circle can be written as a function of $b$:

$$p(b) = p_0 + \rho_f g b,$$

where $g$ is the acceleration due to gravity. Since $C$ is a circle, it is convenient to introduce the variable $\theta$ and integrate $ds$ as $r\, d\theta$, where $r$ is the radius of $C$ and $0 \leq \theta \leq 2\pi$. The pressure field $p(b)$ can be rewritten using $\theta$ as its parameter:

$$p(\theta) = p_0 + \rho_f g r(1 - \sin\theta).$$

The $x$- and $y$-components $p_x(\theta)$ and $p_y(\theta)$ are therefore

$$\begin{aligned} p_x(\theta) = p(\theta)\cos\theta &= [p_0 + \rho_f g r(1 - \sin\theta)]\cos\theta; \\ p_y(\theta) = p(\theta)\sin\theta &= [p_0 + \rho_f g r(1 - \sin\theta)]\sin\theta. \end{aligned} \tag{3.26}$$

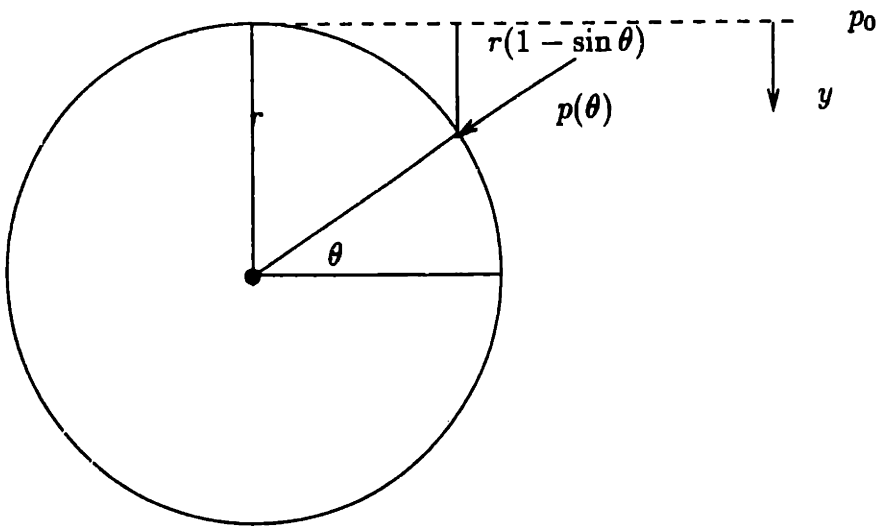



Figure 3-7: Disc Submerged in a Pressure Field

Equation (3.26) can now be substituted into (3.24). This substitution gives

$$\begin{aligned} F_x &= t \oint_c p_x ds; \\ &= t \int_0^{2\pi} p_x(\theta) r \, d\theta; \\ &= rt \int_0^{2\pi} [p_0 + \rho_f g r(1 - \sin\theta)] \cos\theta \, d\theta; \\ &= 0. \end{aligned}$$

Similarly, for the $y$-direction force,

$$\begin{aligned} F_y &= rt \int_0^{2\pi} [p_0 + \rho_f g r(1 - \sin\theta)] \sin\theta \, d\theta; \\ &= t\pi r^2 g \rho_f. \end{aligned}$$

Now substituting (3.26) into (3.25), where $\Delta x = r\cos\theta$ and $\Delta y = r\sin\theta$,

$$\begin{aligned} M_z &= \oint_C p_x(x,y)\Delta y ds + \oint_C p_y(x,y)\Delta x ds; \\ &= t\int_0^{2\pi} p_x(\theta)\sin\theta\, d\theta + t\int_0^{2\pi} p_y(\theta)\cos\theta\, d\theta; \\ &= 2t\int_0^{2\pi} [p_0 + \rho_f g r(1-\sin\theta)]\sin\theta\cos\theta\, d\theta; \\ &= 0. \end{aligned} \tag{3.27}$$

Examining the results, the formulations in (3.24) and (3.25) hold. No $x$-direction force is expected; by inspection all of the forces in the $x$-direction cancel each other exactly. Similarly, no moment $M_z$ is expected: the pressure (and therefore the force) acts normal to the surface of the circle. Therefore the line of action of each force is through the center of the circle and it experiences no net moment.

The result from the $y$-direction integration is exactly that given from Archimedes' Principle, that a body in fluid will experience an upwards force equal to the weight of the fluid it displaces. For this example problem, the volume of the object is $t\pi r^2$ and the weight of the fluid is $g\rho_f$. Then the weight of the fluid it displaces is $t\pi r^2 g\rho_f$, exactly the result above.

### 3.11.3 Numerical Solution for a Polygon

Equations (3.24) and (3.25) can be extended to analyze an arbitrary $n$-sided polygon $C$. The following assumptions are made about $C$:

1. $C$ is closed; side $n$ connects points $n$ and 1.

2. The $n$ points which make up the polygon $C$ are numered such that the points are in counterclockwise order. For example, the arc $P_1P_2P_3$, where $P_1$, $P_2$, and $P_3$ are all points on $C$, traces in a counterclockwise direction.

3. Each side $L_i$ connects points $i$ and $i+1$ (or points $n$ and 1, if $i = n$) with a straight line.

Starting with (3.24) and (3.25), the polygon $C$ can be broken into its $n$ lines $L_i$ and

$$\begin{aligned} F_x &= t \oint_C p_x \, ds &= \sum_{i=1}^{n} t \int_{L_i} p_x \, ds; \\ F_y &= t \oint_C p_y \, ds &= \sum_{i=1}^{n} t \int_{L_i} p_y \, ds; \end{aligned} \tag{3.28}$$

and

$$\begin{aligned} M_z &= \oint_C F_x \Delta y \, ds + \oint_C F_y \Delta x \, ds \\ &= \left( \sum_{i=1}^{n} \int_{L_i} F_x \Delta y \, ds \right) + \left( \sum_{i=1}^{n} \int_{L_i} F_y \Delta x \, ds \right). \end{aligned} \tag{3.29}$$

Figure 3-8 shows an arbitrary polygon side $AB$ with vertices $A(x_1, y_1)$ and $B(x_2, y_2)$, making an angle $\theta$ with the $x$-axis, in a pressure field $P(x, y)$. The pressure exerted by the fluid is of course normal to the polygon line $AB$ and therefore the pressure vector $\vec{\mathbf{n}}$ makes an angle $\alpha = \frac{\pi}{2} + \theta$ with the $x$-axis.

Breaking $\vec{\mathbf{n}}$ into components,

$$n_x = p_x = P(x, y) \cos \alpha;$$

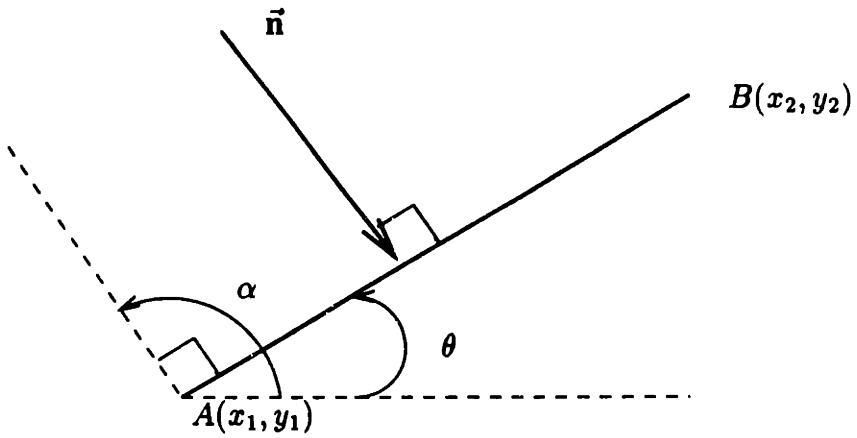



Figure 3-8: An Arbitrary Polygon Side $AB$

$$n_y = p_y = P(x,y)\sin\alpha,$$

where $P(x,y) = \|\vec{n}\|$ is the fluid pressure at point $(x,y)$. Using double-angle formulas,

$$\begin{aligned} p_x = P(x,y)\cos(\frac{\pi}{2}+\theta) &= P(x,y)\left(\cos\frac{\pi}{2}\cos\theta - \sin\frac{\pi}{2}\sin\theta\right) \\ &= -P(x,y)\sin\theta; \\ p_y = P(x,y)\sin(\frac{\pi}{2}+\theta) &= P(x,y)\left(\sin\frac{\pi}{2}\cos\theta + \cos\frac{\pi}{2}\sin\theta\right) \\ &= P(x,y)\cos\theta. \end{aligned} \tag{3.30}$$

Because $AB$ is a line segment, the trigonometric functions can be replaced with expressions which can be computed directly from the coordinates $A$ and $B$:

$$\begin{aligned} p_x &= -P(x,y)\frac{y_2 - y_1}{\sqrt{(y_2-y_1)^2 + (x_2-x_1)^2}}; \\ p_y &= P(x,y)\frac{x_2 - x_1}{\sqrt{(y_2-y_1)^2 + (x_2-x_1)^2}}, \end{aligned} \tag{3.31}$$

and the integrals become

$$\begin{aligned} F_x = t\int_L p_x\, ds &= -t\frac{y_2 - y_1}{\sqrt{(y_2-y_1)^2 + (x_2-x_1)^2}}\int_L P(x,y)\, ds; \\ F_y = t\int_L p_y\, ds &= t\frac{x_2 - x_1}{\sqrt{(y_2-y_1)^2 + (x_2-x_1)^2}}\int_L P(x,y)\, ds. \end{aligned} \tag{3.32}$$

Each of the integrals in equations (3.32) and (3.29) can be evaluated by numerical techniques such as the Newton-Cotes closed integration formulas, Simpson's Rule, etc. The technique used for the integration should be determined by the type of function that describes the pressure field. For instance, the integrals in equations (3.32) can be computed exactly if the pressure field varies linearly or bilinearly.

**Theorem 1** *For a pressure field which varies linearly or bilinearly, the integrals* $F_x = \int_L P(x,y)\, ds$ *and* $F_y = t\int_L P(x,y)\, ds$ *can be computed exactly using the trapezoidal rule.*

*Proof:* From Chapra and Canale (1988), the error for the trapezoidal rule approximation for the integral of $f$ on $[a,b]$ is

$$E_t = -\frac{1}{12}f''(\xi)(b-a)^3, \tag{3.33}$$

where $\xi$ is some arbitrary value on $[a,b]$. For a pressure field $P(x,y)$ which varies bilinearly, the field can be described as

$$P(x,y) = \alpha_0 + \alpha_1 x + \alpha_2 y + \alpha_3 xy,$$

where $\alpha_0$, $\alpha_1$, $\alpha_2$, and $\alpha_3$ are constants. Setting $f = P(x,y)$ in equation (3.33), $\frac{\partial f}{\partial x} = \alpha_1 + \alpha_3 y$ and $\frac{\partial f}{\partial y} = \alpha_2 + \alpha_3 x$. It follows that $\frac{\partial^2 f}{\partial x^2} = 0$ and $\frac{\partial^2 f}{\partial y^2} = 0$, giving $f'' = 0$ for integrating either $p_x$ or $p_y$ on $[a,b]$ so the error $E_t = 0$ and the value obtained using the trapezoidal rule is exact.

The area of a trapezoid is $A = \frac{1}{2}h(b_1 + b_2)$. The the integrals in equation (3.32) evaluate to

$$\begin{aligned}
F_x &= -t\frac{y_2 - y_1}{2\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}}\left(P(x_1,y_1) + P(x_2,y_2)\right) \cdot \sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2} \\
&= -t\frac{y_2 - y_1}{2}\left(P(x_1,y_1) + P(x_2,y_2)\right); \\
F_y &= t\frac{x_2 - x_1}{2\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}}\left(P(x_1,y_1) + P(x_2,y_2)\right) \cdot \sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2} \\
&= t\frac{x_2 - x_1}{2}\left(P(x_1,y_1) + P(x_2,y_2)\right). \qquad (3.34)
\end{aligned}$$

Finally, the forces on the whole body can be determined by substituting these evaluated integrals into equation (3.31):

$$\begin{aligned}
F_x &= -\frac{t}{2}\left\{\sum_{i=1}^{n-1}(y_{i+1} - y_i)[P(x_i,y_i) + P(x_{i+1},y_{i+1})]\right\} \\
&\quad -\frac{t}{2}(y_1 - y_n)[P(x_n,y_n) + P(x_1,y_1)]; \\
F_y &= \frac{t}{2}\left\{\sum_{i=1}^{n-1}(x_{i+1} - x_i)[P(x_i,y_i) + P(x_{i+1},y_{i+1})]\right\} \\
&\quad +\frac{t}{2}(x_1 - x_n)[P(x_n,y_n) + P(x_1,y_1)];
\end{aligned}$$

To calculate the moment experienced by the polygon, $\Delta x$ and $\Delta y$ in (3.29) need to be evaluated. Since the force curve along a side of the polygon due to the pressure

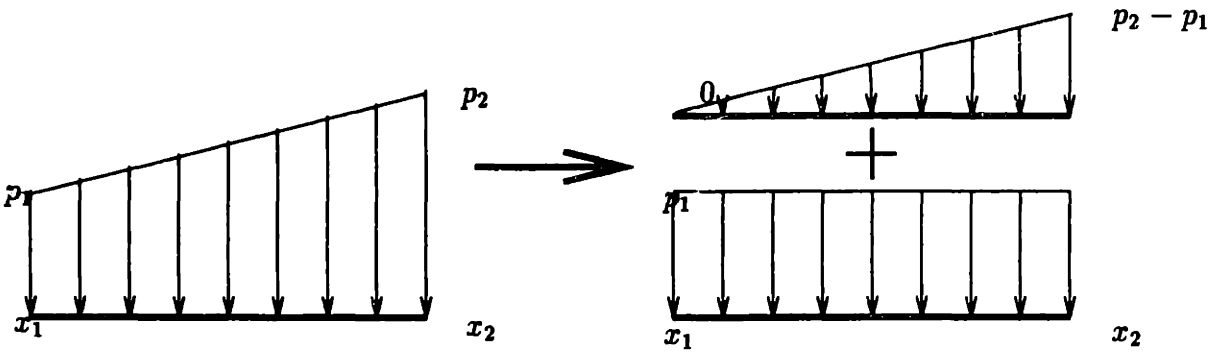



Figure 3-9: Trapezoidal Load into Triangular-plus-Rectangular Load

field is trapezoidal in shape, the force can be considered as the sum of a triangular load curve and a rectangular load curve as in Figure 3-9.

This figure depicts a horizontal polygon side, where $p_1$ and $p_2$ are the values of the distributed pressure forces *normal* to the side. Then the distributed triangular force can be replaced by a point force of $\frac{1}{2}(x_2 - x_1)(p_2 - p_1)$ acting at the point $x_1 + \frac{2}{3}(x_2 - x_1)$. Similarly, the rectangular distributed force can be replaced by a point force of $p_1(x_2 - x_1)$ acting at the point $x_1 + \frac{1}{2}(x_2 - x_1)$.

These two forces can be combined into one resultant point force $p_1(x_2 - x_1) + \frac{1}{2}(x_2 - x_1)(p_2 - p_1)$. The point-of-action for this force can be determined by the parallel-axis theorem:

$$\begin{aligned} \frac{\sum F \cdot d}{\sum F} &= \frac{p_1(x_2 - x_1)(x_1 + \frac{1}{2}(x_2 - x_1)) + \frac{1}{2}(x_2 - x_1)(p_2 - p_1)(x_1 + \frac{2}{3}(x_2 - x_1))}{p_1(x_2 - x_1) + \frac{1}{2}(x_2 - x_1)(p_2 - p_1)} \\ &= x_1 + \frac{x_2 - x_1}{3(p1 + p_2)}(p_1 + 2p_2). \end{aligned} \tag{3.35}$$

The value of $\Delta y$ in (3.29), then, is given by (3.35) and the value of $\Delta x$ in (3.29) is given by an analogous expression.

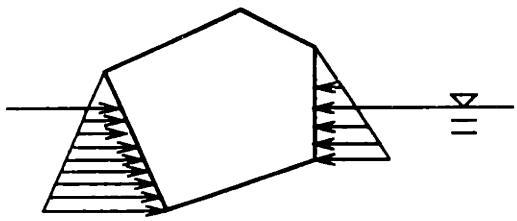


Figure 3-10: Incorrect Computation of Pressure Force for Buoyancy Case

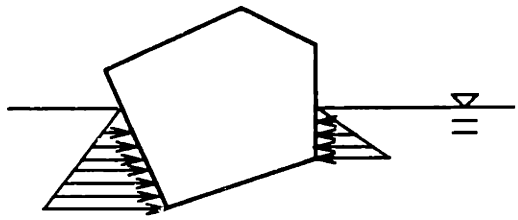


Figure 3-11: Correct Computation of Pressure Force for Buoyancy Case

### 3.11.4 Handling Particle Buoyancy

It is possible that for some point $P_i(x_i, y_i)$ on $C$, the pressure $p(x_i, y_i) = 0$, while for the next point on $C$, $P_{i+1}(x_{i+1}, y_{i+1})$, the pressure $p(x_{i+1}, y_{i+1}) \neq 0$. It is possible that the pressure field $p$ reaches the value 0 just at the point $P_{i+1}$, but the case exists where the pressure field $p$ reaches 0 (and remains at 0) somewhere between $P_i$ and $P_{i+1}$. This is the case where a body is partly submerged in fluid, that is, the "buoyancy" case.

The algorithm above assumes that the pressure varies linearly on a side from $P_i$ to $P_{i+1}$. Therefore the naive algorithm would give an answer like that depicted in Figure 3-10, which is incorrect. Instead the pressure loading must be calculated as in Figure 3-11, which depicts the correct loading on the element.

Since only the coordinates of the cornerpoints of the polygon are known, the intersection point between a polygon's line segment and the fluid mesh must be de-

termined. To save computation time, the intersection points along line segments $L$ which have one positive endpoint pressure and one zero endpoint pressure are the only points which are considered. Note that this restriction excludes cases where the pressures at the endpoints are nonzero but the pressure field is zero along the middle of the line segment somewhere. It is difficult to imagine a physical problem with these conditions, however, so this case is excluded from analysis.

Sedgewick (1992) gives a quick algorithm to determine if two line segments intersect. The line segment $L$ is checked against fluid elements in its neighborhood (using the fast-searching algorithms described in Section 3.13.1, below), and, if the line segments *do* intersect, the intersection point is determined using Cramer's Rule[6].

This procedure determines the location of another point on the polygon $C$, which happens to be exactly the point where the fluid pressure becomes zero. The point can be inserted into the list of points which represent a polygon, again keeping the strict counterclockwise representation. This modification to the algorithm will handle the buoyancy problem while allowing for the use of the same general fluid-force computation techniques described above.

This section has developed expressions to calculate the forces and moments due to a fluid pressure field. The moment calculations require information about the centroid of the polygon; the following sections describe a technique to determine the area and centroid of an $n$-sided polygon $C$.

[6]Care must be taken when computing line-segment intersections to ensure that neither line segment is vertical; in this case the slope of the line is undefined and Cramer's Rule fails.

## 3.12 Area and Centroid Calculations

The preceding sections describe all the mathematics and algorithms necessary to calculate a pressure and velocity field for a fluid and how to use that information to determine the forces acting on an $n$-sided polygon. These computations, however, assume that the location of the centroid of the polygon is known. This section develops a method to compute the area and centroid of a polygon considering the polygon to be a set of several triangles.

### 3.12.1 Area of a Triangle

For a triangle with verticies $(x_i, y_i)$, $(x_j, y_j)$, and $(x_k, y_k)$, where the verticies are numbered such that $i$, $j$, and $k$ are in counterclockwise order, the area $A$ of the triangle is given by

$$\begin{aligned} A &= \frac{1}{2}\begin{vmatrix} 1 & x_i & y_i \\ 1 & x_j & y_j \\ 1 & x_k & y_k \end{vmatrix} \\ &= \frac{1}{2}(x_j y_k + x_k y_i + x_i y_j - x_j y_i - x_i y_k - x_k y_j) \end{aligned} \tag{3.36}$$

(Segerlind, 1984). Note that if the verticies $i$, $j$, and $k$ are *not* in counterclockwise order, rows 1 and 3 of the determinant in (3.36) are swapped and the resulting $A$ is negative.

### 3.12.2 Area of a Polygon

(3.36) can be used to determine the area of any arbitrarily-shaped $n$-sided polygon $C$. Using the same assumptions from Section 3.11.3 and designating point $P_1$ as an "anchor" point, the area of the polygon $C$ can be computed as

$$\begin{aligned} A_C &= \frac{1}{2}\sum_{i=2}^{n-1}\begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_i & y_i \\ 1 & x_{i+1} & y_{i+1} \end{vmatrix} \\ &= \frac{1}{2}\sum_{i=2}^{n-1}(x_i y_{i+1} + x_{i+1}y_1 + x_1 y_i - x_i y_1 - x_1 y_{i+1} - x_{i+1}y_1). \end{aligned} \tag{3.37}$$

Because the determinant in (3.37) can be negative (for clockwise $P_i$, $P_j$, $P_k$), the "cut out" (or nonconvex) sections of the polygon are subtracted from the area of the polygon and the algorithm holds for *all* polygons $C$.

### 3.12.3 Centroid of a Triangle

Consider Figure 3-12, which depicts a triangle $ABC$ (not necessarily a right triangle) with verticies $A(x_1, y_1)$, $B(x_2, y_2)$ and $C(x_3, y_3)$ relative to some fixed coordinate system. The location of the centroid of this triangle with respect to this fixed coordinate system needs to be determined.

On Figure 3-12, construct $\overline{CD}$, with length $h_1$, which is the height of the triangle if $\overline{AB}$ is considered the base of the triangle. The centroidal axis for the triangle $ABC$ is then parallel to $\overline{AB}$, at a distance $h_1/3$ from $\overline{AB}$. This centroidal axis is depicted on Figure 3-12 as $\overline{EH}$; this line intersects side $\overline{AC}$ at point $X$. We can further construct

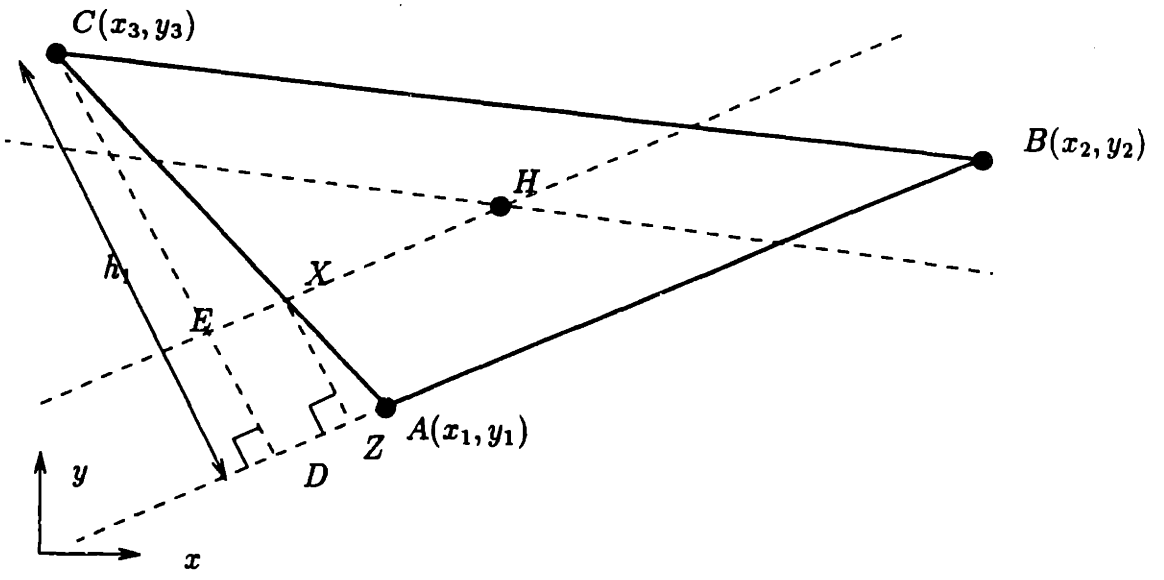



Figure 3-12: Example Triangle for Centroid Calculations

the line $\overline{XZ}$, perpendicular to $\overline{DB}$.

Since $\angle CDA \cong \angle XZA$ and $\angle DAC \cong \angle ZAX$, triangle $DAC$ and $ZAX$ are similar. Therefore,

$$\frac{CD}{XZ} = \frac{CA}{XA},$$

since corresponding sides of similar triangles are proportional. But $\overline{XZ} = \overline{ED}$ and $\overline{ED} = \frac{1}{3}\overline{CD}$ since $\overline{EH}$ is the centroidal axis, so the above expression can be rewritten

$$\overline{XA} = \frac{1}{3}\overline{CA}.$$

The same arguments can be applied again, using $\overline{BC}$ as the base of the triangle. Omitting these calculations for brevity, it is determined that the centroidal point

$H(x_h, y_h)$ is located at

$$\begin{aligned} x_h &= x_1 + \frac{1}{3}(x_3 - x_1) + \frac{1}{3}(x_2 - x_1); \\ y_h &= y_1 + \frac{1}{3}(y_3 - y_1) + \frac{1}{3}(y_2 - y_1), \end{aligned}$$

or, more compactly,

$$\begin{aligned} x_h &= \frac{1}{3}(x_1 + x_2 + x_3); \\ y_h &= \frac{1}{3}(y_1 + y_2 + y_3). \end{aligned} \tag{3.38}$$

### 3.12.4 Centroid of a Polygon

Given that the centroid of any triangle and the area of any polygon can be determined using sets of triangles, it follows that the centroid of the polygon can be determined using the same triangulation.

For any shape which can be broken into $i$ piecewise areas, the centroidal point $(\overline{X}, \overline{Y})$ can be computed as

$$(\overline{X}, \overline{Y}) = (\frac{\sum_i A_i \overline{x}_i}{\sum_i A_x}, \frac{\sum_i A_i \overline{y}_i}{\sum_i A_i}), \tag{3.39}$$

where $\overline{x}_i$ and $\overline{y}_i$ are the distances from the centroid of the area $A_i$ to an arbitrary $x$- and $y$-axis, respectively.

The centroid of the polygon, then, is simple to compute given the above techniques. $\sum_i A_i = A_C$, which can be computed from (3.37). From (3.36) and (3.38), $A_i$, $\overline{x}_i$, and

$\overline{y_i}$ can be computed. As in (3.37), it is useful to use point 1 as an "anchor" point. Combining the equations,

$$\begin{aligned}
\sum_i A_i \overline{x}_i &= \frac{1}{6} \sum_{i=2}^{n-1} \{(x_1 + x_i + x_{i+1}) \cdot \\
&\quad (x_i y_{i+1} + x_{i+1} y_1 + x_1 y_i - x_i y_1 - x_1 y_{i+1} - x_{i+1} y_i)\}; \\
\sum_i A_i \overline{y}_i &= \frac{1}{6} \sum_{i=2}^{n-1} \{(y_1 + y_i + y_{i+1}) \cdot \\
&\quad (x_i y_{i+1} + x_{i+1} y_1 + x_1 y_i - x_i y_1 - x_1 y_{i+1} - x_{i+1} y_i)\};
\end{aligned} \tag{3.40}$$

This can be combined with (3.39) to determine the centroid of the $n$-sided polygon $C$.

## 3.13 Implementation in the MIMES System

A detailed description of the code written to calculate fluid-force loading in MIMES is beyond the scope of this paper. The code itself, implemented mostly in C++, is self-documenting and includes comments to explain various computational techniques, which, while detailed here, may be somewhat obfuscated in the code due to pre-existing data structures, necessary intermediate steps, etc. It is hoped that an understanding of the information flow path in Section 3.3 will explain the general picture of the code and from there the intermediate steps should be clear.

The following sections elaborate on some of the larger issues facing the implementation of the fluid-flow system.

### 3.13.1 Efficient Searching for Nodal Pressures

The fluid-force computations detailed above all require the value of the pressure field $\phi$ at a given point $(x, y)$. This problem is analagous to that described in Section 3.5.2 above, yet the data structures and algorithms presented there will not work particularly well for this problem. The previously-considered problem determined which points in a domain fall inside a given rectangle; here, it is necessary to determine which triangle a given point falls inside. An efficient searching algorithm to answer this question will markedly increase the speed of the fluid-force computations.

One straightforward method is a *grid method*, such as that described in Sedgewick (1992). The fluid domain is broken up into an artificial grid of small squares; each one of these grid squares keeps a list of triangular elements which cross it. To find out which element contains the point $(x, y)$, a simple check of the grid square which would contain the point $(x, y)$ gives a list of triangular elements which cross the grid square. Each of these elements can be checked in turn to see if the point is contained in the candidate element[7]. Once the proper finite element has been selected, the techniques of Section 3.10 can be used to interpolate to the pressure at that point.

The grid method is an efficient algorithm for finite-element use, since the finite-element domain is generally regular so the lists of elements that each grid square keeps track of will be fairly uniform in size. Even if the domain is irregular and the lists are of uneven length, the search time is reduced from having to check all $N$ finite-elements so the algorithm improves the performance of the system.

[7]There are algorithms to determine if a point is inside a polygon in O'Rourke (1994) and Sedgewick (1992).

### 3.13.2 Incorporation into the Structure of MIMES

Fortunately, the MIMES system, developed in C++, uses an object-oriented paradigm which allows for straightforward modification to incorporate the fluid loading. The `Element_State` class (detailed in Rege, 1996) is altered to handle two new vector quantities `flu_force` and `flu_moment`, which are determined using the techniques outlined in this chapter. These quantities are then incorporated into the calculations when the total force and moment are determined in the method `Element::integrate()`. The numerical integration procedures for the rigid-body motion are identical to those already detailed in Rege (1996).

Many new commands and variables have been added to the MIMES shell. These are detailed in Appendix B. Some sample problem files are included in Appendix C; these may also be instructive as they show working problems for the system. The finite-element solver has its own internal parser for defining and solving a problem; the commands for this parser are detailed in Appendix A.

# Chapter 4

# Applications and Results

This chapter presents some applications of the coupled fluid-flow/discrete-element modeling system. Several simple test problems are examined, since the results from the system are readily verifiable.

## 4.1 Test Problem: Flow Underneath a Dam

This first test problem is designed to ensure that the area meshing and fluid-flow computations are accurate. A "dam" structure was created in MIMES, with three connected fluid regions (A, B, and C), as depicted in Figure 4-1. Each of the regions A, B, and C, has a different geometry and a different mesh resolution associated with the geometry. The pressure head $\phi$ is set to be 40 at the top of region A; it is confined to 0 at the top of region C. The rectangular discrete elements are fixed in place to create a channel for particles to flow through. All the boundaries (except those with given $\phi$ values, above) are impermeable, which is the automatic boundary condition for these analyses.

The MIMES file which created the test problem is listed in Section C.1, and a

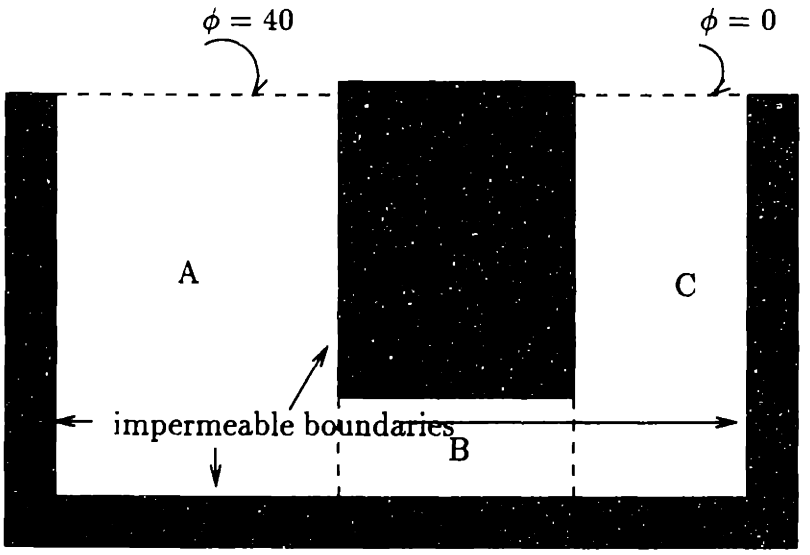



Figure 4-1: Dam Flow Test Problem

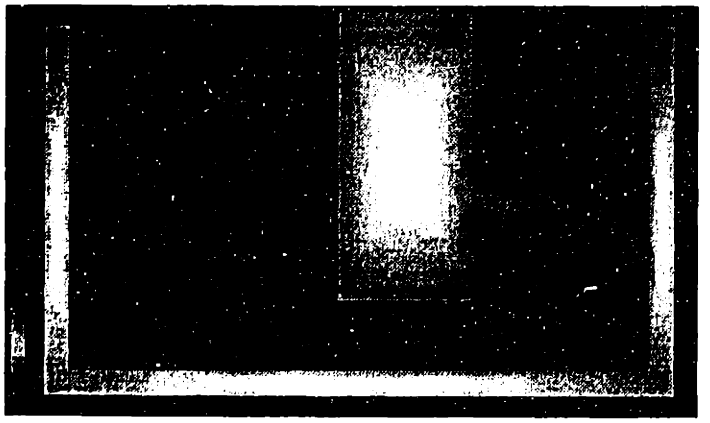


Figure 4-2: Dam Flow Screenshot

screenshot result is depicted in Figure 4-2. While the screenshot does not clearly depict the flow pattern (it is much clearer in color), the problem domain is clear. The adaptive meshing between regions A and B and regions B and C is obvious, however; the system has generated extra finite elements to ensure nodal compatibility across the faces of the elements.

Figure 4-3 shows the equipotential lines[1]. In the figure, the pressure gradient

[1]This graph was generated using *AVSexpress* and **mustafa** from Sandia National Laboratories.

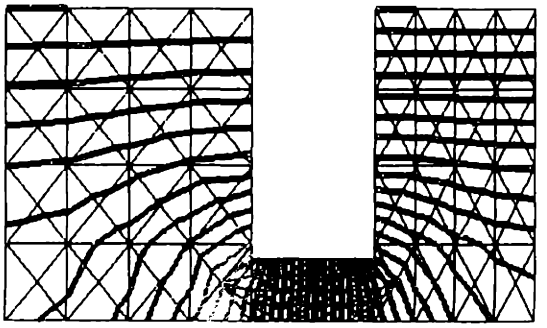


Figure 4-3: Dam Flow Equipotential Lines

between each of the equipotential lines is the same, so areas which have a greater density of equipotential lines indicate a sharper pressure gradient.

The result from the figure is essentially the expected result for a dam-flow problem. The pressure gradient is greatest underneath the dam and the equipotential lines "curve" to show the fluid flow down under the weir.

## 4.2 Test Problem: Particle Buoyancy

Another good test problem is a particle floating in a fluid. Archimedes' Principle states that the buoyancy force upwards on a body is equal to the weight of the fluid displaced by the body. To test the system, bodies can be selected which have densities less than and greater than the fluid; the results from different trials can be compared.

The experimental setup is depicted in Figure 4-4 and the MIMES program listing is in Section C.2. The initial-position screenshot for all experiments is depcited in Figure 4-5. In each trial, a particle (here, a superquadric shape) is allowed to freefall

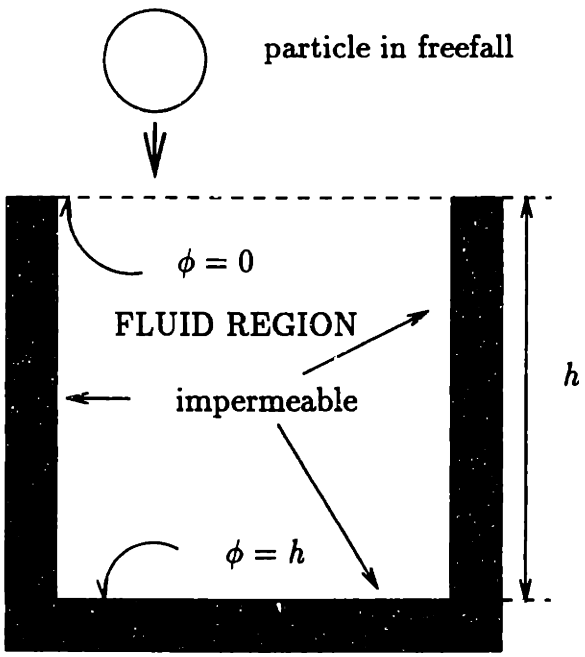



Figure 4-4: Buoyancy Test Problem

into a standing fluid field. The density of the particle is changed for various trials by altering the MIMES input source. The following paragraphs detail the experimental results for different values of $\rho_{particle}$ relative to $\rho_{fluid}$.

**Case:** $\rho_{particle} = \frac{1}{2}\rho_{fluid}$

For the first trial, the density of the particle $\rho_{particle}$ was set to a value exactly one-half of the density of the fluid $\rho_{fluid}$. The particle will be in equilibrium when theweight of fluid displaced exactly equals the weight of the particle. In this case, when half of the particle is submerged in fluid, it will displace fluid weight equal to the total weight of the particle and therefore it will float on the surface of the fluid, half in the fluid and half out of the fluid. This is exactly the end result determined with the MIMES system and is depicted in the Figure 4-6 screenshot.

If the particle is initially rotated somewhat (so one of its axes of symmetry does

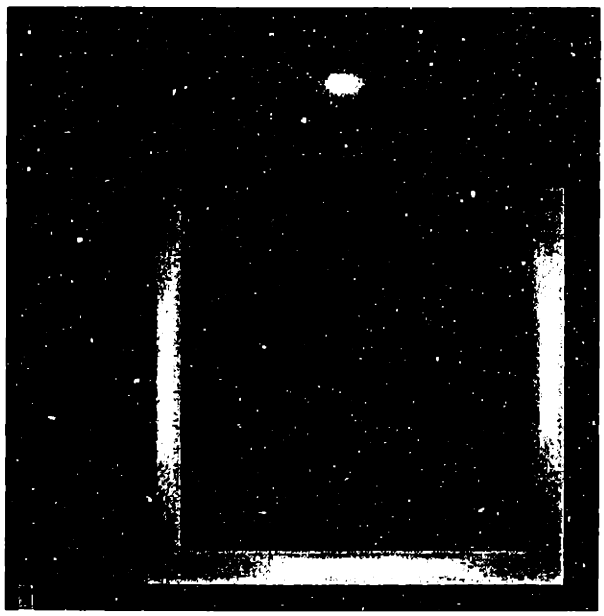


Figure 4-5: Screenshot of Initial Timestep for Particle Buoyancy Problem

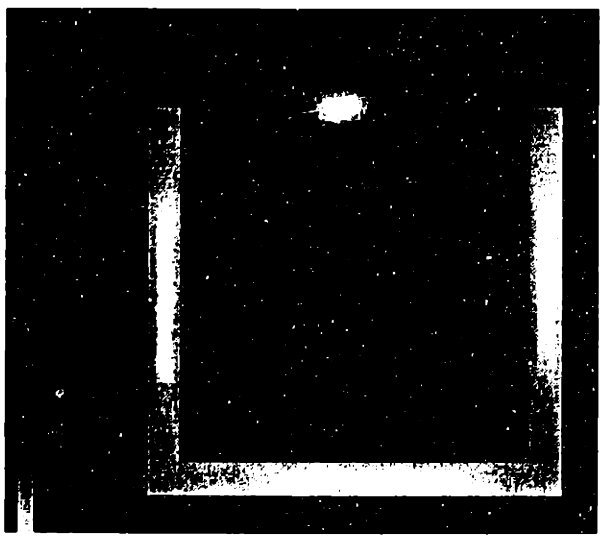


Figure 4-6: Particle Buoyancy Problem: $\rho_{particle} = \frac{1}{2}\rho_{fluid}$

Figure 4-7: Particle Buoyancy Problem: $\rho_{particle} = \rho_{fluid}$

not fall perpendicular to the fluid surface), the moments experienced by the particle will rotate the particle so its final position is also that depicted in Figure 4-6.

**Case:** $\rho_{particle} = \rho_{fluid}$

For the second trial, the density of the particle $\rho_{particle}$ was set to a value exactly that of $\rho_{fluid}$. Again, equilibrium is achieved when the weight of the fluid displaced exactly equals the weight of the particle; this equilibrium is achieved when the particle sits just under the surface of the fluid. This is exactly the end result determined with the MIMES system and is depicted in the Figure 4-7 screenshot.

If the particle is initially rotated or if its initial position is completely within the fluid domain, we would expect that the particle will not move at all, since the equilibirum condition will already be satisfied. Indeed, that is the result for MIMES runs with those conditions.

**Case:** $\rho_{particle} > \rho_{fluid}$

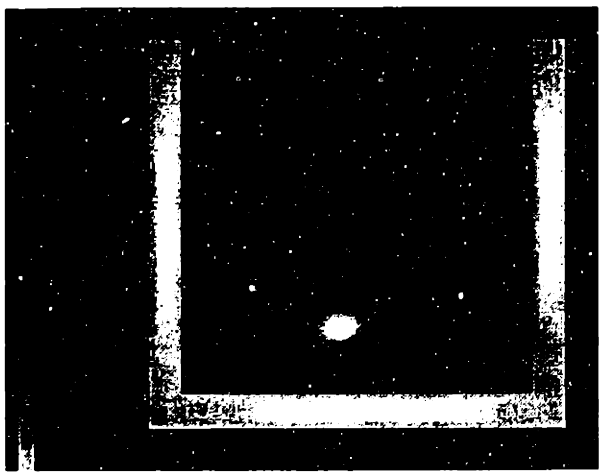


Figure 4-8: Particle Buoyancy Problem: $\rho_{particle} > \rho_{fluid}$

For the third trial, the density of the particle $\rho_{particle}$ was set to a value greater than that of $rho_{fluid}$. Since the weight of fluid displaced by the particle will never be greater than the weight of the particle, the particle will always experience a downward acceleration until it reaches the boundary. Again, this is exactly the result determined from the MIMES trials and is depicted in Figure 4-8.

Of course, the ratio $\rho_{particle}/\rho_{fluid}$ determines the *rate* at which the particle falls. For a ratio only slightly greater than 1, the particle will fall through the fluid, but very slowly. For a ratio much greater than one, the particle will fall much more quickly.

## 4.3 Test Problem: Fluid Pressure Wave

This test problem is designed to verify the transient solution scheme used for the fluid system. Here a fluid region is created where the initial conditions are a pressure head at the bottom of the region and no pressure anywhere else in the system. (See Figure 4-9.) The final, static solution is a linear pressure gradient from $\phi = 0$ at the top of

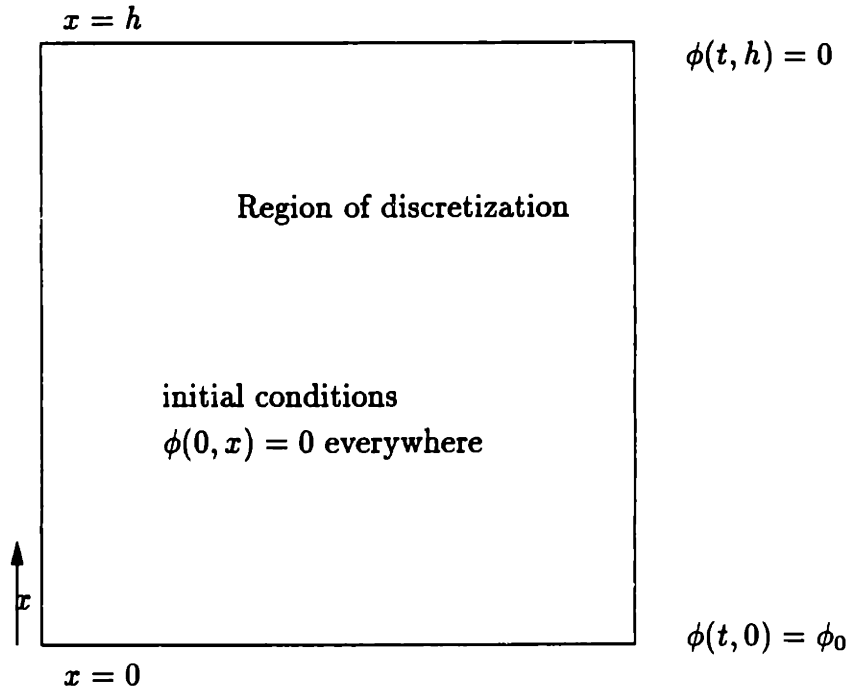



Figure 4-9: Setup for Fluid Pressure Wave Problem

the region ($x = h$)to $\phi = \phi_0$ at the bottom of the region ($x = 0$).

To verify the solution, note that the differential equation

$$\frac{\partial \phi}{\partial t} = \frac{\partial^2 \phi}{\partial x^2} \tag{4.1}$$

describes a dynamic heat-flow equation, or a dyanamic fluid-flow equation such as (3.17) where all the constants have been set to 1. For (4.1) with initial condition $\phi(0, x) = 0$ and boundary conditions $\phi(t, 0) = \phi_0$ and $\lim_{x \to \infty} \phi(t, x) = 0$, the solution is

$$\phi(t, x) = \phi_0 \text{erfc} \frac{x}{2\sqrt{t}}, \tag{4.2}$$

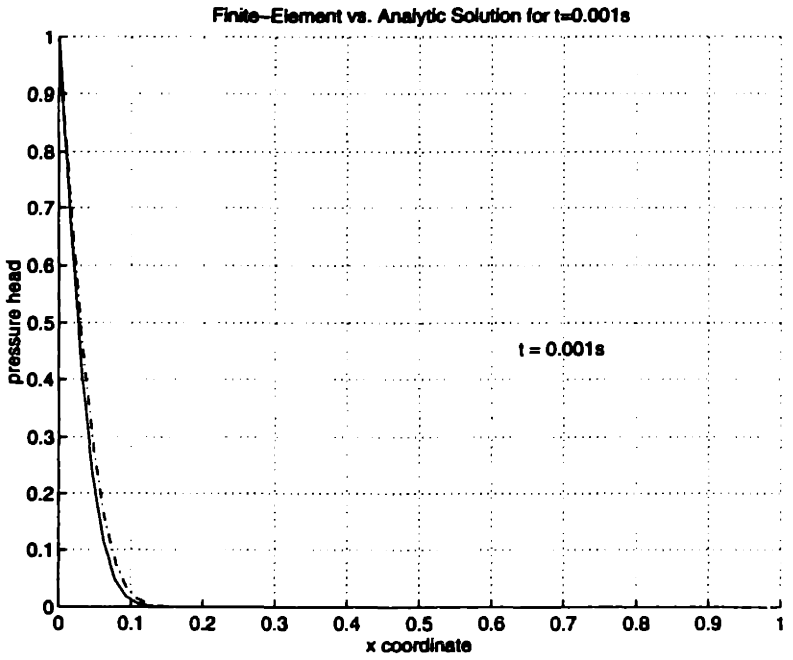



Figure 4-10: Finite-Element (solid) vs. Analytic (dash-dot) Solution for $t = 0.001s$.

where the complementary error function is defined as

$$\text{erfc}(z) = 1 - \frac{2}{\sqrt{\pi}} \int_0^z e^{-u^2} du. \tag{4.3}$$

For the test problem, the boundary conditions are somewhat different; instead of the limit boundary condition (for an infinite rod or fluid field), the pressure at $x = h$ is constrained to $\phi = 0$. However, the solution (4.2) can still be used to verify the numerical algorithm for the first several timesteps–the boundary condition at $x = h$ will not have a large effect on the solution for small $t$. Finite-element meshes of 4, 8, 16, 32, and 64 elements along the $x$-axis were used in separate runs.

The finite-element solutions and the analytic solutions for times $t = 0.001s$, $t = 0.005s$, and $t = 0.010s$ are presented in Figures 4-10, 4-11, and 4-12, respectively. In each figure, the analytic solution is graphed in "dash-dot" and the finite-element

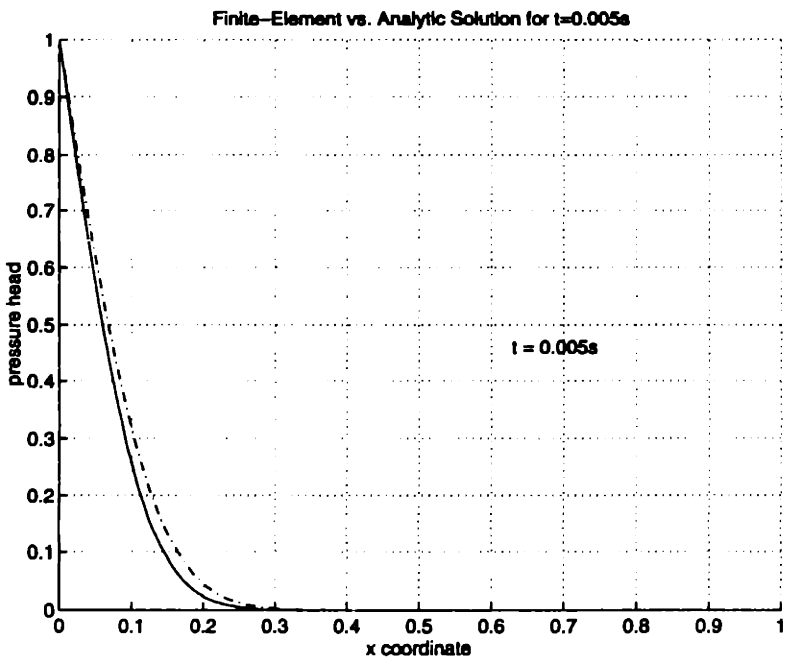



Figure 4-11: Finite-Element (solid) vs. Analytic (dash-dot) Solution for $t = 0.005s$.

solutions from MIMES are graphed in dotted or solid lines. The rightmost dotted line is the solution for 4 elements along the $x$-axis; moving right to left, the graphs are for an 8-element solution (solid), a 16-element solution (dotted), a 32-element solution (solid), and a 64-element solution (dotted).

From Figures 4-10, 4-11, and 4-12, it is clear that the finite-element solution determined by the procedures here gives an answer which is very close to the exact result. The MIMES input source for this problem is listen in Section C.3.

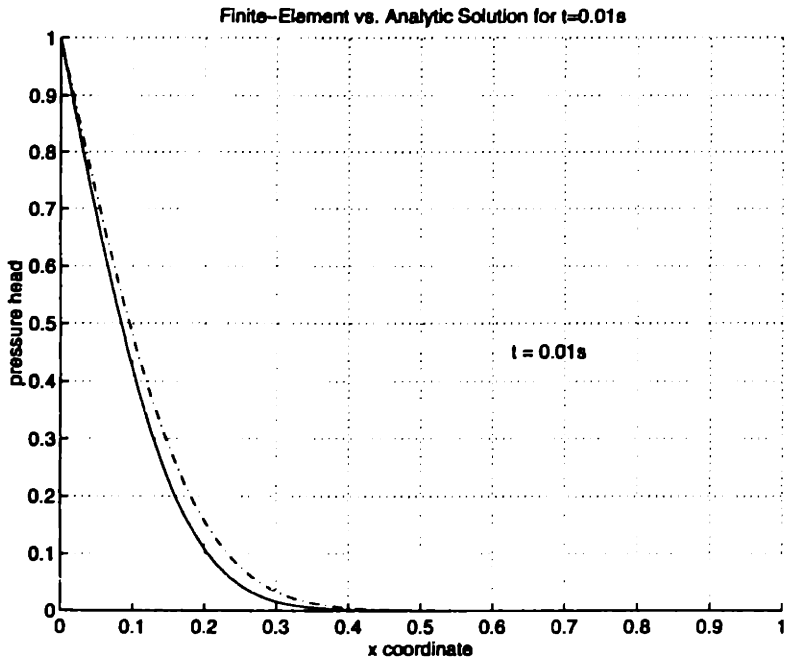



Figure 4-12: Finite-Element (solid) vs. Analytic (dash-dot) Solution for $t = 0.010s$.

# Chapter 5

# Conclusions

Chapters 2 through 4 have discussed the discrete-element method, developed methods of computing fluid flow and fluid forces on discrete elements, and verified the computational methods against analytical solutions. Here is a summary of the results and some suggestions for further study.

## 5.1 Future Uses and Applications

A lot of research which has involved the use of discrete-element simulations has focused on geotechnical problems—e.g. Preece (1993), Rege (1996), Rege and Williams (1996). It is natural, then, to apply the integrated fluid-force/discrete-element system to geotechnical problems.

One area of application includes the field of near wellbore mechanics, or the area of research which answers the question "what happens around a wellbore in an oil well?" When oil is pumped up from a well through a formation, sand is inevitably produced with the oil. This sand must be separated from the oil product (an expensive procedure) and it also clogs machinery and forces costly maintenance and repair.

Oil production companies would like to allieviate this expensive problem. Several *empirical* studies and potential solutions have been presented, e.g. Morita and Boyd (1991), Morita et. al. (1989) but the mechanics of the problem have not been discovered. A discrete-element model which incorporates the fluid dynamics methods outlined here could help to elucidate the mechanics behind sand production in oil wells, and in fact such work is already underway.

Another area of research is the field of soil liquefaction, where

> A loose saturated sand deposit, when subjected to vibration, tends to compact and decrease in volume. If drainage is unable to occur, the pore water pressure increases. If the pore water pressure in the sand deposit is allowed to build up by continuous vibration, a condition will be reached at some time where the overburden pressure will be equal to the pore water pressure. (Das, 1983, p. 351)

Indeed, "[l]iquefaction of saturated sands during earthquakes has been the cause of much damage to buildings, earth embankments, and retaining structures" (Das, 1983, p. 353). The gravity and importance of the phenomenon is proven by the tremendous volume of published work and conference proceedings which deal with the predication and moderation of the problem.

Much soil liquefaction literature takes a finite-element view of soil liquefaction (i.e. Popescu, 1995) or attempts to elucidate the mechanics of soil liquefaction from experimental studies and the empirical relations which can be developed from the data returned by these experiments. With discrete-element simulations, we are able to delve into the *microscopic* interactions among the particles instead of analyzing data at a *macroscopic* or experimental level. With the coupled fluid-flow/discrete-element simulation it may be possible to numerically model a soil sample (as in Rege,

1996), load the sample using an earthquake time history such as the 1940 El Centro earthquake, and observe pore-pressure buildup leading to a state of liquefaction. Some background work has begun in this area as well.

Other areas of application could include particle flows through pipes, especially the flow of suspensions around elbows. Results from these types of simulations could be of interest to engineers who design factories which need to transport suspensions for chemical production. Similarly, the system could be of use to environmental engineers who try to predict contaminant flow through porous media. Perhaps a three-dimensional system would be better suited to handle some of these problems but the two-dimensional methods presented here could give a good "first-cut" representation of many physical phenomena.

## 5.2 Areas of Further Study

This thesis has presented one method of computing a fluid pressure field, computing the fluid forces on discrete elements, and integrating these computations with an exisiting discrete-element system. Work on this subject is by no means complete. Some areas for further study include:

1. The fluid-force computations should include viscosity effects. At present only pressure forces are considered; ignoring the viscosity forces could give inaccurate results for some simulations.

2. A more comprehensive fluid-flow model may be necessary for some problems. The "smeared" Darcy-flow approach works sufficiently well for many discrete-

element problems; however, for problems which do not involve seepage (or other problems away from the Darcy-flow realm), the fluid-flow model here may not be sufficient.

3. The computations described in Chapter 3 should be extended and tested to three-dimensional problems. The integrals involved are somewhat more complicated but are solvable. For a three-dimensional discrete-element simulation, a more comprehensive flow model would also be a good addition since the systems under consideration can be modeled more realistically.

4. From a software standpoint, the graphical user interface for the fluid mesh can be extended and cleaned up somewhat. The underlying C++ functions do not need to be altered, but the interface should be altered to remain consistent with that which already existed in MIMES.

There are of course other areas of research interest. The MIMES package could incorporate several fluid-flow solution techniques (i.e. a lattice gas method in addition to the Darcy flow) and let the user decide which solution package is appropriate to the problem at hand.

## 5.3 Concluding Remarks

The system developed here extends the capabilities of an already-robust computational-mechanics laboratory. The integration of fluid dynamics allows for the solution of a whole new class of problems which have not been considered before. The fluid dy-

namics solution module has been checked for static and dynamic solutions and the results for fluid pressures and particle motion agree well with accepted analytic solutions. It is fair to stipulate that the computational laboratory can be used to predict real-world behavior of physical systems.

The solutions generated here are still *computational* solutions to these problems, however. It is entirely possible that forces exist in these systems which have not been considered here. The work in developing discrete-element simulations will not be complete until a computer system such as MIMES can accurately predict phenomena which can be reproduced in a laboratory or in-situ test, through invasive or non-invasive procedures. With increased computing power and a developing knowledge of computational methods which can accurately model physical systems, there is great hope that discrete-element simulations will become more robust and applicable to problems in multibody mechanics.

# Appendix A

# Finite-Element Subsystem Commands

To solve for fluid pressures, the MIMES system incorporates a finite-element solution system. This system is implemented in an object-oriented paradigm, using C++, and is not specific to fluid finite-element problems. It is a relatively straightforward process to add other finite elements to solve other problems.

The system is command-driven; that is, a parser is given a command which it then interprets and acts upon. While a complete discussion of the implementation is beyond the scope of this document[1], the commands understood by the parser are listed here for clarity in understanding the code.

The command name is listed in **typewriter** font. Required parameters are listed in **emphasized** font; optional parameters are listed in (*parenthesized emphasized*) font. The analysis program is not case-sensitive so any combination of lower- and upper-case letters can be used within the program. Parameters following a command can be separated with commas or with blank space; for instace, the following are identical:

```
n, 1, 10.0, 10.0
n 1 10.0 10.0
```

The command list follows.

bc, *node-number*, *degree-of-freedom*, (*value*) sets a displacement boundary condition for node *node-number*, degree-of-freedom *degree-of-freedom*. The displacement can be set to some optional *value*; if no *value* is given, the displacement is set to 0. Valid inputs for the *degree-of-freedom* field are **x**, **y**, **z**, for $x$-, $y$-, and $z$-displacements, respectively, and **xr**, **yr**, **zr**, for rotations about the $x$-, $y$-, and $z$-axes, respectively.

bcgraph, (*switch*) graph boundary conditions or switch boundary conditions graphing on/off. If bcgraph is specified without any *switch*, boundary condition

[1] A more comprehensive discussion can be found in Klosek (1996).

graphing is turned *on* and the elements are graphed on the screen. Alternately, *switch* can be on or `off` as desired.

`bcprint` print all boundary conditions currently imposed on the structure to the screen or to a text file specified by a previous `outfile` command.

`bye` exit the program.

`deffactor`, *multiplication-factor* sets the *multiplication-factor* used when drawing the deformed shape on the screen. Since structural deflections are often quite small relative to the size of the structure, these often need to be magnified for graphical display. *multiplication-factor* needs to be a value greater than zero.

`deflections` print out the calculated deflections for the problem, either to the terminal screen or to a file previously specified with a previous `outfile` command.

`defshape`, (*switch*) graph the deformed shape or switch deformed-shape graphing on/off. If `defshape` is specified without any *switch*, deformed-shape graphing is turned *on* and the deformed shape is graphed on the screen. Alternately, *switch* can be on or `off` as desired.

`e`, *element-number*, *node-1*, *node-2*, ... define an element. The element number is given by *element-number* and its node connectivity by *node-1*, *node-2*, etc. The number of nodes required depends on the element selected. Note that the nodes must be defined before the element.

`egraph`, (*switch*) graph elements or switch element graphing on/off. If `egraph` is specified without any *switch*, element graphing is turned *on* and the elements are graphed on the screen. Alternately, *switch* can be either on or `off` as desired.

`eltype`, *element-name* change current element type to *element-name*. *element-name* must be one of the elements supported by the program. For the MIMES implementation of the system, the only element supported is called `FETriflu2D`.

`eprint` print all elements (and their node connectivities) to the screen or to a text file specified by a previous `outfile` command.

`exit` exit the program.

`load`, *node-number*, *direction*, *value* define an applied concentrated load *value* at node *node-number*, in the direction *direction*. Valid values for *direction* are `x`, `y`, `z`, `xr`, `yr`, and `zr`. Note that applying two loads on the same load in the same direction has the effect of *adding* the two loads, not replacing the older one with the newer one.

`loadprint` print all applied loads to the screen or to a text file specified by a previous `outfile` command.

`loadgraph`, (*switch*) graph load vectors or switch load vector graphing on/off. If `loadgraph` is specified without any *switch*, load vector graphing is turned *on* and the load vectors are graphed on the screen. Alternately, *switch* can be either `on` or `off` as desired.

`n`, *node-number*, *x-coordinate*, (*y-coordinate*), (*z-coordinate*) define a new node. The node's number is given by *node-number* and its coordinates by *x-coordinate*, *y-coordinate*, and *z-coordinate*, respectively. Note that if you give a *node-number* which already exists, the new `n` command will overwrite the old node information.

`ngraph`, (*switch*) graph nodes (and node numbers) or switch node graphing on/off. If `ngraph` is specified without any *switch*, node graphing is turned *on* and the nodes (and their node-numbers) are graphed on the screen. Alternately, *switch* can be either *on* or *off* as desired.

`nprint` print out node number and coordinates for all of the nodes to the screen or to a text file specified by a previous `outfile` command.

`outfile`, (*filename*) redirect the output from the terminal display to the file *filename*. The command `outfile` without any arguments closes the previously-open output file and directs output back to the terminal.

`solve` solve the finite-element system.

`quit` exit the program.

`reactions` print out the computed reactions for the problem, either to the screen or to a file previously specified with the `outfile` command.

`readfile`, *filename* read commands from the file *filename*. The commands are processed just as if they were input on the command-line one by one. Note that currently you cannot have a `readfile` command within a file you are reading.

`real`, *real-property-set* select which set of real properties is now the current set. *real-property-set* can be any integer number. Note that the active real property will be attached to any elements which are defined (`e` command).

`redraw` redraw the current graphics display.

`rprint` print out all real property sets and the values of the real properties associated with them.

`rprop`, *real-item*, *property-value* set the value of *real-item* to *property-value* for the current real property set. Valid *real-item* labels include `thickness`, `Izz`, `area`, etc., depending on the element used.

# Appendix B

# New MIMES Fluid Commands and Variables

The integration of fluid-force loading into the MIMES system has required the inclusion of several new commands and variables into the MIMES *Tcl/Tk* shell. This section details the new commands and describes the new environment variables.

The command name is listed in `typewriter` font and the parameters for the commands are listed in *emphasized* font. Note that these are now commands that the MIMES *Tcl/Tk* shell understands so they can be used in conjunction with other MIMES commands and script files.

## B.1 New MIMES Shell Commands

These are the new commands understood by the MIMES shell.

`add_fluid_mesh_ui` *x1 y1 x2 y2 x3 y3 x4 y4 ndivx ndivy id* define a domain for the fluid problem. The domain is given by the coordinates (*x1*, *y1*) ...(*x4*, *y4*), which must be in *counterclockwise* order. The region is meshed into *ndivx* divisions in the $x$-direction and *ndivy* divisions in the $y$-direction. *id* is a unique identification number for the region (which must be unique among all objects in the MIMES system).

`add_fluid_bc_ui` *x1 y1 x2 y2 val* define a boundary condition for the fluid problem. Along the line segment from (*x1*, *y1*) to (*x2*, *y2*), the pressure is constrained to *val*.

`change_fluid_id_ui` *oldid newid* a procedure used internally in MIMES; analogous to the `change_id` procedure, but for fluid-mesh items. A fluid-mesh with id *oldid* has its id changed to *newid*.

`clear_fluid_bcs_ui` removes all of the boundary conditions from the fluid-mesh database.

## B.2 New MIMES Shell Variables

Below are the descriptions of the fluid-force shell "flag" variables which control some features of the MIMES fluid-force implementation. They can be set using the standard *Tcl/Tk* command `set`, or their values can be altered using the MIMES menu system.

`FLUIDS_ON` turns fluid force calculations on or off. If off, no fluid-force loading or computations are added into the MIMES simluation.

`FIRST_STEP_STATIC` tells the MIMES fluid-force system to use the static solution $\phi = \mathbf{K}^{-1}\mathbf{q}$ as its first-step answer to the dynamic fluid problem. This is useful in solving problems with already-existing pressure fields. Otherwise the first-step solution is only the boundary conditions to the fluid problem.

`FLUID_PERM` tells the MIMES fluid-force system to allow the discrete elements to modify the fluids' finite-element permeability on a (finite) element-by-element basis. Otherwise no each finite-element is assigned the same permeability.

`FLUID_PERM_HIGH` the highest value allowed for a fluid-element permeability. Calculated values are clamped to be no greater than the value of this variable.

`FLUID_PERM_LOW` the lowest value allowed for a fluid-element permeability. Calculated values are clamped to be no lower than the value of this variable.

`FLUID_GEOMETRY_FILENAME` the name of the file to which the nodal geometry of the fluid mesh is to be outputted. The file is a *MATLAB*-compatible text file and can be read in and processed using *MATLAB*.

`FLUID_GEOMETRY_FLAG` if set, the fluid geometry will be output to the file specified by `FLUID_GEOMETRY_FILENAME`. Otherwise no geometry output is generated.

`FLUID_PRESSURES_FILENAME` the name of the file to which the time-variant fluid pressures are to be outputted. The file is a *MATLAB*-compatible text file and can be read in and processed using *MATALB*.

`FLUID_PRESSURES_FLAG` if set, the time-variant fluid pressures will be output to the file psecified by `FLUID_PRESSURES_FILENAME`. Otherwise no pressure output is generated.

`FLUID_SOLVE_TIMESTEPS` indicates the number of discrete-element timesteps which should be evaluated before a fluid solution is generated. If set to 1, a fluid solution is generated for *each* timestep; if set to 2, a fluid solution is generated for every other timestep, etc. For very small discrete-element timesteps, the fluid solution may not change significantly; this paramater allows you to speed the solution of the system.

# Appendix C

# MIMES Model and Solution Files

This appendix presents the MIMES model and solution files that were used for the projects detailed in this thesis. The files are heavily commented to so the files really do not need any line-by-line explanation.

Familiarity with *Tcl/Tk* and the MIMES system is assumed–this section is not intended to be a rewrite of the MIMES User Manual (in Rege, 1996). However, some study of these files may be instructional for their use of the fluid-dynamics capabilities in MIMES.

## C.1 Problem File: Flow Underneath a Dam

This is the file `dam-flow.tcl`, which generated the problem setup and solution for the problem discussed in Section 4.1. It is a very simple test file which runs only one timestep (to get the static solution for the fluid mesh) and outputs its data in the EXODUS format.

```
#
#  dam-flow.tcl:  verification problem for a sample dam flow simulation
#
#     source dam-flow.tcl to get this up and running.
#
#
#        Justin T. Klosek <jtklosek@iesl.mit.edu>   19 Nov 1996
#
##############################################################################

set_scales_ui 0 0 600 600

##############################################################################
#
#-- add a fluid mesh and some boundary conditions.

add_fluid_mesh_ui 50 50 250 50 250 300 50 300 \
4 4 1000
add_fluid_mesh_ui 250 50 350 50 350 100 250 100 \
4 4 1001
add_fluid_mesh_ui 350 50 480 50 480 300 350 300 \
4 4 1002

add_fluid_bc_ui 50 300 250 300 \
40
add_fluid_bc_ui 350 300 480 300 \
```

```
0

#############################################################################
#
#-- create the "dam" or channel for particles to flow though.

add_rectangle_ui 30 300 50 300 50 30 30 30 \
100 1
set_velocity 1 1 1 0 0 0 100

add_rectangle_ui 30 50 500 50 500 30 30 30 \
101 1
set_velocity 1 1 1 0 0 0 101

add_rectangle_ui 480 30 480 300 500 300 500 30 \
102 1
set_velocity 1 1 1 0 0 0 102

add_rectangle_ui 250 100 250 310 350 310 350 100 \
103 1
set_velocity 1 1 1 0 0 0 103


#############################################################################
#
#-- force MIMES to create a fluid solution; EXODUS output also.

set FLUIDS_ON 1
set FIRST_STEP_STATIC 1
set PRINT_INT 5
set MAX_DT_CR 0.001
set TIMESTEPS 1
set FLUID_PERM 0
set FLUID_SOLVE_TIMESTEPS 2000
set FLUID_PERM_LOW 0.001
set FLUID_PERM_HIGH 5.0
set FLUID_DENSITY 0.1
set EXODUSFLAG 1
set EXODUSOUTPUTSTEP 1
set EXODUSWRITEFILE dam-flow.exo
apply_mimes_params
```

## C.2 Problem File: Particle Buoyancy

This is the file buoyancy.tcl, which generated the problem setup and solution for the problem discussed in Section 4.2. The density of the free-falling particles was altered for particles to be more dense or less dense than the standing fluid; otherwise the setup file is exactly the same for all experiments discussed.

```
#
#   buoyancy.tcl  --  verification problem for a particle buoyancy
#
#    source buoyancy.tcl to get this up and running.
#
#      Justin T. Klosek   <jtklosek@iesl.mit.edu>  19 Nov 1996
#
#############################################################################
#
#-- set up the material; change the density here as required.
#    also add the free-falling particle

set matnum 2
global E(2)
```

```
global nu(2)
global density(2)

create_material_list_ui 2
set E(1) 200
set nu(1) 0.3
set density(1) 1.5
set E(2) 200
set nu(2) 0.3
set density(2) 0.001
apply_material_list 2

set fric(1,1) 0.1
set fric(2,1) 0.1
set fric(2,2) 0.1
apply_fric 2

set k_normal(1,1) 400000
set k_normal(2,1) 400000
set k_normal(2,2) 400000
apply_kn 2

set k_shear(1,1) 600000
set k_shear(2,1) 600000
set k_shear(2,2) 600000
apply_ks 2

add_superquad_ui 131 396 250 321 \
42 2 2 0 1 16 0 0

##########################################################################
#
#-- add a fluid mesh and some boundary conditions

add_fluid_mesh_ui 100 100 300 100 300 300 100 300 \
4 4 10

add_fluid_bc_ui 100 100 300 100 \
200
add_fluid_bc_ui 100 300 300 300 \
0

##########################################################################
#
#-- put the fluid in a box

add_rectangle_ui 100 80 80 80 80 300 100 300 187 2
add_rectangle_ui 100 80 100 100 320 100 320 80 188 2
add_rectangle_ui 300 100 300 300 320 300 320 100 189 2

set_velocity 1  1  1 0 0  0 187
set_velocity 1  1  1 0 0  0 188
set_velocity 1  1  1 0 0  0 189

##########################################################################
#
#-- set up MINES for a fluid solution

set GRAVITY_Y -9.81
set FLUIDS_ON 1
set PRINT_INT 10
set TIMESTEPS 40000
set RAYLEIGH_FACTOR 1.0
set MAX_DT_CR 0.005
set FIRST_STEP_STATIC 1
set FLUID_SOLVE_TIMESTEPS 10000
set FLUID_PERM 0
set FLUID_GEOMETRY_FLAG 0
```

```
set FLUID_PRESSURES_FLAG 0
set FLUID_PERM_LOW 0.0001
set FLUID_DENSITY 1
set FLUID_PERM_HIGH 4.0
set EXODUSWRITEFILE buoyancy.exo
set EXODUSFLAG 0
set EXODUSOUTPUTSTEP 10
apply_mimes_params
```

## C.3 Problem File: Fluid Pressure Wave

This is the file `prop.tcl`, which generated the problem setup and solution for the problem discussed in Section 4.3. The number of finite elements in the mesh is varied by altering the `add_fluid_mesh_ui` statement. The output from the solution is dumped in the flat text files `geometry.flu` and `pressures.flu`, which were processed through various shell commands and graphed using *MATLAB*.

```
#
#  prop.tcl:  verification problem to show fluid pressure wave
#
#     source prop.tcl to get this up and running.
#
#
#        Justin T. Klosek <jtklosek@iesl.mit.edu>   20 Nov 1996
#
#################################################################################
#
#-- add a fluid mesh and some boundary conditions.

set_scales_ui 0 0 150.1 150.1

add_fluid_mesh_ui 1 1 150 1 150 150 1 150 \
1 16 1000


#################################################################################
#
#-- this particle is required to force MIMES to run the experiment.
#    (if there are no particles, it won't execute)

add_rectangle_ui 0 0 0 150 1 150 1 0 \
100 1
set_velocity 1 1 1 0 0 0 100


#################################################################################
#
#-- force MIMES to create a fluid solution; EXODUS output also.

set FLUIDS_ON 1
set FIRST_STEP_STATIC 0
set PRINT_INT 1
set MAX_DT_CR 0.01
set TIMESTEPS 80000
set FLUID_PERM 0
set FLUID_SOLVE_TIMESTEPS 1
set FLUID_PERM_LOW 1.0
set FLUID_PERM_HIGH 1.0
set FLUID_DENSITY 1.0
set EXODUSFLAG 0
set EXODUSOUTPUTSTEP 1
set EXODUSWRITEFILE prop.exo
set FLUID_PRESSURES_FLAG 1
```

```
set FLUID_GEOMETRY_FLAG 1
apply_mimes_params
```

# Bibliography


[1] Arulanaandan, Kandiah, and Scott, Roland F., eds. *Verification of Numerical Procedures for the Analysis of Soil Liquefaction Problems*, vols. 1 and 2. Rotterdam, Netherlands: A. A. Balkema, 1993.

[2] Balasubramaniam, A.S., et. al., eds. *Developments in Geotechnical Engineering, from Harvard to New Delhi 1936-1994*. Rotterdam, Netherlands: A. A. Balkema, 1994.

[3] Bathe, Klaus-Jurgen. *Finite Element Procedures*. Englewood Cliffs, NJ: Prentice-Hall, 1996.

[4] Beer, Ferdinand P., and Johnston Jr., E. Russell. *Mechanics of Materials*, 2d ed. New York: McGraw-Hill, 1992.

[5] Brebbia, C.A., and Chaudouet-Miranda, A., eds. *Boundary Elements in Mechanical and Electrical Engineering*. Boston: Computational Mechanics Publications, 1990.

[6] Brebbia, C.A., and Ferrante, A.J. *Computational Methods for the Solution of Engineering Problems*. London: Pentech Press, 1978.

[7] Brebbia, C.A., and Rencis, J.J., eds. *Boundary Elements XV: Vol. 1, Fluid Flow and Computational Aspects*. Boston: Computational Mechanics Publications, 1993.

[8] Budak, B.M., and Fomin, S.V. *Multiple Integrals, Field Theory, and Series*. Moscow: Mir Publishers, 1973.

[9] Cakmak, A.S., and Herrera, I., eds. *Soil Dynamics and Liquefaction*. Proceedings of the fourth International Conference on Soil Dynamics and Earthquake Engineering, Mexico City, Mexico, October 1989. Boston: Computational Mechanics Publications, 1989.

[10] Cakmak, A.S., Abdel-Ghaffar, A.M., and Brebbia, C.A., eds. *Soil Dynamics and Earthquake Engineering*. Proceedings of the Conference on Soil Dynamics and Earthquake Engineering, Southampton, 13-15 July 1982. Rotterdam, Netherlands: A. A. Balkema, 1982.

[11] Cakmak, A.S., ed. *Soil Dynamics and Liquefaction.* Amsterdam: Elsevier, 1987.

[12] Chapra, S.C., and Canale, R.P. *Numerical Methods for Engineers*, 2d ed. McGraw-Hill, 1988.

[13] Chaudrhy, M.H., and Mays, L.W., eds. *Computer Modeling of Free-Surface and Pressurized Flows.* Boston: Kluwer Academic Publishers, 1993.

[14] Chopra, A.K. *Dynamics of Structures.* Prentice-Hall, 1995.

[15] Chung, T.J. *Finite Element Analysis in Fluid Dynamics.* McGraw-Hill, 1978.

[16] Clough, J., and Penzien, R.W. *Dynamics of Structures*, 2d ed. McGraw-Hill, 1993.

[17] Cochin, I. *Analysis and Design of Dynamic Systems.* Harper and Row, 1980.

[18] Cormen, Thomas H., et. al. *Introduction to Algorithms.* Cambridge, MA: MIT Press, 1990.

[19] Crisfield, M. A. *Non-linear Finite Element Analysis of Solids and Structures.* Wiley, 1991.

[20] Das, Braja M. *Fundamentals of Soil Dynamics.* New York: Elsevier, 1983.

[21] Das, Braja M. *Principles of Geotechnical Engineering*, 2d ed. Boston: PWS-Kent Publishing, 1990.

[22] Daugherty, Robert L., Franzini, Joseph B., and Finnemore, E. John. *Fluid Mechanics with Engineering Applications*, 8th ed. New York: McGraw-Hill, 1985.

[23] Desai, Chandrakant S., and Christian, John T., eds. *Numerical Methods in Geotechnical Engineering.* New York: McGraw-Hill, 1977.

[24] Doolen, G.D. *Lattice Gas Methods.* Cambridge, MA: MIT Press, 1991.

[25] Dungar, R., Pande, G.N., and Studer, J.A., eds. *Numerical Models in Geomechanics.* International Symposium on Numerical Models in Geomechanics, Zurich, 13-17 September 1982. Rotterdam, Netherlands: A. A. Balkema, 1982.

[26] Dvorkin, J., Mavko, G., and Nur, A. "The Effect of Cementation on the Elastic Properites of Granular Material." *Mechanics of Materials* 12 (1991) 207-217.

[27] Eberly, David. *MAGIC: An Object-Oriented Library for Image Analysis, v3.07.* Internal publication, Computer Science Department, University of North Carolina, Chapel Hill, NC, 1995.

[28] Foster, Nick, and Metaxas, Dimitri. *Dynamic Fluid Simulations: Waves, Splashing, Vorticity, Boundaries, Buoyancy.* Internal publication, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, 1996.

[29] Gallivan, K.A., et. al. *Parallel Algorithms for Matrix Computations.* Philadelphia: Society for Industrial and Applied Mathematics, 1990.

[30] Günther, Oliver. *Efficient Structures for Geometric Data Management.* Springer-Verlag, 1988.

[31] Humar, J.L. *Dynamics of Structures.* Prentice-Hall, 1990.

[32] Ishibashi, Isao, Agarwal, Tarun K., and Wang, You. *Hybrid Finite Element/Discrete Element Model for Large Deformation Geotechnical Engineering Problems.* Computing in Civil Engineering, proceedings of the First Congress held in conjunction with A/E/C Systems 1994, Washington D.C, 857-864, 1994.

[33] John, J.E., and Haberman, W.L. *Introduction to Fluid Mechanics*, 2d ed. Prentice-Hall, 1980.

[34] Kardestuncer, Hayrettin, ed. *Finite Element Handbook.* New York: McGraw-Hill, 1987.

[35] Kazda, I. *Finite-Element Techniques in Groundwater Flow Studies.* Amsterdam: Elsevier, 1990.

[36] Klosek, Justin T. *A C++ Implementation of Finite-Element Analysis.* Final project for class 2.093, Computer Methods in Dynamics, Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, MA. Taught by Prof. Klaus-Jurgen Bathe, Spring 1996.

[37] Krylov, V.I. *Approximate Calculation of Integrals.* Macmillan, 1962.

[38] Laszlo, Michael J. *Computational Geometry and Computer Graphics in C++.* Prentice-Hall, 1996.

[39] Loustau, John, and Dillon, Meighan. *Linear Geometry with Computer Graphics.* New York: Marcel Dekker, 1993.

[40] Morita, N., et. al. "Realistic Sand-Production Prediction: Numerical Approach." *SPE Production Engineering*, Feb 1989, 15-25.

[41] Morita, N., and Boyd, P.A. "Typical Sand Production Problems: Case Studies and Strategies." Technical report submitted for presentation at the 66th Annual Technical Conference and Exhibition of the Society of Petroleum Engineers, Dallas, TX, October 6-9, 1991.

[42] Neider, Jackie, Davis, Tom, and Woo, Mason. *OpenGL Programming Guide.* Addison-Wesley, 1993.

[43] O'Connor, Ruaidrhi M. *A Distributed Discrete Element Modeling Environment - Algorithms, Implementation and Applications.* Doctoral thesis, Department of Civil Engineering, Massachusetts Institute of Technology, Cambridge, MA, January 1996.

[44] O'Rourke, Joseph O. *Computational Geometry in C.* Cambridge University Press, 1994.

[45] Pande, G.N., and Van Impe, W.F., eds. *Numerical Models in Geomechanics.* Proceedings on the International Symposium on Numerical Models in Geomechanics, Ghent, 31st March-4th April 1986. Redruth, Cornwall, England: M. Jackson and Son, 1986.

[46] Pande, G.N., and Pietruszczak, S., eds. *Numerical Models in Geomechanics.* Proceedings of the Fourth International Symposium on Numerical Models in Geomechanics—NUMOG IV, Swansea, UK, 24-27 August 1992. Rotterdam, Netherlands: A. A. Balkema, 1992.

[47] Penberty Jr., W.L., and Shaughnessy, C.M. *Sand Control.* Richardson, TX: Society of Petroleum Engineers, 1992.

[48] Pinder, G., and Gray, W.G. *Finite Element Simulation in Surface and Subsurface Hydrology.* New York: Academic Press, 1977.

[49] Preparata, Franco P., and Shamos, Michael Ian. *Computational Geometry: An Introduction.* New York: Springer-Verlag, 1985.

[50] Preece, Dale S., et. al. *Computer Simulation of Rock Blasting: A Summary of Work from 1987 through 1993.* Technical Report SAND92-1027, Sandia National Laboratories, Albuquerque, NM, 1993.

[51] Press, William H., et. al. *Numerical Recipes in C*, 2d. ed. Cambridge: Cambridge University Press, 1995.

[52] Phillips, O.M. *Flows and Reactions in Permeable Rocks.* Cambridge: Cambridge University Press, 1991.

[53] Pironneau, O. *Finite Element Methods for Fluids.* New York: John Wiley and Sons, 1989.

[54] Popescu, Radu. *Stochastic Variability of Soil Properties: Data Analysis, Digital Simulation, Effects on System Behavior.* Doctoral Thesis, Department of Civil Engineering and Operations Research, Princeton University, Princeton, NJ, November 1995.

[55] Ramming, H.G., and Kowalik, Z. *Numerical Modelling of Marine Hydrodynamics.* Amsterdam: Elsevier, 1980.

[56] Rege, Nabha V. *Computational Modeling of Granular Materials.* Doctoral thesis, Department of Civil Engineering, Massachusetts Institute of Technology, Cambridge, Massachusetts, February 1996.

[57] Rogers, David F., and Adams, J. Alan. *Mathematical Elements for Computer Graphics,* 2d. ed. McGraw-Hill, 1990.

[58] Sedgewick, Robert. *Algorithms in C++.* New York: Addison-Wesley, 1992.

[59] Segerlind, Larry J. *Applied Finite Element Procedures*, 2d. ed. New York: John Wiley and Sons, 1984.

[60] Tangora, Martin C., ed. *Computers in Geometry and Topology.* New York: Marcel Dekker, 1989.

[61] Taylor, C., Johnson J.A., and Smith, W.R., eds. *Computational Techniques for Fluid Flow.* Swansea, U.K.: Pineridge Press, 1986.

[62] Torczynski, John R. *A Darcy-Flow Model of Fluid Force on Sand Grains for Discrete Element Method Codes.* Internal publication, Sandia National Laboratories, Albuquerque, New Mexico, June 21, 1996.

[63] Torczynski, John R. *Pressure Gradients in the Vicinity of a Perforation During Flow.* Internal publication, Sandia National Laboratories, Albuquerque, New Mexico, May 29, 1996.

[64] Tronvoll, J., and Fjaer, E. "Experimental Study of Sand Production from Perforation Cavities." *International Journal of Rock Mechanics, Mineral Sciences, and Geomechanics Abstracts*, vol. 31, no. 5 (1994), 393-410.

[65] Williams, John R., Hocking G., and Mustoe, G.W. *The Theoretical Basis of the Discrete-Element Method.* in Proceedings of the 1985 Conference on Numerical Methods in Engineering, Theory and Application, (897-906), 1985.

[66] Williams, John R., and Mustoe, Graham G. W., eds. *Proceedings of the 2d International Conference on Discrete Element Methods (DEM).* Intelligent Systems Engineering Laboratory, Department of Civil Engineering, Massachusetts Institute of Technology, 1993.

[67] Williams, John R., and Rege, Nabha. *The Development of Circulation Cell Structures in Granular Materials Undergoing Compression.* Technical Report, Intelligent Engineering Systems Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1996.

[68] Zhang, R., et. al. *Simulation of Hydraulic Phenomena using the Discrete Element Method.* conference paper in Williams and Mustoe (1993).