12-1993

Rendering of Three-Dimensional Data Sets Derived from Finite-Difference and Spectral Methods

Paul A. Schubert

AFIT/GAE/ENY/93D-24

RENDERING OF THREE-DIMENSIONAL DATA SETS DERIVED FROM FINITE-DIFFERENCE AND SPECTRAL METHODS

THESIS

Paul A. Schubert, Major, USAF

AFIT/GAE/ENY/93D-24

Approved for public release; distribution unlimited

AFIT/GAE/ENY/93D-24

RENDERING OF THREE-DIMENSIONAL DATA SETS DERIVED FROM FINITE-DIFFERENCE AND SPECTRAL METHODS

THESIS

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Aeronautical Engineering

Paul A. Schubert

Major, USAF

December 1993

Preface

The focus of this work is timely visualization of three-dimensional data sets derived from computational fluid dynamics. The rendering algorithm uses an octree to efficiently traverse the data set in rendering an isosurface and is applicable to finite-difference as well as spectral method data sets. The other thrust of this research is investigating the advantages (if any) of rendering a spectral method solution versus a finite-difference one. As a result, majority of the results is centered about steady-state solutions to a model diffusion-convection problem which involves a boundary layer. In addition, limited results for a driven cavity type problem is provided.

I wish to express my love and gratitude to my wife, Rita, and to our children, Alyxandria, Jaime, and Samantha, for their support during our time here - especially for the past four months in which I thought this thesis effort would never end; my general thanks to the school's instructors and my advisor, Phil Beran, for a quality education; and special thanks to Phil Amburn for introducing me to the computer graphics world - without which I would not have enjoyed my stay as nearly as much.

Table Of Contents

List of Figures

List of Tables

Abstract

The timely visualization of three-dimensional data sets and the advantages of using a spectral method solution versus a finite-difference method solution in rendering isosurfaces is described. The Beam-Warming numerical algorithm, which uses implicit-approximate-factorization, is used to generate the steady-state solutions for a model diffusion-convection problem. The Chebyshev collocation operator is used to evaluate the right-hand side of the Beam-Warming algorithm for the spectral solution. Comparing the model problem results with the exact solution, the spectral series solution is truncated to the same degree of accuracy as the finite-difference for comparison of rendering times. The rendering algorithm employs octrees to efficiently traverse the data set to fit the isosurfaces. The actual fitting of polygons to the isosurface uses the marching cubes table look up algorithm. With the spectral series solution, interval math is investigated for guaranteed detection of isosurfaces during the initial octree traversal(s).

RENDERING OF THREE-DIMENSIONAL DATA SETS DERIVED FROM FINITE-DIFFERENCE AND SPECTRAL METHODS

I. Introduction

The basic problem addressed in this thesis is the timely visualization of the results of Computational Fluid Dynamics (CFD) analyses. Specifically, this research investigates the rendering advantages offered by the use of spectral methods (SMs) over finite-difference methods (FDMs) when solutions are rendered as isosurfaces. Three-dimensional (3D) data sets are generated for a model convection-diffusion problem using a traditional FDM and Chebyshev collocation SM. The Beam-Warming algorithm, an implicit, approximate-factorization procedure, is used to generate the steady-state, finite-difference solutions and is modified to generate the spectral solutions. The Chebyshev collocation operator is used to evaluate the right-hand side of the Beam-Warming algorithm to provide the spectral solution at steady-state. Comparing the model problem results with exact solution, the spectral-series solution is truncated to the same degree of accuracy as the finite-difference for comparison of rendering times. The rendering algorithm employs a linear octree to efficiently traverse the data set to fit the isosurfaces. With the spectral-series solution, interval mathematics is investigated for detecting isosurfaces during the initial octree traversal. Limited implementation of the conventional Beam-Warming algorithm to a driven cavity is available. Rendering of the FDM driven cavity over a uniform grid is provided.

1.1. Motivation

Current constraining factors in rendering 3D FDM data sets are the size of the data set and the spacing of the grid nodes. For a 3D, a 100 x 100 x 100 grid (1 million nodes) is not uncommon. Typically, the grid nodes are packed closer together in regions of large flow gradients, e.g., the boundary layer, wakes, etc. Along with the transformation of the physical domain to a uniform rectangular computational domain, this results in a non-uniform distribution of nodes. Such grids pose a serious challenge to any type of volume rendering of the data (Kerlick, 1990:2).

> To be useful, volume visualization techniques must offer understandable data representations, quick data manipulations, and reasonably fast rendering. Scientific users should be able to change parameters and see the resultant image instantly. Few present day systems are capable of this type of performance. (Elvins, 1992:194)

However, another class of CFD algorithm -- SMs -- may be able to bypass these rendering factors. SMs differ from FDMs, since they provide a closed-form, truncated-series solution. Typical series bases are the Fourier series or the Chebyshev polynomials. The main advantage offered by a SM for visualization is the ability to evaluate the closed-form solution where needed in computing an isosurface. The series solution can also be highly truncated when the accuracy level required for volume visualization is much lower than for the computation of the solution. Another advantage of SMs is the ability to use interval mathematics in evaluating the solution over a grid cell. Interval methods are guaranteed not to miss parts of contours or isosurfaces down to a specified size in the viewing region; point sampling of gridded data sets cannot guarantee this (Suffern and Fackerell, 1991:331).

In addition to rendering advantages, SMs in general require fewer degrees of freedom than FDMs (i.e. less nodes). SMs typically offer increased accuracy to many orders of magnitude over a FDM for the same number of grid points. Although a FDM is

faster than a spectral method which uses the same number of points, this advantage is negated when the FDM uses many more grid points to produce a comparably accurate solution (Canuto, et al., 1987:27). The final advantage is the compactness of data sets for comparably accurate solutions. In a SM series solution, any individual series term is on the order of the term's coefficient. Then the SM data set can be filtered to drop those coefficients less than some threshold, i.e. 0.001. For example, when the model problem SM solutions are truncated to the same order of accuracy as the FDM solutions, the ratio of file sizes (SM:FDM) is roughly 5:7000 or about 0.01 percent.

1.2. Background and Prior Work

Presented is the background and prior work on the numerical and rendering algorithms used. Since the intent of this work was to render a 3D FDM versus a 3D SM data set, the Beam-Warming numerical algorithm was chosen for its relative ease of implementation in 3D and ease of modification to a Chebyshev collocation method. FDM's basics, the Beam-Warming algorithm, and extending the algorithm to SMs are discussed. The rendering of 3D data sets is done through either a direct volume rendering or a through a surface fitting algorithm. A surface fitting method was chosen for the rendering method since these methods are faster than direct volume rendering methods. To guarantee surface detection during the initial traversal of the data set, interval mathematics is used.

1.2.1. Finite-Difference Basics

Finite difference methods provide numerical solutions of problems in fluid mechanics based upon direct approximation of the governing equation(s). For example, the second-order-accurate, central-difference operators derived from a Taylor's series expansion are as follows:

$$\frac{\partial u}{\partial x} = \frac{u_{i+1,j} - u_{i-1,j}}{\Delta x} + O[\Delta x^2] = \delta_x u + O[\Delta x^2], \tag{1.1}$$

$$\frac{\partial^2 u}{\partial x^2} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} + O[\Delta x^2] = \delta_{xx} u + O[\Delta x^2], \tag{1.2}$$

where δ_x, δ_{xx} are the finite-difference operators. The 3 point stencil for Eq 1.2 is shown in Figure 1.1. The range of influence is local and encompasses ± 1 additional nodes. As will be seen, the SM's derivative operators have global influence and are applied over the whole domain.

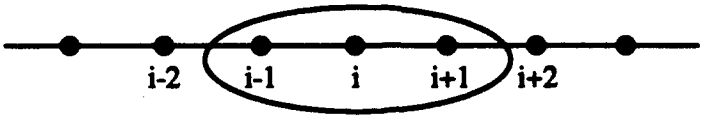

Figure 1.1. FDM Three Point Stencil in 1D

1.2.2. Beam-Warming Numerical Algorithm

Warming and Beam (1978:85-129) introduced an implicit approximate factorization algorithm for a system of conservation laws, such as the compressible Navier-Stokes equations. The algorithm is expressed in the "delta form" (i.e. solves for increments of the conserved variable) and can be either first- or second-order accurate in time. If used to march to steady state, the accuracy of the solution is governed by the accuracy of the derivative operators and boundary conditions applied to the right-hand side of the algorithm. The algorithm uses the Euler and viscous flux Jacobians to linearize the Navier-Stokes equations. The 3D flux Jacobians used were taken from Pulliam and Steger (1980:162). By spatially factoring the finite-difference operator on the left-hand side and by neglecting the higher order terms, the discretization of the left-side in 3D is broken down into three 1D system of equations. By doing so, and since the 1D stencil encompasses only 3 nodes, the algorithm produces a block tridiagonal structure which can be easily inverted at each time step. For example, Eqs 1.3 - 1.7 show a generic algorithm

for a 2D, inviscid flow which is factorized into two 1D systems (the nomenclature is discussed further in Chapters 2 and 3).

$$\left[\mathrm{I} + \delta_x A + \delta_y B\right]\Delta^n U = \Re^n \tag{1.3}$$

$$\left[\mathrm{I} + \delta_x A + \delta_y B\right]\Delta^n U = \left\{\left[\mathrm{I} + \delta_x A\right]\left[\mathrm{I} + \delta_y B\right] - \delta_x \delta_y AB\right\}\Delta^n U \tag{1.4}$$

$$\left[\mathrm{I} + \delta_x A + \delta_y B\right]\Delta^n U \approx \left[\mathrm{I} + \delta_x A\right]\left[\mathrm{I} + \delta_y B\right]\Delta^n U \tag{1.5}$$

$$\left[\mathrm{I} + \delta_x A\right]\left[\mathrm{I} + \delta_y B\right]\Delta^n U = \Re^n \tag{1.6}$$

$$\Delta^n U = \left[\mathrm{I} + \delta_y B\right]^{-1}\left[\mathrm{I} + \delta_x A\right]^{-1}\Re^n \tag{1.7}$$

Without approximate factorization, the term $\left[\mathrm{I} + \delta_x A + \delta_y B\right]^{-1}$ becomes computationally intractable for large number of grid nodes and for the extension to 3D.

1.2.3. Spectral Methods

In contrast, SMs differ conceptually from FDMs. Spectral methods belong to the class of weighted residuals methods. Instead of directly approximating the derivatives as is done in FDMs, the solution is first assumed to be the form of a truncated series; i.e., a 1D steady-state solution is assumed to be: $u = \sum_{i=0}^{N} a_i \phi_i(x)$. Here $\phi_i(x)$'s are known as the approximating or "trail" functions and a_i's are the expansion coefficients. Then the derivatives are approximated indirectly by the derivatives of the assumed solution. For example, $u_x = \sum_{i=0}^{N} a_i \frac{\partial \phi_i}{\partial x} = \sum_{i=0}^{N} b_i \phi_i$, where b_i's can be determined from some linear combination of a_i's and from the orthogonal properties of the trail functions, $\phi_i(x)$'s. Then the expansion coefficients are determined by minimizing the weighted residual -- the error in the differential equation produced by using the truncated approximation instead of the exact solution. SMs are distinguished by the choice of trail functions and also by the type of weighting method.

The choice of trail functions is one of the features which distinguishes SMs from other weighted residual methods and FDMs (Canuto, et al., 1987:1). In SMs, the trail functions are defined over the whole computational domain, are infinitely differentiable,

and are orthogonal to each other. The two most common trail functions are the trigonometric and the Chebyshev polynomials. For evaluation of nonlinear and non-constant coefficient terms with non-periodic boundary conditions, Gottlieb and Orszag (1977: 117) recommend using Chebyshev polynomials. The properties of the Chebyshev polynomial expansions are summarized in Appendix C.

In determining the expansion coefficients, different weighting methods can be used. The 3 most common schemes are Galerkin, collocation, and tau. The collocation method is the simplest to implement and is the method used in this effort. In this approach, the expansion coefficients are determined by satisfying the differential equation exactly at the so-called collocation grid nodes (Canuto, et al., 1987:1). The most effective choice for the grid nodes are those corresponding to quadrature formulas of maximum precision; in this effort, the Gauss-Lobatto grid nodes were used (Canuto, et al., 1987:13).

Reddy (1983) discusses the application of pseudo-spectral approximations to the evaluation of terms in the Beam-Warming algorithm. Pseudo-spectral methods are often interchanged with collocation methods. However, the pseudo-spectral evaluation of nonlinear terms is subject to ailiasing errors (Canuto, et al., 1987:84-87). Reddy uses Fourier pseudo-spectral evaluation of the right-hand side of the algorithm and also uses the thin layer Navier-Stokes form of the equations. For the driven cavity problem solved here, the full compressible conservative form of the Navier-Stokes equations are used.

For other work related to approximate factorization schemes, Street, et al., (1985:53-55) uses a relaxation scheme in which the left-hand side terms are evaluated with finite-difference operator and the right-hand side terms are evaluated with a spectral operator. A variation of their scheme is used in this effort.

1.2.4. Rendering of 3D Data Sets.

The majority of volume visualization algorithms are designed to render gridded 3D data sets (i.e., scientific and medical). These volume visualization algorithms fall into two categories: direct volume rendering and surface fitting algorithms (Elvins, 1992:196).

1.2.5. Direct Volume Rendering Methods.

These methods are characterized by the mapping of elements directly into screen space without using geometric primitives (e.g., triangles or quadrilaterals) as an intermediate representation. Instead, each volume element (voxel) or cell contributes color to the final image through a data classification table. For example, a table may map bone density values to white/opaque, muscle density values to red/semi-transparent, and fat density values to beige/mostly transparent colors. Some disadvantages as pointed out by Elvins (1992:196-197) are:

- to get a reasonable image, care must be taken in setting up the classification table -- slight changes in opacity values can often have a large, unexpected impact on the final image;
- non-uniform grids are handled with difficulty.

Specific algorithms include ray casting (Foley, et al., 1990:701), Sabella method (Sabella, 1988; Foley, et al., 1990:1037) and splatting (Westover, 1990). Since direct volume rendering algorithms are more computationally intensive than surface fitting algorithms, surface fitting algorithms are considered for this effort.

1.2.6. Surface Fitting Methods.

With surface fitting algorithms, the 3D data set is traversed once to extract a surface of constant property value or an "isosurface." The isosurface is fitted with geometric primitives (e.g., triangles) and then is rendered by conventional techniques. Surface rendering is usually faster than direct volume rendering since the volume is traversed only once to render the surface. Some disadvantages as pointed out by Elvins (1992:196) are:

- changing the isosurface value can be time consuming since the entire data set has to be revisited for the new surface extraction;
- these algorithms suffer problems such as occasional false surface pieces and incorrect handling of small features in the data set;
- lastly, only a thin surface is modeled as opposed to the entire data set in direct volume rendering.

The class of surface fitting algorithms includes contour connecting, opaque cubes, and marching cubes. In addition, an octree data-structure can be implemented to reduce the surface extraction and re-extraction times (Wilhelms and Van Gelder, 1992:205).

Contour connecting was one of first methods invented for volume visualization (Elvins, 1992:197). In this method, closed contours are traced in a series of slices and then connected to form an isosurface (Keppel, 1975; Fuchs, et al., 1977). The two main problems with connecting the slices are the "tiling problem" (how to best generate the mesh between any two contours) and the "branching problem" (how to best tile between slices which contain a different number of contours) (Meyers, et al., 1992:230). Also contours that lie within another contour (i.e., a cross section of a torus) and contours from two separate surfaces of the same value are handled with difficulty.

In contrast to contour connecting, the various forms of the cubes' algorithm take a slightly different approach in fitting the surface. Instead of dissecting the data set into individual slices and then fitting the geometric primitives to the slices, these algorithms will fit the primitives directly to the data set cells. The oldest cubes' algorithm is the opaque binary cube or cuberille introduced by Herman and Liu (1979). This algorithm proceeds in two steps: first, the volume is traversed to find all the cells which straddle the isosurface value; and second, the cubes are rendered to form a blocky appearance of the isosurface (Elvins, 1992:198).

Lorensen and Cline dramatically improves this with the marching cubes algorithm. Instead of rendering the whole cube, one to four triangles per cube are fitted to the surface

intersection with the cube's edges. The triangulation is done through a table look up covering the 256 possible cases (Lorensen and Cline, 1987:164-5). These 256 cases can be broken down into 14 distinct major cases through a series of rotations and reflections. These 14 cases are shown in Figure 1.2. However, of these 14 major cases, 6 are ambiguous. False triangles may be generated if special handling is not used (Wilhelms and Van Gelder, 1990; Wyvill, et al., 1986; Nielson and Hamann, 1991).

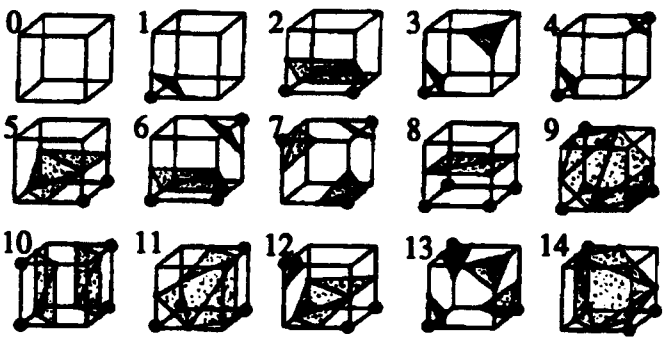

Figure 1.2. Marching Cubes Triangulation Cases (Lorensen and Cline, 1987)

As a means to reduce the surface extraction time, Wilhelms and Van Gelder (1992) introduce branch-on-need octrees to isolate the cells straddling the isosurface value. In a 2D planar case, a quadrant is divided into 4 squares as a start of a quadtree. In 3D, the volume is initially divided into 8 cubes or octants. Then each octant can be further subdivided up into 8 additional octants and so on. Through the use of linear octrees, then any one octant can be mapped into a 3D array containing the surface values (Gargantini, 1981:367). In the algorithm used by Wilhelms and Van Gelder, an octree is generated with its nodes containing minimum and maximum data value found in the node's subtree. In the surface extraction phase, the octree is traversed with a particular surface threshold. Only those branches containing part of the isosurface are visited. If an octree node is encountered with its maximum below or its minimum above the threshold, the node and its children nodes are ignored since the octant does not straddle the isosurface. When a node

at the bottom of the tree (a leaf) is visited, polygons are generated from the cells making up the leaf by using the marching cubes' table look up (Wilhelms and Van Gelder, 1992:206). Since isosurfaces often occupy only 5 - 15% of the total volume, this algorithm provides a means of quickly sifting through the volume. They report surface extraction phase speedups in the range of 1.6 to 11 and overall speedups of 2 - 3 over the regular marching cubes.

An important time-saver noted by Wilhelms and Van Gelder, is the reuse of computed intersection information. Each intersection point requires interpolating to generate a vertex value and the surface normals in the x, y, and z directions. They save this information in a "hash table." When retrieved for the last time, the information is deleted to make room for new vertex information. The branch-on-need portion of their algorithm efficiently allocates memory for viewing data sets which do not conform precisely to a grid dimension of $2^d \times 2^d \times 2^d$, where "d" is the dimension of the octree. However, use of a spectral-series solution negates this requirement since the series can be evaluated exactly at the octree's nodes.

Bloomenthal also used octrees and adaptive subdivision to organize vertex data in Polygonization of implicit functions. However, his octrees were built around one isosurface value known *a priori*.. He found the octree in this use a convenient mechanism for adaptively storing the implicit surface information (Bloomenthal, 1988:354). Also instead of using Lorensen's table-lookup method for polygonizing the cells, he used Wyvill's (1986) method. The advantage Wilhelms and Van Gelder's algorithm has over this is the ability to change the isosurface value and re-extract the resulting surface quicker.

1.2.2. Interval Mathematics

Suffern and Fackerell (1991:331-340) reports on using interval mathematics in conjunction with octrees in rendering isosurfaces of implicit functions. In the above octree algorithm, a surface is detected if any two nodes of a cell straddles the isosurface value.

With an implicit function, the cell's range of values can be used to calculate the natural interval extension of the implicit function. This natural interval extension amounts to the union of possible variations of the function within that cell. The actual function's value is then guaranteed not to lie outside the bounds for this cell. Thus, interval mathematics provides a 100 percent confidence test for rejecting octants not containing the desired isosurface. The main disadvantage of interval mathematics, besides requiring an implicit functional description, is that it requires additional floating-point operations to determine the natural extension. Also if adaptive sub-division is not used, then the usefulness of interval mathematics is constrained by the final plot depth or the final resolution of the octree algorithm (the surface fitting is based upon the cell's actual node values, not its natural interval extension).

1.3. Summary of Results

In Chapter 2, the mathematical formulation of the model problem and the driven cavity is described; Chapter 3 - the numerical algorithms; Chapter 4 - the rendering algorithm; Chapter 5 - the numerical results; and Chapter 6 - the rendering results.

In Chapter 3, the implementation of the Beam-Warming is described for the model problem using the conventional second-order-accurate finite difference operators and spectral Chebyshev collocation operators on the right-hand side of the algorithm. This implementation is for grid nodes located at the Gauss-Lobatto grid points.

In Chapter 4, the modification of the marching cubes algorithm with an octree is described. The application of interval mathematics in conjunction with a Chebyshev series solution and the design of the octree's attempted hash table are provided.

In Chapter 5, a comparison of the finite-difference and spectral accuracy for the model problem is provided. Partial implementation of algorithm to the driven cavity is

available. This partial implementation is for finite-difference operators on a uniformly space grid. Convergence histories of the algorithms are provided. The solutions of the numerical algorithm are used by the rendering algorithm. The spectral data sets containing the expansion coefficients for the Chebyshev truncated series are "filtered" prior to use by the rendering algorithm. To filter the model problem spectral data sets, only those absolute values which exceed a set threshold are retained. The resulting accuracy of the filter data sets are on the order of 10 times the threshold used. For thresholds of 0.001 - 0.00001, typically 80 - 250 expansion coefficients are retained.

In Chapter 6, rendering times for the spectral and finite-difference model problem data sets are compared. The initial time to render the first isosurface is approximate twice as fast for a spectral data set than for a finite-difference data set of comparable accuracy. This difference in timing is solely a function of the relative size of the two data sets. Even though a spectral data set of 86 coefficients renders twice as fast as finite-difference of 33^3 data values, a finite-difference data set of 65^3 data values renders slightly faster a spectral data set of 256 coefficients. However, once the initial isosurface is rendered, the regeneration of isosurfaces for the two methods are equivalent. In successive generation of new isosurfaces, use of octrees to minimize the surface re-extraction time proved to be invaluable. The isosurface build time with octrees are found to be 2 -3 times faster than a generic marching cubes algorithm.

In addition in Chapter 6, the interval mathematics results are provided. In conjunction with a Chebyshev series solution and an octree traversal, interval mathematics were found to have limited usefulness . The interval mathematics are used in calculating an octant's initial minimum and maximum values. Except for the leaves located at the bottom of the octree, a resulting octant range of values is found to be too large for a series solution. As a result, the entire octree is essentially traversed to render an isosurface - at which point, the generic marching cubes algorithm is more effective.

Also in Chapter 6, use of a hash table in conjunction with the octree is provided. The hash table is used to store the interpolated isosurface's vertex and surface normals values for neighboring octant's use. Using the design discussed in Chapter 4, the hash table slowed the octree traversal by a factor of 2 - 3, thus negating the octree's speed advantage.

Lastly in Chapter 6, the quality of the model problem results versus the exact solution and some driven cavity isosurfaces are provided. The model problem isosurfaces degrade significantly when the isosurface value (isovalue) is within 1 error norm of the data set's maximum and minimum value. The maximum error norm of a data set is calculated by comparison with the exact solution. Driven cavity isosurfaces are provided for the pressure field and the velocity magnitude field within the cavity

Extrapolating the model problem results to the rendering of a spectral fluid flow solution is not straight forward. The model problem frequency content inherently is limited. The number of coefficients required for a fluid flow field will undoubtedly be much larger. In this case, the FDM will have lower setup costs. The one advantage, which has not been fully exploited, is the ability to evaluate the SM solution at any arbitrary location in the domain. If the FDM solution is defined over a curvilinear domain (in which the node spacing is a function of two or more coordinate axes), then the rendering of the FDM solutions would require interpolation to avoid "cracks" in the isosurfaces. Then the rendering of FDM cases will significantly increase, while the time required for SM cases would remain relatively constant. This area warrants further investigation.

II. Mathematical Formulation

Two types of problems are examined in attempting to obtain spectral and finite-difference data sets. The first, a model problem, has an exact solution which allows comparison of the SM and FDM absolute accuracy. The second, the driven cavity, was chosen as a generic fluid flow and for it's relative ease of implementation over the domain $\Omega = [-1, 1]^3$. The governing equations and assumptions used for the model problem and the driven cavity are presented herein. Figure 2.1 shows the orientation of the domain and the cavity's lid boundary condition.

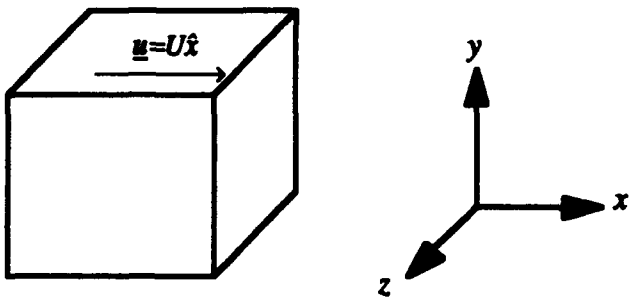

Figure 2.1. Model Problem and Driven Cavity Orientation

2.1. Model Problem

To mimic the form of the Navier-Stokes equations, the following convection-diffusion equation was solved numerically for the model problem:

$$f(f_x + f_y + f_z) = \frac{1}{Re}(f_{xx} + f_{yy} + f_{zz}) + g(x, y, z), \tag{2.1}$$

where $Re = 100$. Using Dirichlet boundary conditions, $g(x, y, z)$ and boundary data are chosen so that the exact solution is

$$f(x,y,z) = \tanh\left(\lambda(1-x^2)\right)\sin\left(\frac{\pi}{2}(1+y^2)\right)\sin\left(\frac{\pi}{2}(1+z^2)\right) \qquad (2.2)$$

The hyperbolic tangent function was chosen in order to mimic a boundary-layer in the x direction. The boundary-layer thickness is then controlled by λ, as listed in Table 2.1 and as shown in Figure 2.2.

Table 2.1. Boundary-Layer Thickness for Model Problem

λ	Boundary-Layer Thickness
2	0.75
5	0.31
8	0.18
10	0.14

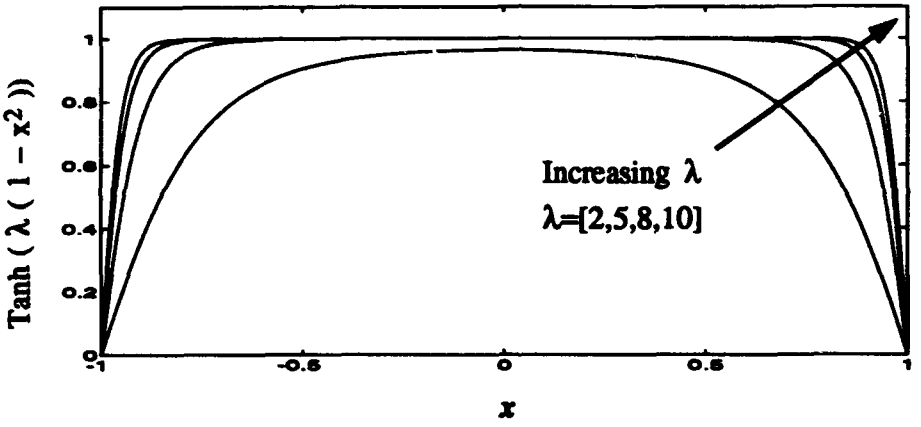

Figure 2.2. Hyperbolic Tangent for Model Problem

2.2. Driven Cavity

For the driven cavity problem, the assumptions, the governing equations, and the boundary conditions used in deriving the steady-state solution are provided. The steady-state solution is determined by time-integrating the 3D, unsteady, Navier-Stokes equations. The fluid medium is assumed to be compressible, thereby allowing the

time-integration to be carried out with the efficient Beam-Warming algorithm. The conservative form of the equations is chosen to allow for implicit treatment of the pressure term. The driven cavity is defined over the domain Ω, as discussed above and is modeled subject to the following assumptions:

- thermally and calorically perfect gas with the specific gas constant, $\gamma = 1.4$
- newtonian fluid
- constant specific heats, c_p and c_v
- the lid moves with a constant velocity, U
- laminar flow with Reynolds number, $Re = 100$
- constant wall temperature, $T_{wall} = 500K$
- subsonic flow with reference Mach number, $M_{ref} = 0.3$
- constant Prandtl number, $Pr = 0.71$
- no-slip boundary condition at wall
- normal derivative of pressure at the wall, $\frac{\partial p}{\partial \eta} = 0$, where η is the normal direction to the wall

The reference Mach number and Reynolds number were chosen in order to improve the convergence rate of the Beam-Warming algorithm. The constant specific heats, gas constant, and Prandtl number are standard assumptions for the wall temperature given.

2.2.1. Navier-Stokes Equations in Conservative Form.

The governing equations are expressed in vector form as (Pulliam and Steger, 1980:159-160)

$$\frac{\partial U}{\partial t} + \frac{\partial E}{\partial x} + \frac{\partial F}{\partial y} + \frac{\partial G}{\partial z} = \frac{\partial E_v}{\partial x} + \frac{\partial F_v}{\partial y} + \frac{\partial G_v}{\partial z}, \tag{2.3}$$

where the state vector of conserved variables, U, and the Euler fluxes, E, F, G, are defined as

$$U = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ E_t \end{bmatrix}, \tag{2.4}$$

$$E = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ (E_t + p)u \end{bmatrix} \quad F = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vw \\ (E_t + p)v \end{bmatrix} \quad G = \begin{bmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + p \\ (E_t + p)w \end{bmatrix}, \tag{2.5a,b,c}$$

The viscous fluxes, E_v, F_v, G_v, are defined as

$$E_v = \begin{bmatrix} 0 \\ \tau_{xx} \\ \tau_{yx} \\ \tau_{zx} \\ \beta_x \end{bmatrix} \quad F_v = \begin{bmatrix} 0 \\ \tau_{xy} \\ \tau_{yy} \\ \tau_{zy} \\ \beta_y \end{bmatrix} \quad G_v = \begin{bmatrix} 0 \\ \tau_{xz} \\ \tau_{yz} \\ \tau_{zz} \\ \beta_z \end{bmatrix}, \tag{2.6a,b,c}$$

Here, ρ is the density; u, v, w are the Cartesian velocity components; p is the pressure; T is the temperature; E_t is the total energy; e is the internal energy; k is the thermal conductivity coefficient; τ_{ij} are the shear stress components; and β_i are the energy components related to the shear stresses and internal energy. The following relationships, derived from the previously mentioned assumptions, relate the above variables appropriately:

$$p = (\gamma - 1)e, \quad T = \frac{e}{c_v} = \frac{p\gamma M_\infty^2}{\rho}, \tag{2.7a,b}$$

$$E_t = \rho e + \tfrac{1}{2}\rho(u^2 + v^2 + w^2), \tag{2.8}$$

$$\tau_{ii} = -\tfrac{2}{3}\mu(u_x + v_y + w_z) + 2\mu\frac{\partial u_i}{\partial x_i}, \quad \tau_{ij} = \tau_{ji} = \mu\left(\frac{\partial u_j}{\partial x_i} + \frac{\partial u_i}{\partial x_j}\right), \tag{2.9a,b}$$

$$\beta_i = \frac{k}{c_v}\frac{\partial e}{\partial x_i} + u\tau_{ix} + v\tau_{iy} + w\tau_{iz}. \tag{2.10}$$

In Eq 2.3, the derivative of viscous fluxes are used ($\frac{\partial E_v}{\partial x}, \frac{\partial F_v}{\partial y}, \frac{\partial F_v}{\partial z}$) which results in cross-derivative terms. In the algorithm, the cross-derivatives terms are handled explicitly and are grouped together. In non-dimensional form, these are as follows:

$$E_{v_1} = \mu \begin{bmatrix} 0 \\ a_2 u_x \\ a_1 v_x \\ a_1 w_x \\ \{a_4 T_x + a_2 u u_x \\ \quad + a_1 (v v_x + w w_x)\} \end{bmatrix} \qquad E_{v_{23}} = \mu \begin{bmatrix} 0 \\ a_3 (v_y + w_z) \\ a_1 u_y \\ a_1 u_z \\ \{a_3 u (v_y + w_z) \\ \quad + a_1 (v u_y + w u_z)\} \end{bmatrix} \qquad (2.11a,b)$$

$$F_{v_2} = \mu \begin{bmatrix} 0 \\ a_1 u_y \\ a_2 v_y \\ a_1 w_y \\ \{a_4 T_y + a_2 v v_y \\ \quad + a_1 (u u_y + w w_y)\} \end{bmatrix} \qquad F_{v_{13}} = \mu \begin{bmatrix} 0 \\ a_1 v_x \\ a_3 (u_x + w_z) \\ a_1 v_z \\ \{a_3 v (u_x + w_z) \\ \quad + a_1 (u v_x + w v_z)\} \end{bmatrix} \qquad (2.12a,b)$$

$$G_{v_3} = \mu \begin{bmatrix} 0 \\ a_1 u_z \\ a_1 v_z \\ a_2 w_z \\ \{a_4 T_z + a_2 w w_z \\ \quad + a_1 (u u_z + v v_z)\} \end{bmatrix} \qquad G_{v_{12}} = \mu \begin{bmatrix} 0 \\ a_1 w_x \\ a_1 w_y \\ a_3 (u_x + v_y) \\ \{a_3 w (u_x + v_y) \\ \quad + a_1 (u w_x + v w_y)\} \end{bmatrix} \qquad (2.13a,b)$$

where

$$a_1 = \frac{1}{Re}, \quad a_2 = \frac{4}{3Re}, \quad a_3 = -\frac{2}{3Re}, \quad a_4 = \frac{1}{(\gamma - 1) Re Pr M_{inf}^2}.$$

2.2.2. Boundary Conditions

On the stationary walls, the boundary conditions require all the velocity components to vanish. Density is related to total energy through Eq 2.8. The final boundary conditions is derived from the assumption of the normal derivative of pressure at the wall to be zero. In non-dimensional form these are as follows:

$$\rho u = \rho v = \rho w = 0, \qquad (2.14a,b,c)$$

$$\rho e = E_t \quad \Rightarrow \quad p = \beta_1 E_t, \qquad (2.14d)$$

$$\frac{\partial p}{\partial \eta} = 0 \quad \Rightarrow \quad \frac{\partial E_t}{\partial \eta} = 0, \qquad (2.14e)$$

where $\beta_1 = \gamma M_{ref}^2(\gamma - 1)$.

For the moving lid, Eqs 2.14a, d, e are modified to reflect $u = 1$:

$$\rho u = \rho, \qquad (2.15a)$$

$$\rho v = \rho w = 0, \qquad (2.15b,c)$$

$$E_t = \rho(e + \tfrac{1}{2}u^2) \quad \Rightarrow \quad p = \beta_2 E_t, \qquad (2.15d)$$

$$\frac{\partial p}{\partial \eta} = 0 \quad \Rightarrow \quad \frac{\partial E_t}{\partial \eta} = \frac{\partial \rho u}{\partial \eta} - \tfrac{1}{2}\frac{\partial p}{\partial \eta}, \qquad (2.15e)$$

where $\beta_2 = \left[\frac{1}{2} + \frac{1}{\beta_1}\right]^{-1} = \left[\frac{1}{2} + \frac{1}{\gamma M_{ref}^2(\gamma-1)}\right]^{-1}$.

III. Numerical Algorithm

This chapter provides an overview of the conventional Beam-Warming algorithm, including an analysis of a variant of this algorithm. The variation incorporates the explicit Chebyshev collocation method. The Chebyshev collocation procedure provides a solution of spectral accuracy at steady-state. The algorithm is first developed with the model problem as described in Chapter 3. The partial extension of the algorithm as described here for the driven cavity is implemented for finite-difference on an uniform grid.

The accuracy of the Beam-Warming scheme is dependent on the choice of space and time differencing operators. Both two- and three-time-level schemes may be used with the Beam-Warming algorithm. The general three-level scheme can be written as follows (Warming and Beam, 1978:93):

$$\frac{\partial U}{\partial t} = \frac{1}{\Delta t}\frac{(1+\theta_2)\Delta - \theta_2\nabla}{1+\theta_1\Delta}U^n + O\left[\left(\theta_1 - \theta_2 - \tfrac{1}{2}\right)\Delta t\right] + O\left[\Delta t^2\right] \tag{3.1}$$

where Δ is the forward-difference operator and ∇ is the backward-difference operator:

$$\Delta^n U = U^{n+1} - U^n, \qquad \nabla^n U = U^n - U^{n-1}. \tag{3.2a,b}$$

The type of time integration represented by Eq 3.1 is controlled by θ_1 and θ_2. Table 3.1 lists some of the standard schemes and their accuracy.

Table 3.1. Typical Integration Schemes

Integration Scheme	θ_1	θ_2	Accuracy (temporal)
Trapezoidal Rule	1/2	0	2nd
Euler Implicit	1	0	1st
3-Point Formula	1	1/2	2nd

In deriving the steady-state solutions within this effort, the Euler implicit integration scheme was used.

3.1. Beam and Warming Finite-Difference Method (Conventional)

In delta form, Eq 2.3 is written as

$$\frac{\partial}{\partial t}(\Delta^n U) = \frac{\partial}{\partial x}(\Delta^n E_v - \Delta^n E) + \frac{\partial}{\partial y}(\Delta^n F_v - \Delta^n F) + \frac{\partial}{\partial z}(\Delta^n G_v - \Delta^n G). \qquad (3.3)$$

Through the introduction of the *flux Jacobians*, Eq 3.1 may be written solely in terms of differences of U, $\Delta^n U$. The flux Jacobians are used to linearize the flux differences in Eq 3.1 as follows:

$$\Delta^n E \approx A^n \Delta^n U \quad \Delta^n F \approx B^n \Delta^n U \quad \Delta^n G \approx C^n \Delta^n U, \qquad (3.4a,b,c)$$

where

$$A^n \equiv \left(\frac{\partial E}{\partial U}\right)^n \quad B^n \equiv \left(\frac{\partial F}{\partial U}\right)^n \quad C^n \equiv \left(\frac{\partial G}{\partial U}\right)^n. \qquad (3.5a,b,c)$$

The flux Jacobians are 5 x 5 matrices for the 3D driven cavity problem. The components of all the Jacobians matrices are given in Appendix A. The viscous terms are somewhat more complicated to linearize since they involve derivative terms of the state vector U. The derivation of the term $\Delta^n E_v$ is shown as an example (Beran, 1993:Aero 753).

$$\Delta^n E_v = \Delta^n E_{v_1} + \Delta^n E_{v_{23}}, \qquad (3.6)$$

$$\Delta^n E_{v_1} \approx P^n \Delta^n U + R^n \Delta^n U_x = (P - R_x)^n \Delta^n U + (R^n \Delta^n U)_x, \qquad (3.7)$$

$$\Delta^n E_{v_1} \approx (R^n \Delta^n U)_x, \qquad (3.8)$$

$$\Delta^n E_{v_{23}} \approx \Delta^{n-1} E_{v_{23}}, \qquad (3.9)$$

$$\Delta^n E_v \approx (R^n \Delta^n U)_x + \Delta^{n-1} E_{v_{23}}, \qquad (3.10a)$$

where

$$P^n \equiv \left(\frac{\partial E_{v_1}}{\partial U}\right)^n \quad R^n \equiv \left(\frac{\partial E_{v_1}}{\partial U_x}\right)^n. \qquad (3.11,3.12a)$$

The term $(P - R_x)$ governs the spatial variation of the thermal conductivity and viscosity. Since this term will appear on the left-hand side of the algorithm, it has no effect on the steady-state solution of the driven cavity. By assuming the viscosity and conductivity are locally constant, the term $(P - R_x)$ is neglected. The term $\Delta^n E_{v_{23}}$ pertains to the

cross-derivative variables. It is handled explicitly in order to produce a block tridiagonal structure for the algorithm. The other viscous terms are as follows:

$$\Delta^n F_v \approx (S^n \Delta^n U)_y + \Delta^{n-1} F_{v_{12}}, \quad \Delta^n G_v \approx (T^n \Delta^n U)_z + \Delta^{n-1} G_{v_{12}}, \tag{3.10b,c}$$

where

$$S^n \equiv \left(\frac{\partial F_{v_2}}{\partial U_y}\right)^n, \quad T^n \equiv \left(\frac{\partial G_{v_3}}{\partial U_z}\right)^n. \tag{3.12b,c}$$

By incorporating Eqs 3.3 and 3.10 into Eq 3.1 and by taking advantage of approximate factorization yields the Beam-Warming algorithm in factored form (Beran, 1993:Aero 753).

$$\left[I + \alpha_1(\delta_x A^n - \delta_{xx} R^n)\right]\left[I + \alpha_1(\delta_y B^n - \delta_{yy} S^n)\right]\left[I + \alpha_1(\delta_z C^n - \delta_{zz} T^n)\right]\Delta^n U = \Re^n, \tag{3.13}$$

$$\Re^n = \alpha_2 \Delta^{n-1} U + \alpha_3\left[(E_v - E)_x + (F_v - F)_y + (G_v - G)_z\right]^n \tag{3.14}$$

$$+\alpha_2\left[(\Delta E_{v_{23}})_x + (\Delta F_{v_{13}})_y + (\Delta G_{v_{12}})_z\right]^{n-1},$$

where I is the Identity matrix, and

$$\alpha_1 = \frac{\theta_1 \Delta t}{1+\theta_2}, \quad \alpha_2 = \frac{\theta_2}{1+\theta_2}, \quad \alpha_3 = \frac{\Delta t}{1+\theta_2}.$$

The terms δ_i, δ_{ii} are the standard FDM derivative operators as defined in Eqs 1.1 and 1.2. For the model problem described in Chapter 3, Eqs 3.13 and 3.14 reduce to the following:

$$\left[1 + \alpha_1(\delta_x f^n - \tfrac{1}{Re}\delta_{xx})\right]\left[1 + \alpha_1(\delta_y f^n - \tfrac{1}{Re}\delta_{yy})\right]\left[1 + \alpha_1(\delta_z f^n - \tfrac{1}{Re}\delta_{zz})\right]\Delta^n f = \Re^n, \tag{3.15}$$

$$\Re^n = \alpha_2 \Delta^{n-1} f + \alpha_3\left[\tfrac{1}{Re}(f_{xx} + f_{yy} + f_{zz}) - (e_x + e_y + e_z) + g\right]^n, \tag{3.16}$$

where $e = \frac{1}{2} f^2$, and g is defined in Eq 2.1.

To maintain the efficiency of the algorithm, the boundary conditions must be incorporated while preserving the tridiagonal structure and achieving the desired steady-state accuracy. To maintain the tridiagonal structure, first-order-accurate boundary conditions are implemented for the implicit portion of the algorithm. Then after $\Delta^n U$ is calculated and the solution has been updated, the second-order boundary conditions are explicitly enforced.

The model problem is implemented using a rectilinear grid comprised of irregularly spaced grid points at the Gauss-Lobatto quadrature points, $x_j = cos\frac{\pi j}{N}$. The Gauss-Lobatto points are the extrema of the Chebyshev polynomials (Gottlieb and Orszag, 1977:40). The variation of the grid metrics is a stretching/compression of any one Cartesian axis. The second-order-accurate finite-difference operators are modified to account for the non-uniform grid spacing, i.e. $h_i = x_{i+1} - x_i$ (Anderson, et al., 1984:55; Canuto, et al., 1987:144). For the boundary nodes, a second-order polynomial fit is used. The modified operators are provided in Appendix B.

3.2. Beam and Warming w[illegible] Chebyshev Collocation Method

At steady state, the solution's accuracy is governed by the method of evaluating the right-hand side and the method of enforcing of the boundary conditions. These areas are modified to yield a solution of spectral accuracy. Also, a relaxation scheme is added to the algorithm.

Instead of using finite-difference operators to evaluate the derivatives of the right-hand side, Eqs 3.14 and 3.16, Chebyshev collocation differentiation is used. This can be accomplished through the use of a Fast Fourier Transform (FFT) or through matrix multiplication.

> If the discrete Chebyshev transforms are computed by an FFT algorithm which takes advantage of the reality and the parity of the function $\tilde{u}(\theta) = u(cos\,\theta)$, the total number of operations required to differentiate in physical space is $(5log_2 N + 8 + 2q)N$, where q is the order of the derivative. ... If the collocation derivative is computed by matrix multiplication, the total number of operations is $2N^2$. (Canuto, et al., 1987:69-70)

Canuto et al. also present relative timings of the two Chebyshev methods. These are summarized in Table 3.2

Table 3.2. Timing (in msec) for Chebyshev Collocation Derivatives

	Cyber 855 (scalar)		Cyber 205 (vector)	
N	Matrix Multiply	Chebyshev Transform	Matrix Multiply	Chebyshev Transform
8	0.11	0.18	0.001	0.002
16	0.38	0.39	0.003	0.004
32	1.54	0.91	0.012	0.009
64	5.74	1.92	0.046	0.022
128	24.02	4.20	0.178	0.044
256	93.49	9.40	0.706	0.102

(Canuto, et al., 1987:45)

In vector mode, the two methods are equivalent for a moderate number of nodes, N_j. Equivalency is achieved between N=16 and N=32. Also, when applying the FFT method, ailiasing can be introduced in the evaluation of the nonlinear terms if care is not taken(Canuto, et al., 1987:84). One method is to use 50 percent more nodes during the FFT (the 3/2 rule). Then the additional coefficients are set to zero when evaluating the nonlinear terms via the convolution sum (Canuto, et al., 1987:82-85). For the spectral accuracy required here, a moderate number of nodes along a coordinate direction will be used (i.e. 25 - 33). Since the transform and matrix methods require approximately the same amount of time for this node size, the matrix method was used to avoid the issue of ailiasing. The actual derivative matrices are listed in Appendix C.

As for the boundary conditions, the explicit updating of the Neumann conditions needs to be modified for the Chebyshev matrix derivative. This can be shown for the 1D case, in which $\frac{dU}{dx} = 0$ at $n = 1, N$. The explicit form of $\frac{dU}{dx}$ is

$$\frac{dU}{dx} = \begin{bmatrix} u_1 & u_2 & \cdots & u_N \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} \end{bmatrix}, \tag{3.17}$$

where $A = a_{mn}$ is the derivative matrix and $U = u_m$ is the array of collocation points.

At the boundaries, the following conditions then apply:

$$\frac{du_1}{dx} = 0 = u_1 a_{11} + u_2 a_{21} + \cdots + u_{N-1} a_{N-1,1} + u_N a_{N1}, \tag{3.18}$$

$$\frac{du_N}{dx} = 0 = u_1 a_{1N} + u_2 a_{2N} + \cdots + u_{N-1} a_{N-1,N} + u_N a_{NN}. \tag{3.19}$$

Since $a_{11} = -a_{NN}$ and $a_{N1} = -a_{1N}$ (see Appendix C), then u_1 and u_N can be calculated:

$$u_1 = \frac{1}{b}(a_{11} s_1 + a_{N1} s_N), \tag{3.20}$$

$$u_N = -\frac{1}{b}(a_{N1} s_1 + a_{11} s_N), \tag{3.21}$$

where

$$s_1 = u_2 a_{21} + \cdots + u_{N-1} a_{N-1,1},$$

$$s_N = u_2 a_{2N} + \cdots + u_{N-1} a_{N-1,N},$$

$$b = a_{N1}^2 - a_{11}^2.$$

The relaxation scheme used here is a variation of the scheme used by Street et al. (1985:52-55). As they pointed out:

> The crucial property that a relaxation scheme should possess ... is that it damp effectively the high-frequency components of the error. It need not be especially effective in the low-frequency range, so long as it does not amplify any components. (Street, et al., 1985:52)

Their modified form of the approximate-factorization algorithm, expressed for two space dimension, is

$$[\alpha I - \delta_x H][\alpha I - \delta_y H]\Delta U = \alpha M, \tag{3.22}$$

where $H = \partial M / \partial U$ is the flux Jacobian of M; $\alpha = \frac{1}{\Delta t}$; and δ_x, δ_y are the second-order-accurate finite-difference operators. The right-hand side matrix, M, is calculated with a Chebyshev collocation method. Their relaxation scheme was to vary α in the range of $[\alpha_l, \alpha_h]$ by using the rule

$$\alpha^k = \alpha_h \left(\frac{\alpha_l}{\alpha_h} \right)^{(k-1)/(K-1)}, \tag{3.23}$$

where k is the time-level index and K denotes the number of different α values. Note that Street's α starts at α_h and ends at α_l. Since Eqs 3.13 - 3.16 are obtained through

multiplication by Δ_t, the following adaptation of Eqs 3.22 and 3.23 is used for this investigation:

$$\Im_x \Im_y \Im_z \Delta^n U = (\alpha^k)^3 \Re^n \tag{3.24a}$$

where

$$\Im_x \equiv \left[\alpha^k I + \alpha_1 (\delta_x A^n - \delta_{xx} R^n)\right] \tag{3.24b}$$

$$\Im_y \equiv \left[\alpha^k I + \alpha_1 (\delta_y B^n - \delta_{yy} S^n)\right] \tag{3.24c}$$

$$\Im_z \equiv \left[\alpha^k I + \alpha_1 (\delta_z C^n - \delta_{zz} T^n)\right] \tag{3.24d}$$

$$\alpha^k = \alpha_h \left(\frac{\alpha_l}{\alpha_h}\right)^{(K-k)/(K-1)} \tag{3.25}$$

and $k = 1, 2, \ldots, K$. Here, the α starts at α_l and ends at α_h. Typical values used are [0.1, 0.8 - 1.0] for $[\alpha_l, \alpha_h]$ and $K = 500 - 1000$.

3.3. Program Overview and Flowchart

Figure 3.1 presents the general flow of the program BW3D. The block solver (Beran, 1993:Aero 753) is vectorized over each sweep line. The implementations of the Chebyshev algorithms are taken from Canuto et al. (1987:Appendix B). The actual spectral coefficients are extracted from the converged solution using Canuto's 1D Chebyshev transform applied in 3 successive sweeps. The convergence criteria is:

$$\left|\Delta^n U\right|_{max} \leq \varepsilon, \tag{3.26}$$

where ε is set to 10^{-13} (near machine zero). A relatively small value of ε is required for the spectral solutions to converge to the minimum error solution as determined through examination of the model problem results.

The driven cavity output for graphics consists of the following solution parameters: the Cartesian velocity components, u, v, w; the scalar magnitude of the velocity vector field, $q = \frac{1}{2}\left(u^2 + v^2 + w^2\right)^{\frac{1}{2}}$; pressure, p, and temperature, T. The overall objective is to

compare the rendering of FDM and SM solutions which are of the same order of accuracy. To achieve this, the SM solutions are "filtered" by using only those expansion coefficients whose absolute value exceeds a set threshold. Typical thresholds are 0.001, 0.0001, and 0.00001. By doing so, the size of the SM data sets were typically reduced to the order of 100 coefficients. The resulting accuracy of the filtered model problem data sets are approximately 10 times the threshold used.

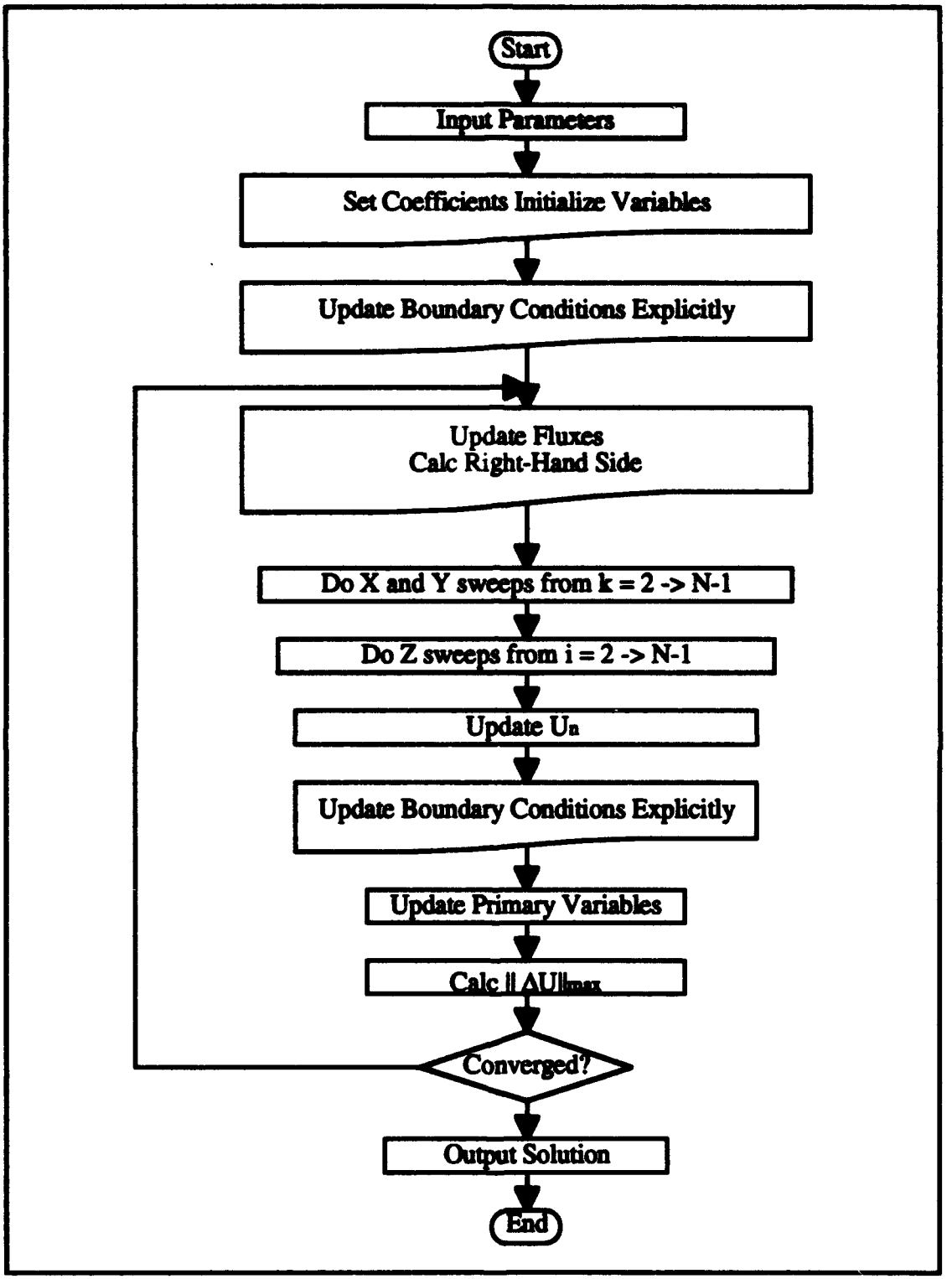

Figure 3.1 Main Flowchart of Numerical Algorithm

IV. Rendering Algorithm

This chapter describes how rendering of a single parameter data set is accomplished with octrees modified to take advantage of interval mathematics. The implementation mechanics for multiple parameters (i.e., the driven cavity) are discussed in Chapter 6. Rather, this chapter discusses the implementation of the generic marching cubes algorithm, the overlaying of the octree, the traversal algorithm, the attempted hash table addition (for reuse of vertex data among octants), and data structures used in the program.

4.1. Generic Marching Cubes Implementation

The marching cubes algorithm can be divided up into 3 rendering phases: input, extraction, and display. Input covers reading the data set from disk into the 3D data array or the "Cube". Extraction involves traversing every cell in the Cube with an isosurface value and fitting triangles to those cells which intersect the isosurface. Display covers the actual the process of sending the isosurface polygons to the graphics hardware.

4.1.1. Input Phase.

For large data sets containing sampled volume data, the input phase takes the majority of the total rendering time to accomplish. As will be seen, up to 90% of the total time can be spent in reading an ASCII file format. In the case of filtered SM data sets, this time is minimal since the file sizes are relatively small. Instead, the evaluation of the truncated spectral series at every node location in the Cube will consume the majority of the time.

4.1.2 Extraction Phase.

Once the user specifies the isosurface value or the "threshold", every cell of the Cube is visited by sweeping in pairs of *xy* planes. The algorithm is shown in Figure 4.1.

```
function marching_cubes
{
    get 1st xy plane of Cube data and normals at the nodes
    interpolate_xy_plane
    for (iz=1; iz<zdim; iz++)
    {
        get next xy plane of Cube data and normals at the nodes
        interpolate_xy_plane
        interpolate_z_edges_between_xy_planes
        calculate table index & triangulate polygons
        output polygon data
        swap xy planes
    }
}
function interpolate_xy_plane
{
    for (iy=0; iy<ydim; iy++)
    for (ix=0; ix<xdim; ix++)
    {
        if (surface_cross_x_edge) interpolate x values and normals
        if (surface_cross_y_edge) interpolate y values and normals
    }
}
define surface_cross_edge
    (node 1 value ≤ isovalue and isovalue < node 2 value)  or
    (node 2 value ≤ isovalue and isovalue < node 1 value)
```

Figure 4.1. Marching Cubes Algorithm

The normals at the Cube's nodes are calculated by normalizing the gradient of parameter, as shown in Eq 4.1. The gradient is approximated through a first order FDM along each of the coordinate axis.

$$\bar{n} = \frac{f_x \hat{i} + f_y \hat{j} + f_z \hat{k}}{\left(f_x^2 + f_y^2 + f_z^2\right)^{\frac{1}{2}}} \tag{4.1}$$

where $\bar{n}$ is the unit normal vector of the parameter f. If a cell is encountered that intersects the isosurface, polygons are generated representing the portion of isosurface within that cell. If any two nodes of a cell straddle the isosurface value, then that cell intersects the isosurface. The actual triangulation of the cell is done through Lorensen and Cline's table

lookup method (1987:165). For cells containing an ambiguous face, the ambiguity can be resolved by evaluating the function at the center of the ambiguous face (Wilhelms and Van Gelder, 1990:80). However, this has not been implemented for this effort.

4.1.3. Display Phase.

In this effort, instead of sending the polygon information directly to graphics hardware, the information is stored in a vertex and polygon array for later display at the user convenience. This allows the option to display multiple isosurfaces without having to traverse the octree for each surface regeneration. These data structures are briefly described below in section 4.3.

4.2 Octree Implementation

Prior to discussing the specifics of the rendering phases, physical layout of the octree are discussed. The rendering phases with an octree are similar to that of marching cubes except for setup and traversal of the octree. The algorithm will be described assuming interval mathematics will be used to determine the initial minimum and maximum values of the octants. Use of a hash table is attempted. The hash table's design is provided. Lastly, incorporating a cutting plane into the algorithm is discussed.

4.2.1. The Cube and the Octree Layout.

First the Cube (the 3D data array) layout is described; second, the octree layout; lastly, the "dimension" of a octree leaf is related to the number of cells and nodes it covers in the Cube.

The size of the Cube (the number of nodes contained within) is related to the plot depth resolution by the formula: $\left(2^p + 1\right) \text{x} \left(2^p + 1\right) \text{x} \left(2^p + 1\right)$, where p refers to the plot depth of the algorithm. For this effort, the maximum plot depth used is 6 (or a Cube size of 65x65x65). Higher plot depths could have been easily used. The number of cells contained in the Cube is related by a similar formula: $\left(2^p\right) \text{x} \left(2^p\right) \text{x} \left(2^p\right) = 8^p$. For

example, if the plot depth = 0, then the Cube would contain 1 cell and 8 nodes. If the plot depth = 1, then the Cube would contain 8 cells and 27 nodes. Also, the plot depth can be thought of as the "octant dimension" of the Cube. This will be useful in relating an octree's leaf to the Cube's nodes and cells.

Similarly, the dimension of the octree, *odim*, is defined as the number of levels contained in the octree. The top level of the octree (*octlevel* = 0) corresponds to the entire Cube. At the next level down (*octlevel* = 1), the Cube is subdivided into 8 octants. Then at the next level down (*octlevel* = 2), each octant is subdivided again into 8 additional octants, and so on. The number of octree nodes or octants at any one level is determined by the formula: $8^{octlevel}$. At the bottom of the tree, there are 8^{odim} leaves.

As for the size of a leaf, the number of Cube nodes and cells contained within a leaf are defined by similar formulas: #nodes = $\left(2^m + 1\right)\mathrm{x}\left(2^m + 1\right)\mathrm{x}\left(2^m + 1\right)$ and #cells = 8^m. Here, m is defined as the octant dimension of the leaf. For sake of clarity, the dimension of a leaf will be referred to as the *mcdim*. As noted prior, if a leaf is visited which intersects the isosurface, then polygons are fitted using the marching cubes algorithm, hence the *mcdim* label. Figure 4.2 shows the Cube and the different octree levels for $p = 4$, *odim* = 3, and *mcdim* = 1. In this example, the Cube contains 17^3 nodes and any one leaf contains 8 cells or 27 nodes of the Cube. The plot depth and octree dimension and leaf dimension are related by: $p = odim + mcdim$. Also note, the term $\left(2^m + 1\right)$ is the *xdim*, *ydim*, and *zdim* in the marching cubes algorithm, Figure 4.1. Following Wilhelms and Van Gelder suggestion, the minimum leaf dimension was *mcdim* = 1 (1992:206). If the plot depth, $p = mcdim$, and *odim* = 0, then the generic marching cubes algorithm is implemented.

4.2.2. Rendering Phases Overview.

As before with the marching cubes algorithm, the rendering can be divided into input, extraction and display phases. The display phase is the same as before. The amount

of time the other two phases take is dependent on whether interval mathematics is used. The impact on the input phase will be discussed next, followed by a discussion of interval mathematics, surface extraction by the octree traversal, and the attempted use of hash table for temporary storage of vertex information among octants.

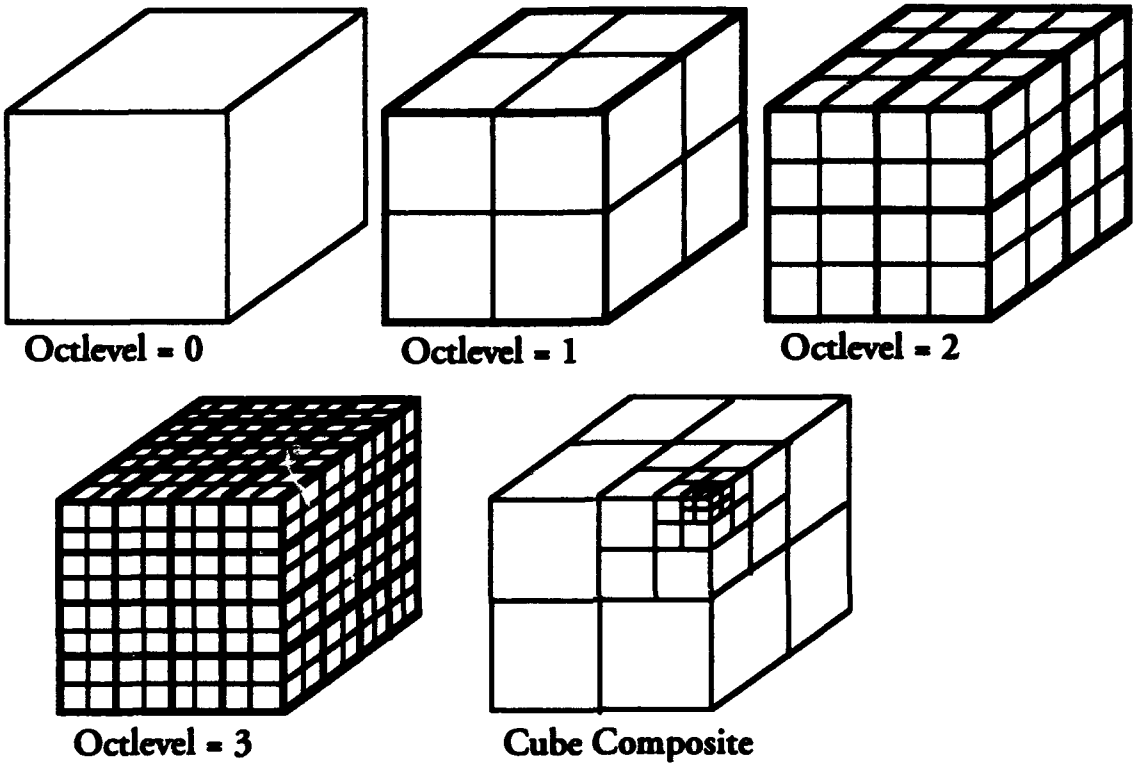

Figure 4.2. Octree Overlay, Plot Depth = 4, Octree Dimension = 3

4.2.3. Octree Input Phase.

In addition to allocating and assigning data values to the Cube, an octree is created that contains at each node the maximum and minimum data values found in that node's subtree and a pointer to the minimum (x,y,z) location in the Cube. Even though a Cube's index can be calculated directly from the octree node's address (Gargantini, 1981:366), Wilhelms and Van Gelder's approach is adopted to speed up the octree traversal (1992:212). Compared to the generic marching cubes input phase, the octree input phase is slightly longer since the octree nodes have to be populated. If using interval mathematics in

conjunction with a SM solution, the octree nodes are assigned minimum and maximum values from the top down instead of the normal bottom's up initialization. Only when a leaf is visited are the Cube nodes assigned a data value.

4.2.4. Interval Methods.

The probability of detecting contours or isosurfaces by using the process of point sampling (as is done in the marching cube algorithm) is a function the plot depth used. Figure 4.3 shows 4 cases in which point sampling will fail to detect a contour in 2D.

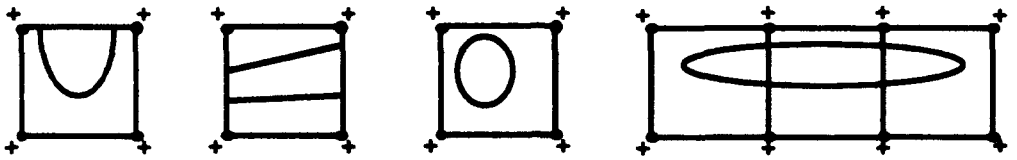

Figure 4.3. Undetected Contour Cases (Suffern and Fackerell, 1991:332)

Suffern and Fackerell (1991:332) also provide the basic operations of interval mathematics which are as follows:

$$[a,b]+[c,d]=[a+c,b+d], \tag{4.2a}$$

$$[a,b]-[c,d]=[a-d,b-c], \tag{4.2b}$$

$$[a,b]\otimes[c,d]=[\min(ac,ad,bc,bd),\max(ac,ad,bc,bd)], \tag{4.2c}$$

$$[a,b]/[c,d]=[a,b]\otimes[\tfrac{1}{c},\tfrac{1}{d}], \quad \text{provided} \quad 0\notin[c,d]. \tag{4.2d}$$

Eq 4.2d can also be extended to cases when $0\in[c,d]$, but this particular operation is not applicable in deriving the natural interval extension for the truncated series used here. To derive the natural interval extension of a function, the fundamental property of interval mathematics is used:

$$\text{if } x\in X, \text{ then } f(x)\in F(X), \tag{4.3}$$

where $F(X)$ is calculated by substituting the interval $X=[a,b]$ into $f(x)$ and using the interval operations defined in Eqs 4.2a-d (Suffern and Fackerell, 1991:332). In applying

this to the octree, every octant is defined over some 3D interval. By calculating the spectral series' natural interval for an octant yields the maximum and minimum for that octant. The actual surface value for that octant is guaranteed to lie within this natural interval. The natural interval extension for a Chebyshev polynominal is shown in Appendix D. This provides a means of branching into the octree without first evaluating the spectral series at every node location in the Cube. Since the surface fitting is only dependent on node values contained the octree's leaves, the interval mathematics is not applied to the bottom level of the octree. The primary drawback is that every natural interval calculation for an octant requires 6 additional series evaluations which cannot be used in the actual surface fitting. If the first isosurface selected intersects a small portion of the total volume, then using interval mathematics is a good trade-off. Then while the user is examining the first few isosurfaces, the program can evaluate the remaining unfilled nodes of the Cube in the background.

4.2.5. Octree Traversal and Extraction Phase

Every node in the octree is an octant which is made up of either 8 additional octants or (at the bottom level) at least 8 cells or at least 27 data values in the Cube. Traversal of the octree is accomplished by performing a preordered traversal that recursively visits each node and its subtree. Figure 4.4 shows the traversal algorithm. The octree's data structure is briefly described in section 4.3. The one modification made from Wilhelms and Van Gelder's setup is the addition of octant full flag. This is used in the binary decision to "fill" the octant using interval mathematics. The "ijk" index is the precalculated index into the Cube's array and stored in each leaf.

```
function Build_Surface
{
    start at top of octree, octlevel = 0, get the top octant
    while ( not done )
    {
        if( octant is not full ) fillOctant using Interval Math
        if( surfaceNotPresent ) get the next octant
        else
        {
            if( octlevel is not at bottom of octree )
            {
                subdivide:
                drop down to next octlevel and get the 1st of 8 octants
            }
            else
            {
                March through the leaf starting at ijk index of the Cube
                get the next octant
            }
        }
        while( all 8 octants at this level have been visited )
        {
            if( octlevel is back at the top ) done = TRUE
            else
            {
                go back up to previous octlevel
                get the next octant
            }
        }
    }
}
define surfaceNotPresent
    octant min > isovalue or octant max < isovalue
```

Figure 4.4. Octree Traversal Algorithm (Suffern and Fackerell, 1991:339).

4.2.6. Hash Table Design.

In addition, a hash table should be used to save previously calculated surface vertices. Since each surface vertex is generally used in four neighboring polygons, having to recalculate the vertex information (the location on an edge and the vertex's normal) can negate the octree speed advantage (Wilhelms and Van Gelder, 1992:221). As suggested by Wilhelms and Van Gelder, each cell edge in the Cube is given unique key. They suggested using the leaf's index into the Cube as the basis of the edge key. Specially, this is given as:

4 * index + direction_code, where direction_code is 1 for x, 2 for y, or 3 for z (Wilhelms and Van Gelder, 1992:225). This is also assuming the dimension of a leaf (*mcdim*) is 1. Then due to traversal order, the 3 edges adjacent to the "origin" of the cell will never be visited again (Wilhelms and Van Gelder, 1992:215).

In the design attempted here, 3 hash tables are used, one for each coordinate direction. This allows the leaf's index into the Cube to be used directly as the key. If an edge intersection is calculated which is not on the "origin axis" of the leaf, then the key, the interpolated vertex and 3 normals values are stored in the appropriate hash table at the first available table entry. If a leaf's edge is found to intersect the isosurface, then the appropriate hash table is checked with the index key. If the edge is on a "origin axis" of the leaf, a delete flag is set for that hash entry. With this setup, accessing the hash tables, slowed the traversal by a factor of 2 - 3.

4.2.7. Modifying the Octree for a Cutting Plane.

If displaying multiple isosurfaces, a desired feature is a cutting plane. A simple cutting plane in either the xy, yz, or xz plane is easily implemented with an octree. For example, if yz cutting plane is to be set at $x = 0.5$, then prior to marching a leaf, the x value of the leaf's origin is checked. If it is less than 0.5, then march it. Otherwise, discard it and move on to the next octant.. The modification to the octree traversal algorithm is shown in italicized font in Figure 4.5.

```
if( surfaceNotPresent ) get the next octant
else
{
    if( octlevel is not at bottom of octree )
    {
        subdivide:
        drop down to next octlevel and get the 1st of 8 octants
    }
    else
    {
        if( octant x, y, or z < cutting plane x, y, or z)
            March the octant starting at ijk location in the Cube
        get the next octant
    }
}
```

Figure 4.5. Cutting Plane Algorithm Modification

4.3. Program Data Structures

The program is written in standard C programming language. The data structures for 4 of the key variables, the Cube, the octree, the vertex array, and the polygon array, are shown in Table 4.1.

Table 4.1. Key Data Structures

Cube	Octree	Vertex	Polygon
int full; unsigned int x; unsigned int y; unsigned int z; float nodeValue; vertex n { float x; float y; float z; }	int full; unsigned long ijk; float min; float max;	vertex v { float x; float y; float z; } vertex n { float x; float y; float z; }	unsigned long v1; unsigned long v2; unsigned long v3;

V. Numerical Results

Presented herein are the model problem accuracy comparisons for the FDM and SM cases, the convergence histories, the filtering of the SM data sets and their resulting accuracy. In this context, FDM represents a second-order-accurate, finite-difference method. SM represents a Chebyshev collocation method. All results pertain to steady-state solutions. The results for the driven cavity at the Gauss-Lobatto grid nodes are not available. Even though the modification of the finite-difference derivatives for the Gauss-Lobatto nodes worked for the model problem, the extension to the driven cavity is not working. All of the driven cavity results are from using FDMs on an uniform grid.

5.1. Model Problem Accuracy Comparisons

Each numerical solution is compared with the exact solution, and the absolute error is calculated at each grid node. The maximum error is the maximum norm of the absolute error matrix, i.e.,

$$\varepsilon_{max} = \left| Exact - U_{calculated} \right|_{max} \tag{5.1}$$

The different cases analyzed and the maximum error for the data sets are shown in Table 5.1 and plotted versus the number of degrees of freedom for a 1D case in Figure 5.1. The convergence criteria is based upon the maximum norm of the ΔU being less than some small epsilon, ε (see Eq 3.26). To allow for the SM cases to converge to their lowest error solution, ε is set to near machine zero (O [10^{-13}]). As expected, for the same number of degrees of freedom, SM produces solutions 2 - 5 orders of magnitude more accurate than FDM.

The overall accuracy of the SM cases is a function of the number of nodes inside the boundary layer. The number of Chebyshev polynomials required resolve boundary layer distances is on the order δ^{-1}, where δ is the boundary layer thickness (Gottlieb and Orszag, 1977:40). For the worse case, δ=0.14, the number of nodes required to resolve this is 3. For the 1D case in which the degrees of freedom is 17, there are just 3 nodes located within the boundary layer. This case exhibits the worst spectral error, 0.294. In fact, none of the cases in which N=17 would converge without first a FDM solution. In all other cases, the SM would converge from an impulsive start or a linearly interpolated 2D solution.

Table 5.1. Model Problem Accuracy for FDM and SM

λ	Boundary-Layer Thickness, δ	N	E_{max}, FDM	E_{max}, SM
2	0.75	17	1.40×10^{-1}	1.41×10^{-4}
		25	4.87×10^{-2}	2.32×10^{-6}
		33	2.59×10^{-2}	2.27×10^{-8}
5	0.31	17	5.60×10^{-1}	1.48×10^{-2}
		25	7.28×10^{-2}	3.11×10^{-4}
		33	3.73×10^{-2}	8.90×10^{-6}
8	0.18	17	no convergence	4.50×10^{-2}
		25	1.21×10^{-1}	1.84×10^{-3}
		33	5.71×10^{-2}	1.80×10^{-4}
		49	2.37×10^{-2}	3.90×10^{-7}
10	0.14	17	no convergence	2.94×10^{-1}
		25	1.84×10^{-1}	9.34×10^{-3}
		33	7.70×10^{-2}	3.89×10^{-4}
		49	3.00×10^{-2}	4.83×10^{-6}
		65	1.60×10^{-2}	n/a

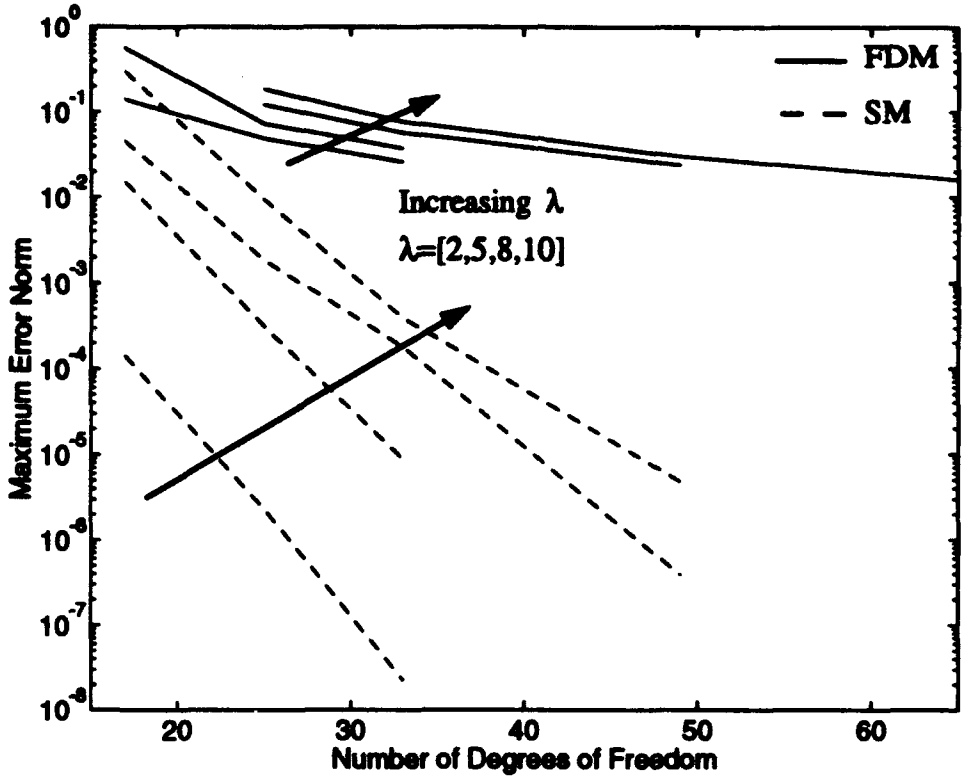

Figure 5.1. Solution Accuracy for Number of 1D Degrees of Freedom

5.2. Convergence History

Typical convergence plots are shown in Figures 5.2 and 5.3 for the model problem and the driven cavity, respectively. The model problem plots are from cases in which a 2D solution was linearly interpolated to the 3D domain for the initial start. The SM cases typically required a much smaller initial Δt for convergence than the FDM cases ($\Delta t = .01$ versus .05). For the model problems, a simple variable time step is implemented. At the end of each iteration step, the program checks the rate of change of ΔU and applies a proportional change to Δt, i.e., for a converging solution, Δt is increased. The change in

Δt is also weighted to favor a decrease, as opposed to an increase, in Δt. The limits of Δt is set to [0.001, 0.1].

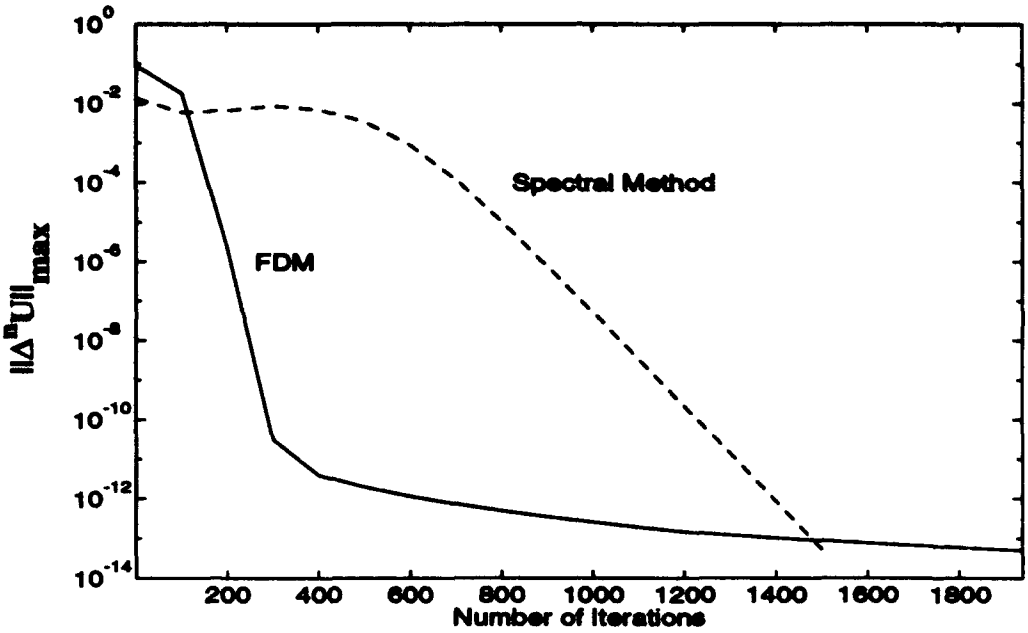

Figure 5.2. Model Problem Convergence History, N = 49

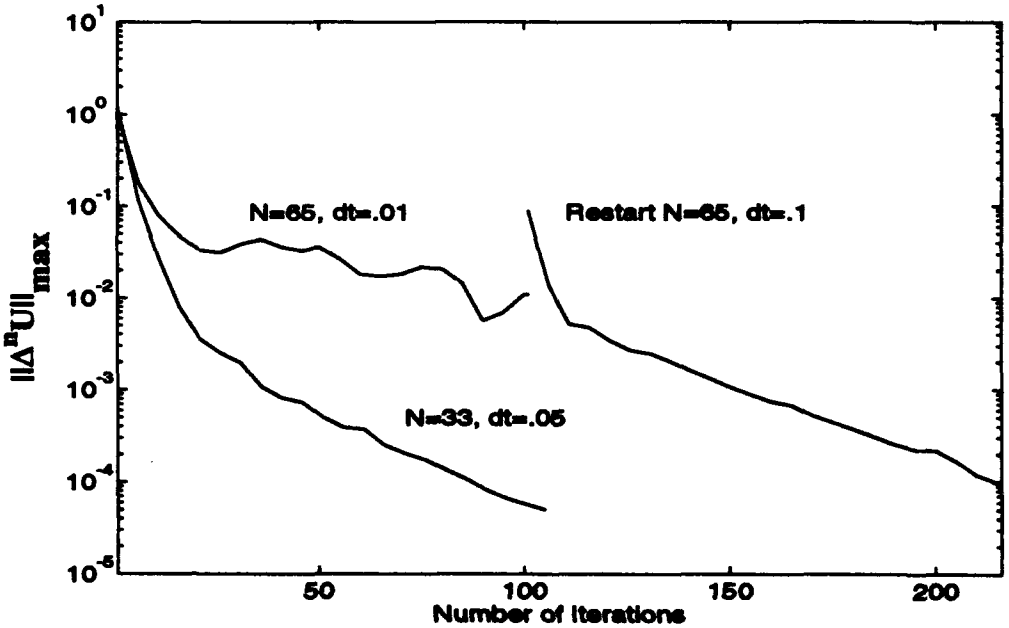

Figure 5.3. Driven Cavity Convergence History, Uniform Grid

In the cases shown in Figure 5.2, the initial Δt is set .01 and .05. With the initial start conditions described previous, the initial ΔU norms are 0.1 or less. The FDM cases, in general, converge relatively quick to the SM cases. After a certain norm level is reached though (in this example, 10^{-11}), the FDM cases would slow dramatically for the machine-zero ε (Eq 3.26). In the driven cavity cases (Figure 5.3), convergence is defined as $\varepsilon = 0.001\Delta t$. Also the initial solution is an impulsive start. The break in the N=65 case is a result of performing 100 iterations at $\Delta t = 0.01$ and then restarting the code with $\Delta t = 0.1$. If the initial Δt is set to 0.05, the code does not converge for N = 65.

5.3. Filtering of Expansion Coefficients for Graphics

In order to compare the SM and FDM solutions at the same degree of accuracy, and in order to reduce the computational workload for evaluating the SM series solution, the SM solutions are filtered. All expansion coefficients, whose absolute value was below a certain threshold, are discarded. Various thresholds from 0.00001 to 0.001 are used. The effect of the threshold on the rendering is examined in Chapter 6. Figure 5.4 shows the distributions of two SM data sets ($\lambda = 10$ and N=25,33). As seen, the majority of the coefficients are within the interval [-0.0001,0.0001]. Also note, as the number of degrees of freedom increases, the mode of distribution shifts toward the left (10^{-8} to 10^{-11}).

The resulting accuracy of the SM data sets, after filtering at 0.001 and 0.0001, are listed in Table 5.2. If the initial accuracy of the data set is less than the filtering threshold, then the resulting filtered accuracy is approximately 1 order of magnitude above the threshold. If the initial accuracy is greater than the threshold, the resulting filtered accuracy is on the same order as the initial accuracy. Also note, the number of coefficients retained is dramatically reduced from the original data set. Since the data sets are representations of an analytic function, the data sets should be inherently frequency limited. Hence, the

reduction in coefficients is not surprising. Unfortunately, without a driven cavity solution at the appropriate grid spacing (i.e., the Gauss-Lobatto points), the extraction of the spectral coefficients would yield impracticable results.

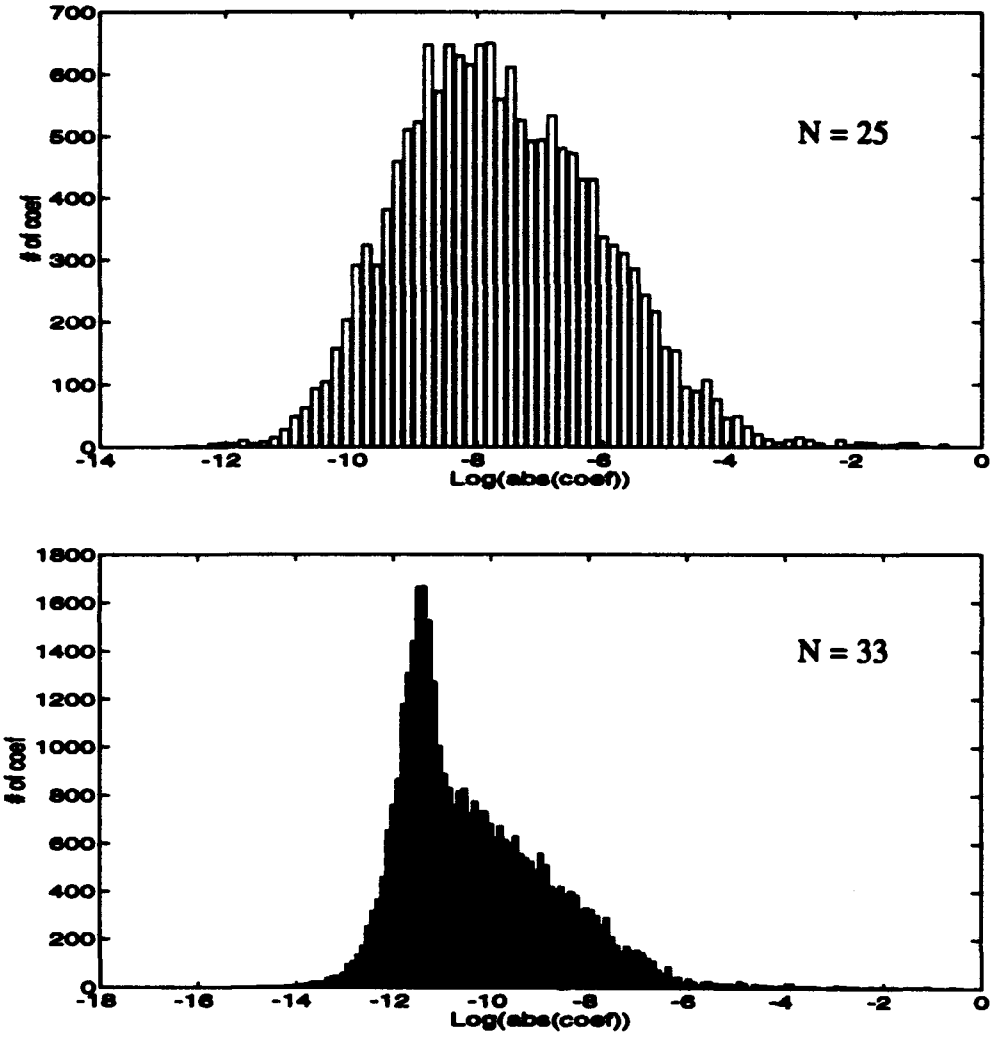

Figure 5.4. Histogram of Full Expansion Coefficient Data Sets: $\lambda = 10$, N = 25, 33

Table 5.2. Spectral Accuracy for Filtered Data Sets

λ	N	Full Series	Truncated Series					
			Threshold = 0.0010			Threshold = 0.0001		
		E_{max}	E_{max}	#Coef	%Filtr	E_{max}	#Coef	%Filtr
2	17	1.41×10^{-4}	1.10×10^{-2}	45	0.9	7.29×10^{-4}	83	1.7
	25	2.32×10^{-6}	1.10×10^{-2}	45	0.3	7.29×10^{-4}	83	0.5
	33	2.27×10^{-8}	1.10×10^{-2}	45	0.1	7.29×10^{-4}	83	0.2
5	17	1.48×10^{-2}	1.02×10^{-2}	66	1.3	1.43×10^{-2}	267	5.4
	25	3.11×10^{-4}	7.50×10^{-3}	66	0.4	1.17×10^{-3}	112	0.7
	33	8.90×10^{-6}	7.49×10^{-3}	66	0.2	1.10×10^{-3}	112	0.3
8	17	4.50×10^{-2}	1.98×10^{-2}	90	1.8	4.29×10^{-2}	500	10.2
	25	1.84×10^{-3}	8.91×10^{-3}	83	0.5	2.46×10^{-3}	155	1.0
	33	1.80×10^{-4}	8.75×10^{-3}	85	0.2	1.17×10^{-3}	144	0.4
	49	3.90×10^{-7}	8.74×10^{-3}	85	0.1	1.16×10^{-3}	144	0.1
10	17	2.94×10^{-1}	2.27×10^{-1}	297	6.1	2.79×10^{-1}	1485	30.2
	25	9.34×10^{-3}	7.09×10^{-3}	89	0.6	6.48×10^{-3}	237	1.5
	33	3.89×10^{-4}	9.38×10^{-3}	86	0.2	1.91×10^{-3}	155	0.4
	49	4.83×10^{-6}	9.38×10^{-3}	86	0.1	1.73×10^{-3}	156	0.1

Lastly, the number of coefficients retained is also related to the initial data set's accuracy. For example, for the case in which the threshold = 0.0001 and $\lambda = 10$, the number of coefficients retained decreases from 1485 to 155 as the number of degrees of freedom increases from 17 to 33. This is the same trend as seen in the histograms and the shift of the mode towards the left. It relates to the level of noise or error contained in each coefficient. For the cases in which N = 33 or 49, the accuracy of the full data set is the same order of magnitude as the threshold or lower than the threshold. For this reason, the filtered data sets with N = 33 are compared with the FDM data sets with N = 33, and the filtered data sets with N = 49 with the FDM data sets with N = 69. To visualize the effect of filtering, a 1D case is shown in Figure 5.5 for the hyperbolic tangent function.

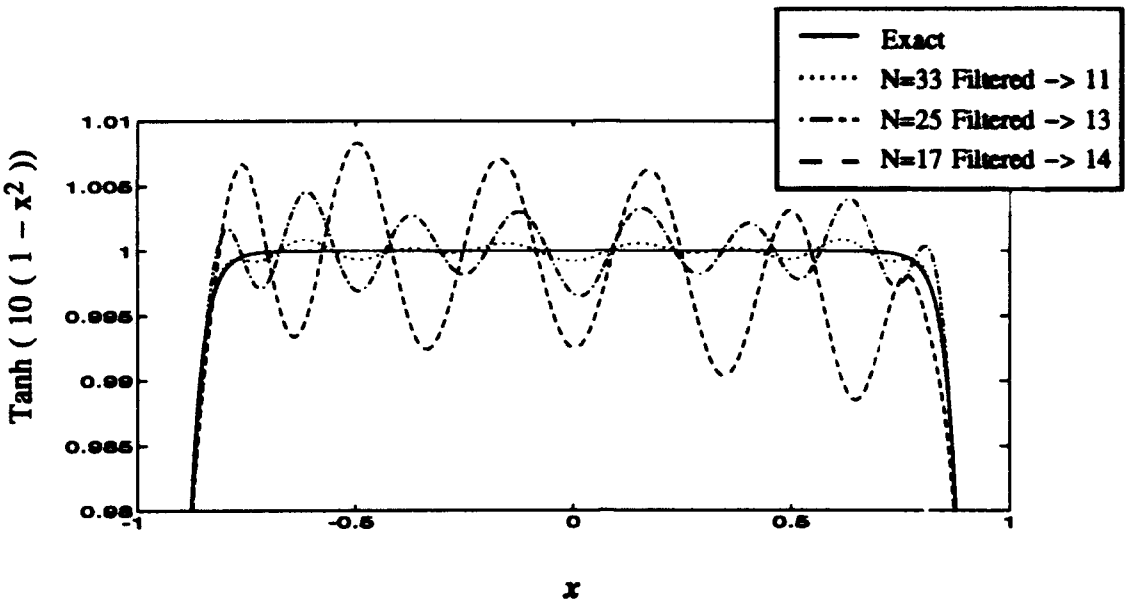

Figure 5.5. Resulting Truncated Fit of Hyperbolic Tangent, Threshold = 0.001

In the 1D filtered case, the x direction expansion coefficients are extracted from the 3D SM data sets (N=17, 25, 33) for $y, z = 0$. The extracted coefficients are then filtered at a threshold = 0.001, resulting in reducing the number of coefficients to 14, 13, and 11 respectively. The plots versus the exact solution are shown enlarged in Figure 5.5. As seen, the solution's error exhibits itself as an oscillatory behavior about the exact solution. Since the Chebyshev polynomials are closely related to the Fourier cosine series, i.e., $T_n(\cos\theta) = \cos(n\theta)$, this behavior is consistent with a truncated Fourier series fit to a straight line. In Chapter 6, isosurfaces are shown that display this oscillatory behavior if the desired isosurface value is within 1 error norm (ε_{max}) of the data set's upper limit.

VI. Rendering Results and Conclusions

In this chapter, the following results are presented:

- rendering timings of various model problem cases,
- the effect of the data set's error on model problem isosurfaces, and
- multiple isosurface views of both the model problem and the driven cavity.

The effect of interval mathematics and the attempted use of a hash table on rendering times is discussed. The effect of using a cutting plane is shown in the multiple isosurfaces views. The cases with the smallest boundary layer are used primarily for the comparison discussion since these cases contained the larger maximum error norms, ε_{max} (see Eq 5.1). The model problem cases used are listed in Table 6.1.

Table 6.1. Model Problem Data Sets Used

File	Original Deg of Freedom	λ	Boundary Layer	Filter Threshold	# Filtered Coef	ε_{max}
FD3310	33^3	10	0.14	n/a	n/a	0.0770
FD6510	65^3	10	0.14	n/a	n/a	0.0160
SM3310	33^3	10	0.14	0.00100	86	0.0094
SM4910.1	49^3	10	0.14	0.00010	156	0.0017
SM4910.01	49^3	10	0.14	0.00001	258	0.0001
SM338	33^3	8	0.18	0.00100	85	0.0088
SM335	33^3	5	0.31	0.00100	66	0.0075
SM332	33^3	2	0.75	0.00100	45	0.0110

6.1. Rendering Timings

The various rendering times without using interval mathematics for two plot depths are listed in Table 6.2 (files FD6510, SM3310, SM4910.1 and SM4910.01) and Table 6.3 (files FD3310 and SM3310). All timings are from a Silicon Graphics VGX R4000,

50Mhz workstation. Columns 3 - 6 are initialization costs in setting up the Cube and the octree. Only the build surface timing is applicable to rendering a new isosurface. With an octree at the highest resolution (plot depth = 6), this is less than a second. Any calculations of normals are based upon gradient of the scalar field and do not change with subsequent new renderings. However, the decrease in calculating the normals between the pure marching cubes algorithm (Octree dimension, *odim* = 0) and the octree algorithm results from calculating the normals pertaining to that isosurface instead of the whole Cube.

Table 6.2. Rendering Times Without Interval Math, Plot Depth = 6, Isovalue = 0.95

File	Odim	Cube/Octree Alloc & Input (sec)	Function Eval & Octree Init (sec)	Total Init (sec)	Calc Normals (sec)	Build Surface (sec)
FD6510	0	50.7	0.3	51.0	2.6	2.8
SM3310	0	0.4	21.0	21.4	2.6	2.9
SM4910.1	0	0.5	37.3	37.8	2.6	2.9
SM4910.01	0	0.5	60.1	60.6	2.6	2.8
Analytic	0	0.4	3.0	3.4	2.6	2.9
FD6510	4	51.9	2.2	54.1	0.2	0.4
SM3310	4	0.4	22.7	23.1	0.4	0.8
SM4910.1	4	0.5	39.1	39.6	0.5	0.7
SM4910.01	4	0.5	62.2	62.7	0.4	0.7
Analytic	4	0.4	4.8	5.2	0.4	0.7
FD6510	5	51.7	3.4	55.1	0.1	0.4
SM3310	5	0.5	24.4	24.5	0.3	0.7
SM4910.1	5	0.5	40.2	40.7	0.3	0.7
SM4910.01	5	0.5	63.6	64.2	0.3	0.7
Analytic	5	0.5	5.9	6.4	0.3	0.7

Interestingly, the build surface timings for the FDM are approximately half that of the SM data sets and the analytic solution. The rendering algorithm fits approximately half the number of polygons to the FDM data sets as compared with the others for the this isosurface. For instance, the number of polygons for the FD6510 isosurface case in

Table 6.3. Rendering Times Without Interval Math, Plot Depth = 5, Isovalue = 0.95

File	Odim	Cube/Octree Alloc & Input (sec)	Function Eval & Octree Init (sec)	Total Init (sec)	Calc Normals (sec)	Build Surface (sec)
FD3310	0	6.4	0.1	6.5	0.3	0.4
SM3310	0	0.1	2.6	2.7	0.3	0.4
Analytic	0	0.1	0.4	0.5	0.3	0.4
FD3310	3	6.6	0.2	6.8	0.1	0.1
SM3310	3	0.1	2.8	2.9	0.1	0.2
Analytic	3	0.1	0.5	0.6	0.1	0.2
FD3310	4	6.6	0.3	6.9	0.0	0.1
SM3310	4	0.1	2.9	3.1	0.1	0.2
Analytic	4	0.1	0.6	0.7	0.1	0.2

Table 6.2 is 8303 versus 16295 and 16231 for the SM and analytic solutions respectively. As a result, the time to build the isosurface is cut in half. The reduced polygon count for the FDM is a result of the node spacing used in the Cube. The FDM data values are located at the Gauss-Lobatto points which is more sparsely distributed in the interior of the Cube. As a result, less cells are traversed in the surface fitting. In contrast, the SM and analytic isosurfaces are extracted from a Cube and octree that are set up assuming uniformly spaced nodes. For isosurface values which are located in the denser FDM grid region (i.e., an isovalue = 0.1), the two methods produce comparable polygon count and build timings.

In comparing the octree versus the pure marching cubes, the results are similar to those found by Wilhelms and Van Gelder (1992). The octree significantly reduces the building and re-building times for isosurfaces. The only difference here, these timings are based upon not using a hash table to reuse vertex information among octants. The attempted addition of the hash table is discussed below.

In comparing SM cases with FDM ones, the SM model problem cases are approximate twice as fast as the FDM cases in which the accuracy of the two solutions are comparable. This relative timing, for the domain of the model problem, is solely a function

of either reading the data values from disk (FDM) or evaluating the function at the nodes. Since the model problem frequency content is inherently limited, the number of coefficients required for a fluid flow field will undoubtedly be much larger. In this case, the FDM will have lower setup costs.

The one advantage, which has not been fully exploited, is the ability to evaluate the SM solution at any arbitrary location in the domain. If the FDM solution is defined over a curvilinear domain (in which the node spacing is a function of two or more coordinate axes), then the rendering of the FDM solutions would require interpolation to avoid "cracks" in the isosurfaces. Then the rendering of FDM cases will significantly increase, while the time required for SM cases would remain relatively constant.

Another attempt to exploit this SM advantage is in the use of interval mathematics in determining an octant's initial minimum and maximum value. The relative timings with interval mathematics are shown in Table 6.4. The "% Calculated" column is the percent of the Cube's nodes filled during the octree first traversal. The "Additional Initialization" timings result from filling the nodes not visited during that first traversal and can be done in the background while waiting for user input.

Table 6.4. Rendering Times With Interval Math, Plot Depth = 6, Isovalue = 0.95

File	Odim	w/o Int Math	w/ Int Math		
		Total Init Build (sec)	Total Init Build (sec)	Add'l Init (sec)	Init % Calc
SM3310	4	24.3	27.2	1.7	100
	5	25.9	27.7	7.4	77
SM4910.1	4	40.7	42.8	1.7	100
	5	41.6	39.8	10.8	77
SM4910.01	4	63.8	66.1	1.5	100
	5	65.1	59.1	16.1	77
SM332	4	13.4	9.6	5.3	63
	5	14.5	7.8	8.7	39

In this application, interval mathematics is found to be of limited use. The main problem is the natural interval extension for a Chebyshev series is not effective until the 4th level of the octree (*odim*= 5); the resulting "width" of the natural interval is too large at the higher octree levels. The width of an interval is defined by: $w([a,b]) \equiv b-a$. If $f(x)$ is continuous, then the following is true (Snyder, 1992:123):

$$w(X) \to 0 \;\Rightarrow\; w(f(X)) \to 0 \tag{6.1}$$

For instance, the natural interval extension for whole domain of the Cube (the top octree level) for SM3310 case is [-1.83,2.44]. The widths of the octant interval remain large at the higher octree levels to include all isosurface values and the whole Cube is effectively traversed along the octree. Octree traversal of the entire Cube is relatively inefficient as compared to the pure marching cubes traversal. Interval mathematics may be useful if the plot depth is greater than 6 or if recursive subdivision based upon the isosurface curvature is employed.

Lastly, the hash table impact on build times is discussed. Table 6.5 lists the increase in build timings for when the hash table, as described in Chapter 4, is enabled. The number of hashes refers to the number of visits made to the table and isosurface vertex information was found from a previous octant. As shown, using the hash table slowed the build tin e by a factor of 2 - 3. Even so, the octree timings without the hash table are significantly faster than the pure marching cubes method.

Table 6.5. Hash Table Rendering Times, Isovalue = 0.95

File	Plot Depth	Odim	Build w/o Table (sec)	Build w/ Table (sec)	Increase Ratio	# Hashes	% Vertex Reused
Analytic	6	5	0.69	2.06	3.0	8522	26
SM3310	6	5	0.65	2.10	3.2	8614	26
Analytic	5	4	0.17	0.29	1.7	2151	25
SM3310	5	4	0.17	0.28	1.6	2152	25

6.2. Quality of Numerical Solution Graphics

Figures 6.1 - 6.5 show isosurfaces for FD3310, FD6510, SM3310, SM4910.1, and SM4910.01 data sets. For these isosurfaces, the accuracy of the data set is significant, i.e., the isovalue within the range of 1 error norm of the data set's maximum value. For these cases, the model problem's maximum value is 1. Each figure shows the degradation of the isosurfaces as the isovalue approaches 1. The sequence of Figures 6.1 - 6.5 shows the subsequent improvement in the isosurfaces as the accuracy of the data set improves.

In all cases, whenever the isovalue is within 1 error norm of the upper (or lower) limit of the data set, degradation of the surface occurs (as compared to the rendering of the analytic solution). The onset of the degradation of the FDM isosurfaces is more gradual than the SM cases. For instance, FD3310 is noticeably degraded for an isovalue = 0.9638, while $1-\varepsilon_{max} = 0.923$; whereas, SM3310 is noticeably degraded for an isovalue = 0.9923, while $1-\varepsilon_{max} = 0.991$. Also the SM isosurface degradation spread relatively uniform over the surface and is oscillatory. This was seen in the 1D filtered fits to the hyperbolic tangent function in Chapter 5. Lastly, Figure 6.5 shows by decreasing the filter threshold to 0.00001, the SM data set essentially tracks the analytic solution to the limits of the display.

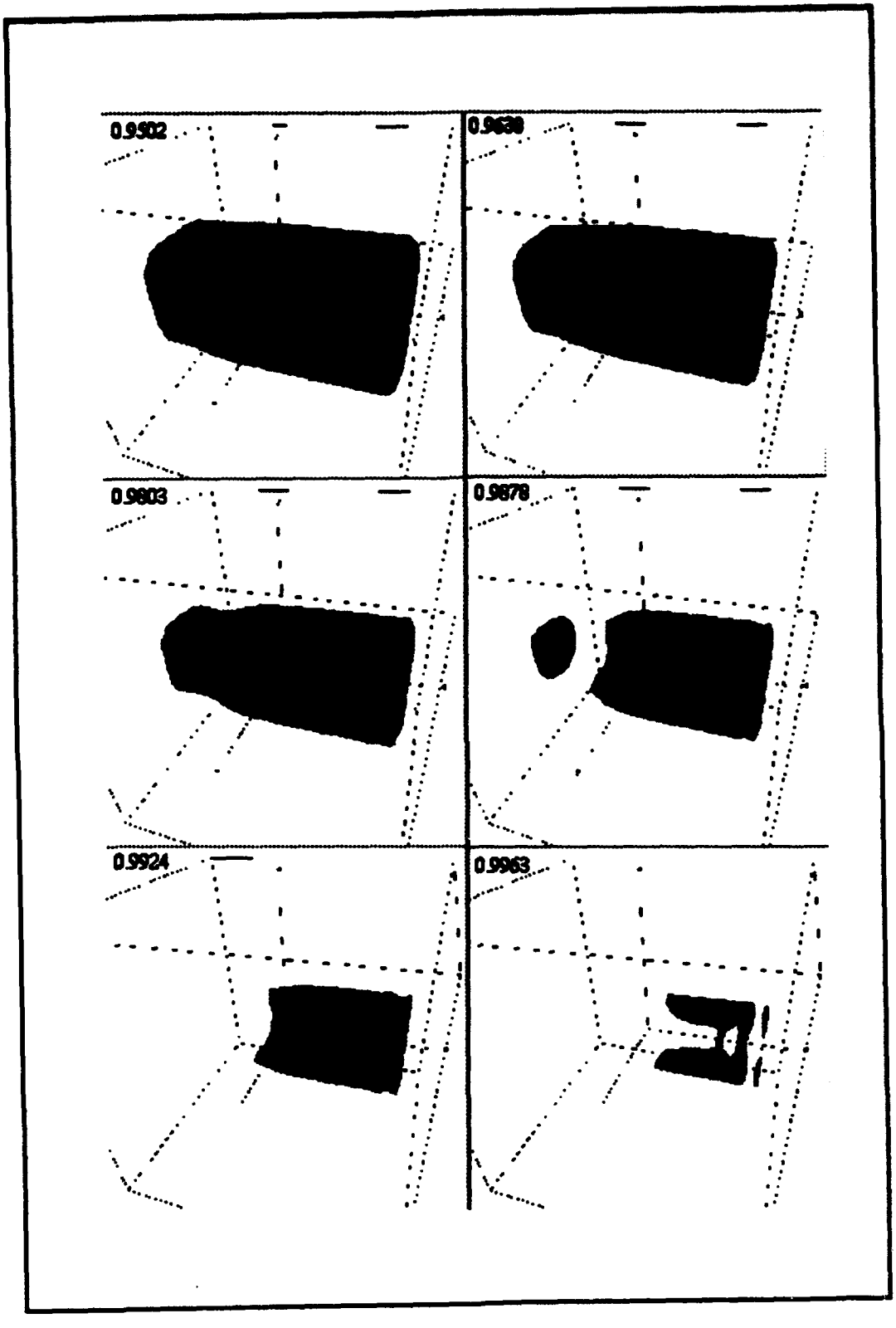

Figure 6.1. Isosurfaces for FD3310 Data Set, $1-\varepsilon_{max}=0.923$

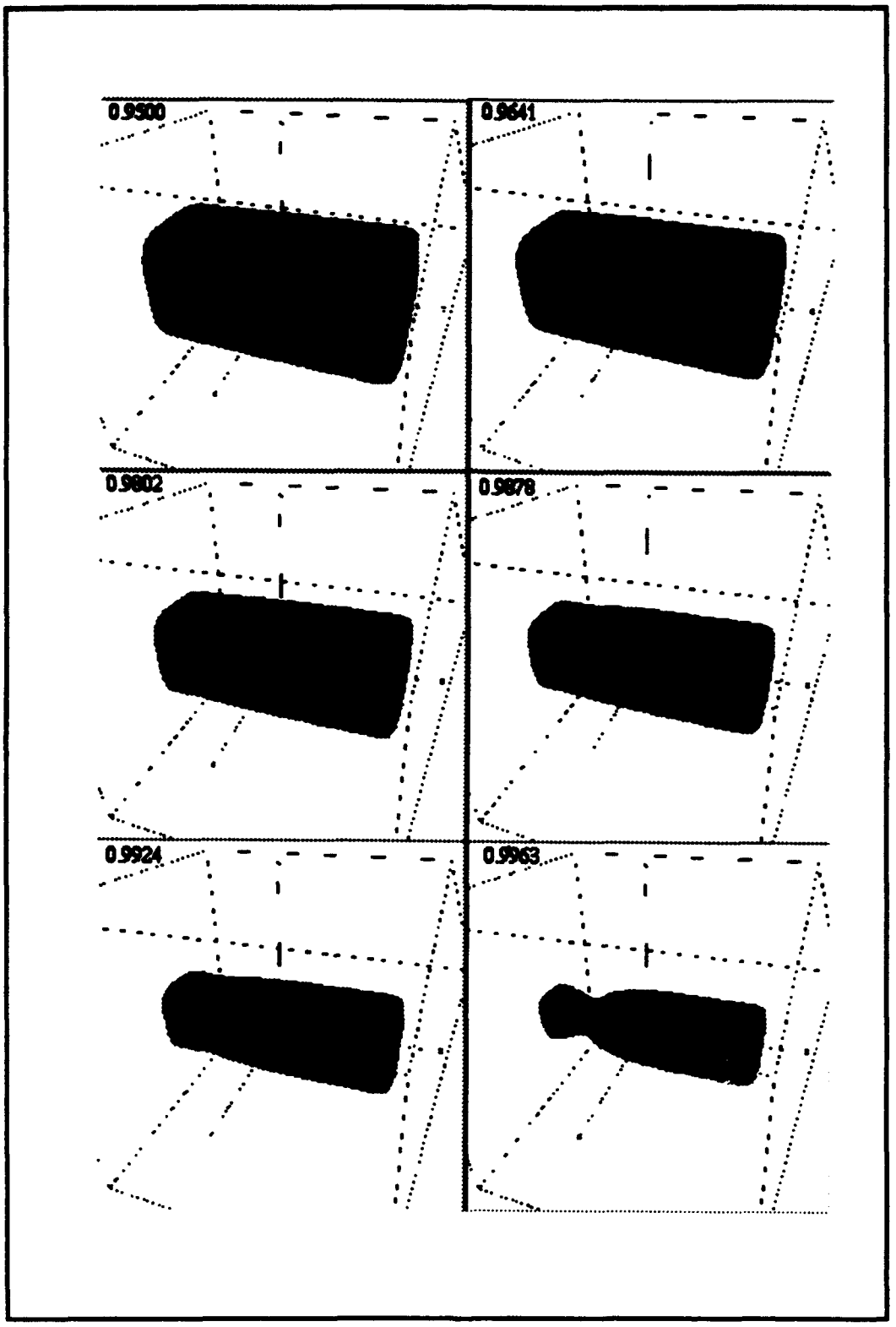

Figure 6.2. Isosurfaces for FD6510 Data Set, $1-\varepsilon_{max}=0.984$

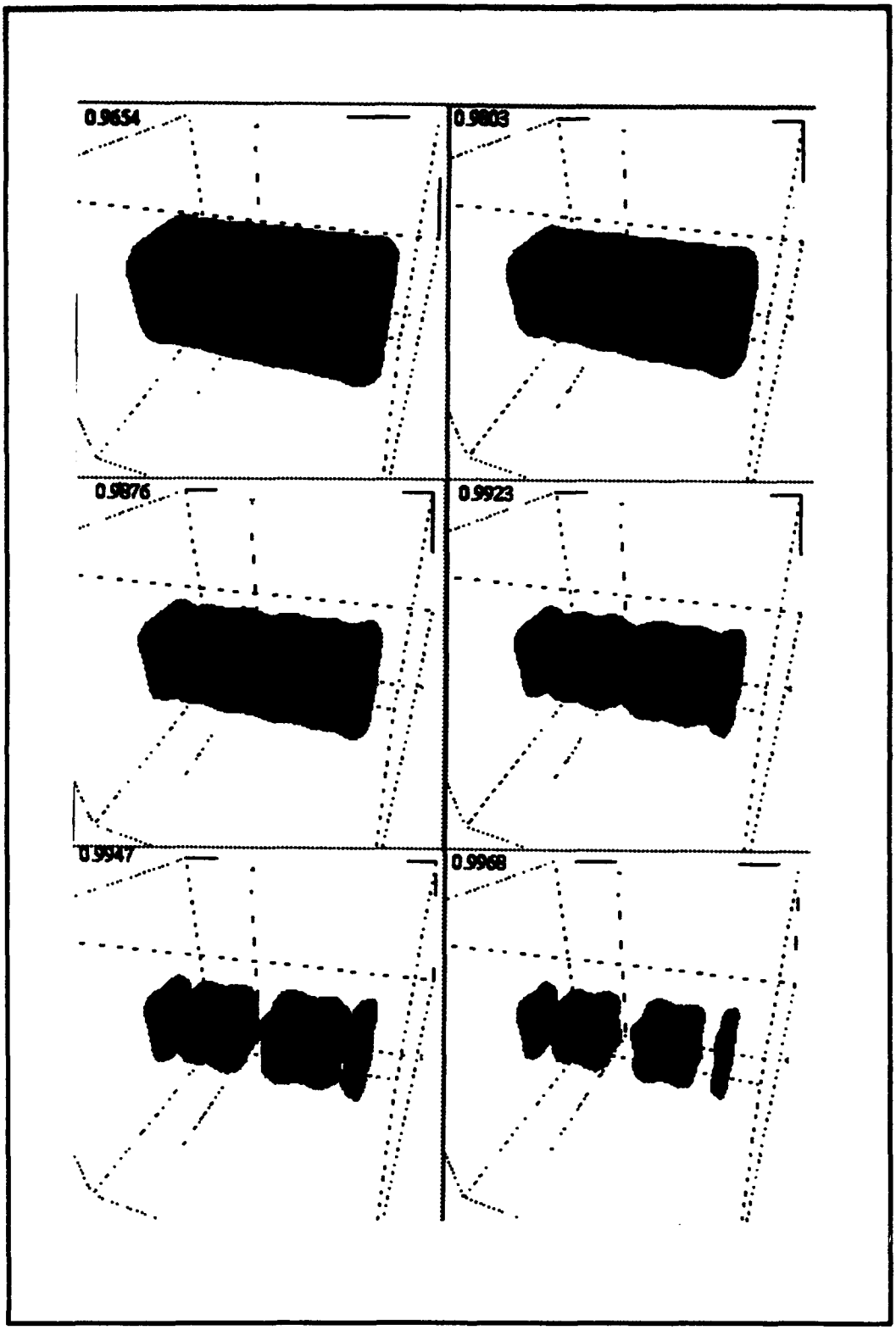

Figure 6.3. Isosurfaces for SM3310 Data Set, $1-\varepsilon_{max}=0.991$

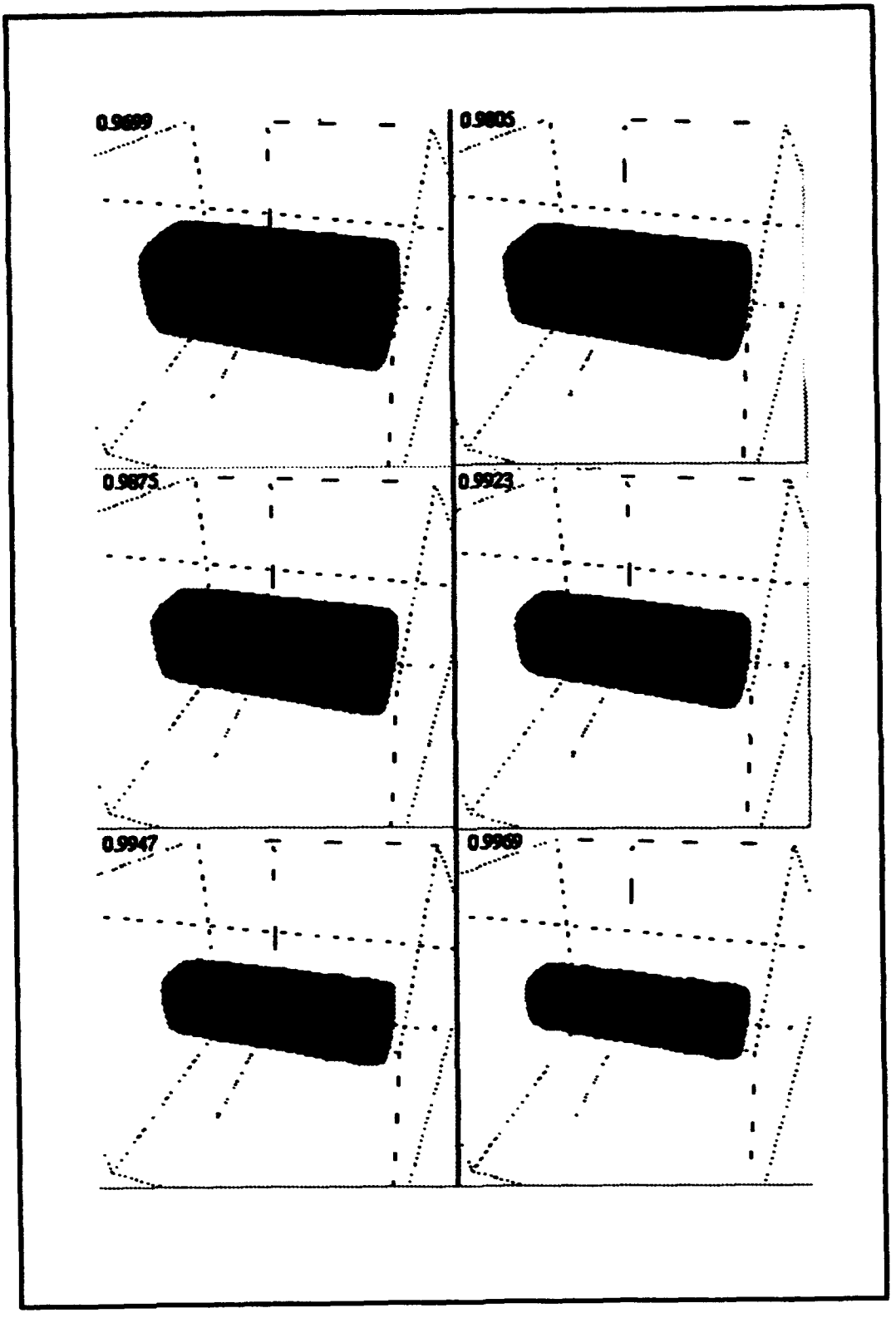

Figure 6.4. Isosurfaces for SM4910.1 Data Set, $1-\varepsilon_{max}=0.998$

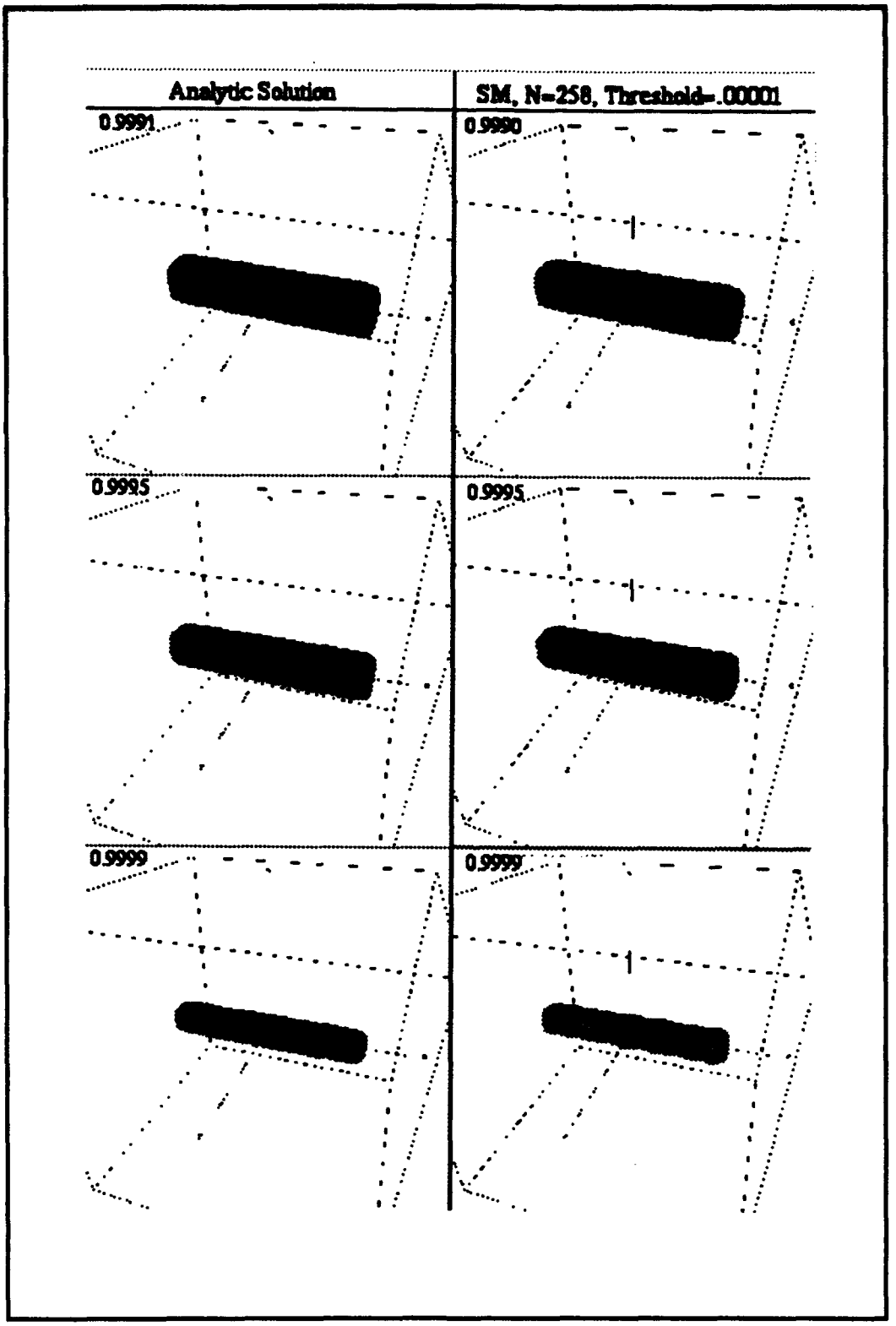

Figure 6.5. Isosurfaces for SM4910.01 Data Set versus Analytic Solution

6.3. Model Problem

Figure 6.6 portrays the effect of viewing multiple isosurface for the model problem. A cutting plane is employed to eliminate portions of the outer isosurfaces for unobstructive viewing of the inner isosurfaces. Currently, these surfaces are opaque. However, transparency could be easily implemented for an alternate viewing capability.

6.4. Driven Cavity

Figures 6.7 and 6.8 show various isosurfaces for the scalar value of the velocity vector field. Each node value is determine by the following relation: $q = \frac{1}{2}(u^2 + v^2 + w^2)^{\frac{1}{2}}$, where u, v, and w are Cartesian velocity components. The vortex structure is easily seen. In both figures, the xy cutting plane has been used at progressively increasing z locations for isosurfaces 1 -2 or 1 - 5 respectively. In Figure 6.9, the isosurfaces for N=65 case has relatively smoother surface than for N=33 case. This "jagged" effect is also apparent in the isosurfaces for the pressure field, as shown in Figure 6.10. The increased accuracy for N=65 is readily apparent in Figures 6.9 and 6.10.

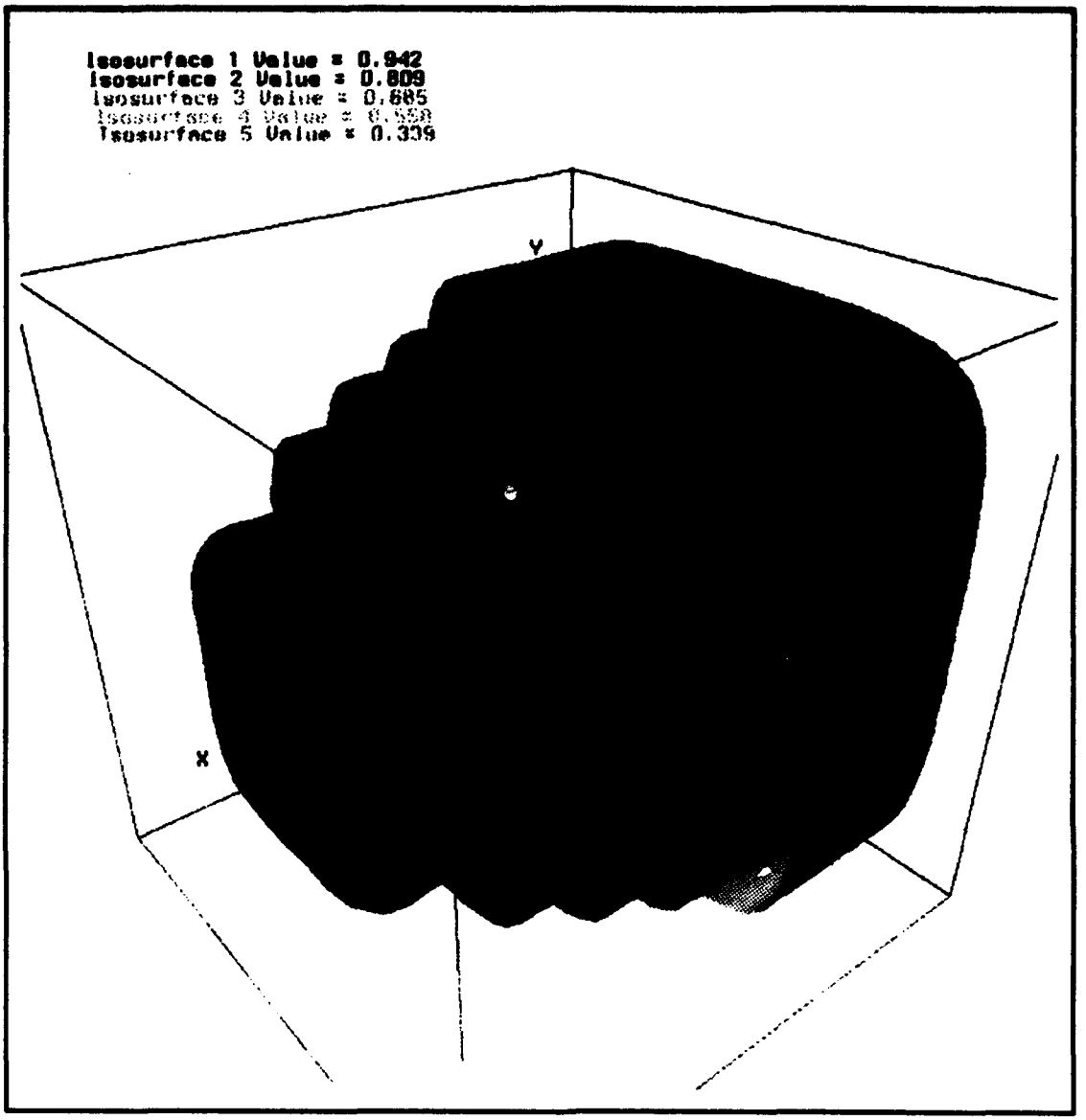

Figure 6.6. Cutaway Views of the Model Problem

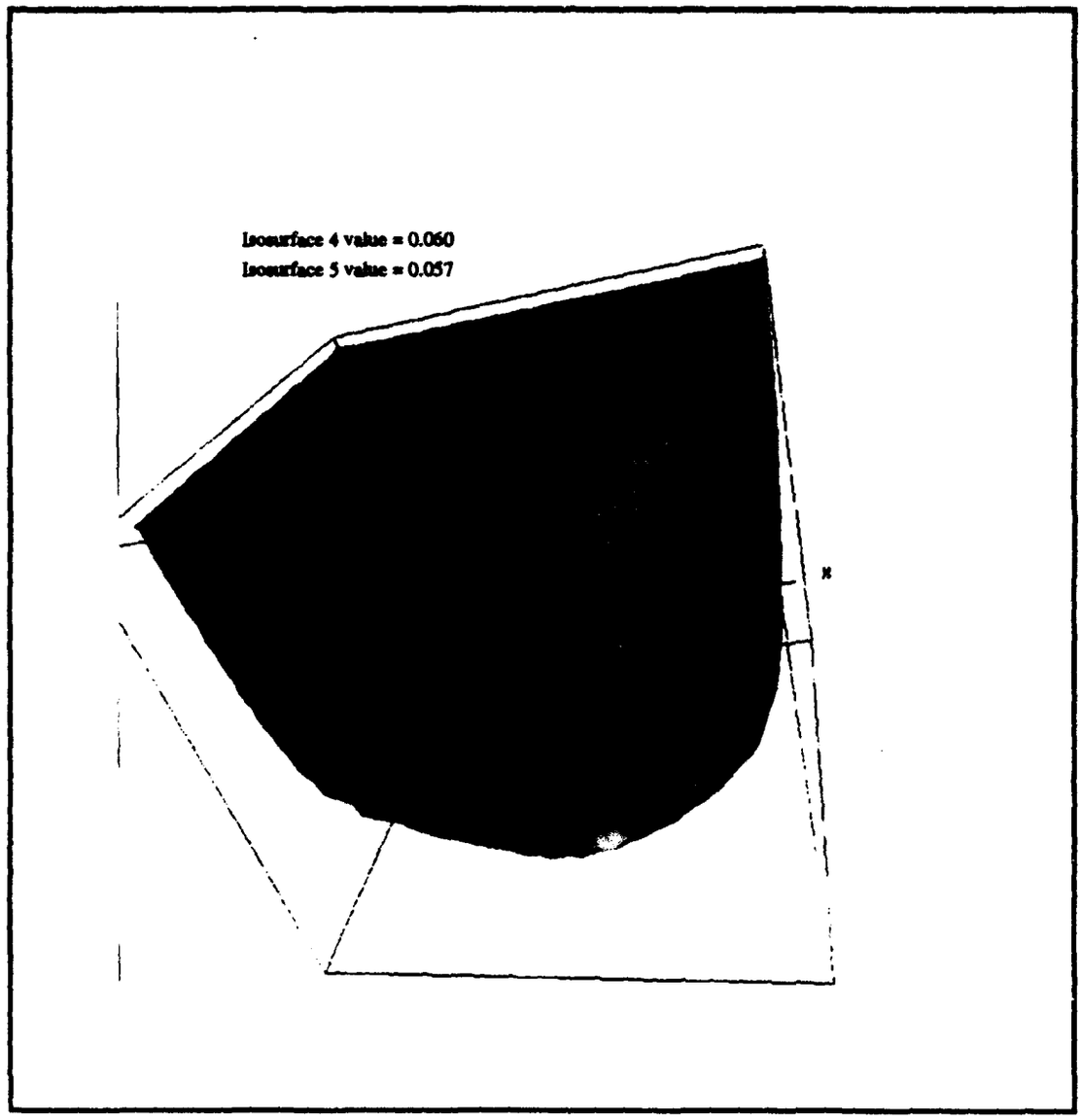

Figure 6.7. The Velocity Vortex Tube for the Driven Cavity

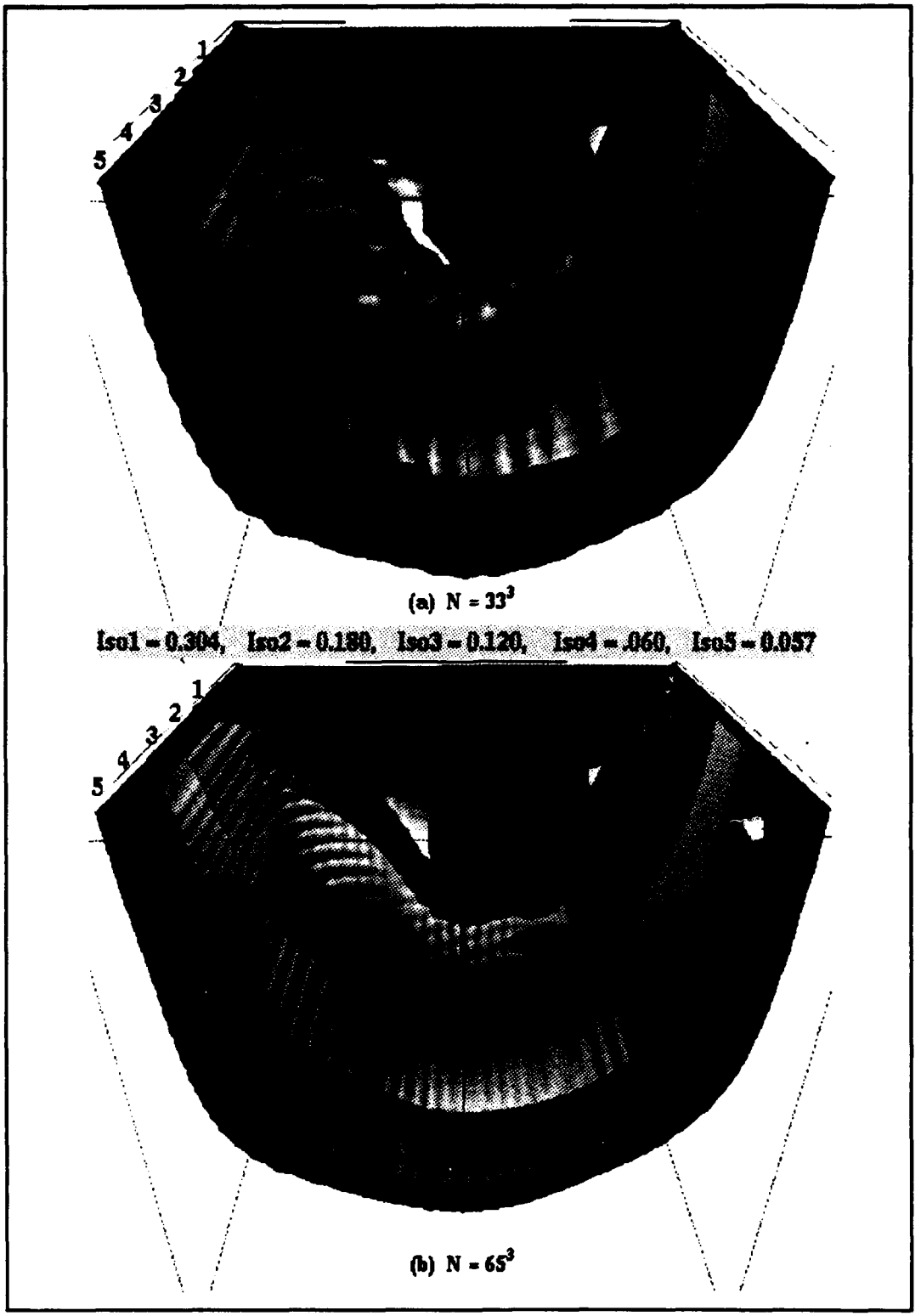

Figure 6.8. Multiple Velocity Isosurfaces for the Driven Cavity

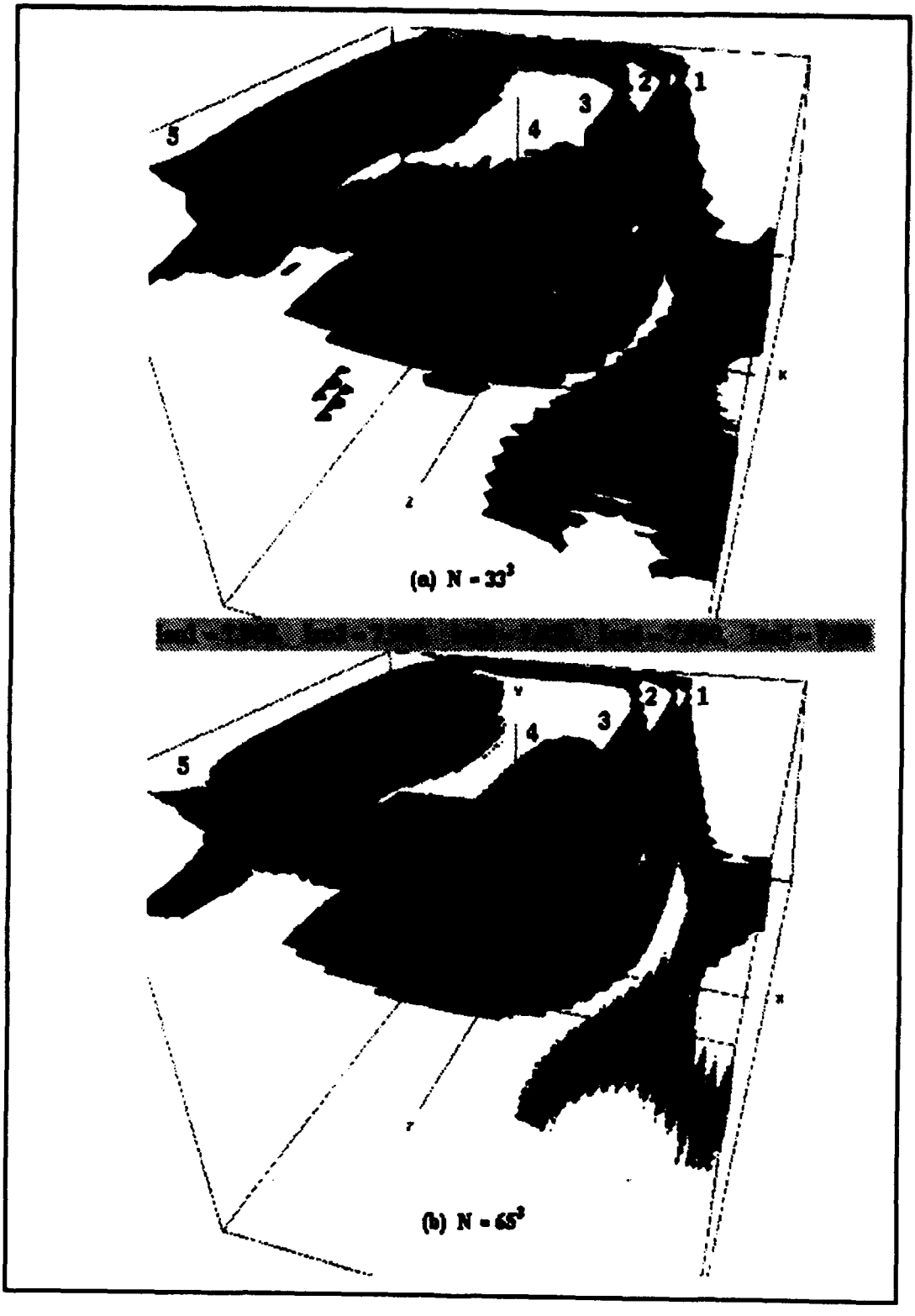

Figure 6.9. Multiple Pressure Isosurfaces for the Driven Cavity

6.5. Conclusions

Extrapolating the model problem results to the rendering of a spectral solution of a fluid flow is not straight forward. The model problem frequency content is inherently limited. The number of coefficients required for a flow field will undoubtedly be much larger. In this case, the FDM will have lower setup costs. The one advantage, which has not been fully exploited, is the ability to evaluate the SM solution at any arbitrary location in the domain. If the FDM solution is defined over a curvilinear domain (in which the node spacing is a function of two or more coordinate axes), then the rendering of the FDM solutions would require interpolation to avoid "cracks" in the isosurfaces. Then the rendering of FDM cases would significantly increase, while the time required for SM cases would remain relatively constant. This area warrants further investigation.

Appendix A: Flux Jacobians

The flux Jacobians are defined as

$$A \equiv \frac{\partial E}{\partial U} \qquad B \equiv \frac{\partial F}{\partial U} \qquad C^n \equiv \frac{\partial G}{\partial U}, \tag{A.1a,b,c}$$

$$R \equiv \frac{\partial E_{v_1}}{\partial U_x} \qquad S \equiv \frac{\partial F_{v_2}}{\partial U_y} \qquad T \equiv \frac{\partial G_{v_3}}{\partial U_z}, \tag{A.2a,b,c}$$

where the state vector of conserved variables, U, the Euler fluxes, E, F, G, and the components of the viscous fluxes, E_{v_1}, F_{v_2}, G_{v_3}, are defined in Eqs 2.4 - 2.13. Pulliam and Steger (1980:162) provide these 5x5 matrices in general form as follows:

$$A, B, C = \begin{bmatrix} 0 & K_A & K_B & K_C & 0 \\ K_A\phi^2 - u\theta & \theta - K_A(\gamma-2)u & K_B u - K_A(\gamma-1)v & K_C u - K_A(\gamma-1)w & K_A(\gamma-1) \\ K_B\phi^2 - v\theta & K_A v - K_B(\gamma-1)u & \theta - K_B(\gamma-2)v & K_C v - K_B(\gamma-1)w & K_B(\gamma-1) \\ K_C\phi^2 - w\theta & K_A w - K_C(\gamma-1)u & K_B w - K_C(\gamma-1)v & \theta - K_C(\gamma-2)w & K_C(\gamma-1) \\ \theta\left[2\phi^2 - \gamma\left(\frac{E_1}{\rho}\right)\right] & \left\{K_A\left[\gamma\left(\frac{E_1}{\rho}\right) - \phi^2\right] - (\gamma-1)u\theta\right\} & \left\{K_B\left[\gamma\left(\frac{E_1}{\rho}\right) - \phi^2\right] - (\gamma-1)v\theta\right\} & \left\{K_C\left[\gamma\left(\frac{E_1}{\rho}\right) - \phi^2\right] - (\gamma-1)w\theta\right\} & \gamma\theta \end{bmatrix}, \tag{A.3}$$

where

$$\phi^2 = \tfrac{1}{2}(\gamma-1)\left(u^2 + v^2 + w^2\right), \tag{A.4}$$

$$\theta = K_A u + K_B v + K_C w. \tag{A.5}$$

For a uniform grid spacing, to obtain A,

$$K_A = 1, \quad K_B = K_C = 0. \tag{A.6}$$

In a similar manner,

$$R, S, T = \frac{\mu}{\rho}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ -u(a_1 + a_5 K_E) & a_1 + a_5 K_E & 0 & 0 & 0 \\ -v(a_1 + a_5 K_F) & 0 & a_1 + a_5 K_F & 0 & 0 \\ -w(a_1 + a_5 K_G) & 0 & 0 & a_1 + a_5 K_G & 0 \\ -\begin{bmatrix} a_6 \frac{E_1}{\rho} + a_7 q^2 \\ + a_5\left(u^2 K_E + v^2 K_F + w^2 K_G\right) \end{bmatrix} & u(a_7 + a_5 K_E) & v(a_7 + a_5 K_F) & w(a_7 + a_5 K_G) & a_6 \end{bmatrix}, \quad \text{(A.7)}$$

where

$$a_1 = \frac{1}{Re} \quad a_5 = \frac{1}{3Re} \quad a_6 = \frac{\gamma}{PrRe} \quad a_7 = \frac{Pr-\gamma}{PrRe}, \quad \text{(A.8)}$$

$$q^2 = u^2 + v^2 + w^2. \quad \text{(A.9)}$$

Likewise, to obtain R for a uniform grid spacing,

$$K_E = 1, \quad K_F = K_G = 0. \quad \text{(A.10)}$$

Appendix B: Finite-Difference Equations for Gauss-Lobatto Points

For the Gauss-Lobatto points, $x_i = cos\frac{\pi i}{N}, \quad i = 0, 1, \ldots, N$, the second-order accurate finite-difference derivative operators are as follows:

- Central-Difference:

$$\delta_x f = \frac{-\alpha^2 f_{i-1} + (\alpha^2 - 1) f_i + f_{i+1}}{h_i(1+\alpha)} \quad \text{(Anderson, et al.,1984:55),} \tag{B.1}$$

$$\delta_{xx} f = \frac{1}{h_{i-1} h_{avg}} f_{i-1} - \frac{2}{h_{i-1} h_i} f_i + \frac{1}{h_i h_{avg}} f_{i+1} \quad \text{(Canuto, et al.,1987:144),} \tag{B.2}$$

where

$$\alpha = \frac{h_i}{h_{i-1}}, \quad h_i = x_i - x_{i+1}, \quad h_{avg} = \frac{h_i + h_{i-1}}{2};$$

- Forward-Difference:

$$\delta_x f = \frac{-\alpha(2+\alpha) f_i + (1+\alpha)^2 f_{i+1} - f_{i+2}}{h_i (1+\alpha)}, \tag{B.3}$$

where

$$\alpha = \frac{h_{i+1}}{h_i}, \quad h_i = x_i - x_{i+1};$$

- Backward-Difference:

$$\delta_x f = \frac{\alpha(2+\alpha) f_i - (1+\alpha)^2 f_{i-1} + f_{i-2}}{h_{i-2}(1+\alpha)}, \tag{B.4}$$

where

$$\alpha = \frac{h_{i-1}}{h_{i-2}}, \quad h_{i-1} = x_{i-1} - x_i;$$

The forward- and backward-difference operators are derived from a second-order polynomial fit.

Appendix C: Chebyshev Polynomial Properties and Derivative Matrices

The following properties of Chebyshev polynomial expansions are taken from Gottlieb and Orszag, (1977:Appendix A):

- The Chebyshev polynomial of degree n, $T_n(x)$, is defined by

$$T_n(\cos\theta) = \cos(n\theta). \tag{C.1}$$

  Thus, $T_0(x) = 1$, $T_1(x) = x$, $T_2(x) = 2x^2 - 1$, $T_3(x) = 4x^3 - 3x$, and so on.

- The Chebyshev polynomials are the solutions of the differential equation

$$\left(1-x^2\right)^{1/2} \frac{d}{dx}\left(1-x^2\right)^{1/2} \frac{dT_n}{dx} + n^2 T_n = 0 \tag{C.2}$$

  that are bounded at $x = \pm 1$. They satisfy the orthoganality relation

$$\int_{-1}^{1} T_n(x) T_m(x) \left(1-x^2\right)^{-1/2} dx = \frac{\pi}{2} c_n \delta_{nm}, \tag{C.3}$$

  where $c_0 = 2$, $c_n = 1$ for $n > 0$, and $\delta_{nm} = 0$ for $n \neq m$, $\delta_{nm} = 1$ for $n = m$.

- Some properties of Chebyshev polynomials are

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), \tag{C.4}$$

$$|T_n(x)| \leq 1, \tag{C.5}$$

$$T_n(\pm 1) = (\pm 1)^n, \quad T_{2n}(0) = (-1)^n, \quad T_{2n+1}(0) = 0. \tag{C.6a,b,c}$$

- The following formulae relate the expansion coefficients a_n in the series

$$f(x) = \sum_{n=0}^{N} a_n T_n(x), \tag{C.7}$$

  to the expansion coefficients b_n of the series

$$Lf(x) = \sum_{n=0}^{N} b_n T_n(x), \tag{C.8}$$

where c_n are defined in Eq C.3 and for various linear operators L. Two formulae are:

$$Lf = \frac{d}{dx} f(x): \quad b_n = \frac{2}{c_n} \sum_{\substack{p=n+1 \\ p+n \text{ odd}}}^{N} p a_p, \tag{C.9}$$

$$Lf = \frac{d^2}{dx^2} f(x): \quad b_n = \frac{1}{c_n} \sum_{\substack{p=n+2 \\ p+n \text{ even}}}^{N} p(p^2 - n^2) a_p. \tag{C.10}$$

Canuto et al. (1987:69) present the Chebyshev collocation derivative operator, D_N, in matrix form for the Gauss-Lobatto points, $x_i = \cos\frac{\pi i}{N}$.

$$D_N f(x_{ij}) = \sum_{j=0}^{N} (D_N)_{ij} f(x_j) \quad (i = 0, \ldots, N) \tag{C.11}$$

where

$$(D_N)_{ij} = \begin{cases} \dfrac{c_i(-1)^{i+j}}{c_j(x_i - x_j)} & i \neq j, \\ \dfrac{-x_j}{2(1-x_i^2)} & i = j, \end{cases} \tag{C.12a, b}$$

$$(D_N)_{11} = -(D_N)_{NN} = \frac{2N^2+1}{6}, \tag{C.12c}$$

$$c_1 = c_N = 2, \quad c_i = 1.$$

Appendix D: Natural Interval Extension for a Chebyshev Polynomial

In order to determine the natural interval extension for a Chebyshev series, the cyclic behavior or the 'monotonicity intervals' of the nth-order Chebyshev polynomial has to be taken into account.

> For example, an inclusion function evaluation method can be defined for the cosine operator, based on the observation that the cosine function is monotonically decreasing in the interval $[\pi 2n, \pi(2n+1)]$, and monotonically increasing in the interval $[\pi(2n+1), \pi(2n+2)]$, for integer n. (Snyder, 1992:123)

Similarly, a method for evaluating the natural interval extension of a nth-order Chebyshev polynomial is defined. Recalling Eq C.1, $T_n(\cos\theta) = \cos(n\theta)$ and $x = \cos\theta$, then

$$T_n(\cos\theta) = \pm 1, \tag{D.1}$$

when $n\theta = \pi p, \quad p = 0, 1, \ldots, n$. For example, the x positions of the 4th-order Chebyshev polynomial extrema are

$$\begin{aligned} &n = 4, \quad \theta = \tfrac{\pi}{4} p \quad (p = 0, 1, 2, 3, 4); \\ &\theta = 0, \quad \tfrac{\pi}{4}, \quad \tfrac{\pi}{2}, \quad \tfrac{3\pi}{4}, \quad \pi; \\ &x = 1, \quad 0.71, \quad 0, \quad -0.71, \quad -1; \\ &T_n = 1, \quad -1, \quad 1, \quad -1, \quad 1. \end{aligned} \tag{D.2a - d}$$

Given the following interval $X = [a, b] \Rightarrow \Theta = [\theta_a, \theta_b]$, then the natural interval extension of a nth-order Chebyshev polynomial is defined as

$$T_n([\theta_a, \theta_b]) \equiv \begin{cases} [-1, 1], & \text{if } \theta_b - \theta_a \geq \frac{\pi}{n} \\ [-1, \max\{T_n(\theta_a), T_n(\theta_b)\}], & \text{if } \theta_b - \theta_a < \frac{\pi}{n} \text{ and } p_a \text{ is even, } p_b \text{ odd} \\ [\min\{T_n(\theta_a), T_n(\theta_b)\}, 1], & \text{if } \theta_b - \theta_a < \frac{\pi}{n} \text{ and } p_a \text{ is odd, } p_b \text{ even} \\ [\min\{T_n(\theta_a), T_n(\theta_b)\}, \max\{T_n(\theta_a), T_n(\theta_b)\}], & \text{if otherwise} \end{cases} \tag{D.3}$$

where $p_a = int\left(\frac{n\theta_a}{\pi}\right), \quad p_b = int\left(\frac{n\theta_b}{\pi}\right)$.

Bibliography

Anderson, Dale A., et al., *Computational Fluid Mechanics and Heat Transfer*. New York: Hemisphere Publishing Corporation, 1984.

Beran, Philip Class handout, Aero 753, Advanced Computational Fluid Dynamics. Graduate School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB OH, Summer Quarter 1993.

Bloomenthal, Jules, "Polygonization of Implicit Surfaces," *Computer Aided Geometric Design*, Volume 5, Number 4: 341-355 (November 1988).

Canuto, et al., *Spectral Methods in Fluid Dynamics*. Berlin: Springer-Verlag, 1987.

Elvins, T. Todd, "A Survey of Algorithms for Volume Visualization," *Computer Graphics*, Volume 26, Number 3: 194-201 (August 1992).

Foley, J. D., van Dam, A., Fiener, S.K., and Hughes, J.F., *Computer Graphics:, Principles and Practice*, Addison-Wesley, 1990.

Fuchs, et al., "Optimal Surface Reconstruction from Planar Contours," *Communications of the ACM*, Volume 20, Number 10: 693-702 (October 1977).

Gargantini, I., "Linear Octrees for Fast Processing of Three-dimensional Objects," *Computer Graphics and Image Processing*, Volume 20, Number 4: 365-374 (December 1982).

Gottlieb, D., and Orszag, S.A., *Numerical Analysis of Spectral Methods: Theory and Applications*. Philadelphia: Society for Industrial and Applied Mathematics, 1977.

Herman, G.T., and Liu, H.K., "Three-dimensional Display of Human Organs from Computed Tomograms," *Computer Graphics and Image Processing*, Volume 9, Number 1: 1-21 (January 1979).

Keppel, E., "Approximating Complex Surfaces by Triangulation of Contour Lines," IBM Journal of Research and Development, Volume 19, Number 1: 2-11 (January 1975).

Kerlick, G.D, "ISOLEV: a Level Surface Cutting Plane Program for Fluid Flow Data," *SPIE Volume 1259, Extracting Meaning from Complex Data: Processing, Display, Interaction*, Volume 1259: 2-13 (1990).

Lorensen, W.E. and Cline, H.E., "Marching Cubes: a High Resolution 3D Surface Construction Algorithm," *Computer Graphics*, Volume 21, Number 4: 163-169 (July 1987).

Meyers, D., Skinner, S, and Sloan, K., "Surfaces from Contours," *ACM Transactions on Graphics*, Volume 11, Number 3: 228-258 (July 1992).

Nielson, G.M. and Hamann, B., "The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes," *Proceedings of the IEEE Visualization '91 Conference*, IEEE Computer Society Press: 83-91 (October 1991).

Pulliam, Thomas H., and Steger, Joseph L., "Implicit Finite-Difference Simulations of Three-Dimensional Compressible Flow," *AIAA Journal*, Volume 18, Number 2: 159-167 (February 1980).

Reddy, K. C., "Pseudospectral Approximation in a Three-Dimensional Navier-Stokes Code," *AIAA Journal*, Volume 21, Number 8: (August 1983).

Sabella, P., "A Rendering Algorithm for Visualizing 3D Scalar fields," *Computer Graphics*, Volume 22, Number 4: 51-58 (July 1988).

Snyder, J.M, "Interval Analysis for Computer Graphics," *Computer Graphics*, Volume 26, Number 2: 121-131 (July 1992).

Street, Craig L., et al., "Spectral Multigrid Methods with Applications to Transonic Potential Flow," *Journal of Computational Physics*, Volume 57: 43-76 (1985).

Suffern, K.G., and Fackerell, E.D., "Interval Methods in Computer Graphics," *Computer Graphics in Australia*, Volume 15, Number 3: 331-340 (1991).

Warming, R. F., and Beam, Richard M., "On the Construction and Application of Implicit Factored Schemes for Conservation Laws," *SIAM-AMS Proceedings*, Volume 11 85-129 (1978).

Westover, L., "Footprint Evaluation for Volume Rendering," *Computer Graphics*, Volume 24, Number 4: 367-376 (August 1990).

Wilhelms, J. and Van Gelder, A., "Topological Considerations in Isosurface Generation, Extended Abstract," *Computer Graphics*, Volume 24, Number 5: 79-86 (November 1990).

Wilhelms, J. and Van Gelder, A., "Octrees for Faster Isosurface Generation," *ACM Transactions on Graphics*, Volume 11, Number 3: 201-227 (July 1992).

Wyvill, G., McPheeters, C., and Wyvill, B., "Data Structure for Soft Objects," *The Visual Computer*, Volume 2, Number 4: 227-234 (August 1986).

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1993	Master's Thesis

4. TITLE AND SUBTITLE
RENDERING OF THREE-DIMENSIONAL DATA SETS DERIVED FROM FINITE-DIFFERENCE AND SPECTRAL METHODS

5. FUNDING NUMBERS

6. AUTHOR(S)
Paul A. Schubert, Major, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GAE/ENY/93D-24

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
WL/FIM
Wright-Patterson AFB, OH 45433

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The timely visualization of three-dimensional data sets and the advantages of using a spectral method solution versus a finite-difference method solution in rendering isosurfaces is described. The Beam-Warming numerical algorithm, which uses implicit-approximate-factorization, is used to generate the steady-state solutions for a model diffusion-convection problem. The Chebyshev collocation operator is used to evaluate the right-hand side of the Beam-Warming algorithm for the spectral solution. Comparing the model problem results with the exact solution, the spectral series solution is truncated to the same degree of accuracy as the finite-difference for comparison of rendering times. The rendering algorithm employs octrees to efficiently traverse the data set to fit the isosurfaces. The actual fitting of polygons to the isosurface uses the marching cubes table look up algorithm. With the spectral series solution, interval math is investigated for guaranteed detection of isosurfaces during the initial octree traversal(s).

14. SUBJECT TERMS	15. NUMBER OF PAGES
Finite-Difference, Spectral Methods, Numerical Analysis, Chebyshev Collocation, Computer Graphics, Octrees, Marching Cubes	83
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL