1998

Three-Dimensional Parallel Finite Element Simulation of Natural Gas Flow in a Porous Media.

James David Callahan
Louisiana State University and Agricultural & Mechanical College

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

THREE DIMENSIONAL PARALLEL FINITE ELEMENT SIMULATION OF NATURAL GAS FLOW IN A POROUS MEDIA

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

In

The Department of Computer Science

by
James David Callahan
B.S., McNeese State University, 1990
December 1998

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

A parallel-three-dimensional finite element simulation of natural gas flow in a porous media was developed for use in multiple well reservoirs. Developed with this simulation was a set of graphical applications to provide a geometric description of the reservoir, viewing of the generated mesh and viewing of results generated by the simulation. An adaptive mesh scheme for the dynamic refinement and derefinement of the mesh during the solution process is presented. The adaptive mesh scheme utilizes a mesh storage technique designed to reduce the space requirements of the mesh. This adaptive mesh technique was applied to an unstructured mesh. A parallel algorithm for the frontal solution technique was developed and implemented in C++ on small clusters of Microsoft Windows NT 4.0, Linux, and PowerPC workstations using MPI. The increased accuracy of these simulations was verified for the Eugene Island natural gas reservoir, located off the coast of Louisiana. The results for the Eugene Island example are more accurate than results from previous finite difference solutions for the same simulation.

INTRODUCTION

Based on the advantages of FEM, a finite element – dynamic adaptive – single phase – three dimensional – prototype simulator was designed and developed for this dissertation. Gas was chosen for the single phase to be simulated because of the availability of actual reservoir data. This gas flow simulation includes the modeling of multiple well-bore areas and has only minimal assumptions as to the physical characteristics of the governing equations, e.g. neglecting gravitational forces. The primary variable is the pressure within the reservoir, although the gas (condensate) and water phases are included in the calculation of the production rate. The reservoir pressure is approximated for selected discrete locations in three dimensions at many different - discrete times. The accuracy of the simulation is demonstrated by history matching results from an offshore Louisiana gas reservoir, Eugene Island.

The finite element prototype simulator for this research contains several features, which to our knowledge were previously unavailable for natural gas flow studies and simulations. This finite element prototype simulator uses an adaptive mesh in three dimensions. This mesh is stored using a new technique which only stores the nodes and pointers to their neighboring nodes (Wang98). This technique has been shown to reduce the storage requirements for a finite element mesh. Also developed were new mesh refinement / derefinement routines to allow the inclusion of more complex domains. For cost effective

visualization, an algorithm has been developed for the conversion of nodal-based finite element method meshes to VRML (virtual reality modeling language).

The prototype simulator has additionally been implemented on a distributed memory parallel architecture using the MPI (Message Passing Interface) communication libraries, MPICH/NT (Gropp96) version 0.9b. The MPI libraries are standard communication libraries, which support the use of a single set of parallel source code on multiple machine architectures (MPIF94). It is significant to note that distributed memory parallel architectures may have additional communication costs but their prevalence in the market place and flexibility in an enterprise make them ideally suited for industrial applications. MPI libraries are available for both distributed memory and shared memory parallel architectures.

PREVIOUS WORK

Adaptive Finite Element Method

The use of the finite element method (FEM) to obtain approximate solutions for differential equations is a popular area of research. The use of the FEM provides a more flexible domain and a more stable solution to many equations previously solved using finite difference techniques (Kocberber95, Pepper95, Löhner87). In addition, it has been shown that finite element grids provide many mathematical advantages in Control-Volume Finite Element (CVFE) schemes (Eymard93). Whether using compositional CVFE schemes or conventional FEM schemes, the flexible gridding techniques of finite element methodologies are useful in reducing storage requirements while providing adequate coverage of an entire reservoir.

An active finite element research focus is mesh generation. One reason for this is the effect the mesh has on the FEM results. Some issues involved in mesh generation are:

1. The number of dimensions (one, two or three) of the mesh required for the domain has an enormous influence on both the size and complexity of the problem.
2. The shapes of the element that form the mesh must be chosen. In 2D, rectangular and triangular elements are popular, and their extensions to 3D are also frequently used.

3. The mesh storage structure profoundly affects the amount of memory used, the CPU time necessary for assembly and the complexity of the programming. Hierarchical structures (e.g. quadtrees, octrees) and hybrid storage structures are two popular choices for current FEM work.
4. Mesh adaptation is used to reduce the CPU time needed and to make more efficient use of memory. The adaptation of the mesh can occur either statically (before FE calculation), dynamically (during FE calculation) or both.

Adaptive FEMs reduce both the storage and CPU time required to more accurately solve a given physical problem (Löhner87). Although two-dimensional adaptive FEM simulations were being used during the 1980s, the extension of these methods into three-dimensional domains has been a very recent phenomenon. For truly three-dimensional approaches to be viable, the adaptive procedures must not require more storage and CPU time than can be saved during the assembly and solution of the element equations. Several different truly three-dimensional adaptive procedures are being studied that may meet this requirement (Pepper95, Deb95, Sethi95).

An adaptive mesh for FEM provides additional elements in areas where there are large changes occurring (less stable) and reduces the number of elements in areas that are less active (very stable). Static adaptive methods provide additional elements during the initial mesh generation in areas that are known to be less stable. Dynamic adaptive methods modify the mesh during the whole solution process by adding or removing elements from the mesh using:

- Relocation – Elements are moved from areas of less activity to areas where additional elements are needed.
- Remeshing – The elements of the mesh are recreated using an automatic mesh generation procedure that adds additional elements in areas of high activity.
- Enrichment – For *h*-adaptive enrichment, elements of the mesh are split into smaller elements. *p*-adaptive enrichment involves increasing the order of the element shape functions. Combinations of these two different techniques may be used.

The relocation of elements can be very effective for some problem domains. Because of the overhead associated with insertion or removal of elements and the regeneration of the mesh, there are advantages in only relocating nodes (Löhner87, Hassan98). When too many nodes require modification, this method is no longer effective and other mesh adaptation methods are better.

Any remeshing technique requires an automatic generation of the mesh during the solution. Remeshing techniques have been developed for use with both hexahedral (Greaves98) and tetrahedral (O'Dwyer97, Okuda97, Peraire87) elements. As the error in the solution increases, using these techniques, the mesh is regenerated with additional elements reducing the error. Any information that is needed from the previous mesh has to be mapped into this new mesh. The complexity of three-dimensional meshes makes this technique very difficult but desirable. The extensions of the two-dimensional advancing front (Jin93),

Delaunay triangulation (Üler94, Schroeder88, Schroeder90) and other hybrid systems employing different gridding techniques have enjoyed some success (Burnett87). Though effective when areas of the mesh are changing, remeshing itself consumes a considerable amount of CPU time for mesh generation and transfer of information between meshes (Connell94).

Mesh enrichment schemes used are most efficient when the changes take place in a small percentage of the total domain. Since mesh enrichment time can be excessive if the mesh is adapted often, the error evaluation scheme must be efficient and reliable (Löhner87, Zienkiewicz87). *H*-adaptive (Pepper95, Tani97, Sethi95, Ding93), *p*-adaptive and combined *ph*-adaptive (Deb95) procedures have all been used.

The data structure used with these adaptive schemes has an impact on the memory and CPU time required. The common storage classes used to store meshes are element-based structures using octrees or quadtrees (Greaves98, Jin93, Deb95), face/edge/node (Connell94) structures and node structures (Ruede93, Wang98). It is very important with adaptive mesh procedures to be *able to quickly find neighboring elements (determine valid connectivity between* elements before refinement). The conservation of the memory is always important. Face/edge/node structures need large amounts of memory and are difficult to maintain but are excellent for finding neighboring elements. Tree structures are very useful in maintaining a refinement history but are not efficient when finding neighboring elements. Node structures provide an efficient method to store meshes but finding neighboring elements can be time consuming;

however algorithms have been developed to facilitate their use in adaptive procedures.

Most adaptive schemes that use three-dimensional elements do only a two-dimensional adaptation. These are not truly three-dimensional. The vertical component of each element is limited in how it can be enriched. For example, in some schemes the addition of vertical elements requires the addition of a whole new layer of elements. The use of a new *h*-adaptive enrichment scheme is presented in this dissertation as part of our natural gas simulation prototype. It is completely three-dimensional and is a flexible method for the addition of elements to the mesh. Our adaptive simulation uses a node based storage structure that provides flexible and efficient use of memory (Wang98). This node based storage structure reduces the redundancy of nodal information stored in multiple elements, but adds some complexity to the mesh structure.

Eugene Island Reservoir

The techniques that are developed have been implemented in a prototype reservoir simulation of the Eugene Island condensate field. This reservoir is a working condensate field off the Louisiana gulf coast. We selected the Eugene Island gas reservoir because we had the necessary data for the reservoir and its production history. Eugene Island, Block 305, is a gas-bearing sand reservoir at a depth of 10,300 feet in the Louisiana gulf coast region. The reservoir is owned by Chevron USA, which has allowed the use of their data for this reservoir. Production of gas from the Eugene Island reservoir started in August of 1979. There are six wells within the reservoir that are produced. The Eugene Island,

Block 305 has been the subject of work in the Department of Petroleum Engineering at Louisiana State University. They have provided summary information related to production and performance of the Eugene Island gas reservoir.

In 1987, Doreen Arcaro and Zaki Bassiouni used the Eugene Island reservoir, Block 305 as a basis for a detailed economic and technical feasibility study (Arcaro87). This study was centered around the use of co-production to extend the life of a reservoir. During this study the reservoir was simulated to determine the reservoir pressure and the cumulative gas production. It was predicted that the co-production technique would increase the life of the reservoir by six years. The prediction was made using a material-balance approach, which approximates the production without using the geometry of the reservoir.

Keith Halford presented a simulation of the Eugene Island reservoir in his 1985 master's thesis (Halford85). Halford used a radial reservoir simulator in this study. Pressure throughout the reservoir was averaged and solved using a simple tank model program based on material balance. The Van Everdingen and Hurst's aquifer model was selected based on the radial geometry. In order to determine the best history match, the permeability of the reservoir and the angle open to flow measurement was varied. The best solution was compared to the actual pressure averages for all wells. A resulting minimum error of 170 psi was obtained.

The accurate simulation of this reservoir was of interest to the co-production study of the LSU Petroleum Engineering Department. The results presented in this work show a very good match with actual data. The adjustment in the permeability and selection of the reservoir geometry to achieve a good history match are discussed in this dissertation.

A TRULY THREE-DIMENSIONAL ADAPTIVE FINITE ELEMENT METHOD

Coordinate System and Terminology

The coordinate system used for our new adaptive procedure, the mesh generation, and refinement algorithms are presented in this section. The mesh used for the FE reservoir simulation was a three-dimensional Cartesian coordinate system. The x and y axis form the plane parallel to the surface. The z-axis is indicative of depth and increases as depth below the surface increases. Figure 1(a) shows the orientation of these axes. For consistency, all references to directions of any location within the mesh uses the mesh origin. Given this orientation, the "right/left" of a node means an increasing/decreasing X direction from the node; in "front/back" of a node implies an increasing/decreasing Y direction from the node; "above/below" a node implies a decreasing/increasing Z direction from the node.

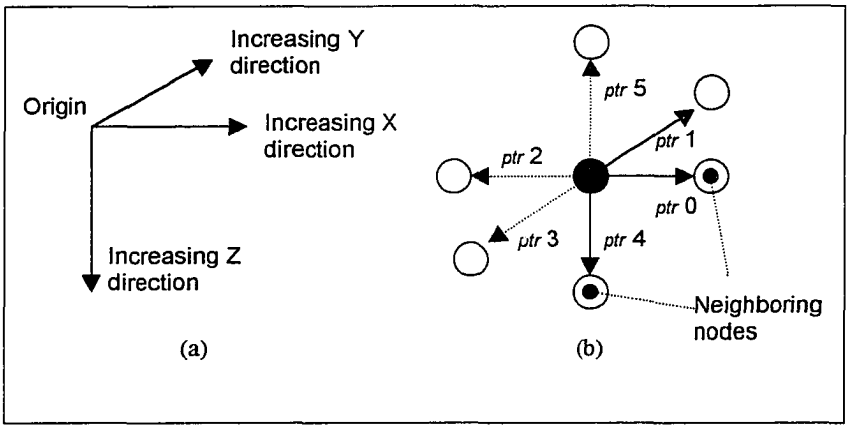


Figure 1 – (a) Coordinate system used (b) Directional pointers to neighboring nodes

For each node, a geographic 3-tuple (x, y, z) is maintained. In addition to the nodal 3-tuple, six pointers to nearest neighboring nodes, a pointer to the parent element (refinement history), the refinement level, the permeability and the current pressure are stored for each node. The node-based mesh (Ruede93, Wang98) is formed by the interconnection of these nodes. The six pointers (shown as arrows on Figure 1(b)) connect a node (shown as solid black) with the neighboring nodes. It is possible to traverse these pointers to any other node; therefore the mesh is fully connected. The node-neighbor pointers are labeled *ptr* 0 to *ptr* 5 on Figure 1(b). The pointer *ptr* 0 connects the node to the neighboring node that is to the "right" (increasing x direction) of the node. The pointer *ptr* 1 points to the neighboring node that is in "back" (increasing y direction) of the node. The pointer *ptr* 4 points to the neighboring node that is "below" (increasing z direction) the node.

Mesh Generation

The initial mesh contains a structured set of elements that partition the domain. An initial mesh is usually coarsely generated and serves as the basis for future adaptation. Describing the geometric boundaries of the domain for mesh generation is often tedious. The process used to generate our initial mesh is similar to the automatic generation algorithm presented by Schneider (Schneider95).

For complex domains, some additional guidelines have been developed which maintain a consistent and extensible mesh. The guidelines are not always required for the use of the mesh in well-block regions, but are a requirement for

reservoirs that are made up of a collection of sub-meshes. These guidelines provide a basis for the combination of multiple sub-meshes into a large mesh. For demonstration purposes, we use the two dimensional Figure 2. The guidelines however extend naturally into three dimensions.

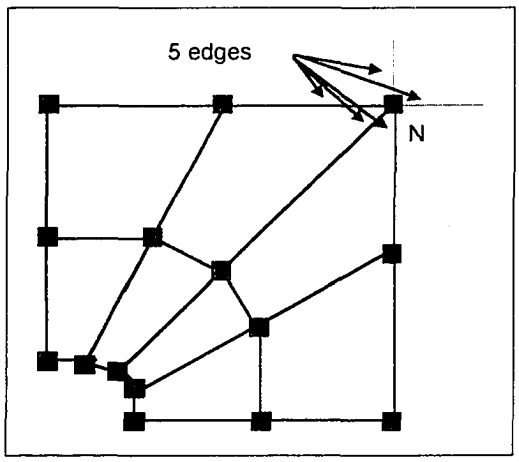


Figure 2 – Embedded sub-mesh in larger problem

1. It is necessary to maintain the consistency of directional node pointers. The consistency of directional node pointers, means that all subsequent links are made in the desired direction. All grids must be fully connected (see the two dimensional "radial" domain in Figure 2). If we embed the grid in a larger mesh, the problem shown in Figure 2 could occur. There are five directional node pointers from node **N**, but there are only 4 node pointers for each node object in two dimensions (6 node pointers for three dimensions) and this is inconsistent.
2. It is necessary to maintain the connectivity of the node pointers. If any of the node pointers are removed then the desired connectivity is lost. Mislinked or

unlinked node pointers destroy the structure of the mesh. Each of the directional node pointers must either be connected to a neighboring node or be a null pointer.

3. It is necessary to maintain 1-irregular connectivity. The connectivity rule for 1-irregular elements requires that the refinement level of an element cannot differ from the refinement level of the neighboring elements by more than one. When refining an element, it is necessary to check the elements that are neighbors to see if their refinement level differs by more than one from the original element of interest. In this cases, either the element cannot be refined or the neighboring elements that differ by more than one must be refined before continuing.

To enter and describe the reservoir domain for subsequent mesh generation, a graphic user interface has been developed. This graphic interface supports the interactive entry of hexahedrons. Each hexahedron defines a reservoir volume (part of the domain) plus stores information about the geometry of the reservoir and the permeability of its media. The combination of this information provides the data for the entire reservoir.

As hexahedrons are added and removed, the display is updated to show their relative position and size. The graphical interface is very useful and provides an interactive visualization of the reservoir. Although the computer display provides an interactive visualization of the reservoir, the user still has the following two responsibilities:

1. Hexahedrons must be added so that they are fully connected. Any holes within the reservoir are not automatically detected in this initial grid.
2. The placement of hexahedrons must be planned so that they properly model any wells within the reservoir.

A hexahedron *b* is defined by its location (X_b, Y_b and Z_b) and size (W_b, D_b and H_b) and each hexahedron is so defined. The bounding region of the reservoir is first determined by finding the minimum X_b, Y_b and Z_b of all hexahedrons and the maximum X_b+W_b, Y_b+D_b and Z_b+H_b. Figure 3 shows an example of a bounding region for a set of hexahedrons. The following pseudo code demonstrates the procedure used to create new mesh from the hexahedrons:

Procedure GenerateMesh

LOOP zCoord FROM *minimum* Z_b TO *maximum* Z_b+H_b STEP *element height* DO
  LOOP yCoord FROM *minimum* Y_b TO *maximum* Y_b+D_b STEP *element depth* DO
    LOOP xCoord FROM *minimum* Y_b TO *maximum* Y_b+W_b STEP *element width* DO
      IF location (xCoord, yCoord, zCoord) *within a hexahedron* THEN
        Create a new node
        Traverse the list of current nodes to find (xCoord – *element width*, yCoord, zCoord)
        IF *neighbor node is found* THEN
          [*new node*].ptr 2 = [*neighbor node*]
          [*neighbor node*].*ptr* 0 = [*new node*]
        ENDIF
        *{similarly connect ptr 3 with neighbor at (*xCoord, yCoord – *element depth,* zCoord*)}*

*{similarly connect ptr 5 with neighbor at (*xCoord, yCoord, zCoord – *element height)}*

ENDIF

ENDDO

ENDDO

ENDDO

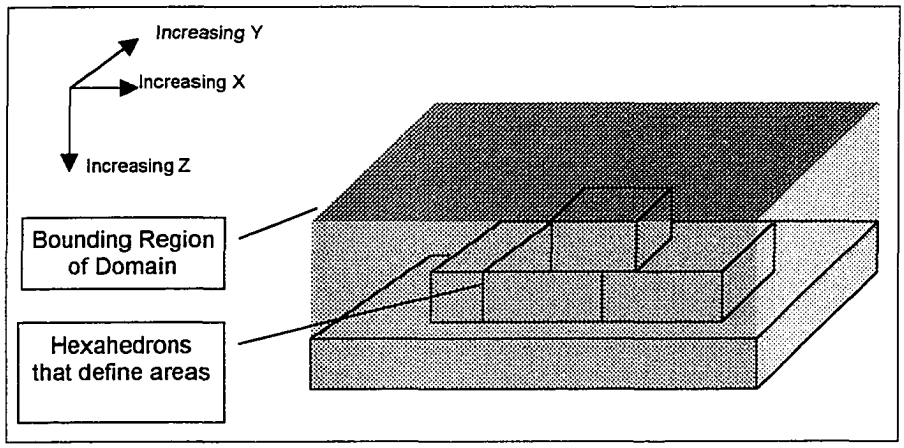


Figure 3 – Geometric description of reservoir

Every potential node inside the enclosed reservoir region and within a hexahedron is added to the mesh. The addition of nodes to the mesh proceeds in an increasing direction from the origin ("front", "upper", "left" corner of bounding region). The only neighboring nodes that need to be linked are the neighbor node pointers that point in a decreasing direction (*ptr* 2, *ptr* 3 and *ptr* 5 of Figure 1(b)). After these three pointers are set, the increasing direction pointers of the neighboring nodes (*ptr* 0, *ptr* 1 and *ptr* 4) are set to point to the new node.

Mesh Adaptation

Hexahedral Transition Element

The three-dimensional transition element used for the FE simulation is a bilinear hexahedral element with 26 nodes (Morton95). Since the element contains nodes on its edges, it permits connectivity of elements when they are adapted. However, there is only one node on each edge so that no more than two elements can be connected on that edge. Enrichment schemes that split the element into three smaller elements (Ding93) cannot be used with this element. Before adapting an element, it is necessary to determine if its neighbors have the same refinement level (number of times the element has been refined) and ensure that it will not invalidate the element's connectivity. Figure 4 shows the node locations and the numbering scheme for the transition element.

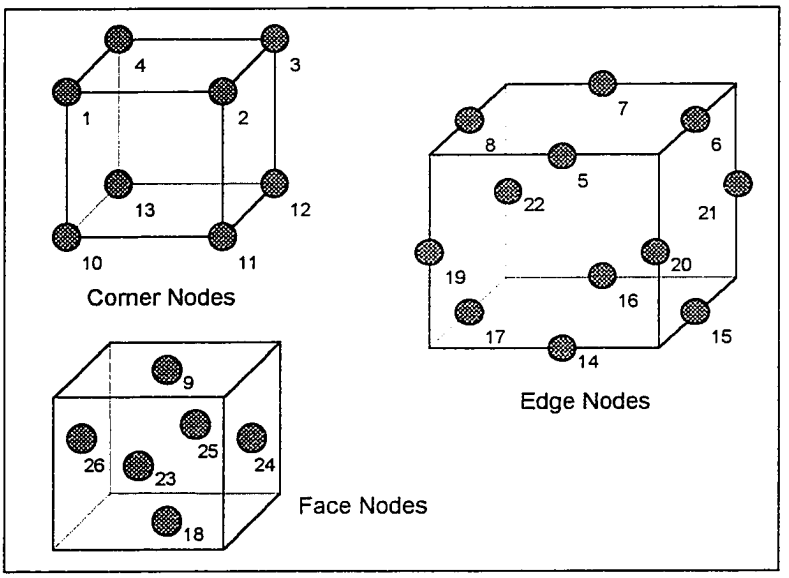


Figure 4 – Transition element node locations and numbering scheme

A big advantage of the nodal storage is that the elements of the mesh do not remain in memory throughout the computation. This conserves space since the elements are assembled only when they are needed by the FEM calculation and then removed. This element assembly process is very efficient and uses a minimal amount of CPU time (Wang98).

Adaptation Criterion

During the simulation it is necessary to determine where and when additional elements are needed. The addition of elements increases the accuracy of the solution (decreases the error). The regions in close proximity to wells in reservoirs often have high rates of flow, and within these regions, additional elements are beneficial. The pressure gradient for gas flow (Eq. 1) was a good indication of the need for refinement.

$$\nabla P^{t(e)} = \left(\bar{\hat{P}}^{t(e)} - \bar{\hat{P}}^{t-1(e)}\right) \qquad \text{Eq. 1}$$

For each time step, the element pressure is calculated, the gradient is formed and compared to a predetermined threshold value. If the gradient change is larger than this threshold value, the element is marked for refinement.

A change in the production rate of a well causes spikes in the gradient that can result in the mesh being refined (or derefined) many times for a single time step. This can lead to an over utilization of mesh adaptation, which reduces the efficiency of the solver. This phenomenon is controlled by limiting the adaptation of the mesh to one refinement or derefinement within each time step of the simulation. This was an effective deterrent to the over adaptation of the mesh.

Given a finite element mesh for the gas reservoir simulation, the adaptation of the mesh uses the following procedure.

1. Calculate the finite element solution for the mesh.
2. Mark elements that have a pressure gradient larger than the predetermined thresholds and can be refined (or derefined). For odd numbered time steps, check for pressure gradients less than the lower threshold, and for even numbered time steps, check for pressure gradients greater than the upper threshold value.
3. Modify the FE mesh by adding or removing nodes (refining or derefining the mesh).
4. Check for the completion of the simulation, and if not complete, return to step 1 to calculate the next time step.

h-Refinement

The *h*-refinement of an element is accomplished by splitting the element into eight new elements. By adding additional elements, the discretization error of the problem is reduced. The following three tasks refine the element:

1. Each of the elements in the mesh is checked to determine whether it needs to be refined.
2. The elements that need to be refined are checked to verify that their neighbors are at the same refinement level or one refinement level higher.
3. The element is split by adding new nodes at its center and along its edges. The nodes are then reconnected to neighboring nodes.

This adaptation criterion is used to determine if an element should be adapted. When an element meets these conditions, it must also be checked to determine if its adaptation would invalidate the connectivity of the mesh and all its neighboring elements must be located to determine their refinement level. Since the neighboring elements of the mesh are not available in our storage scheme, we use the nodal connections to determine the neighboring elements.

The parent node of an element is defined as the node from which the element is assembled (node 1 in Figure 4). Each neighboring element is determined by the parent node used to assemble that element. The parent nodes whose assembled element shares a node with the element to be adapted (except for neighboring corner elements), are the parent nodes of neighboring elements. Therefore, to find the neighboring elements is to find the parent node of the neighboring elements. The search for neighboring nodes can be very complex because most meshes are irregularly shaped. A systematic search technique was developed and based on the structure of the bilinear element. The bilinear element has at most one node between the endpoints of any edge, i.e., at most six node connections (two in each dimension) separate parent nodes of neighboring elements. For each node in the element being refined, a search of the neighbor node pointers of depth six (two in each direction) is performed and a list of parent nodes is built. Before being added to the list, it is verified that the parent node shares a node with the element being adapted. This search for neighboring elements is only necessary in the decreasing direction on each axis because the nodes of an element are in the increasing direction from

the parent node of the element. This reduces the amount of time needed to search for neighboring elements.

After determining the neighbors of the element, the diagonal neighbors are removed from consideration since their proper connectivity is implied with the proper connectivity of the adjacent edge and face neighbors. Each of the remaining edge and face neighbors' refinement level is checked. For two elements to have a valid connectivity, their refinement level must differ by no more than one. We verify that the element has not been refined more than any neighboring element. Each element's parent node stores the refinement level of an element and this can be checked without additional computation.

After the list of elements to be refined is built, each of these elements is split into eight new elements and their neighboring node pointers updated. Figure 5 depicts the steps of the refinement procedure for each element. The element refinement starts with the original element Figure 5(a) and ends with element being divided into smaller elements Figure 5(d). As nodes are added, their permeability and pseudo-pressure are determined (averaged) from neighboring nodes.

To refine an element:

1. The existence of edge nodes numbered 5-8, 14-17 and 19-22 for the element are checked. If a node does not exist, then add a node at the location. The neighboring node pointers of the new node and the neighboring nodes must be updated.

2. Check the element for the existence of the face nodes numbered 9, 18 and 23-26. If a node does not exist then it must be added.
3. Add a node into the center of the element (this node does not have a number in the element). The neighboring node pointers for this node are connected with the six nodes that "surround" it.
4. The parent nodes of the new elements are the nodes numbered 5, 8, 9, 19, 23, 26 and the new center node. These nodes set their parent element pointer to the parent node of the original element (node number 1). For an example of this, see Figure 6.
5. The refinement level of the refined element and the new elements is increased by one.

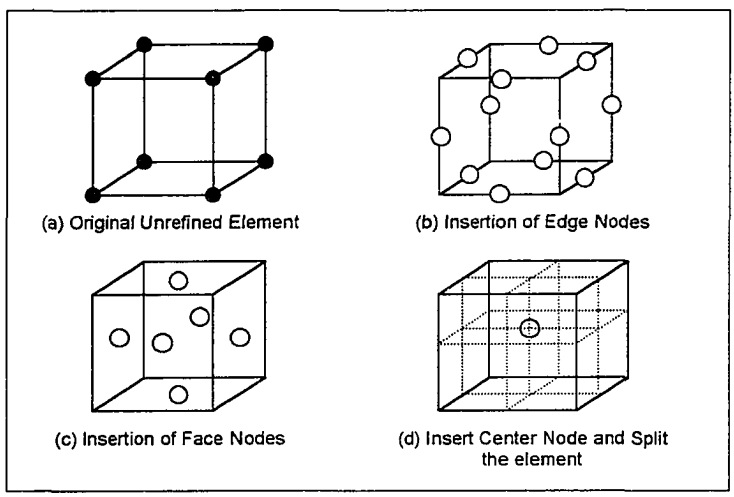


Figure 5 – Steps of element refinement

Derefinement

The derefinement of a set of elements is similar to a reversal of the refinement process. Elements that were split from one element are joined with

the same element. The history of the refinement is maintained in the parent element pointers. The derefinement of the mesh is the reversal of this history for each refined element. Though the ordering of derefinement for distinct groups of elements may be different from their initial refinement, they are joined with the same parent.

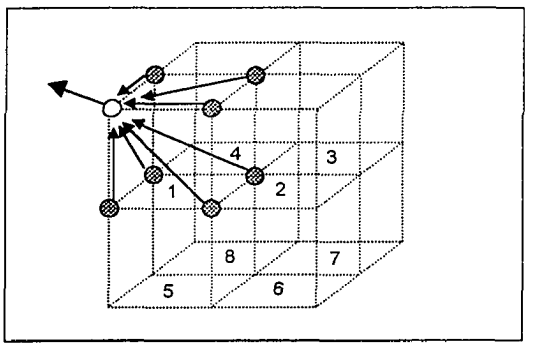


Figure 6 – Parent element pointers of refined elements

The requirements for a group of elements to be derefined are the following.

- A set of elements (eight sibling elements split from the same parent) must all be selected for refinement based on their pressure gradient being less than the prescribed threshold.
- Each of the elements within the group must not be a transition element. A transition element has one or more edge or face nodes.
- All of the eight sibling elements except for the "upper", "forward left" element will have parent nodes that point (via their parent element pointer) to the same parent (Figure 6).

A set of eight elements is recombined into a single element by removing nodes and reconnecting the node neighbor pointers of surrounding nodes. The following procedure is used to recombine the elements.

1. Remove the center node of the set of elements. This is node 7 of the "upper", "forward left" element of the set (element 1 of Figure 6). After this node has been removed and the neighbor node links connected, the element is reassembled into a single element.
2. Although the set of elements has been recombined into a single element, removal of unnecessary face and edge nodes from the newly derefined element is important. Face nodes numbered 9, 18 and 23-26 are checked to determine if they are still needed. If the neighbor node pointer in the direction of the neighboring element perpendicular to the face is not connected, then the node can be removed.
3. Similarly, the edge nodes are removed from the element if they are unnecessary. All neighboring node pointers (except for the pointers in the directions parallel to the edge) are checked for edge nodes that are unconnected.
4. The refinement level of the parent node is reduced.

SOFTWARE SYSTEM DEVELOPMENT

The simulation of the Eugene Island gas reservoir was divided into three separate applications (MESHGEN, FESERV, FEIMP) (see Figure 7). Each of the applications compartmentalizes different phases of the solution. The MESHGEN application is a set of utilities that together provide different services needed in the mesh generation. These utilities do most of the more complex and laborious tasks in the mesh generation. These utilities include the use of graphic tools that are an integral part and indispensable when generating meshes for large domains. In fact, the mesh generation phase would be impossible without them (Aziz93, Löhner87). The FESERV application is the implementation of the three-dimensional finite element solver prototype. The mesh description is loaded from a disk file and computed. The output from the FESERV application is a series of solution files with the coordinate of the node in the mesh and its pressure value (psia). This application is built so that it can be recompiled using C++ compilers on different computer hardware platforms and operating systems. The FEIMP application converts these ASCII solution files into database records. After the records have been stored in the database, they can be queried using an SQL (structured query language) query. These three applications each perform tasks for different stages of the finite element simulation. The applications used together make up the software system for the simulation. This includes the creation of the necessary input files, the computation of the approximate solution, and the organization of the results into a usable format.

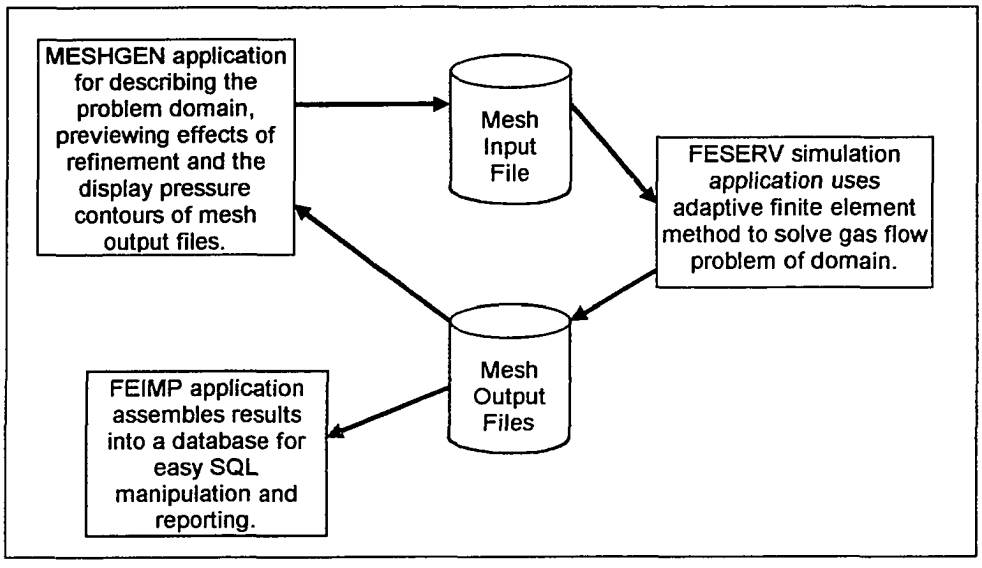


Figure 7 - Applications used in simulation of Eugene Island

The hardware/software platform, that all these applications were written and supported for, is the Win32 platform (Microsoft Windows 95 and NT). This is a microcomputer application whose computing platform is low cost with widespread availability. These attributes are strongly related and point toward our most abundant group of technology users, who have limited financial resources and therefore limited computing resources. It has been a focus of this research to provide advanced solution methods for this generally affordable class of machines. This means that this simulation must have reduced storage requirements and make efficient use of CPU time while still maintaining an ease of use. By doing this, we have provided an affordable solution to gas reservoir problems that can fulfill most current industry needs. Many of the typical United States gas fields are owned and operated by small businesses.

MESHGEN Application

The MESHGEN application (Figure 8) generates the following data for the FESERV solver:

1. A description of the problem domain (including geometry and physical characteristics that vary spatially) is stored on disk. The single physical characteristic that is needed for the Eugene Island reservoir is the permeability. Physical characteristics that do not vary over the reservoir (ex. temperature) are entered into the FESERV solver directly.
2. The well withdrawal rate schedule is a production factor provided in thousand cubic feet per day. A production factor is maintained separately for each well.
3. Post-processing information for the initial mesh is stored. The post-processing information contains the coordinates of regions in which an additional level of refinement is performed after the initial mesh generation, but before the start of the solution computation.
4. Simulation values are stored (time increment between steps and maximum number of iterations).

MESHGEN also has the following convenience features:

1. The generation of WRL (world) files for VRML (virtual reality modeling language) viewing. VRML viewers provide a three dimensional environment on the computer display though which the user can navigate using the keyboard or a computer mouse. The display is used to verify that the mesh structure (i.e. vertices and edges) is what is desired and that it is constructed properly.

2. Three-dimensional previewing of the generated mesh within the MESHGEN application. This allows the viewing on the computer display of the mesh rotated at different angles. The refinement and derefinement steps can be temporarily applied before the actual mesh generation. The refinement/derefinement situations, which may exist during the simulation, can be evaluated. This previewing takes place within the software as opposed to the generation of a file and use of an external viewer as described, item 1, above.
3. Three-dimensional contouring of the output for an individual time step. Output files from the FESERV application (one for each time step of the simulation) are used as input. For a specific user selected reservoir depth, a two dimensional, color contoured plot of the reservoir pressures is displayed on the computer screen. A low and high-pressure value is used to assign a relative color value (from red to blue) to pressures within the selected level of the reservoir.

FESERV Application

The FESERV application does not have a graphical interface. Instead, it is a console application written specifically for portability to multiple platforms. The application is built of C++ classes for element assembly, refinement and derefinement. The reusability of these classes is limited because of their tight coupling, however their structure can still be extended through inheritance. The use of object-oriented software development techniques has not been a primary goal of this project. Its use has improved the level of abstraction within the code

module and has been very useful in the development of the prototype simulation for Eugene Island reservoir. These classes form the basis for future extensions allowing multiple finite element shapes.

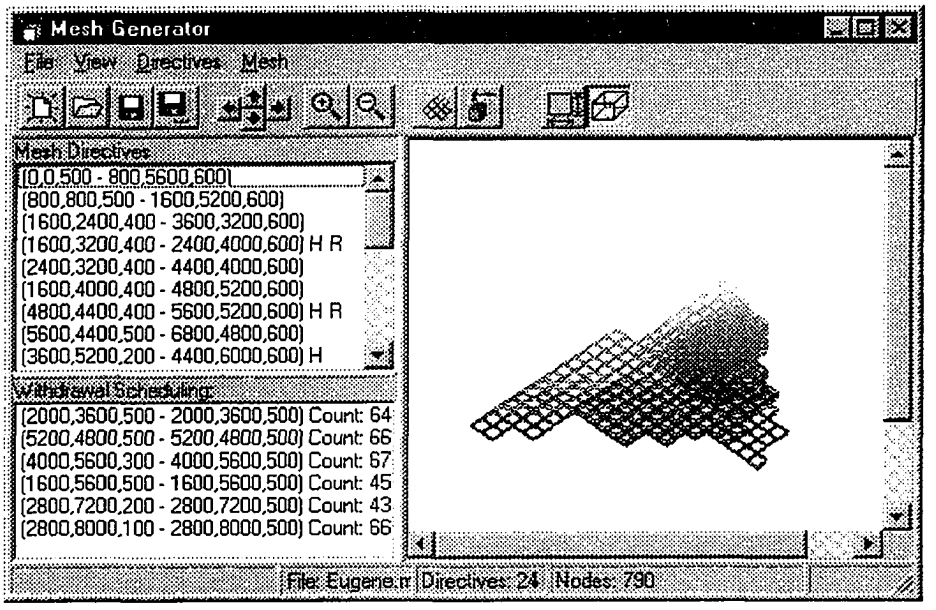


Figure 8 - MESHGEN application screen for Eugene Island

The input for this computational application is the mesh description file output by MESHGEN. The number of iterations and their step size are loaded from this file. At each step of the execution of this module, output files are created. These output files contain the current pressure value at each of the nodes as well as a VRML representation of the structure and pressure distribution of the FEM mesh. The module also performs the following tasks:

1. The initial mesh is generated from the mesh description file. This is a coarse mesh that serves as the basis for future refinement.

2. The initial mesh is pre-refined in selected areas by performing a refinement step before the simulation starts.
3. The approximate solution is calculated at each time step by assembling the stiffness matrix and load vector from the finite elements and then solving the simultaneous system of equations formed.
4. The refinement and derefinement of the mesh are performed at alternating steps to obtain a more accurate approximate solution.
5. The VRML files for the visualization of the solution and the current finite element mesh structure are created.

FEIMP Application

For each time step with the prototype simulation, there are results (3 dimensional node coordinate and pressure value) at each node in the mesh. With 2042 time steps for the Eugene Island prototype and assuming a minimum node count of 790 nodes within the FEM mesh, there are 1,613,180 individual results. The large number of results is overwhelming for the typical user and it is desirable to organize this information into a more usable form.

The FEIMP application was written specifically for this task. After executing the FESERV application, the FEIMP application assembles each of the output files and inserts them within a database. After all results of the simulation have been added to the database, the database can then be manipulated and queried using SQL (structured query language). An example query would be to determine the average pressure within a specific area of the reservoir. Here is one such query: "select Avg(Pressure) where (x>400) and

(x<425) and (y>500) and (y<525) and (z<300)". The results of the query would be returned in tabular form with columns for the x, y, and z coordinates and the average approximate solution.

THE USE OF VRML FOR VISUALIZATION

The development of a 3D finite element simulation is an enormously complex problem. Some of this complexity would be removed if one could adequately visualize the 3D mesh during the simulation. An adequate visualization of the mesh is difficult because of its geometric complexity and the amount of information that must be presented. We developed a prototype finite element reservoir simulator with an initial mesh of 790 nodes (forming 560 elements). Even this moderate initial mesh size grows rapidly with the dynamic adaptation of the mesh during the solution. Larger mesh sizes exacerbate its visualization. The visualization of the quality of information being generated in this type of problem is only available using proprietary systems. These systems are expensive and often require advanced hardware. A very cost-effective and adequate visualization for this type of problem is presented which uses virtual reality modeling language (VRML) viewers. Specifically, the VRML language that we generate conforms to the VRML97 specification (VRML97) (revised version of VRML 2.0).

VRML as a Debugging Tool

During development of our 3D finite element method prototype, we found it very difficult to determine whether our mesh was being correctly adapted. The elements of the mesh used in finite element method are refined (or derefined) between consecutive time steps based on their solution at the previous time step. The refinement of elements with a nodal mesh structure is error prone

(Wang98). A VRML description of the mesh is a useful tool for debugging the code of a mesh refinement algorithm. The VRML visualization of the mesh structure provides an excellent view of the interconnection of mesh nodes.

The mesh visualization used for debugging is accomplished by generating the VRML description of the line segments connecting the nodes of the mesh. Then this VRML description is stored in a "World" (.wrl) file. This file is loaded into a VRML viewer to be interpreted and the result of this is displayed on the computer screen. VRML viewers support navigation by a user through three dimensional structures using input from their PCs. That is, we can "fly" in and around the structure using this navigation. To explain this navigation we offer Figure 9, Figure 10 and Figure 11. Figure 9 shows an actual finite element mesh for the Eugene Island reservoir. Viewing Figure 9 we were unable to determine whether our mesh was constructed correctly. Determining correctness required a "closer look" at the part of the mesh that looks like a black glob near the top. Figure 10 is obtained by "flying" into this area of interest. This "glob" begins to reveal its mesh structure but the desired details are still not visible. Figure 11 finally shows the mesh structure in this area and also its nodes (where mesh lines intersect). It is imperative in the finite element method for these lines and nodes to form elements of a particular shape. We were able to successfully view the desired structure only through the use of the conversion routines and VRML viewers.

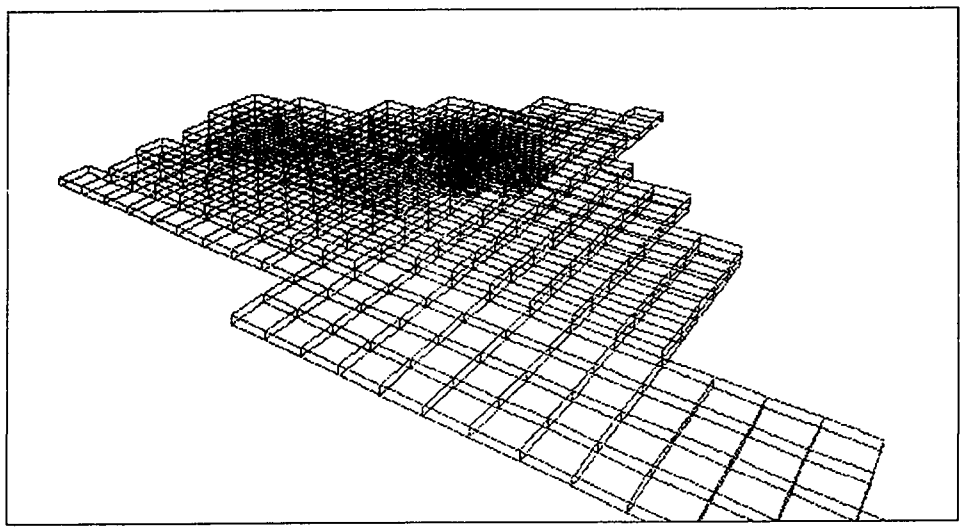


Figure 9 – Viewing an example finite element mesh using a VRML Viewer

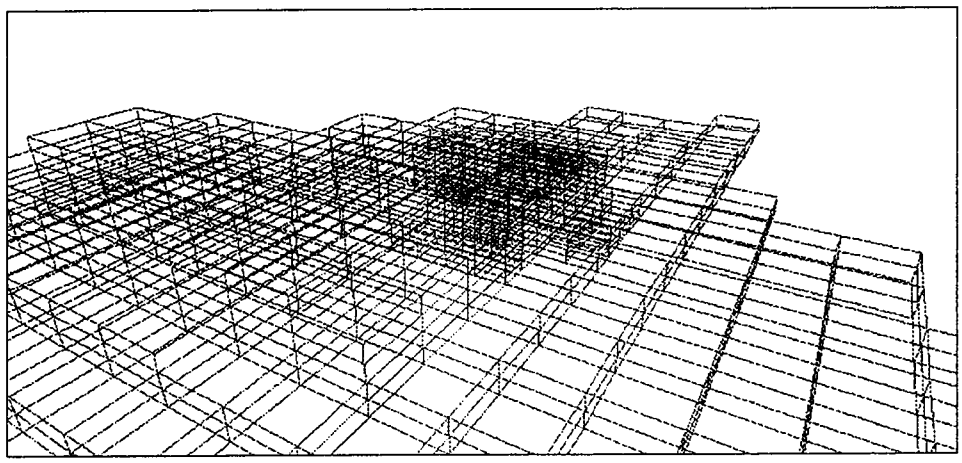


Figure 10 – Navigation of the VRML Viewer for a closer view of a well bore

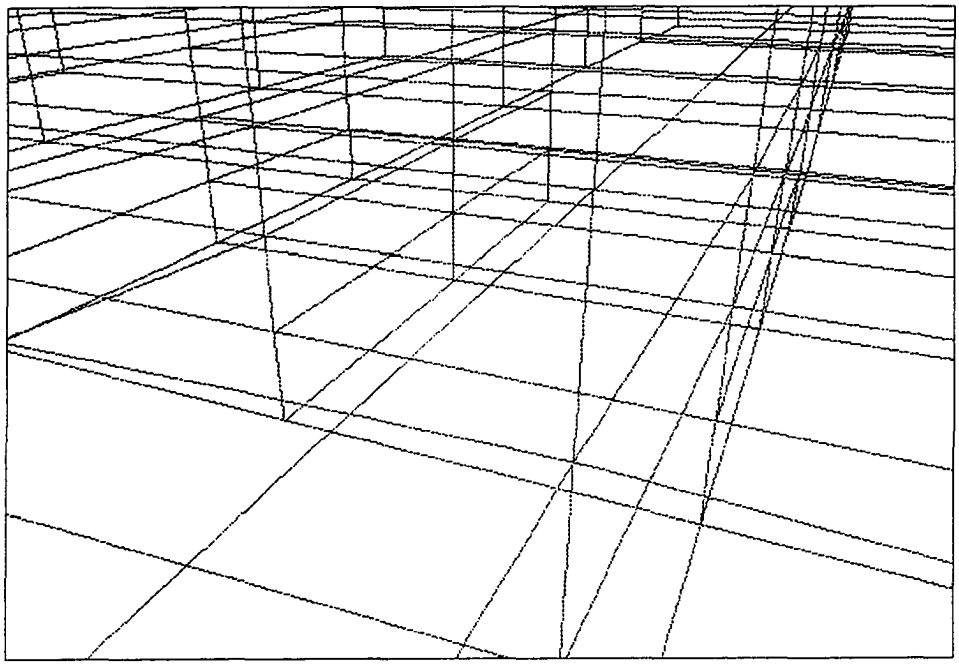


Figure 11 – *Using VRML to view finite element node connections*

The VRML language contains many constructs for the description of structures in three dimensions plus their appearance, sound, interaction with other structures, etc. The "World" file contains a scene of different types of VRML nodes. To export our mesh structures to VRML, an **IndexedLineSet** VRML node was used (Ames97, Taubin98). Each vertex of the mesh is mapped to an entry in the **Coordinate** VRML node, which contains the Cartesian coordinate for each location in the finite element mesh. Each edge between vertices of a single element or two elements (any two vertices connected by an edge) is mapped to an entry in the **coordIndex** field of the **IndexedLineSet** VRML node. Figure 12 presents an example of this creation process. Below is the part of the structure the IndexedLineSet VRML node used.

```
IndexedLineSet {
        exposedField SFNode coord NULL
        field MFInt32 coordIndex []              # [-1, ∞)
}
```

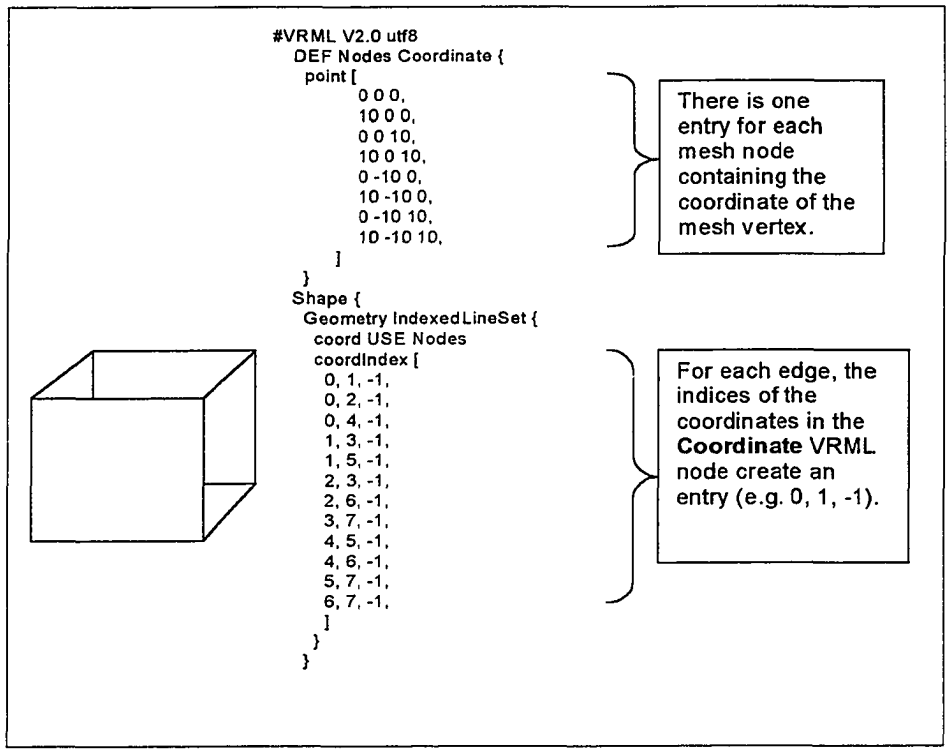


Figure 12 – VRML Debugging Example

When mapping vertices from the finite element mesh to the VRML **Coordinate** node, it was necessary to convert between the coordinate systems. Figure 13 shows a comparison of these two coordinate systems. To generate the world file for our finite element mesh, the finite element coordinates are converted to the VRML coordinates using the following mapping:

$$X_{VRML} = X_{FEM}$$
$$Y_{VRML} = -Z_{FEM}$$
$$Z_{VRML} = Y_{FEM}$$

where X_{VRML}, Y_{VRML} and Z_{VRML} are the coordinates of the world file and X_{FEM}, Y_{FEM} and Z_{FEM} are the coordinates of the finite element mesh.

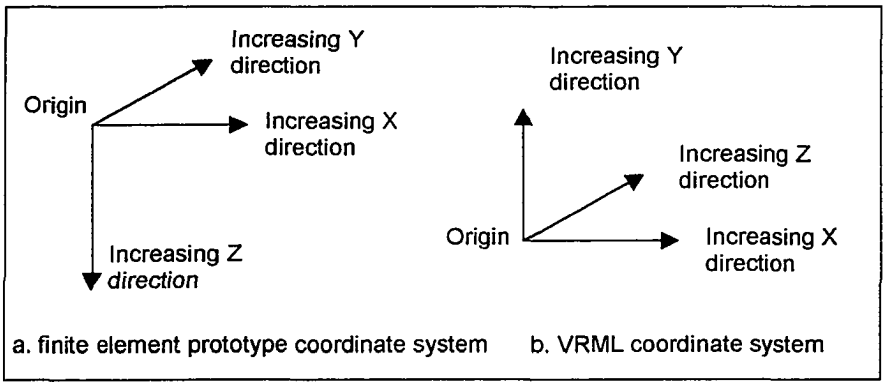


Figure 13 – Comparison of coordinate systems

Visualizing Results with VRML

In addition to use as a debugging tool, VRML descriptions of the mesh are better known for the visualization of solutions of problems solved with the finite element method. For our Eugene Island reservoir solution, visualizations of the cross-sections of the reservoir were desired. We used the **IndexedFaceSet** VRML node to visualize different layers of the reservoir. We used the same **Coordinate** VRML node (vertices of the mesh) that was created for the structure. For clarity in viewing, only faces for the top and bottom of the elements were created. The specification of the **IndexedFaceSet** fields follows:

```
IndexedFaceSet {
        exposedField SFNode color NULL
        exposedField SFNode coord NULL
        field SFBool ccw TRUE
        field SFBool colorPerVertex TRUE
        field SFBool convex TRUE
        field MFInt32 coordIndex []              # [-1, ∞)
        field SFBool solid TRUE
}
```

The color field, in addition to the **coord** field of the **IndexedFaceSet** VRML node, denotes the color of the vertices. The **Color** VRML node was used to define a different color to represent each vertex in the finite element mesh for the coordinates. Color is specified by three numbers, (R, G, B) representing the amount of red, green and blue. The colors of the vertices were selected so the mesh node with the lowest pressure (psi) was red (1, 0, 0) and the highest pressure was blue (0, 0, 1). Interpolation was used to assign colors to other vertices (between red and blue depending on its simulation-calculated pressure). The color of the surface of the face is defined as a smooth gradient and the **colorPerVertex** field was set to TRUE. Figure 14, the VRML code example below, is an actual sample of the output. In this example, the conversion for a single element is given. The **IndexedFaceSet** VRML node is created for the top and bottom faces of the element. The sides of the element are not reconstructed in the VRML file. The **coordIndex** field contains two face specifications. The first is made up of the vertices 0, 1, 3, and 2 from the **Coordinate** VRML node given at the top of the example. In the example, vertex 0 corresponds to location (0, 0, 0) in the **Coordinate** VRML definition and location (0, 0, 0) in the mesh structure. Vertex 1 is location (10, 0, 0) in the VRML file and location (10, 0, 0) in the mesh. Vertex 3 is location (0, 0, 10) in the VRML file and location (0, 10, 0) in the mesh. Vertex 2 is location (10, 0, 10) in the VRML file and location (10, 10, 0) in the mesh. These locations are combined in a counter-clockwise direction to form the top face of the element. The bottom face of the element is made up of VRML vertices 4, 5, 7, and 6.

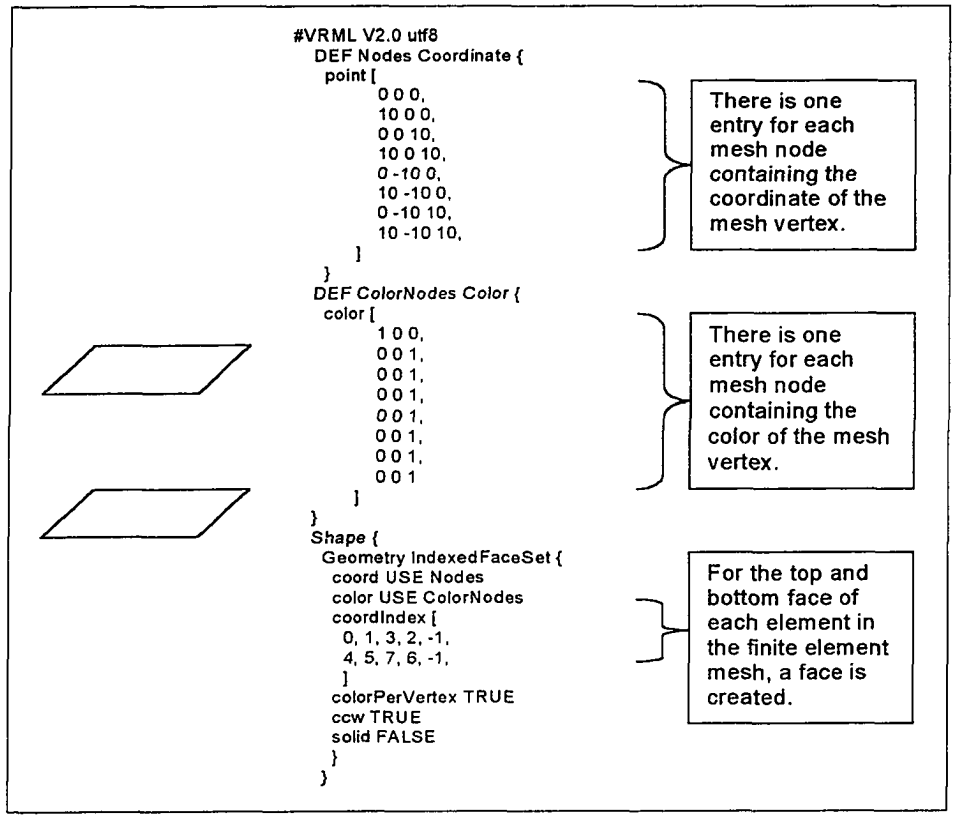


Figure 14 – VRML finite element method solution example

One limitation of the VRML file used in this version is the difficulty viewing layers at lower depths in the reservoir (upper layers occlude the lower layers of the VRML display). By separating layers into different objects within the VRML file using **Shape** nodes, transparency can be applied individually to each layer using an **Appearance** VRML node. Upper layers were assigned a transparency value that allowed viewing of the lower layers (Figure 15). The multiple VRML shapes were grouped using the **children** field of the **Group** VRML node.

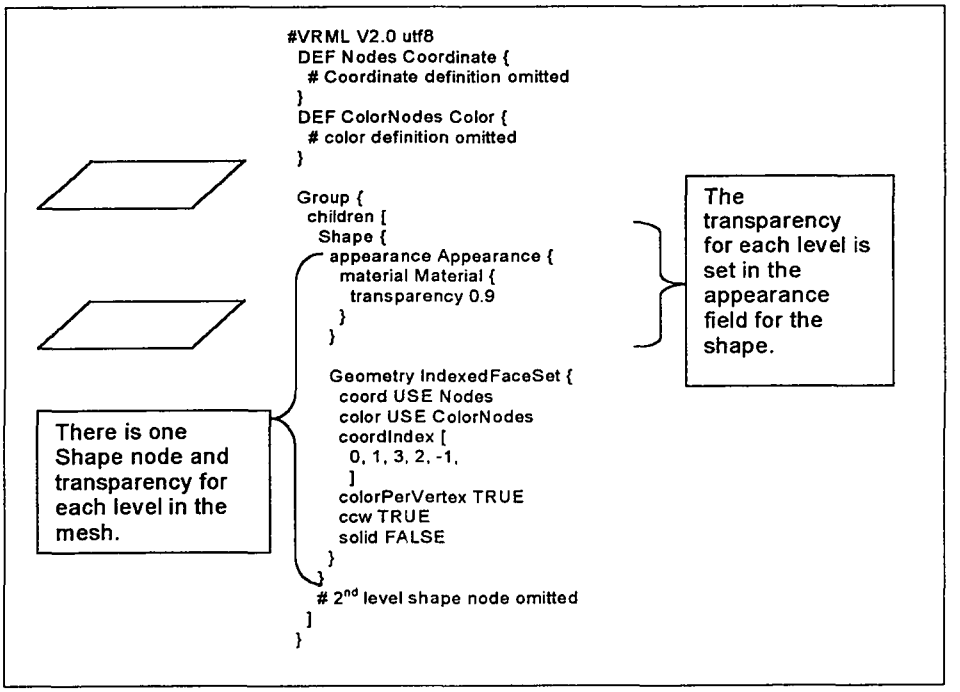


Figure 15 – Creation of a Layered VRML for mesh

A VRML file was created whenever the mesh had been adapted or the reservoir withdrawal rate changed for each of the time steps during the simulation. This produced an excellent view of the structural changes occurring in the mesh and their effect on the solution. Figure 16 shows an example VRML file for a time step of the Eugene Island reservoir prototype. The dark spot in the center of the figure is the location of a production well. The darker region shows a pressure decline around the area of the well due to the production of condensate. The multiple slices of the reservoir can be seen at the top of the figure. The transparency of the levels allows the lower level pressure gradients to be seen. The viewer allows the user's perspective to be changed by using the controls on the left and bottom sides of the figure.

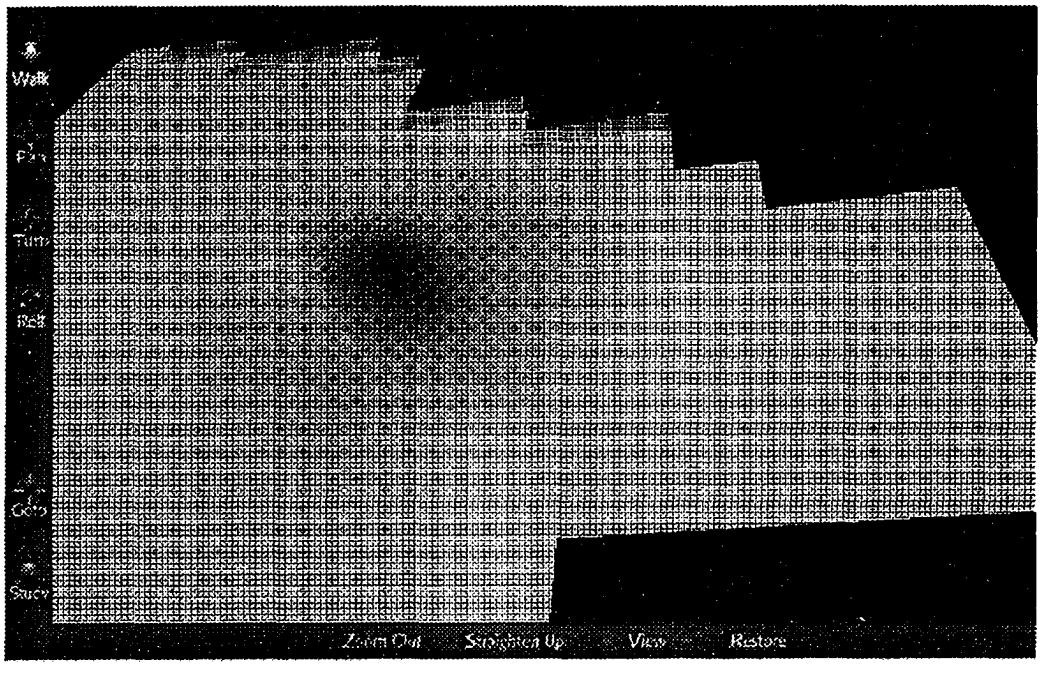


Figure 16 – Sample Output from finite element method Prototype

In conclusion, VRML has been a valuable tool for our development of the finite element method prototype. The completion of our development would not have been possible without inexpensive VRML viewers. The VRML file creation algorithm can be included in a finite element method project for quick visualization. The algorithm is flexible enough to modify the color selection and element faces viewed. Since there are VRML viewers for most computers, these files are also powerful tools for collaboration. The use of VRML for low-resolution visualization of problems is an excellent complement to any current visualization techniques.

THREE DIMENSIONAL MODEL DEVELOPMENT

The development of a finite element solution, in many ways, is based on an approach to minimize the residual error in the problem domain. In the finite element method, there are two major methods of minimizing the equation for the residual error. One is to use variational calculus to find the theoretical minimum of the model. A variational method takes advantage of the nature of the flow equations in question and provides a function that determines the theoretical minimum or the next best case for the optimization. Although very effective, this method can be very difficult, even for simple problems, because it requires an energy functional. The other common method is the method of weighted residuals (MWR). The method of weighted residuals, which includes the Galerkin method, is designed to minimize the residual by weighting segments of the residual equation. In this method, a weighting is applied to segments of the problem to numerically find the most accurate solution. There are many MWR methods. The Galerkin method of weighted residuals was selected and used for this research. The Galerkin method weights the residual equation using a shape function, which is discussed later, and integrates the weighted equation over the element domain.

Development of Gas Flow Equations

The first step in the finite element solution is to derive the element equation from the original problem domain. Each problem domain has a controlling mathematical model and a dependency on physical properties, which

provide each individual problem its uniqueness. From Aziz and Setari (Aziz83), we obtained equation 2 for the flow of gas without assuming constant compressibility.

$$\nabla^2 P^2 = \frac{\varphi \mu c}{K}\frac{\partial P^2}{\partial t} + \frac{2ZRT\mu}{MK}\tilde{q} \qquad \text{Eq. 2}$$

where,

M is molecular weight
R is the universal gas constant
T is absolute temperature
P is pressure
K is permeability
φ is porosity
μ is viscosity
c is gas compressibility
Z is the compressibility factor (z - factor)
$\tilde{q}$ is the withdrawal rate at reservoir conditions

Equation 2 is transformed using the gas pseudo pressure variable, which is:

$$\Phi(P) = 2\int_{P_c}^{P} \frac{\lambda}{\mu(\lambda)Z(\lambda)} d\lambda \qquad \text{Eq. 3}$$

This transformation yields the following equation:

$$\nabla^2 \Phi = \frac{\varphi \mu c}{K}\frac{\partial \Phi}{\partial t} + \frac{2RT}{MK}\tilde{q} \qquad \text{Eq. 4}$$

This is the governing equation used in each problem domain for this research (Eq. 4).

Shape functions

The shape functions presented in this section are directly affected by the physical structure of the element used. These shape functions are the weighting

for residual minimization of the flow equations and hence the selection of the element will ultimately affect the accuracy of the solution. After designating a physical structure for the element, the shape functions are defined so that:

1. The values of the shape functions are one at their node and zero at all other nodes in a finite element.
2. Between any two nodes that are linked, the shape functions are equivalent to an interpolating function between the two nodes.

Figure 4 shows the element that was derived from the three-dimensional transition element developed by Morton (Morton95). In this hexahedral element there are 26 nodes as labeled in Figure 4. For each of the nodes, there is a corresponding shape function.

The label of the local nodes of the element is different from Morton's labels. These changes were necessary in this research to optimize the dynamic adaptive mesh refinement routines. The shape functions have been transformed by mapping Morton's node numbers to the scheme presented in Figure 4.

Derivation of Element Equations

The residual (Eq. 5) is weighted by the shape functions (N_n^e) and integrated over the volume of an element (e) resulting in equation 6. This is the Galerkin Method of weighting the residuals.

$$R(x,y,z;a(t)) = \nabla^2\Phi^{(e)}(x,y,z;a(t)) - \frac{2RT}{MK}\tilde{q}(x,y,z,t) - \frac{\varphi\mu c}{K}\frac{\partial\Phi^{(e)}(x,y,z;a(t))}{\partial t} \quad \text{Eq. 5}$$

$$\iiint_{(e)} R(x,y,z;a(t))N_i(x,y,z)dV = 0, \text{ where } i = 1,2,\ldots n \quad \text{Eq. 6}$$

There is one weighted residual for each node (each degree of freedom) of each element. The residual *R* is substituted into equation 6 to form equation 7. The equation is rewritten as a sum of integrals as presented in equation 8.

$$\iiint_{(e)} N_i \left[\nabla^2 \Phi^{(e)} - \frac{2\mathrm{R}T}{\mathrm{M}K} \tilde{q} - \frac{\varphi \mu c}{K} \frac{\partial \Phi^{(e)}}{\partial t} \right] dV = 0 \qquad \text{Eq. 7}$$

$$\iiint_{(e)} N_i \nabla^2 \Phi^{(e)} dV - \iiint_{(e)} \frac{2\mathrm{R}T}{\mathrm{M}K} \tilde{q} N_i dV - \iiint_{(e)} \frac{\varphi \mu c}{K} \frac{\partial \Phi^{(e)}}{\partial t} N_i dV = 0 \qquad \text{Eq. 8}$$

Green's theorem (divergence theorem) is:

$$div(N_i \nabla \Phi) = N_i \nabla^2 \Phi + \nabla N_i \cdot \nabla \Phi \qquad \text{Eq. 9}$$

furthermore,

$$\hat{n} \cdot (N_i \nabla \Phi) = N_i (\hat{n} \cdot \nabla \Phi) \qquad \text{Eq. 10}$$

the expression $\hat{n} \cdot \nabla \Phi$ is the directional derivative of Φ in the direction of the outward normal vector $\hat{n}$ of the surface S in the divergence theorem. If we denote this derivative by $\frac{\partial \Phi}{\partial \hat{n}}$, the formula for the divergence theorem becomes:

$$\iiint (N_i \nabla^2 \Phi + \nabla N_i \cdot \nabla \Phi) dV = \iint_S N_i \frac{\partial \Phi}{\partial \hat{n}} dA \qquad \text{Eq. 11}$$

This formula is called Green's First Theorem. We may also write this in the form:

$$\iiint N_i \nabla^2 \Phi dV = -\iiint (\nabla N_i \cdot \nabla \Phi) dV + \iint_S N_i \frac{\partial \Phi}{\partial \hat{n}} dA \qquad \text{Eq. 12}$$

Using Green's Theorem, we rewrite the first term of equation 8.

$$\iiint_{(e)} N_i \nabla^2 \Phi^{(e)} dV = -\iiint_{(e)} \left(\nabla N_i \cdot \nabla \Phi^{(e)}\right) dV + \iint_S N_i \frac{\partial \Phi^{(e)}}{\partial \hat{n}} dA \qquad \text{Eq. 13}$$

$$\tilde{\tau}_n^{(e)} \equiv \frac{\partial \Phi^{(e)}}{\partial \hat{n}} = \left[\hat{N} \cdot \nabla \Phi^{(e)}\right] \qquad \text{Eq. 14}$$

Making a substitution of equation 14 into equation 13, the equation for the first term becomes:

$$\iiint_{(e)} N_i \nabla^2 \Phi^{(e)} dV = -\iiint_{(e)} \left(\nabla N_i \cdot \nabla \Phi^{(e)}\right) dV + \iint_S N_i \tilde{\tau}_n^{(e)} dA \qquad \text{Eq. 15}$$

Equation 8 can be rewritten using this new first term as:

$$\begin{aligned} &\iint_S N_i \tilde{\tau}_n^{(e)} dA - \iiint_{(e)} \left(\nabla N_i \cdot \nabla \Phi^{(e)}\right) dV \\ &- \iiint_{(e)} \frac{2RT}{MK} \tilde{q} N_i dV - \iiint_{(e)} \frac{\varphi \mu c}{K} \frac{\partial \Phi^{(e)}}{\partial t} N_i dV = 0 \end{aligned} \qquad \text{Eq. 16}$$

The last term of the left-hand side of equation 16 includes a time dependent partial derivative. Using a finite difference approximation for the derivative with respect to time (Zienkiewics82), we simplify the complexity of equation 16 as follows:

$$\begin{aligned} \iiint_{(e)} \frac{\varphi \mu c}{K} \frac{\partial \Phi^{(e)}}{\partial t} N_i^{(e)} dV &\cong \iiint_{(e)} \frac{\varphi \mu c}{K} \left(\frac{\Phi^{(e)^{t+1}} - \Phi^{(e)^t}}{\Delta t} \right) N_i^{(e)} dV \\ &= \iiint_{(e)} \frac{\varphi \mu c}{K} \frac{\Phi^{(e)^{t+1}}}{\Delta t} N_i^{(e)} dV - \iiint_{(e)} \frac{\varphi \mu c}{K} \frac{\Phi^{(e)^t}}{\Delta t} N_i^{(e)} dV \end{aligned} \qquad \text{Eq. 17}$$

Substituting the finite difference approximation of the time derivative (equation 17) in place of the term for the partial of the pseudo pressure with respect to time, we obtain equation 18.

$$\iint_S N_i \tilde{\tau}_n^{(e)} dA - \iiint_{(e)} \left(\nabla N_i \cdot \nabla \Phi^{(e)}\right) dV - \iiint_{(e)} \frac{2RT}{MK} \tilde{q} N_i dV$$
$$- \iiint_{(e)} \frac{\varphi \mu c}{K} \frac{d\Phi^{(e)}}{\Delta t} \Phi^{(e)t+1} N_i dV + \iiint_{(e)} \frac{\varphi \mu c}{K} \frac{\Delta \Phi^{(e)}}{\Delta t} \Phi^{(e)t} N_i dV = 0 \qquad \text{Eq. 18}$$

The desired final result is an equation in the form $K_{ij}^{(e)} a_j + C_{ij}^{(e)} a_j = F_i^{(e)}$, which is solved as a system of simultaneous equations in matrix form. To do this, we divide the system of equations into terms that are indicative of their contribution to the gas flow. These are $K_{ij}^{(e)}$ (the stiffness matrix), $C_{ij}^{(e)}$ (the capacity matrix), and $F_i^{(e)}$ (the load vector which includes parts of the derivative at time *t*). Additionally we know that over an element, that is not on a boundary, the net flux will be equal to zero since gas flows into and out of the element at an equal rate. Therefore, we can eliminate the first term in equation 18, except on the boundary of the domain where the flux is not equal to zero, yielding:

$$[K] = \iiint_{(e)} \left(\nabla N_i^{(e)} \cdot \nabla \Phi^{(e)t+1}\right) dV \qquad \text{Eq. 19}$$

$$[C] = \iiint_{(e)} \frac{\varphi \mu c}{K} \frac{\Delta \Phi}{\Delta t} \Phi^{(e)t+1} N_i^{(e)} dV \qquad \text{Eq. 20}$$

$$\hat{F} = -\iiint_{(e)} \frac{2RT}{MK} \tilde{q} N_i^{(e)} dV + \iint_S \tilde{\tau}_n^{(e)} N_i^{(e)} dA + \iiint_{(e)} \frac{\varphi \mu c}{K} \frac{\Delta \Phi}{\Delta t} \Phi^{(e)t} N_i^{(e)} dV \qquad \text{Eq. 21}$$

The trial solution Φ^t is also defined in terms of the finite element shape functions and coefficients. The following five equations make use of the finite element solution.

$$\Phi^t(x, y, z, t; a) \approx \sum_{j=1}^{n} a_j(t) N_j(x, y, z) \qquad \text{Eq. 22}$$

$$\Phi^{t+1}(x,y,z,t;a) \approx \sum_{j=1}^{n} a_j(t+\Delta t) N_j(x,y,z) \qquad \text{Eq. 23}$$

$$\nabla\Phi^{t+1} \approx \sum_{j=1}^{n} a_j(t+\Delta t)\left[\hat{i}\frac{\partial N_j}{\partial x} + \hat{j}\frac{\partial N_j}{\partial y} + \hat{k}\frac{\partial N_j}{\partial z}\right] \qquad \text{Eq. 24}$$

$$\nabla N_j \approx \hat{i}\frac{\partial N_j}{\partial x} + \hat{j}\frac{\partial N_j}{\partial y} + \hat{k}\frac{\partial N_j}{\partial z} \qquad \text{Eq. 25}$$

$$\iiint (\nabla N_j \cdot \nabla\Phi^{t+1}) dV \approx \sum_{i=1,j=1}^{n,n} a_j(t+\Delta t)\left[\frac{\partial N_i}{\partial x}\frac{\partial N_j}{\partial x} + \frac{\partial N_i}{\partial y}\frac{\partial N_j}{\partial y} + \frac{\partial N_i}{\partial z}\frac{\partial N_j}{\partial z}\right] \qquad \text{Eq. 26}$$

Using these equations, we rewrite the stiffness matrix, capacity matrix, and load vector.

$$K_{ij} = \iiint_{(e)} \frac{\partial N_i^{(e)}}{\partial x}\frac{\partial N_j^{(e)}}{\partial x} dV + \iiint_{(e)} \frac{\partial N_i^{(e)}}{\partial y}\frac{\partial N_j^{(e)}}{\partial y} dV + \iiint_{(e)} \frac{\partial N_i^{(e)}}{\partial z}\frac{\partial N_j^{(e)}}{\partial z} dV \qquad \text{Eq. 27}$$

$$C_{ij} = \iiint_{(e)} \frac{\varphi\mu c}{K}\frac{\Delta\Phi}{\Delta t} N_i^{(e)} N_j^{(e)} dV \qquad \text{Eq. 28}$$

$$F_i = -\iiint_{(e)} \frac{2RT}{MK}\tilde{q} N_i^{(e)} dV + \iint_S \tilde{\tau}_n^{(e)} N_i^{(e)} dA + \iiint_{(e)} \frac{\varphi\mu c}{K}\frac{\Delta\Phi}{\Delta t}\Phi^{(e)t} N_i^{(e)} dV \qquad \text{Eq. 29}$$

In the load vector, the terms c and $\Phi^{(e)t}$ are quantities which may be approximated within each finite element solution. The first term, compressibility, is approximated using a first order finite difference solution with values for P, $Z(P)$ and Φ calculated at the previous time step. For a particular element, $\Phi^{(e)t} = \frac{\sum_{k=1}^{n} \Phi_k^t}{n}$, or the average of the pseudo pressure at each node of the element in the previous step.

Numerical Integration of Equations

To numerically integrate the stiffness matrix, capacity matrix and load vector, the equations are rewritten in terms of the parent shape functions (Burnett87). By changing the variables to the shape functions, the physical location of the nodes will not change the value of the integral and hence provide a more flexible interface.

$$K_{ij} = \iiint_{(e)} \frac{\partial N_i^{(e)}}{\partial x}\frac{\partial N_j^{(e)}}{\partial x}\left|J^{(e)}\right| d\xi d\eta d\varsigma + \iiint_{(e)} \frac{\partial N_i^{(e)}}{\partial y}\frac{\partial N_j^{(e)}}{\partial y}\left|J^{(e)}\right| d\xi d\eta d\varsigma + \iiint_{(e)} \frac{\partial N_i^{(e)}}{\partial z}\frac{\partial N_j^{(e)}}{\partial z}\left|J^{(e)}\right| d\xi d\eta d\varsigma \qquad \text{Eq. 30}$$

$$C_{ij} = \iiint_{(e)} \frac{\varphi \mu c}{K}\frac{d\Phi}{\Delta t} N_i^{(e)} N_j^{(e)}\left|J^{(e)}\right| d\xi d\eta d\varsigma \qquad \text{Eq. 31}$$

$$F_i = -\iiint_{(e)} \frac{2\mathrm{R}T}{\mathrm{M}K}\tilde{q} N_i^{(e)}\left|J^{(e)}\right| d\xi d\eta d\varsigma + \iint_S \tilde{\tau}_n^{(e)} N_i^{(e)} dA + \iiint_{(e)} \frac{\varphi \mu c}{K}\frac{\Delta\Phi}{\Delta t}\Phi^{(e)t} N_i^{(e)}\left|J^{(e)}\right| d\xi d\eta d\varsigma \qquad \text{Eq. 32}$$

The calculation of the determinant of the Jacobian matrix (i.e. $\left|J^{(e)}\right|$) is necessary in the numerical integration of the equations. The following equations demonstrate the necessary calculations needed for the determinant of the Jacobian.

$$
\begin{aligned}
\left|J^{(e)}\right| &= \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \varsigma} & \frac{\partial y}{\partial \varsigma} & \frac{\partial z}{\partial \varsigma} \end{bmatrix} \\
&= \begin{bmatrix} J_{11}^{(e)}(\xi,\eta,\varsigma) & J_{12}^{(e)}(\xi,\eta,\varsigma) & J_{13}^{(e)}(\xi,\eta,\varsigma) \\ J_{21}^{(e)}(\xi,\eta,\varsigma) & J_{22}^{(e)}(\xi,\eta,\varsigma) & J_{23}^{(e)}(\xi,\eta,\varsigma) \\ J_{31}^{(e)}(\xi,\eta,\varsigma) & J_{32}^{(e)}(\xi,\eta,\varsigma) & J_{33}^{(e)}(\xi,\eta,\varsigma) \end{bmatrix} \\
&= \frac{\partial x \partial y \partial z}{\partial \xi \partial \eta \partial \varsigma} + \frac{\partial x \partial y \partial z}{\partial \varsigma \partial \xi \partial \eta} + \frac{\partial x \partial y \partial z}{\partial \eta \partial \varsigma \partial \xi} - \frac{\partial x \partial y \partial z}{\partial \varsigma \partial \eta \partial \xi} - \frac{\partial x \partial y \partial z}{\partial \xi \partial \varsigma \partial \eta} - \frac{\partial x \partial y \partial z}{\partial \eta \partial \xi \partial \varsigma}
\end{aligned}
\qquad \text{Eq. 33}
$$

where,

$$
\begin{aligned}
J_{11}^{(e)} &= \sum_{k=1}^{n} x_k^{(e)} \frac{\partial N_k(\xi,\eta,\varsigma)}{\partial \xi} \quad J_{12}^{(e)} = \sum_{k=1}^{n} y_k^{(e)} \frac{\partial N_k(\xi,\eta,\varsigma)}{\partial \xi} \quad J_{13}^{(e)} = \sum_{k=1}^{n} z_k^{(e)} \frac{\partial N_k(\xi,\eta,\varsigma)}{\partial \xi} \\
J_{21}^{(e)} &= \sum_{k=1}^{n} x_k^{(e)} \frac{\partial N_k(\xi,\eta,\varsigma)}{\partial \eta} \quad J_{22}^{(e)} = \sum_{k=1}^{n} y_k^{(e)} \frac{\partial N_k(\xi,\eta,\varsigma)}{\partial \eta} \quad J_{23}^{(e)} = \sum_{k=1}^{n} z_k^{(e)} \frac{\partial N_k(\xi,\eta,\varsigma)}{\partial \eta} \\
J_{31}^{(e)} &= \sum_{k=1}^{n} x_k^{(e)} \frac{\partial N_k(\xi,\eta,\varsigma)}{\partial \varsigma} \quad J_{32}^{(e)} = \sum_{k=1}^{n} y_k^{(e)} \frac{\partial N_k(\xi,\eta,\varsigma)}{\partial \varsigma} \quad J_{33}^{(e)} = \sum_{k=1}^{n} z_k^{(e)} \frac{\partial N_k(\xi,\eta,\varsigma)}{\partial \varsigma}
\end{aligned}
\qquad \text{Eq. 34}
$$

The partial derivatives of the shape functions with the element parameters ξ, η and ς are calculated analytically from the parent shape functions. The partial derivatives of the shape functions with respect to the variables x, y and z are not known beforehand and must be calculated. The method of calculating these partial derivatives in three dimensions is an extension of Burnett's (Burnett87) handling of the two-dimensional equations for the partial derivatives of the shape functions with respect to x and y. Using the chain rule of differentiation, the partial derivatives can be written in the following form.

$$
\frac{\partial N_i}{\partial \xi} = \frac{\partial N_i}{\partial x}\frac{\partial x}{\partial \xi} + \frac{\partial N_i}{\partial y}\frac{\partial y}{\partial \xi} + \frac{\partial N_i}{\partial z}\frac{\partial z}{\partial \xi} \qquad \text{Eq. 35}
$$

$$\frac{\partial N_i}{\partial \eta} = \frac{\partial N_i}{\partial x}\frac{\partial x}{\partial \eta} + \frac{\partial N_i}{\partial y}\frac{\partial y}{\partial \eta} + \frac{\partial N_i}{\partial z}\frac{\partial z}{\partial \eta} \qquad \text{Eq. 36}$$

$$\frac{\partial N_i}{\partial \varsigma} = \frac{\partial N_i}{\partial x}\frac{\partial x}{\partial \varsigma} + \frac{\partial N_i}{\partial y}\frac{\partial y}{\partial \varsigma} + \frac{\partial N_i}{\partial z}\frac{\partial z}{\partial \varsigma} \qquad \text{Eq. 37}$$

Rewriting equations 35-37 in matrix form produces equation 38.

$$\begin{Bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \\ \frac{\partial N_i}{\partial \varsigma} \end{Bmatrix} = \left[J^{(e)}\right] \begin{Bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{Bmatrix}, i = 1,2,\cdots,n \qquad \text{Eq. 38}$$

By multiplying each side of equation 38 by the inverse of the Jacobian matrix, the vector of the partial derivatives of interest is obtained on the left-hand side (equation 39).

$$\begin{Bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{Bmatrix} = \left[J^{(e)}\right]^{-1} \begin{Bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \\ \frac{\partial N_i}{\partial \varsigma} \end{Bmatrix}, i = 1,2,\cdots,n \qquad \text{Eq. 39}$$

The inverse of the Jacobian matrix is rewritten in the form of equation 40 using the formula $A^{-1} = \frac{1}{\det(A)} Adj(A)$.

$$\left[J^{(e)}\right]^{-1} = \frac{1}{\left|J^{(e)}\right|} \begin{bmatrix} \frac{\partial y}{\partial \eta}\frac{\partial z}{\partial \varsigma} - \frac{\partial z}{\partial \eta}\frac{\partial y}{\partial \varsigma} & \frac{\partial z}{\partial \eta}\frac{\partial x}{\partial \varsigma} - \frac{\partial x}{\partial \eta}\frac{\partial z}{\partial \varsigma} & \frac{\partial x}{\partial \eta}\frac{\partial y}{\partial \varsigma} - \frac{\partial y}{\partial \eta}\frac{\partial x}{\partial \varsigma} \\ \frac{\partial z}{\partial \xi}\frac{\partial y}{\partial \varsigma} - \frac{\partial y}{\partial \xi}\frac{\partial z}{\partial \varsigma} & \frac{\partial x}{\partial \xi}\frac{\partial z}{\partial \varsigma} - \frac{\partial z}{\partial \xi}\frac{\partial x}{\partial \varsigma} & \frac{\partial y}{\partial \xi}\frac{\partial x}{\partial \varsigma} - \frac{\partial x}{\partial \xi}\frac{\partial y}{\partial \varsigma} \\ \frac{\partial y}{\partial \xi}\frac{\partial z}{\partial \eta} - \frac{\partial z}{\partial \xi}\frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \xi}\frac{\partial x}{\partial \eta} - \frac{\partial x}{\partial \xi}\frac{\partial z}{\partial \eta} & \frac{\partial x}{\partial \xi}\frac{\partial y}{\partial \eta} - \frac{\partial y}{\partial \xi}\frac{\partial x}{\partial \eta} \end{bmatrix} \qquad \text{Eq. 40}$$

Therefore,

$$\frac{\partial N_i}{\partial x} = (J_{11})^{-1}\frac{\partial N_i}{\partial \xi} + (J_{12})^{-1}\frac{\partial N_i}{\partial \eta} + (J_{13})^{-1}\frac{\partial N_i}{\partial \varsigma}$$
$$\frac{\partial N_i}{\partial y} = (J_{21})^{-1}\frac{\partial N_i}{\partial \xi} + (J_{22})^{-1}\frac{\partial N_i}{\partial \eta} + (J_{23})^{-1}\frac{\partial N_i}{\partial \varsigma} \qquad \text{Eq. 41}$$
$$\frac{\partial N_i}{\partial z} = (J_{31})^{-1}\frac{\partial N_i}{\partial \xi} + (J_{32})^{-1}\frac{\partial N_i}{\partial \eta} + (J_{33})^{-1}\frac{\partial N_i}{\partial \varsigma}$$

where,

$$(J_{11})^{-1} = \frac{1}{|J^{(e)}|}\left(\frac{\partial y}{\partial \eta}\frac{\partial z}{\partial \varsigma} - \frac{\partial z}{\partial \eta}\frac{\partial y}{\partial \varsigma}\right)$$
$$(J_{21})^{-1} = \frac{1}{|J^{(e)}|}\left(\frac{\partial z}{\partial \eta}\frac{\partial x}{\partial \varsigma} - \frac{\partial x}{\partial \eta}\frac{\partial z}{\partial \varsigma}\right)$$
$$(J_{31})^{-1} = \frac{1}{|J^{(e)}|}\left(\frac{\partial x}{\partial \eta}\frac{\partial y}{\partial \varsigma} - \frac{\partial y}{\partial \eta}\frac{\partial x}{\partial \varsigma}\right)$$
$$(J_{12})^{-1} = \frac{1}{|J^{(e)}|}\left(\frac{\partial z}{\partial \xi}\frac{\partial y}{\partial \varsigma} - \frac{\partial y}{\partial \xi}\frac{\partial z}{\partial \varsigma}\right)$$
$$(J_{22})^{-1} = \frac{1}{|J^{(e)}|}\left(\frac{\partial x}{\partial \xi}\frac{\partial z}{\partial \varsigma} - \frac{\partial z}{\partial \xi}\frac{\partial x}{\partial \varsigma}\right)$$
$$(J_{32})^{-1} = \frac{1}{|J^{(e)}|}\left(\frac{\partial y}{\partial \xi}\frac{\partial x}{\partial \varsigma} - \frac{\partial x}{\partial \xi}\frac{\partial y}{\partial \varsigma}\right)$$
$$(J_{31})^{-1} = \frac{1}{|J^{(e)}|}\left(\frac{\partial y}{\partial \xi}\frac{\partial z}{\partial \eta} - \frac{\partial z}{\partial \xi}\frac{\partial y}{\partial \eta}\right)$$
$$(J_{32})^{-1} = \frac{1}{|J^{(e)}|}\left(\frac{\partial z}{\partial \xi}\frac{\partial x}{\partial \eta} - \frac{\partial x}{\partial \xi}\frac{\partial z}{\partial \eta}\right) \qquad \text{Eq. 42}$$
$$(J_{33})^{-1} = \frac{1}{|J^{(e)}|}\left(\frac{\partial x}{\partial \xi}\frac{\partial y}{\partial \eta} - \frac{\partial y}{\partial \xi}\frac{\partial x}{\partial \eta}\right)$$

The Gaussian Quadrature method was used to integrate the stiffness matrix and load vector. Two Gauss points were used for each dimension, yielding a total of eight Gauss points. The weight of each Gauss point is 1. The

function that defines each element of the stiffness matrix is evaluated for each Gauss point and the results summed.

$$\iiint_e K_{ij} = \sum_{k=1}^{n}\sum_{l=1}^{n}\sum_{m=1}^{n} w_{nk} w_{nl} w_{nm} f(\xi_k, \eta_l, \varsigma_m)$$
where, $n = 2$
and $\xi_1 = -\frac{1}{\sqrt{3}}, \xi_2 = \frac{1}{\sqrt{3}}, \eta_1 = \xi_1, \eta_2 = \xi_2, \varsigma_1 = \xi_1, \varsigma_2 = \xi_2$
and $w_{nk} = w_{nl} = w_{nm} = 1$ Eq. 43

The entries of the load vector were also calculated using numerical integration. The finite elements used have an $O(h^2)$ accuracy and the quadrature integration an $O(h^6)$ accuracy. By using numerical integration, no more error is introduced than had already been introduced through the finite element approximations.

The use of numerical integration allows finite element implementation to be more flexible. More specifically, if we change the equations that are being solved, the determination of the integral of the finite element equations is not necessary. No appreciable slowdown is introduced into the calculation for the model since the Gaussian Quadrature integration requires a fixed number of steps.

Solution of Time Differential Term

The general matrix form of the element equations for differential equations containing a time differential in finite elements is:

$$[C]\left\{\frac{da(t)}{dt}\right\} + [K]\{a(t)\} = \{F(t)\} \quad \text{Eq. 44}$$

By substituting a backward difference approximation for the time differential term of Eq. 44, the element equations becomes:

$$[C]\left\{\frac{a(t+\Delta t)-a(t)}{\Delta t}\right\}+[K]\{a(t+\Delta t)\}=\{F(t+\Delta t)\} \qquad \text{Eq. 45}$$

Eq. 45 can then be rewritten as the following of a recurrence relation:

$$[[C]+\Delta t[K]]\{a(t+\Delta t)\}=\Delta t\{F(t+\Delta t)\}+[C]\{a(t)\} \qquad \text{Eq. 46}$$

Eq. 46 is a fully implicit representation.

A fully implicit representation is the most stable to solve but uses the most computer time. For this reason, if an explicit solution is stable it is often used. The explicit representation of the gas equations for this simulation was stable, and the time differential term of Eq. 4, $\frac{\partial \Phi}{\partial t}$ is solved using a forward difference approximation. The substitution of this difference approximation in Eq. 17 provides an explicit time calculation of each time step. This forward difference approximation provides less stability than a central difference or backward difference approximation (Burnett87), but it greatly reduces the execution time and space requirements of the 3D FEM approximation.

PARALLEL IMPLEMENTATION OF A FRONTAL FINITE ELEMENT SOLVER

This section presents timings and other information for a parallel-prototype application developed for the solution of a finite element problem. This prototype was originally developed using MPI (MPIF94) and C++ on a cluster of Window NT workstations and has been successfully recompiled, implemented, and executed on three additional computer platforms: Linux cluster, PowerPC cluster, and multiprocessor SGI workstation. Benchmarks to measure the speedup in the time required for execution are reported for the different platforms listed. The approach and effort that was required for the MPI implementation on each platform is presented plus the job scheduling, resource partitioning and parallel commands used for each.

Parallel computing can reduce the execution time required for computations by taking advantage of multiple processors. The parallel-prototype application used in this dissertation was developed for the finite element problem, and its solution eventually becomes a linear systems problem. Although substantial effort has been used in the solution of linear systems; the frontal solution technique has received very little of this effort. This may be attributed to the inherent serial nature of the algorithm used in a frontal solution.

Löhner (Löhner87) made a distinction between mildly parallel machines (fewer than ten processors) and massively parallel machines. The frontal solution technique is well known to not make efficient use of massively parallel computers. However, its use with mildly parallel machines is promising. With

mildly parallel machines, a frontal solution has the potential to overlap both the assembly and elimination of elements from the stiffness matrix and load vector. One target platform for this research is a small cluster of Windows NT workstations. Windows NT workstations are more accessible than massively parallel computers. These NT workstations are available to almost everyone including small engineering firms. This type of workstation cluster is less expensive and provides an already existing platform for the utilization of parallel code in industry.

Algorithm

Irons (Irons70) originally developed the frontal solution algorithm as an alternative to algorithms that used a banding technique for the coefficients in the stiffness matrix. This algorithm is even more efficient when there are nodes other than corner nodes (nodes 1-4, 10-13 of Figure 4) in the elements of the finite element mesh. Finite elements that contain nodes in addition to corner nodes frequently occur in unstructured grids and adaptive mesh schemes that better describe the environment in a reservoir (aquifer) simulation. The efficiency of the algorithm emphasizes an ordering of elements rather than nodes. Several methods have been published for the renumbering of meshes to minimize the frontal-width of a mesh and the resultant linear equations (Pina81, Sloan83, Sloan86).

The frontal solution technique is based on the Gaussian elimination method to solve the linear system of equations. When all contributions to the stiffness matrix and load vector have been accumulated for a variable with this

method, the equation for that variable can be eliminated from the linear system. The elimination of the equation for this variable does not affect the elimination of other variables. By assembling elements that contribute to that equation as soon as possible, the equation can be eliminated before all the elements are assembled. This allows the assembly of elements to be processed by one or more processors while another processor simultaneously handles the elimination of equations. A more detailed description of the frontal solution technique has been published (Irons70) and is available to the interested reader.

Element Dependencies

To determine all the dependencies of a node (variable) in the "assembled" equations, a data structure (see Figure 17) was designed containing information about the dependencies of all the elements. For each element, a list of element numbers is maintained for all the elements that share nodes with it. When all elements that share a mesh node have been assembled, the contributions to that node's equation in the stiffness matrix have been determined. This structure needs to be modified whenever an element is refined to reflect any new dependencies.

An element dependency structure is created from the initial mesh interconnection of nodes and each parent node (node 1 Figure 6) is assembled. For each node of an element, all the directional pointers are traversed to determine its neighboring elements. The parent node numbers of the elements in this collection of neighbors from this traversal are added to the element

dependency array. Then the index of the first neighbor's parent node number in the dependency array is stored in an index array.

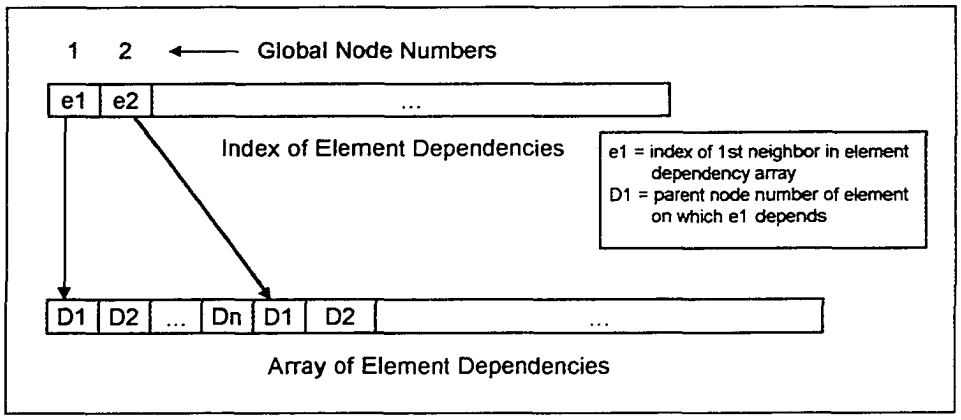


Figure 17 - Structure Maintained for Element Dependencies

Solution Process

The element assembly is done simultaneously on multiple processors. Each processor has its own set of elements to partially assemble. As the elements are partially assembled, their contribution to the stiffness matrix and load vector is transmitted back to the root process (the process with rank 0 in MPI). These contributions to the stiffness matrix and load vector are accumulated. Any withdrawal of condensate from the reservoir is also accumulated in the load values during the solution process. When the equations are completely assembled, they are candidates for elimination as well as all the other nodes that make up the element. These equations are being eliminated while other elements are being assembled simultaneously.

To accomplish speedup in this finite element assembly process, the elements are interleaved on different processors (Figure 18). Before the

elements are interleaved to different processors, they are placed in an order that minimizes the frontwidth of the linear system. The frontwidth is the maximum numbers of nodes that are required to eliminate equations. The frontal node renumbering algorithms provide an ordering which attempts to minimize frontwidth. The initial mesh construction uses a numbering scheme for nodes that resulted in a small frontwidth. The assignment of elements to processors for assembly can be interleaved because connected elements are assembled at the same time. The first element is sent to the first processor, the second element to the second processor, etc., and this continues for all the processors available for assembly. If the frontwidth is greater than the number of processors, then the maximum $\left\lceil \frac{\text{Number of Processors}}{\text{Frontwidth}} \right\rceil$ steps of the assembly process is used on each processor before beginning the elimination.

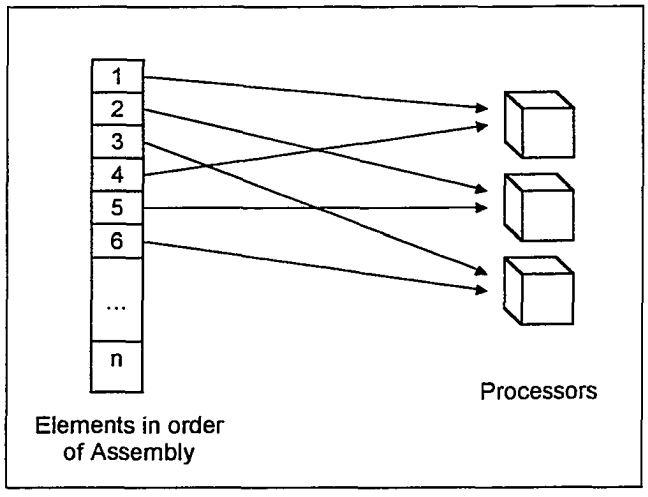


Figure 18 - Interleaved Assignment of Elements to Processors

When the elimination processor (rank 0 in MPI) receives an assembled element, it adds this contribution to the stiffness matrix and load vector. Parent nodes that have not been assembled are assigned an initial status of 0. When parent nodes are assembled, their status is changed to 1. To determine if an equation (e) is a candidate for elimination, the dependency array is checked for the node with global node number (e). If each of the parent nodes of the elements have a status of 1, then the equation (e) can be eliminated. Some nodes are not in the dependency array because they are not parent nodes. These non-parent nodes can only be eliminated after their parent node.

To eliminate an equation only the non-zero columns for this equation in the stiffness matrix are used, reducing computations. After all equations are assembled and eliminated, the solution is stored in the load vector. This solution is used to make decisions about mesh adaptation. It is stored in the nodes for use with the next time-step.

The following summarizes the solution process:

1. The initial mesh is built.
2. The dependency array for the elements is created.
3. The elements are assembled on multiple processors.
4. When the elements are assembled, this is communicated to the root process where their status is kept, updated, and checked to determine if the equations can be eliminated.
5. Equations are eliminated and the status of each of their nodes updated.

6. Any necessary mesh adaptation is done.
7. Equation elimination results are communicated to remaining processes.
8. Continue at step 3 until completion.

The solution process is not a balanced algorithm. As processors are added, the assembly process will take less time than the elimination. This algorithm is well known to not be scaleable to a large number of processors. For mildly parallel clusters of workstations, those with only a few processors, this algorithm results in a speedup. The complexity of this algorithm is less than most other solutions; hence its implementation cost is also less.

Hardware Platforms

NT Workstation Cluster

To demonstrate how easily a Windows NT workstation cluster can be obtained for parallel use, 2 Windows NT workstations were used as the hardware platform in my home. Both of these were purchased with my funds and were being used for other work. Each Windows NT workstation had a 100 MHz Intel Pentium processor, Windows NT 4.0, 32 megabytes of RAM, 1 gigabyte of disk storage, and a 10-megabit Ethernet port. The free WinMPich MPI libraries were downloaded from Mississippi State University to be used for communication between processors. The WinMPich libraries are based on MPICH, a joint project of Mississippi State University and Argonne National Laboratory. The WinMPich MPI libraries were installed with no previous experience with this type of installation. The document at

http://www.erc.msstate.edu/mpi/mpiNT.html was located and I asked my MPI installation questions. shane@erc.msstate.edu responded promptly and after 2 hours of additional MPI installation effort, the WinMPich MPI libraries were available for use. The Microsoft Visual C++ compiler (Microsoft97) was successfully used to compile the C++ source with the MPI functions included. For benchmarking purposes, the Windows NT cluster of the Physics department at Southern University was used. This cluster has 4 nodes available for parallel code. The WinMPich MPI environment and compiler are the same as the previous NT system. The 4 machines used for benchmarking were:

1. 300 MHz Intel Petium II, 128 megabytes of RAM.
2. 133 MHz Intel Pentium, 82 megabytes of RAM.
3. 166 MHz Intel Pentium w/MMX, 32 megabytes of RAM.
4. 166 MHz Intel Pentium w/MMX, 32 megabytes of RAM.

A 10-megabit Ethernet network was used for network connectivity (Figure 19).

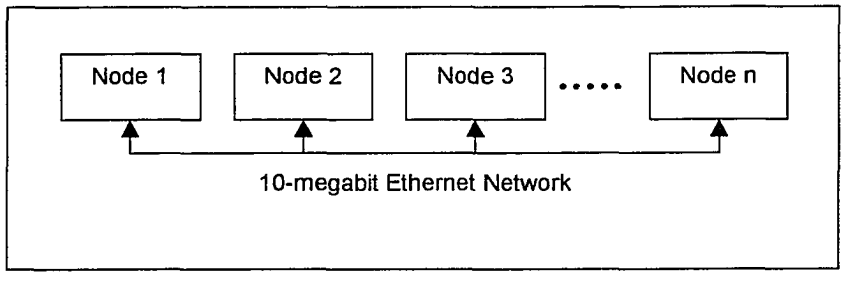


Figure 19 – Diagram of Microsoft Windows NT cluster

The WinMPich MPI environment was implemented as an NT service installed on each of the NT workstations. No job scheduling was supported and only immediate execution was used. Jobs can be submitted from each

workstation using a configuration file. This file is frequently used with MPI and contains a list of the processes to be created (executable names) and a list on which workstations they should be executed. It should be noted that a shared memory execution is also supported by this MPI implementation for the SMP (symmetric multiprocessor) version of Windows NT, although we did not use it.

Linux Workstation Cluster

The Linux workstation cluster used is located in the Computer Science Department at the University of Montana in Missoula, Montana. It was very interesting because it was implemented with freeware (the effort of D. J. Morton). This Linux workstation cluster consists of 9 (including the front-end machine) 100 MHz Intel Pentium processors, each with 64 megabytes of RAM (Figure 20). The cluster communicated with a 100-megabit Ethernet network. A 10-megabit Ethernet network connects the front-end machine with the outside network. The compiler used for C++ was a GNU C++ compiler. A LAM (Local Area Multicomputer) MPI implementation is used on the cluster to provide communication between processors.

The LAM implementation provides centralized management of the MPI environment. The front-end computer initialized the MPI environment on all of the Linux workstations and submitted jobs. All nodes were considered homogenous (same amount of memory, disk storage and processing speed). No job scheduling is provided and jobs execute immediately when submitted using the "mpirun" command.

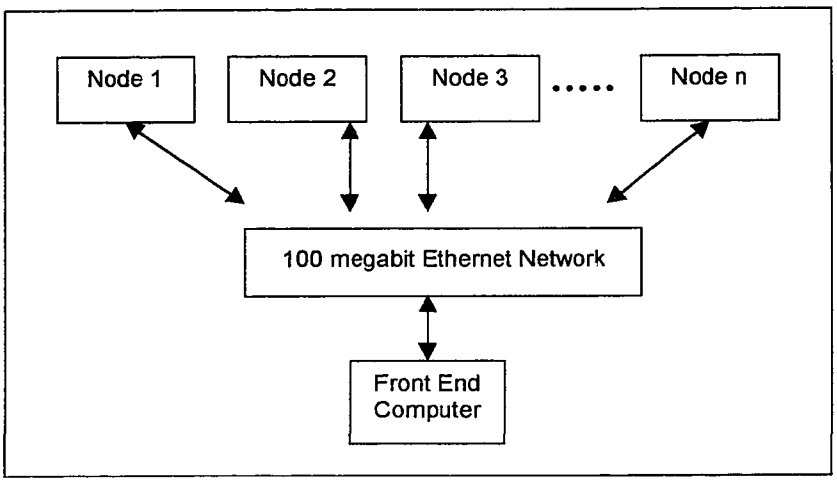


Figure 20 – Diagram of Linux cluster

PowerPC Workstation Cluster

The PowerPC cluster used consists of IBM RS/6000 PowerPC workstations. The workstations are connected with a 10-megabit Ethernet network (Figure 21). The MPI environment is managed through a single workstation (home node) using the POE (IBM Parallel Operating Environment v2.2) software (IBM96). The mpCC C++ compiler, a basic part of the POE installation, was used to compile the source code.

The execution of MPI jobs is performed immediately after submission to the POE. Before the execution of the job, the executable and any necessary input files must be copied onto the remote machines using the "mprcp" command. Then the "poe" command is used to set the environment options and execute a job.

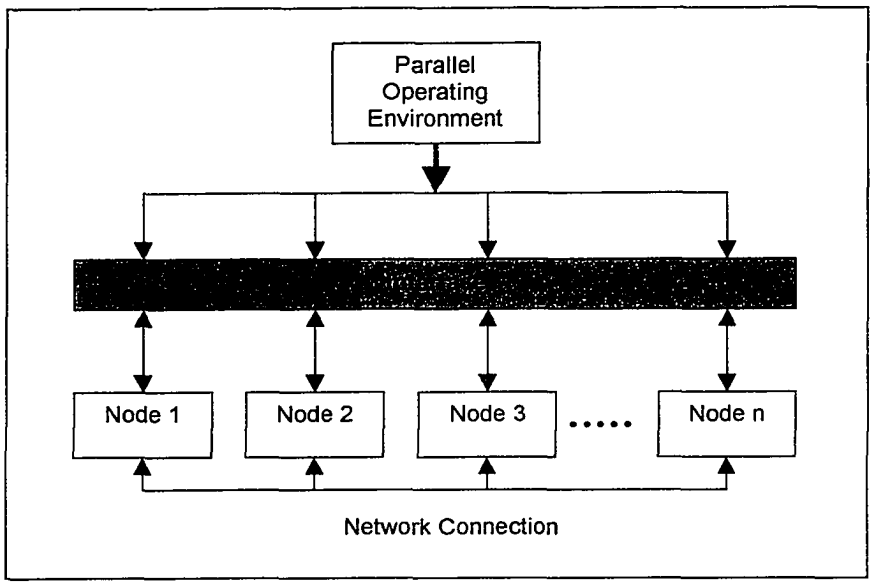


Figure 21 – Diagram of PowerPC cluster

Silicon Graphics Origin Multiprocessor Workstation

The Silicon Graphics Origin workstation used is a 4 processor shared memory machine. It had 4 MIPS R10000 CPUs with 1 gigabyte of main memory. The MIPSpro C++ Compiler version 7.20 was used to compile the source code. The MPI environment was Silicon Graphics' proprietary implementation of MPI. The total time to convert the source code to this environment was less than 30 minutes. Only minor changes to the structure of the source files were necessary.

The jobs are executed when they are submitted and no job scheduling is provided in the basic MPI environment. Although the MPI implementation is proprietary, it closely follows MPI standards. The "mpirun" command was used to submit jobs.

Results

Our parallel frontal FEM solver has been tested on all the platforms listed and a benchmark was developed for comparison. This benchmark consisted of the first 31 days of the Eugene Island Reservoir simulation using the parallel-prototype FEM application. Exclusive use of some of these platforms for longer periods of time was not available in all cases, so this reduced simulation time permitted each benchmark to run on dedicated hardware without sharing resources with any other processes. By running the prototype simulation for only 31 days of the simulation, the total execution time is reduced. This reduction in execution time also reduces machine dependent effects while maintaining sufficient execution time to be a representative test. These results are not measures of the absolute best performance of the hardware, but are measures of the relative performance of the parallel frontal algorithm.

The chart of results (Figure 22) shows the total time to complete the month of simulation for the prototype for different hardware platforms and different number of processors. The time of the sequential algorithm (not frontal algorithm) was 1349 seconds on one of the Windows NT workstations used for the parallel benchmarking. For 2 CPUs, the Windows NT cluster took 888 seconds to execute the prototype. The speedup for 2 CPUs is 1.51, and the speedup for 3 CPUs (1.68) is only slightly better than with 2 CPUs. This points toward potential additional gains in the future by distributing the elimination process in addition to the assembly of the stiffness and load matrices. For 4 CPUs the speedup is 1.69.

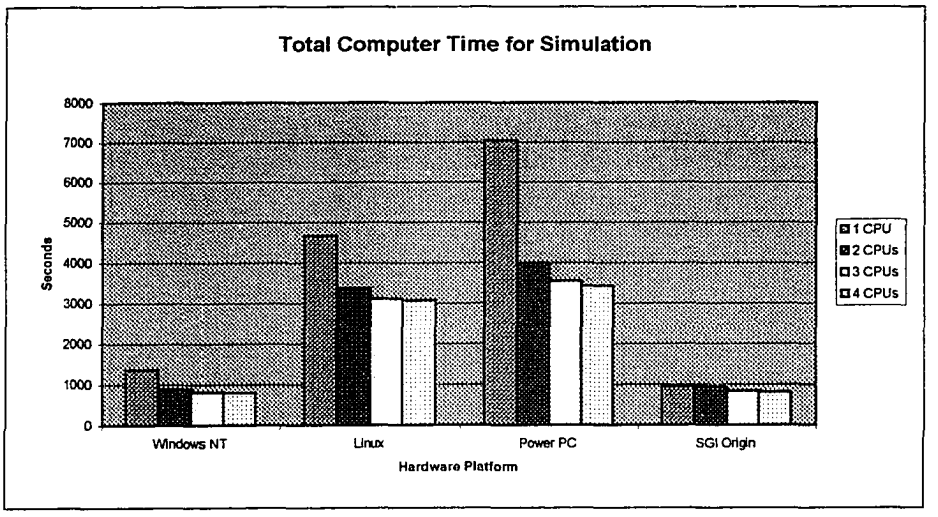


Figure 22 – Total computer time for simulation

Other environments executed the parallel MPI code as presented on Figure 22. The total time necessary to complete the benchmark was reduced as additional processors were used. For the Linux cluster the time for a single CPU using the sequential algorithm was 4659 seconds. For 2 CPUs the time was 3366 seconds. For 3 CPUs the time was 3110 seconds and for 4 CPUs the time was 3056 seconds. The Power PC cluster took 7038 seconds with the sequential algorithm on a single CPU. The cluster was timed at 3985 seconds for 2 CPUs, 3536 seconds for 3 CPUs, and 3412 seconds for 4 CPUs. The times measured for the SGI Origin workstation were 942 seconds for a single CPU using the sequential algorithm, 932 seconds for 2 CPUs, 828 seconds for 3 CPUs, and 811 seconds for 4 CPUs. The parallel source code could be compiled and executed on these platforms without significant effort.

As expected, the overall execution time decreased with additional processors for all the platforms used. The reduction in execution time diminished

at a very small number of processors because of the imbalance in the algorithm. During the development of the parallel code, the assembly process for the finite elements was optimized. Through optimization, the time for assembly of an element's contribution to the stiffness matrix and load vector was substantially reduced. This reduction in the calculations caused the imbalance in the algorithm and more time was required for elimination. With our future addition of p-refinement for the elements, this time will increase relative to the elimination of equations from the system of equations and tend toward a better balance in the algorithm.

By using the parallel programming interface (MPI) and programming language (C++), we have been able to port our parallel prototype application to different computer platforms with minimal effort. These results do not indicate the relative performance of the operating systems; e.g., the Windows NT workstations executed the prototype much faster than the Linux cluster. This is because the nodes of the Windows NT cluster (166 MHz MMX Intel CPU) are more powerful than the nodes of the Linux cluster (100 MHz Intel CPU). Each of the platforms performed well during the benchmarking.

A final consideration is that given the small number of massively parallel machines available versus the large number of Windows NT workstations that can be formed into clusters, which will form a basis for mainstream parallel computations?

EUGENE ISLAND HISTORY MATCHING

The physical characteristics for our demonstration problem are from Eugene Island. This data has been used as input for the three dimensional FEM prototype. The data available for Eugene Island was documented in field units and required conversion to make it useful as input to the prototype. The formulas and conversions are presented along with the results of the prototype simulation and its match to the actual production history. These results show agreement and are more descriptive than those previously presented in the one dimension case provided by Halford (Halford85).

Physical Reservoir Properties

The condensate in the reservoir was analyzed and found to be composed of the components listed in Table 1. The composite molecular weight and specific gravity are used in the calculation of viscosity and z-factor during the reservoir simulation.

Table 1 – Eugene Island Fluid Properties

Components	Mol%	Molecular Weight	per Comp	Specific Gravity	Per Comp
Carbon Dioxide	0.0009	44.01	0.039609	1.5194	0.001367
Nitrogen	0.0007	28.016	0.019611	0.9672	0.000677
Methane	0.9292	16.042	14.90623	0.555	0.515706
Ethane	0.0253	30.068	0.76072	1.046	0.026464
Propane	0.0093	44.094	0.410074	1.547	0.014387
Iso-Butane	0.003	58.12	0.17436	2.067	0.006201
N-Butane	0.0036	58.12	0.209232	2.071	0.007456
Iso-Pentane	0.0019	72.146	0.137077	2.4906	0.004732
N-Pentane	0.0018	72.146	0.129863	2.4906	0.004483
Hexanes	0.0029	86.172	0.249899	2.9749	0.008627
Heptanes Plus	0.0214	175	3.745	0.8033	0.017191
Composite	1		20.78167		0.607291

The pressure to z-factor table (Table 2) was used for the prototype simulator instead of a generalized equation for better agreement with the actual production data and previous simulations. This table was converted to a cubic polynomial using least squares. The actual z-factor data versus this least squares polynomial are presented on Figure 23. The agreement is excellent with an R^2 value of greater than 0.99.

Table 2 - Z Factor versus Pressure for Eugene Island

Pressure(PSIA)	Z factor
0	1
1477	0.9022
1993	0.8936
2675	0.8974
3183	0.9109
4000	0.948
5000	1.0123
6000	1.0908
7000	1.1771
7500	1.2224
7680	1.2393
8139	1.2821
8445	1.3112
8895	1.354

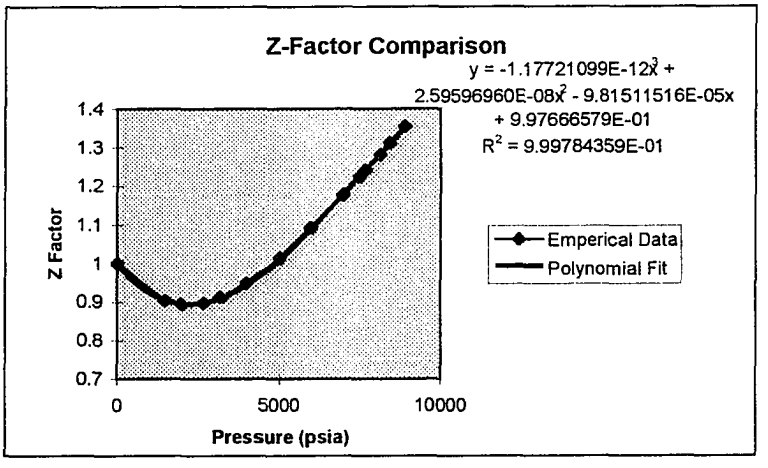


Figure 23 - Z - Factor Calculation

Conversion of Surface Data for FEM Prototype Input

The original input data was in field units (barrels, psi) at surface conditions. These surface quantities were converted into units that agree with those used in the prototype simulator. This was necessary to convert the reservoir condensate quantities produced into units that could be used for a gas. Using the standard equation of state for a gas (Eq. 47), we converted surface volumes into well-bore volumes.

$$Pv = ZnRT \qquad \text{Eq. 47}$$

The number of molecules of condensate is the same at the surface as well-bore and this equality produces equation 48.

$$\frac{P_s v_s}{Z_s R T_s} = \frac{P_w v_w}{Z_w R T_w} \qquad \text{Eq. 48}$$

Where the subscript s is at surface conditions and w is at well-bore conditions. The well-bore volume (v_w) can be obtained from Eq. 48 yielding Eq. 49. The surface volume in terms of other known quantities is:

$$v_w = \frac{Z_w T_w P_s v_s}{Z_s T_s P_w} \qquad \text{Eq. 49}$$

For the Eugene Island reservoir:

T_S	=	519° (Rankin)
P_S	=	14.96 (psia)
Z_S	=	0.98
T_W	=	654° (Rankin)
P_W	=	8138.96 (psia)
Z_W	=	1.37

It was also necessary to convert the water produced from the Eugene Island reservoir into an equivalent amount (mass) of gas to account for the water

produced and to obtain the correct pressure change. Equation 50 shows the conversion of produced water to gas. The produced water is converted from bbl/day of water to its equivalent in mcf/day of gas.

$$\text{gas (mcf/day)} = 7.39 \times \text{water(bbl/day)} \qquad \text{Eq. 50}$$

Similarly, the oil produced in the condensate is also converted to an equivalent amount of gas using equation 51. The specific gravity and molecular weight of the condensate (Table 1) is used in this relationship.

$$\text{gas(mcf/day)} = 133000 \frac{sp.grav}{MW} \frac{\text{oil(bbl/day)}}{1000} \qquad \text{Eq. 51}$$

These conversions are also shown in Appendix A. The gas withdrawn from the Eugene Island Reservoir contains oil and water. This oil and water is included in the gas production value as their equivalent in gas from Eq. 50 and Eq. 51.

Simulation Results for Eugene Island

Simulation Testing Methodology

To evaluate the accuracy of the approximate solution using this new three-dimensional-FEM prototype, simulation results were compared to the actual well test data and production reports. The prototype simulation was run on 1,981 days of the reservoir production which began production on August 1, 1979. This segment of the reservoir production was chosen because it was a very representative time interval and there was accurate available input data. During this time interval, wells were put into production in a staggered fashion as they were completed. The production continued for all wells until some of the wells produced water at a rate that was no longer economical and they were

shutdown. When a well "watered" out, the production from the reservoir was reduced.

Halford (Halford85) presented a simulation of the Eugene Island reservoir in his 1985 master's thesis using a radial reservoir simulator in which the pressure throughout the reservoir was averaged. This is sometimes referred to as a simple tank model. To determine the best history match, Halford varied the permeability of the reservoir. Halford's best solution was compared to the actual pressure averages for all wells and a minimum error of 170 psi was obtained.

The data collected on production reports is two-dimensional. The pressure at the well-bore was recorded as a single value regardless of the depth of the measurement. In contrast to Halford's one-dimension simulation, the new prototype simulation is three-dimensional and calculates the pressure at all locations throughout the reservoir and at all depths. The results from the prototype in the area of the well-bore were manually averaged to get a single value only to make comparison to the original production data. Because of this "averaging" of the true reservoir results, the increased resolution of the prototype simulation is not fully presented in this comparison of results.

The actual production rates of Eugene Island were used as input for our prototype. The production of water and oil was converted to an equivalent amount of gas for the reservoir. The permeability used for the simulation was the average permeability for the reservoir (Halford85). To find a best solution with the prototype for comparison with Halford's, the initial average permeability was reduced 18 md. The initial pressure of the reservoir was assumed to be a

constant value and was used as the initial pressure for each well as it went on-stream. Further evaluation of the original production reports show evidence that this was not the case, but more accurate initial pressures were not available. If the initial pressure of the well-bore was not homogeneous, then differences in final approximate well-bore pressures from actual well measurements would be expected.

Comparison of Observed / Calculated Well Bore Pressures

Figure 24 - Figure 29 present the calculated results at each of the 6 well-bore locations over the first 1,981 days of the production of the condensate reservoir. Presented along with the calculated results are the pressures at the well-bore from actual well measurements performed during production (Table 3).

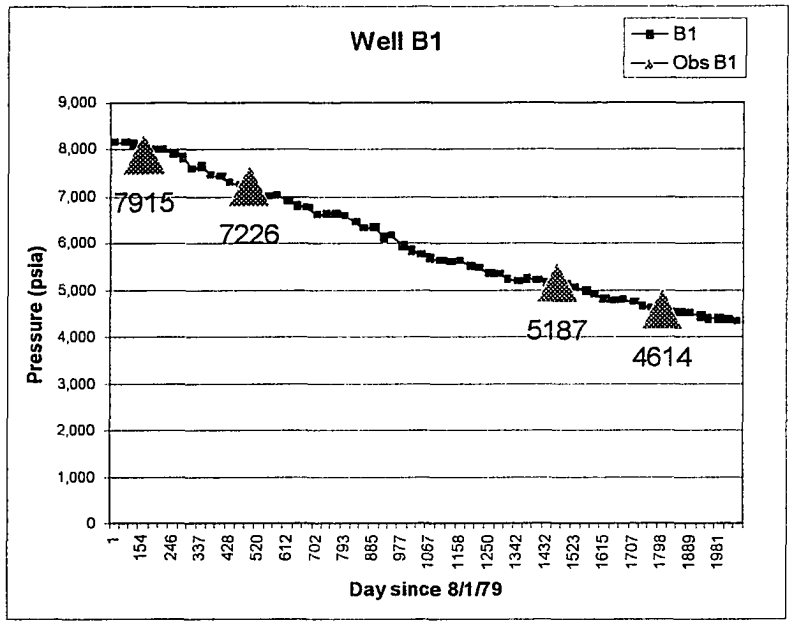


Figure 24 – Results for Well B1

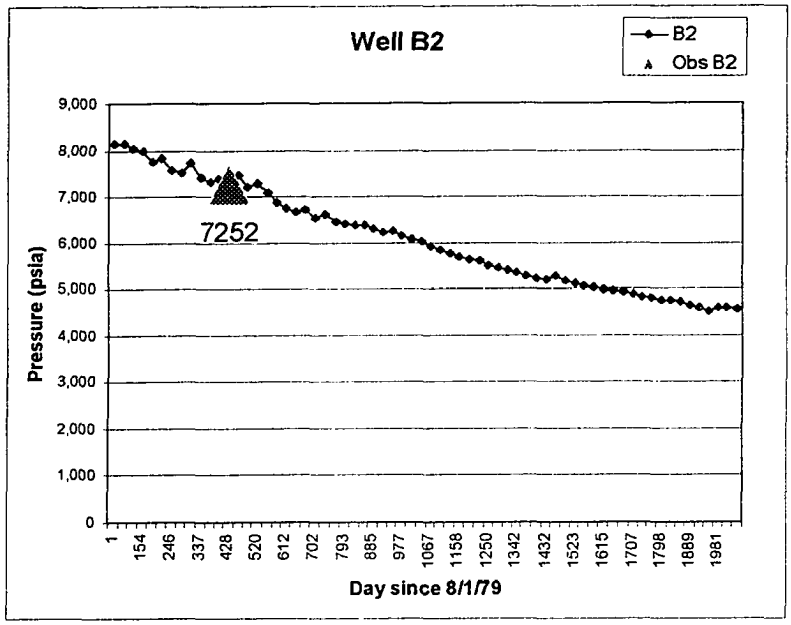


Figure 25 – Results for Well B2

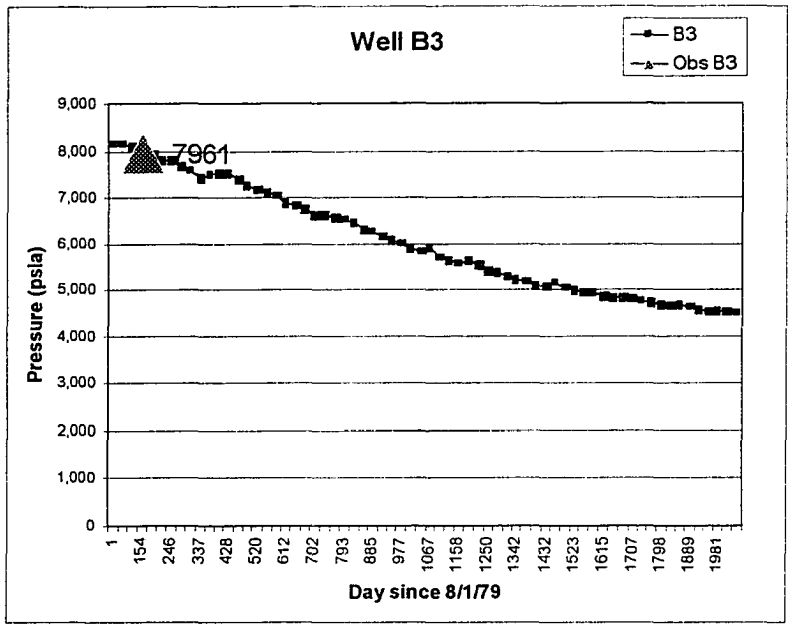


Figure 26 – Results for Well B3

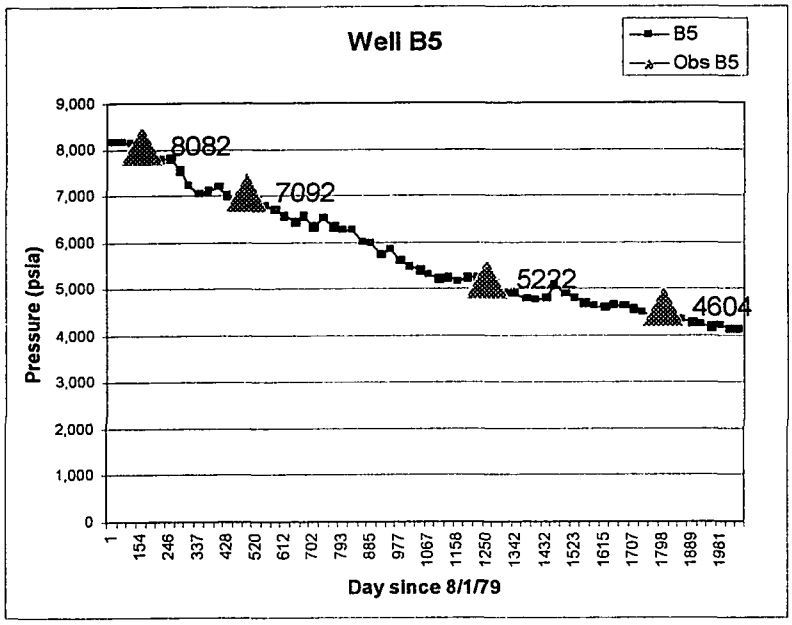


Figure 27 – Results for Well B5

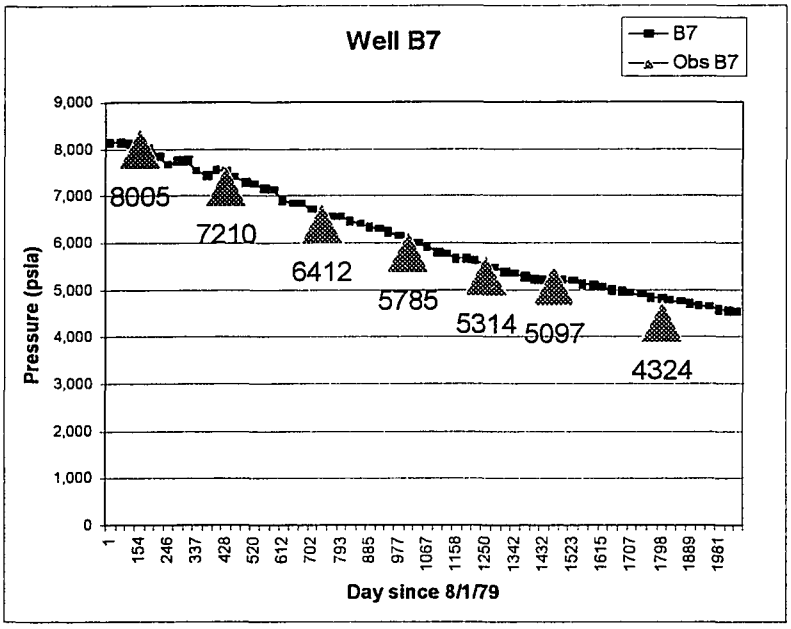


Figure 28 – Results for Well B7

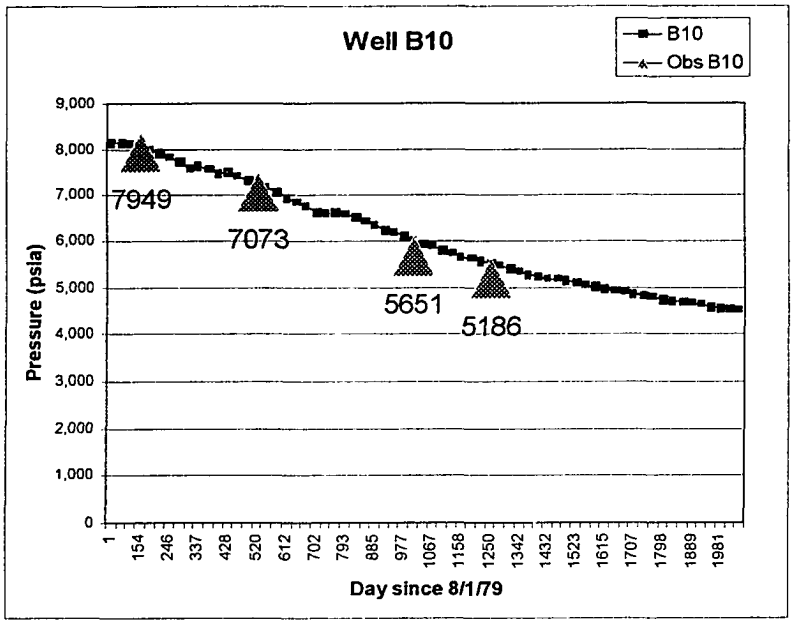


Figure 29 – Results for Well B10

Table 3 – Well-bore Pressure Data

Day	B1	B2	B3	B5	B7	B10	Obs B1	Obs B2	Obs B3	Obs B5	Obs B7	Obs B10
1	8,138	8,138	8,138	8,138	8,138	8,138						
93	8,138	8,138	8,138	8,138	8,138	8,138						
124	8,127	8,033	8,096	8,117	8,132	8,119						
154	8,033	7,993	8,087	7,959	8,114	8,072	7915		7961	8082	8005	7949
185	7,997	7,763	7,928	7,881	8,008	7,992						
215	7,985	7,827	7,805	7,782	7,838	7,918						
246	7,919	7,567	7,796	7,803	7,644	7,814						
277	7,826	7,534	7,684	7,548	7,762	7,737						
306	7,582	7,739	7,572	7,202	7,752	7,586						
337	7,639	7,409	7,424	7,024	7,533	7,616						
367	7,430	7,313	7,491	7,097	7,458	7,553						
398	7,414	7,385	7,531	7,186	7,548	7,476						
428	7,290	7,444	7,518	7,003	7,519	7,502		7252			7210	
459	7,235	7,482	7,397	6,854	7,388	7,401						
490	7,269	7,209	7,254	7,081	7,281	7,323	7226			7092		
520	7,080	7,286	7,150	6,910	7,231	7,188						7073
551	7,002	7,082	7,117	6,762	7,167	7,153						
581	7,011	6,878	7,033	6,704	7,103	7,071						
612	6,931	6,752	6,882	6,549	6,915	6,922						
643	6,811	6,667	6,812	6,443	6,839	6,824						

(TABLE continued)

(TABLE continued)

671	6,774	6,705	6,737	6,552	6,835	6,757						
702	6,610	6,539	6,608	6,338	6,687	6,632						
732	6,627	6,616	6,602	6,508	6,604	6,597					6412	
763	6,627	6,460	6,555	6,332	6,551	6,623						
793	6,578	6,406	6,522	6,259	6,534	6,578						
824	6,460	6,380	6,458	6,250	6,476	6,514						
855	6,317	6,378	6,308	5,985	6,383	6,402						
885	6,344	6,300	6,251	5,964	6,347	6,332						
916	6,126	6,237	6,138	5,737	6,273	6,241						
946	6,157	6,263	6,078	5,849	6,238	6,177						
977	5,952	6,147	5,983	5,595	6,139	6,092						
1008	5,848	6,074	5,900	5,478	6,060	6,004					5785	5651
1036	5,761	6,005	5,816	5,387	5,980	5,924						
1067	5,694	5,913	5,907	5,302	5,902	5,880						
1097	5,626	5,837	5,692	5,202	5,792	5,786						
1128	5,605	5,760	5,635	5,229	5,744	5,714						
1158	5,607	5,682	5,555	5,148	5,668	5,637						
1189	5,515	5,618	5,636	5,251	5,668	5,624						
1220	5,472	5,611	5,524	5,252	5,619	5,570						
1250	5,361	5,506	5,405	5,035	5,510	5,497				5222	5314	5186
1281	5,321	5,449	5,362	5,112	5,458	5,467						
1311	5,218	5,404	5,265	4,876	5,392	5,399						
1342	5,180	5,333	5,208	4,880	5,331	5,333						
1373	5,249	5,266	5,159	4,780	5,269	5,278						
1401	5,215	5,217	5,095	4,759	5,224	5,229						
1432	5,157	5,193	5,052	4,797	5,186	5,179						
1462	5,157	5,259	5,142	5,071	5,222	5,163	5187				5097	
1493	5,135	5,154	5,036	4,889	5,200	5,137						
1523	5,049	5,119	4,976	4,790	5,171	5,097						
1554	4,990	5,065	4,938	4,700	5,133	5,055						
1585	4,892	5,036	4,921	4,622	5,090	5,017						
1615	4,825	4,985	4,856	4,585	5,046	4,968						
1646	4,773	4,949	4,815	4,632	4,998	4,920						
1676	4,778	4,931	4,824	4,619	4,956	4,890						
1707	4,724	4,874	4,788	4,562	4,918	4,857						
1738	4,662	4,816	4,758	4,500	4,880	4,823						
1767	4,614	4,797	4,712	4,418	4,843	4,785						
1798	4,582	4,738	4,674	4,387	4,802	4,744	4614			4604	4324	
1828	4,536	4,743	4,648	4,345	4,764	4,708						
1859	4,528	4,722	4,658	4,341	4,728	4,684						
1889	4,492	4,641	4,629	4,274	4,697	4,658						
1920	4,449	4,587	4,559	4,223	4,663	4,618						
1951	4,406	4,525	4,507	4,180	4,624	4,574						
1981	4,401	4,578	4,531	4,190	4,588	4,552						
2012	4,363	4,590	4,507	4,137	4,560	4,527						
2042	4,325	4,568	4,482	4,111	4,535	4,501						

In comparing these results to actual well measurements, there is considerable agreement for wells B1, B2, B3 and B5. The maximum difference in these wells throughout the simulation is a difference of 216 psi in well B5 after five years. There was only one actual well measurement for well B2, so very few comparisons can be made for the well. It would be expected that the results for this well would follow the other wells.

The results for wells B7 and B10 showed more of a difference with actual measured data. The maximum difference in the well-bore pressure of these wells was 478 psi after 5 years. Even though these two wells showed more of a difference from actual data, their differences are relatively small.

Other Simulation Results

The FEM results for the Eugene Island Reservoir simulation are available at any location in the reservoir including the well-bore. Since this FEM simulation is three dimensional, results for any point within the reservoir domain are easily obtained. The reservoir pressure for a condensate is very important at different depths within the reservoir and at locations other than the well-bore (Figure 30). These results are for a reservoir location a few hundred feet from well B-10 (Figure 31). One year into the simulation, the pressure at a depth of 100 feet from the top of the reservoir is 7663 psia, 7660 at a depth of 300 feet, and 7678 at a depth of 600 feet. Three years into this simulation, these pressures become 6144 at a depth of 100 feet, 6137 at a depth of 300 feet, and 6150 at a depth of 600 feet. These pressures are different than expected because they do not

necessarily increase with depth. This means that the dew point could occur in part of the reservoir other than in the vicinity of a well-bore.

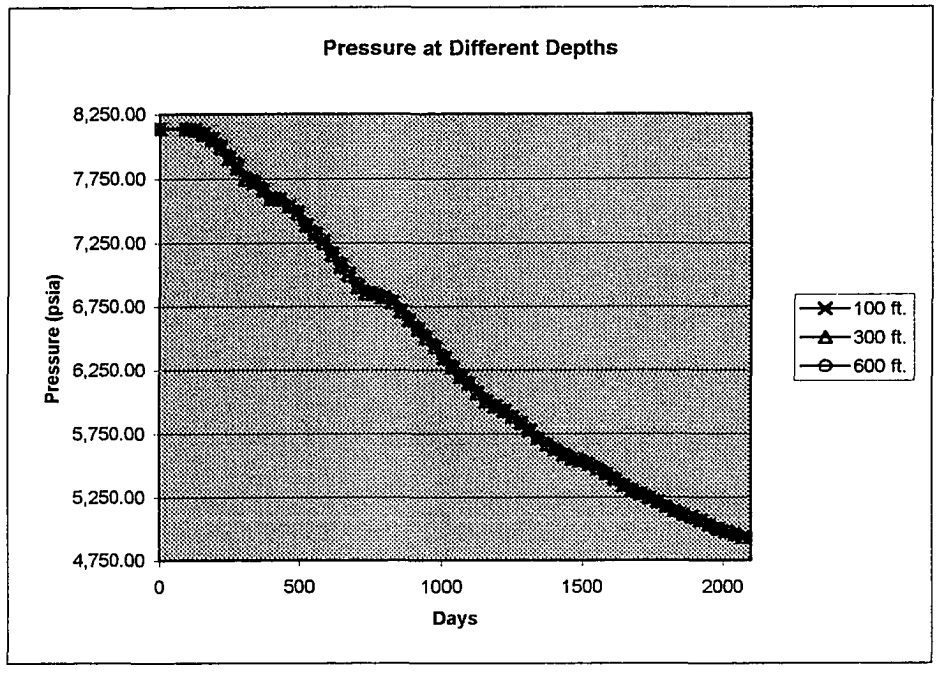


Figure 30 - Pressure Graph of Results at Different Depths

Results, other than at mesh nodes are obtained as follows. The shape functions of the hexahedral element used for the FEM prototype are evaluated to provide the pressure values for any location within the element. The physical coordinate of each node (x, y, z) is a mapping into the finite element which contains the location. The solution is approximated by summing the pressure at each node of the element with each nodal pressure weighted by the value of the shape functions at the points of interest (Eq. 22). This provides a continuous-approximate solution for the pressure throughout the reservoir.

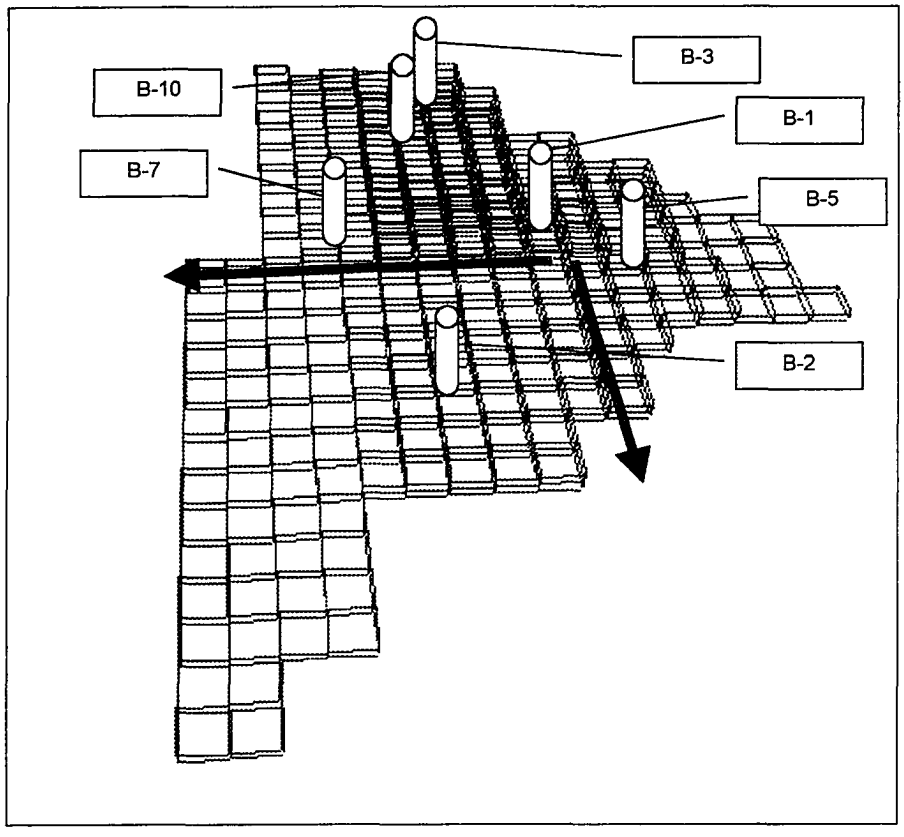


Figure 31 - Well Locations in Eugene Island Reservoir

Using this FEM capability, the reservoir pressure as presented on Figure 32, was evaluated at various distances from well B1 toward the south in the reservoir at 1-4 years into the simulation, as shown on Figure 31. Figure 32 shows the pressure increases as the distance from the well-bore increases. This pressure increase is the most pronounced during the increased production that occurred three years into the simulation. In year four, the pressure began to level as the production rate dropped. Figure 33 shows reservoir pressures for years one through four of the simulation versus distance toward the south of well B-1 at a depth of 200 feet. At 200 feet of depth, the reservoir pressure decreases

for the first 400 feet out from the initial location, as shown on Figure 33. At 500 feet, the pressure begins to increase as shown on Figure 32.

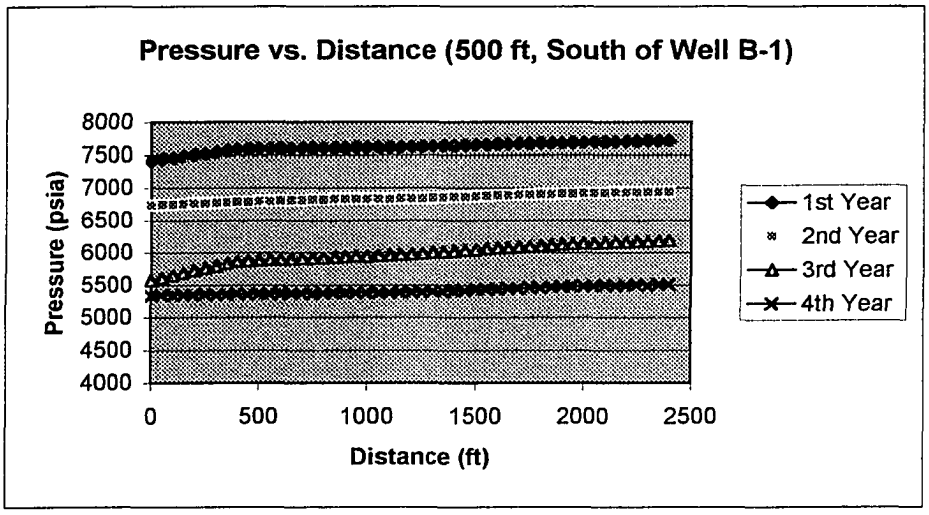


Figure 32 - Pressure vs. Distance (500 ft, South of Well B-1)

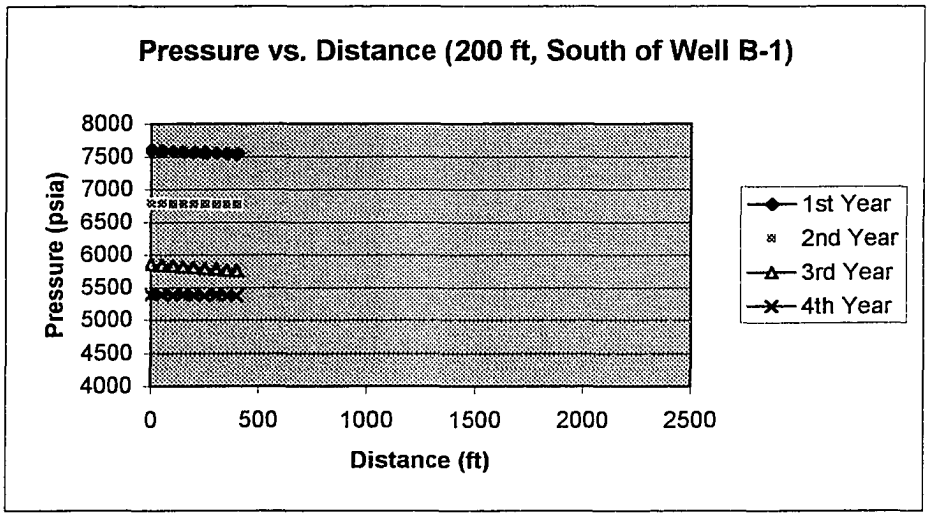


Figure 33 - Pressure vs. Distance (200 ft, South of Well B-1)

Figure 34 presents the pressure versus distance toward the west (Figure 31 shows this direction) from well B-1 and the pressure increases continuously at a depth of 500 feet. The pressure is generally increasing versus the distance from the well vicinity. This increase is gradual at year one and year four in the simulation. But for years two and three, the pressure change is sharper. In particular, the pressure drop near the well vicinity is more pronounced and at a distance of 2400 feet. The pressure in these areas is affected by well B-7 in Figure 31. After a reduction in the production of well B-7 in the fourth year, this pressure decline is not as apparent and the reservoir pressure begins to recover. The vertical differences in the pressure are seen by comparing Figure 34 at a depth of 500 feet with Figure 35 at a depth of 200 feet. For year one, two and three the pressure increase is more gradual at a depth of 200 feet (Figure 35). This steep initial change in pressure at a depth of 500 feet is very different and not seen at 200 feet.

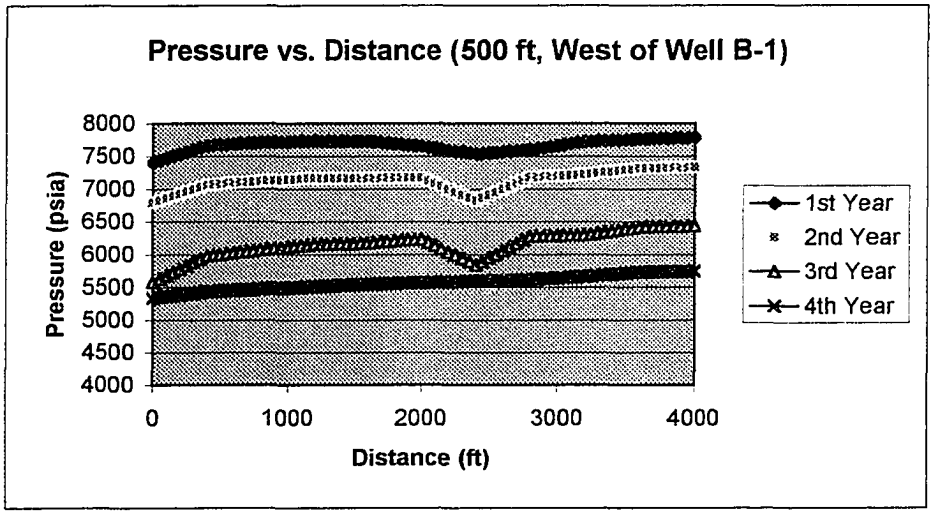


Figure 34 - Pressure vs. Distance (500 ft, West of Well B-1)

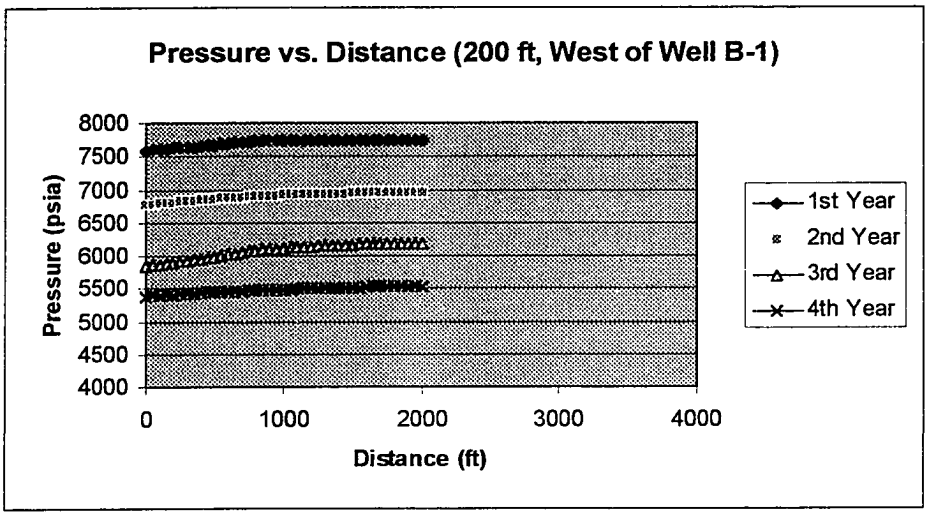


Figure 35 - Pressure vs. Distance (200 ft, West of Well B-1)

For a more complete view of pressure changes throughout the reservoir, a reservoir pressure plot for the first four years of the simulation is shown as a surface on Figures 36-39. The 500 feet depth was chosen because it extends to the boundaries of the reservoir and gives a better representative view of the reservoir. The boundaries of the reservoir are where the pressure becomes zero on these surface plots.

The pressure surface at one year into the simulation shown on Figure 36 is consistent with the known reservoir history. The upper-left-hand corner of the reservoir has a higher pressure due to a lack of production in this region. At each well location there is a noticeable drop in the pressure caused by the production of condensate. At two years into the simulation (Figure 37), the impact on the pressure caused by condensate production at well locations is even more visible. The region of the reservoir in the vicinity of wells B-3 and B-10 show a marked decrease in pressure. The upper-left-hand corner of the

reservoir is now declining in pressure, but not as rapidly as the remainder of the reservoir because no well is located there. The pressure in regions around well-bores show the results of increased production of the reservoir in the third year (Figure 38). The pressure surface is more consistent due to the approach of steady state because the affect of the boundary of the reservoir has occurred. In the fourth year of the simulation (Figure 39), the pressure drop at wells B-7 and B-1 has nearly disappeared with their decreased production (three wells were shut-in). A smoother pressure surface returns in this part of the reservoir.

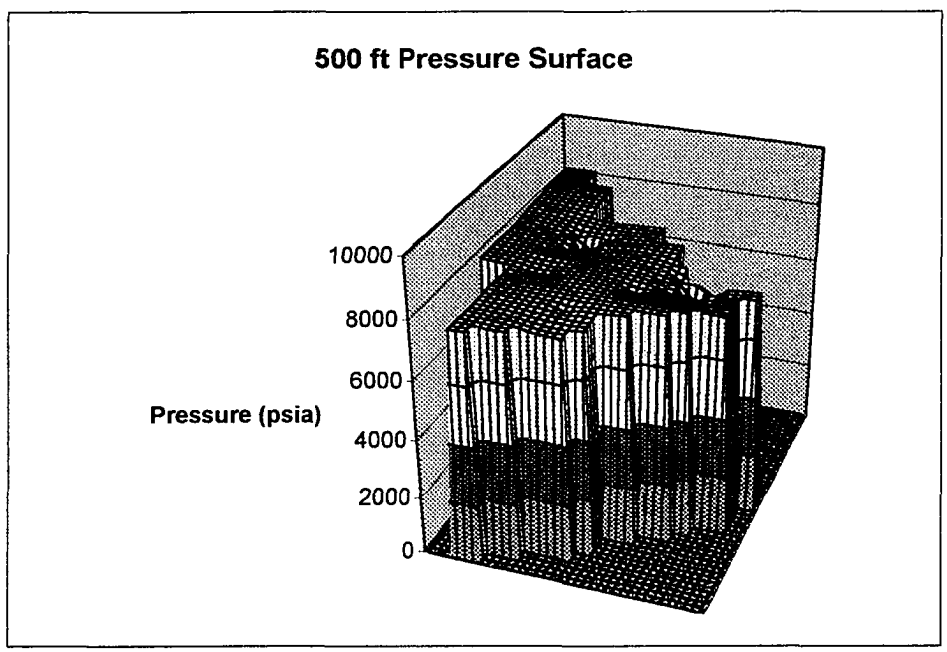


Figure 36 - Pressure Surface at a Depth of 500 ft. - 1st Year

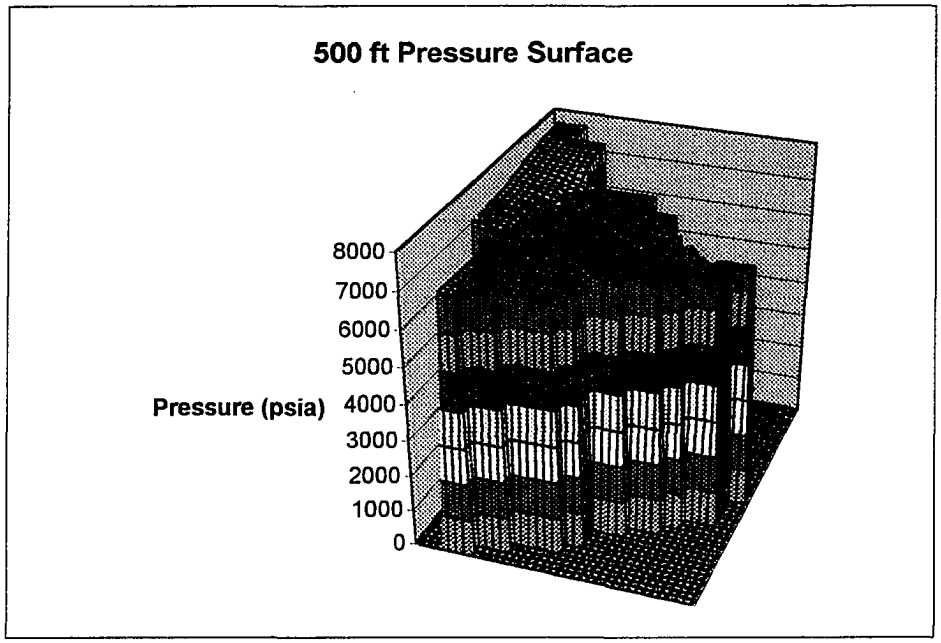


Figure 37 - Pressure Surface at a Depth of 500 ft. - 2nd Year

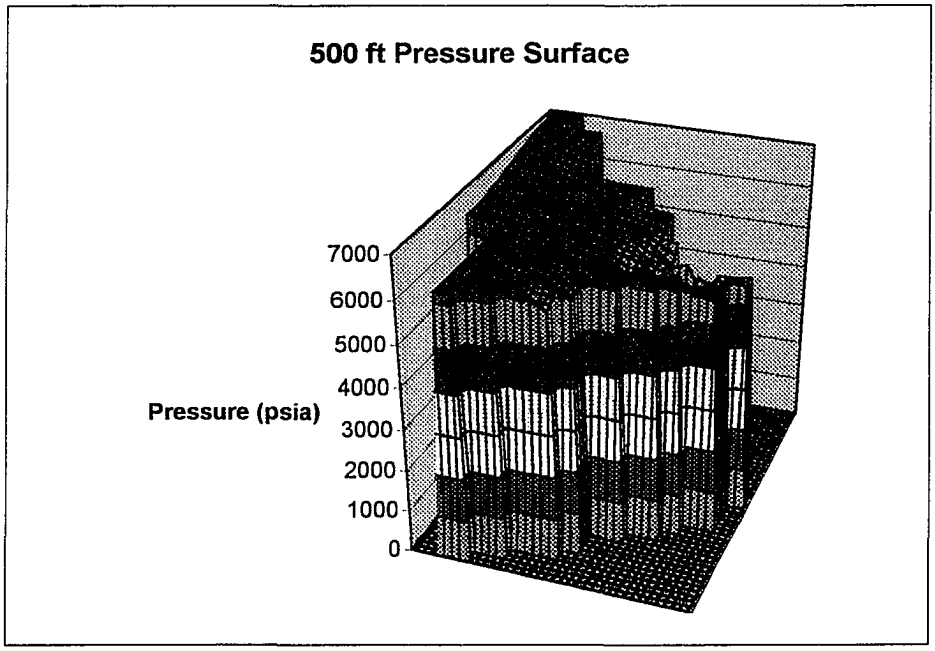


Figure 38 - Pressure Surface at a Depth of 500 ft. - 3rd Year

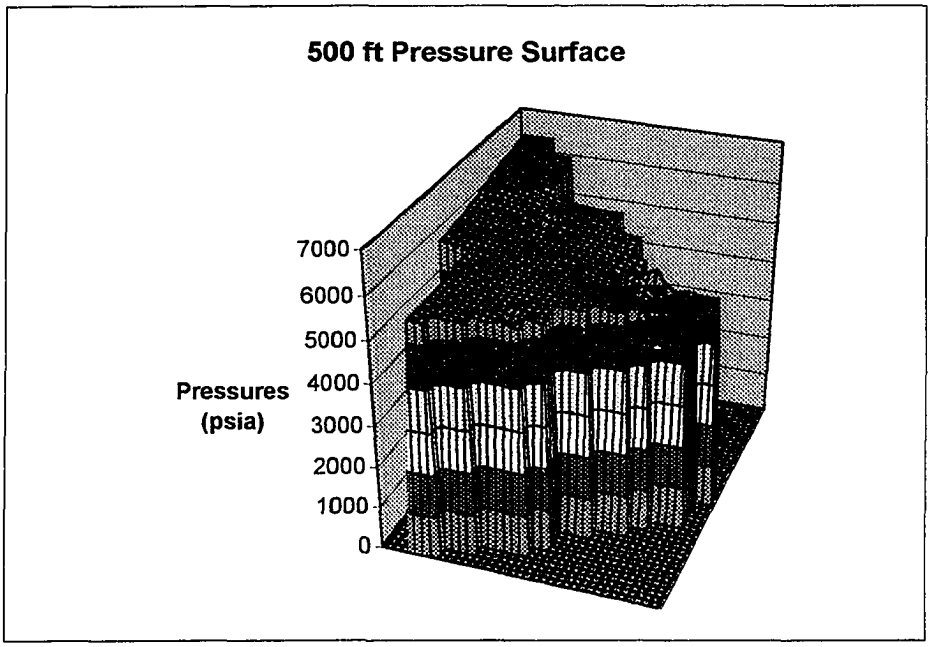


Figure 39 - Pressure Surface at a Depth of 500 ft. - 4th Year

CONCLUSION

Contributions

Storage structures and adaptive refinement algorithms have been developed and presented for a fully three-dimensional FEM simulation. The nodal structure is more efficient for the storage of finite element meshes than other mesh storage methods. Although an increase in complexity of mesh adaptation has been observed, robust algorithms for mesh generation and adaptation have proved useful in implementing a prototype reservoir simulation. The prototype was shown effective as a gas reservoir simulator.

The use of this fully three-dimensional FEM simulation and the presented mesh generation scheme allows a greater level of detail in describing condensate reservoirs. The reservoir's physical characteristics can be different at each node throughout the FEM mesh. This provides excellent capabilities for the modeling of high permeability streaks and other heterogeneities within the reservoir (better than FDM). The FEM prototype simulation supports these heterogeneities without requiring a separate simulation of the area of heterogeneity. Since the entire reservoir is modeled instead of a small block, the interaction of these areas can be observed.

The use of VRML language for the debugging of FEM simulations is novel. It provides a very effective way to visualize the development of the three-dimensional adaptive FEM code and to isolate any errors in the adaptive algorithms. The accessibility of VRML viewers and their "ease of use" make this

debugging technique desirable for other three-dimensional software development projects.

This is the first adaptation of the frontal matrix solution technique to parallel FEM simulation. By overlapping the assembly of the stiffness and load matrices with the elimination of equations in the resulting system of equations, the overall computer time can be decreased. Its use provides a basis for additional work in optimizing parallel FEM algorithms.

Future Work

Finite element methods are often not used for fluid flow simulation despite their advantages because they require a deeper knowledge of mathematics plus more management of the mesh than the "simpler" finite difference method. Future work might be to develop a kernel that will encapsulate many of the tedious details of the finite element method into object oriented classes. This kernel would also provide tools that the finite element practitioner could use to experiment and modify. However this kernel would not completely replace a reasonable knowledge of finite element methods but permit one to focus more on the application than the finite element method.

Future research should also include the study of the use of p-refinement for the mesh in the well region. By providing a higher order element with this region, additional accuracy can probably be gained. The single-phase flow prototype can be extended to solve simulations of two-phase flow for use with a greater number of application areas.

Additional work is desired for the parallel three-dimensional FEM algorithm to improve its scalability and performance on clusters of microprocessors. Although the algorithm presented makes considerable progress into the use of parallel algorithm on a small cluster, the search for more efficient solvers is an area of ongoing research.

REFERENCES

Ames97 — A. Ames et al., *VRML 2.0 Sourcebook*, John Wiley and Sons, New York, 1997.

Arcaro87 — D. Arcaro and Z. Bassiouni, 'The Technical and Economic Feasibility of Enhanced Gas Recovery in the Eugene Island Field by Use of the Coproduction Technique', *Journal of Petroleum Technology*, 585-590 (May 1987).

Aziz83 — K. Aziz and A. Settari, *Petroleum Reservoir Simulation*, Applied Science Publishers, 1983.

Aziz93 — K. Aziz, "Reservoir Simulation Grids: Opportunities and Problems", *Journal of Petroleum Technology*, 658-663 (July 1993).

Burnett87 — D.S. Burnett, *Finite Element Analysis—From Concepts to Applications*, Addison-Wesley Publishing Company, Reading, MA, 1987.

Carey88 — G. Carey, M. Sharma and K. Wang, 'A Class of Data Structures for 2-D and 3-D Adaptive Mesh Refinement', *Internation Journal for Numerical Methods in Engineering* **26**, 2607-2622 (1988).

Connell94 — S. Connell and D. Holmes, 'Three-Dimensional Unstructured Adaptive Multigrid Scheme for the Euler Equations', *AIAA Journal* **33**, 1626-1632 (August 1994).

Deb95 — M. Deb and M. Reddy, 'A New Generation Solution Adaptive Reservoir Simulator', SPE 30720 presented at the *SPE Annual Technical Conference & Exhibition*, Dallas, Texas, Oct. 22-25, 1995.

Ding93 — Y. Ding and P. Lemonnier, 'Development of Dynamic Local Grid Refinement in Reservoir Simulation', SPE 25279 presented at the *12th SPE Symposium on Reservoir Simulation*, New Orleans, Louisiana, Feb. 28 – March 3, 1993.

Dow95 — J.O. Dow, M.J. Sandor, "Submodeling Approach to Adaptive Mesh Refinement", *AIAA Journal*, **33**, No. 8, 1550-1554 (1995).

Eymard93 R. Eymard and F. Sonier, 'Mathematical and Numerical Properties of Control-Volume, Finite-Element Scheme for Reservoir Simulation', SPE 25267 presented at the *12th SPE Symposium on Reservoir Simulation*, New Orleans, Louisiana, Feb. 26 – March 3, 1993.

Greaves98 D. Greaves and A. Borthwick, 'On the Use of Adaptive Hierarchical Meshes For Numerical Simulation of Separated Flows', *International Journal for Numerical Methods in Engineering* **26**, 303-322 (1998).

Gropp96 Gropp, Lusk, Skjellum. "A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard", accepted to *Parallel Computing*, to appear in 1996.

Halford85 K. Halford, 'Screening of Co-Production Prospects', LSU master's thesis, December 1985.

Hassan98 O. Hassan, E. Probert and K. Morgan, 'Unstructured Mesh Procedures for the Simulation of Three-Dimensional Transient Compressible Inviscid Flows with Moving Boundary Components', *International Journal for Numerical Methods in Fluids* **27**, 41-55 (1998).

Huebner75 Huebner, *Finite Element for Engineers*, John Wiley & Sons, New York, 1975.

IBM96 *IBM Parallel Environment for AIX: Hitchhiker's Guide V2.2*, IBM Corporation (November 1996).

Irons70 Bruce, Irons, "A Frontal Solution Program for Finite Element Analysis", *International Journal for Numerical Methods in Engineering*, **2**, 5-32 (1970).

Jin93 H. Jin and R. Tanner, 'Generation of Unstructured Tetrahedral Meshes by Advancing Front Technique', *International Journal for Numerical Methods in Engineering* **36**, 1805-1823 (1993).

Kocberber91 S. Kocberber and R.E. Collins, "Gas Well Test Analysis in Complex Heterogeneous Reservoirs", *Society of Petroleum Engineering Journal*, 1991.

Kocberber95 S. Kocberber, 'An Automatic, Unstructured Grid-Generation System for Geologically Complex Reservoirs', *SPE Computer Applications*, 105-111 (October 1995).

Löhner87 R. Löhner, 'Finite Elements in CFD: What Lies Ahead', *International Journal for Numerical Methods in Engineering* **24**, 1741-1756 (1987).

Microsoft97 *Microsoft Visual C++ 5.0 Online Users Guide*, Microsoft Corporation (1997).

Morton95 D. Morton, J.M. Tyler and J. Dorroh, 'A New 3D Finite Element for Adaptive h-Refinement in 1-Irregular Meshes', *International Journal for Numerical Methods in Engineering*, **38** (1995).

MPIF94 Message Passing Interface Forum, University of Tennessee, Knoxville, Report No. CS-94-230, May 5, 1994.

O'Dwyer97 J. O'Dwyer and P. Evans, 'Triangular Element Refinement in Automatic Adaptive Mesh Generation', *IEEE Transactions on Magnetics* **33**, 1740-1743 (March 1997).

Oden94 J.T. Oden, W. Wu, M. Ainsworth, "An A Posteriori Error Estimate for Finite Element Approximations of the Navier-Stokes Equations", *Computer Methods in Applied Mechanics and Engineering*, **111**, 185-202 (1994).

Okuda97 H. Okuda, T. Yashiki and G. Yagawa, 'Three-Dimensional Finite Element Adaptive Analysis of Incompressible Flow Based on Control of Node Density Distribution and a posteriori Error Estimation', *Advances in Engineering Software* **28**, 173-187 (1997).

Pepper95 D. Pepper and D. Stephenson, 'An Adaptive Finite-Element Model for Calculating Subsurface Transport of Contaminant', *Ground Water* **33**, 486-496 (1995).

Peraire87 J. Peraire, M. Vahdati, K. Morgan and O. Zienkiewicz, 'Adaptive Remeshing for Compressible Flow Computations', *Journal for Computational Physics* **72**, 449-466 (1987).

Pina81 H. Pina, "An Algorithm for Frontwidth Reduction", *International Journal for Numerical Methods in Engineering*, **17**, 1539-1546 (1981).

Ruede93 U. Ruede, *Mathematical and Computational Techniques for Multilevel Adaptive Methods*, SIAM, Philadelphia, 1993.

Schneiders95 R. Schneiders and R. Bünten, 'Automatic Generation of Hexahedral Finite Element Meshes', *Computer Aided Geometric Design* **12**, 693-707 (1995).

Schroeder88 W. Schroeder and M. Shephard, 'A Combined Octree / Delaunay Method for Fully Automatic 3-D Mesh Generation', *International Journal for Numerical Methods in Engineering* **26**, 2503-2515 (1988).

Schroeder90 W. Schroeder and M. Shephard, 'Geometry-Based Fully Automatic Mesh Generation and the Delaunay Triangulation', *International Journal for Numerical Methods in Engineering* **29**, 37-55 (1990).

Sethi95 H. Sethi and D. Pepper, 'Contaminated Groundwater Transport Using an Adaptive 3-D Finite Element Model', *Proceedings of the 6th Annual International Conference on High Level Radioactive Waste Management*, Las Vegas, 1995.

Sloan83 S.W. Sloan, M.F. Randolph, "Automatic Element Reordering for Finite Element Analysis with Frontal Solution Schemes", *International Journal for Numerical Methods in Engineering*, **19**, 1153-1181 (1983).

Sloan86 S.W. Sloan, "An Algorithm for Profile and Wavefont Reduction of Sparse Matrices", *International Journal for Numerical Methods in Engineering*, **23**, 239-251 (1986).

Tani97 K. Tani and T. Yamada, 'H-Version Adaptive Finite Element Method Using Edge Element for 3D Non-Linear Magnetostatic Problems', *IEEE Transactions on Magnetics* **33**, 1756-1759 (March 1997).

Taubin98 G. Taubin, W. Horn, F. Lazarus and J. Rossignac, "Geometry Coding and VRML", *Proceedings of the IEEE* **86**, 1228-1243 (June 1998).

Turner56 M.J. Turner, R.W. Clough, H.C. Martin and L.J. Topp, "Stiffness and Deflection Analysis of Complex Structures", *J. Aeron Sci*, **23,** No. 9, 805-823 (1956).

Tworzydlo92 W.W. Tworzydlo, J.T. Oden, E.A. Thorton, "Adaptive Implicit/Explicit Finite Element Method for Compressible Viscous Flows", *Computer Methods in Applied Mechanics and Engineering,* **95**, 397-440 (1992).

Üler94 F. Üler and O. Mohammed, 'A 3-D Finite Element Mesh Generator for Complex Volumes', *IEEE Transactions on Magnetics* **30**, 3539-3542 (September 1994).

VRML97 *The Virtual Reality Modeling Language*, International Standard ISO/IEC 14772-1:1997, electronic print, 1997.

Wang98 G.H. Wang, J.M. Tyler, J.S. Weltman and J.D. Callahan, 'Node-based Dynamic Adaptive Grid With Quadrilateral and Hexahedral Elements', to appear in *Advances in Engineering Software including Computing Systems in Engineering*, (1998).

Zienkiewicz82 O.C. Zienkiewicz and K. Morgan, *Finite Elements and Approximation*, John Wiley & Sons, New York,1982.

Zienkiewicz87 O. Zienkiewicz and J. Zhu, 'A Simple Error Estimator and Adaptive Procedure for Practical Engineering Analysis', *International Journal for Numerical Methods in Engineering* **24**, 337-357 (1987).

APPENDIX A - HISTORICAL PRODUCTION INFORMATION

The following data are actual production rates for the Eugene Island reservoir. The first 21 months of the production data is presented as a representative set of data. The first five columns are provided for the reservoir. Columns 6- 8 are calculations that convert the oil and water production into equivalent gas production. The calculations allow the simulation of the reservoir using the three dimensional finite element gas simulation. Tables 4-9 contain a partial listing of the data that was used as input for the FESERV application for Eugene Island reservoir.

Table 4 – Production Rates for Well B-1

Month	Days On	Oil (BPD)	Wtr (BPD)	Gas (MCF)	Oil (MCF)	Wtr (MCF)	Tot (MCF)
May-79	31	131	0	0	509	0	509
Jun-79	0	0	0	0	0	0	0
Jul-79	0	0	0	0	0	0	0
Aug-79	4	56	0	782	218	0	1000
Sep-79	24	453	5	9913	1761	37	11711
Oct-79	23	449	5	8054	1745	37	9836
Nov-79	8	90	0	2060	350	0	2410
Dec-79	8	101	0	2807	393	0	3200
Jan-80	9	166	0	4541	645	0	5186
Feb-80	27	886	18	24183	3444	133	27760
Mar-80	0	0	0	0	0	0	0
Apr-80	21	553	0	18941	2149	0	21090
May-80	25	386	0	11174	1500	0	12674
Jun-80	26	757	0	21494	2942	0	24436
Jul-80	30	738	0	19789	2868	0	22657
Aug-80	11	354	0	8803	1376	0	10179
Sep-80	30	1103	0	27145	4287	0	31432
Oct-80	31	1014	0	25761	3941	0	29702
Nov-80	18	507	0	13960	1970	0	15930
Dec-80	19	486	25	13653	1889	185	15727

Table 5 - Production Rates for Well B-2

Month	Days On	Oil (BPD)	Wtr (BPD)	Gas (MCF)	Oil (MCF)	Wtr (MCF)	Tot (MCF)
May-79	0	0	0	0	0	0	0
Jun-79	0	0	0	0	0	0	0
Jul-79	0	0	0	0	0	0	0
Aug-79	15	58	0	7246	225	0	7471
Sep-79	13	319	0	7736	1240	0	8976
Oct-79	26	824	34	19936	3203	251	23390
Nov-79	16	503	38	11728	1955	281	13964
Dec-79	28	1047	32	24689	4069	236	28995
Jan-80	25	886	37	21746	3444	273	25463
Feb-80	27	192	6	4898	746	44	5689
Mar-80	30	876	27	22937	3405	200	26541
Apr-80	28	993	20	22362	3859	148	26369
May-80	27	497	0	12763	1932	0	14695
Jun-80	10	248	0	5907	964	0	6871
Jul-80	2	39	0	936	152	0	1088
Aug-80	19	688	0	15645	2674	0	18319
Sep-80	12	274	0	6163	1065	0	7228
Oct-80	20	692	0	15617	2690	0	18307
Nov-80	27	899	0	24206	3494	0	27700
Dec-80	30	1050	0	25738	4081	0	29819

Table 6 - Production Rates for Well B-3

Month	Days On	Oil (BPD)	Wtr (BPD)	Gas (MCF)	Oil (MCF)	Wtr (MCF)	Tot (MCF)
May-79	0	0	0	0	0	0	0
Jun-79	0	0	0	0	0	0	0
Jul-79	0	0	0	0	0	0	0
Aug-79	13	195	0	5635	758	0	6393
Sep-79	5	131	0	2793	509	0	3302
Oct-79	23	884	0	20697	3436	0	24133
Nov-79	30	1192	12	28833	4633	89	33554
Dec-79	14	635	0	16370	2468	0	18838
Jan-80	21	858	36	22602	3335	266	26203
Feb-80	28	938	82	24521	3646	606	28773
Mar-80	31	1526	0	38967	5931	0	44898
Apr-80	14	685	0	15419	2662	0	18081
May-80	0	0	0	0	0	0	0
Jun-80	0	0	0	0	0	0	0
Jul-80	15	473	0	11045	1838	0	12883
Aug-80	20	827	26	22416	3214	192	25822
Sep-80	29	967	0	23140	3758	0	26898
Oct-80	22	671	0	16181	2608	0	18789
Nov-80	22	714	0	18709	2775	0	21484
Dec-80	31	1043	0	27219	4054	0	31273

Table 7 - Production Rates for Well B-5

Month	Days On	Oil (BPD)	Wtr (BPD)	Gas (MCF)	Oil (MCF)	Wtr (MCF)	Tot (MCF)
May-79	0	0	0	0	0	0	0
Jun-79	0	0	0	0	0	0	0
Jul-79	0	0	0	0	0	0	0
Aug-79	6	52	0	970	202	0	1172
Sep-79	24	228	0	8776	886	0	9662
Oct-79	22	304	0	10143	1182	0	11325
Nov-79	18	486	15	12672	1889	111	14672
Dec-79	6	277	0	8456	1077	0	9533
Jan-80	17	636	6	19442	2472	44	21958
Feb-80	29	1017	65	32484	3953	480	36917
Mar-80	31	1158	101	35369	4501	746	40616
Apr-80	28	833	35	24462	3238	259	27958
May-80	28	469	5	15721	1823	37	17581
Jun-80	29	793	8	23182	3082	59	26323
Jul-80	31	923	0	27809	3587	0	31396
Aug-80	11	417	0	11199	1621	0	12820
Sep-80	27	698	0	19124	2713	0	21837
Oct-80	31	861	0	22742	3346	0	26088
Nov-80	27	695	0	22347	2701	0	25048
Dec-80	31	780	0	27653	3032	0	30685

Table 8 - Production Rates for Well B-7

Month	Days On	Oil (BPD)	Wtr (BPD)	Gas (MCF)	Oil (MCF)	Wtr (MCF)	Tot (MCF)
May-79	0	0	0	0	0	0	0
Jun-79	0	0	0	0	0	0	0
Jul-79	0	0	0	0	0	0	0
Aug-79	0	0	0	0	0	0	0
Sep-79	0	0	0	0	0	0	0
Oct-79	10	46	0	4960	179	0	5139
Nov-79	24	553	0	11752	2149	0	13901
Dec-79	24	917	0	19275	3564	0	22839
Jan-80	11	289	3	6075	1123	22	7220
Feb-80	7	136	4	2702	529	30	3260
Mar-80	17	602	38	11787	2340	281	14408
Apr-80	16	533	34	10952	2072	251	13275
May-80	2	16	0	377	62	0	439
Jun-80	0	0	0	0	0	0	0
Jul-80	13	192	48	6181	746	355	7282
Aug-80	19	190	117	8960	738	865	10563
Sep-80	22	183	183	6736	711	1352	8800
Oct-80	22	150	216	5556	583	1596	7735
Nov-80	20	122	63	5806	474	466	6746
Dec-80	31	285	392	10805	1108	2897	14810

Table 9 - Production Rates for Well B-10

Month	Days On	Oil (BPD)	Wtr (BPD)	Gas (MCF)	Oil (MCF)	Wtr (MCF)	Tot (MCF)
May-79	0	0	0	0	0	0	0
Jun-79	0	0	0	0	0	0	0
Jul-79	0	0	0	0	0	0	0
Aug-79	5	74	0	864	288	0	1152
Sep-79	14	363	0	6669	1411	0	8080
Oct-79	19	475	5	8573	1846	37	10456
Nov-79	16	356	4	7298	1384	30	8711
Dec-79	20	746	0	15358	2899	0	18257
Jan-80	19	731	0	14729	2841	0	17570
Feb-80	27	1296	0	26385	5037	0	31422
Mar-80	0	0	0	0	0	0	0
Apr-80	1	19	0	4011	74	0	4085
May-80	31	643	0	11802	2499	0	14301
Jun-80	3	56	0	1203	218	0	1421
Jul-80	11	383	0	8536	1489	0	10025
Aug-80	10	325	0	6753	1263	0	8016
Sep-80	25	545	0	17082	2118	0	19200
Oct-80	23	158	0	9967	614	0	10581
Nov-80	19	507	0	11274	1970	0	13244
Dec-80	31	935	0	20855	3634	0	24489

APPENDIX B - UNITS USED IN CALCULATIONS

Q (withdrawal rate) - mcf/d

P (pressure) - Psia

K (absolute permeability) - darcy

ϕ (porosity) - ratio (fraction)

μ (viscosity) - cp

t (time) - days

Φ (pseudo-pressure) - $psia^2/cp$

M (molecular weight) - kg/mol

L (length) - ft

T (temperature) - Rankin°

Z (gas compressibility factor) - fraction

VITA

James Callahan was born in Oakdale, Louisiana, on December 21, 1967. He received a bachelor of science degree in computer science from McNeese State University in 1990. That same year he received a second bachelor of science degree in Mathematics and another bachelor of science degree in Mathematical Statistics. He is currently the senior software developer for InControl Technologies, Inc. in Houston, Texas. He will receive the degree of Doctor of Philosophy in 1998.

DOCTORAL EXAMINATION AND DISSERTATION REPORT

Candidate: James David Callahan

Major Field: Computer Science

Title of Dissertation: Three Dimensional Parallel Finite Element Simulation of Natural Gas Flow In a Porous Media

Approved:

John M. Tyler
Major Professor and Chairman

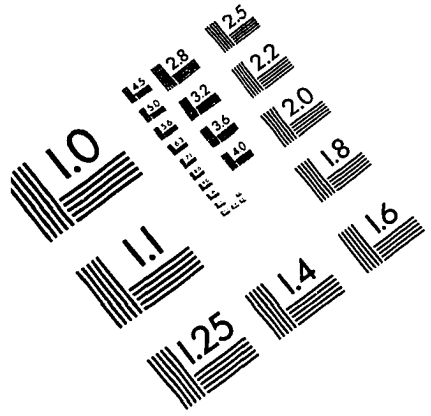
Dean of the Graduate School

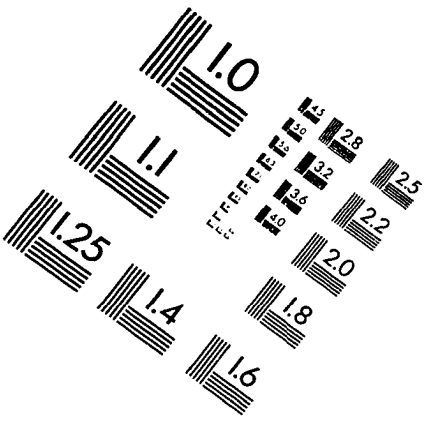
EXAMINING COMMITTEE:

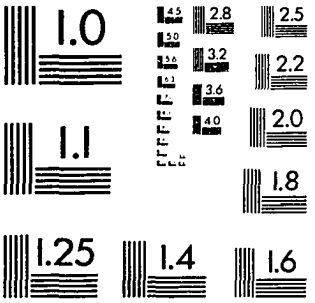
Date of Examination:

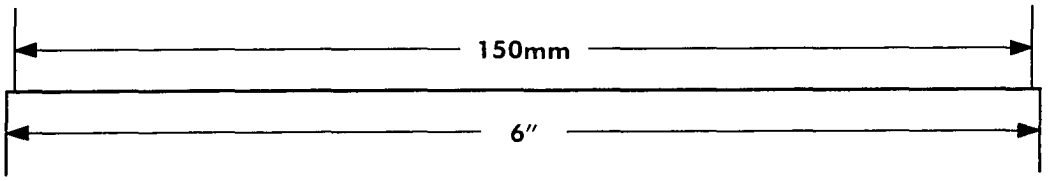
October 12, 1998

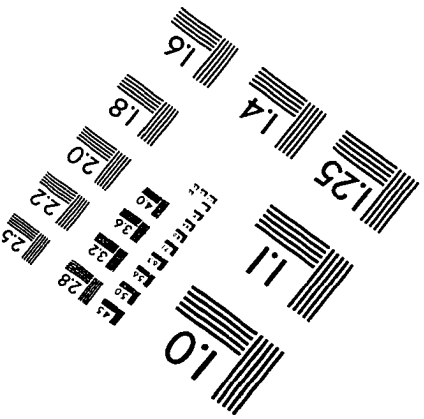
IMAGE EVALUATION TEST TARGET (QA-3)

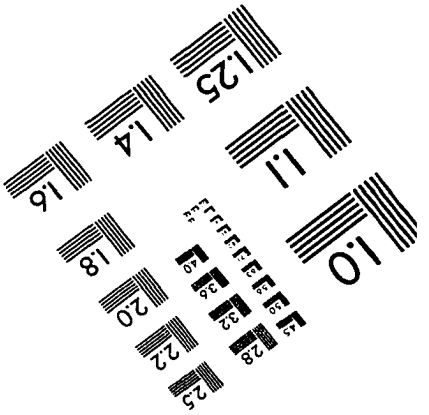












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989