BC/14802-8 Vol II
F/96

IMPROVED RECOVERY FROM GULF OF MEXICO RESERVOIRS

DOE/BC/14802--8-Vol. 2

Volume II (of 4)

Task 5, Modify Publicly Available Simulators
(subtask 5.1, subtask 5.2, subtask 5.3)

RECEIVED

JUN 10 1997

OSTI

Grant Number: DE-FG22-95BC14802

Louisiana State University
Department of Petroleum Engineering
Baton Rouge, LA 70803-6417

Award Date: February 14, 1995
Completion Date: October 13, 1995
Total Government Award: \$1,266,667.00

Principal Investigator: W. Clay Kimbrell
Zaki A. Bassiouni
Adam T. Bourgoyne

Contracting Officer's Representative: Edith C. Allison

Final Report: Covering February 14, 1995 - October 13, 1996

Report Date: January 13, 1997

RECEIVED
USDOE/PETC
97 JAN 17 AM 8:58
A. J. JONES & ASSOCIATES, INC.

MASTER

HH
PROCESSED FROM BEST AVAILABLE COPY

US/DOE Patent Clearance is not required prior to publication of this document.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED *hg*

CLEARED BY
PATENT COUNSEL

Grant Number DE-FG22-95BC14802

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Volume II
Task 5, Modify Publicly Available Simulators
(subtask 5.1, subtask 5.2, subtask 5.3)

TABLE OF CONTENTS

Table of Contents	
List of Figures	
List of Tables	
Abstract	
Executive Summary	

I.	Minimizing Development Overhead with Partial Parallelization	
A.	Introduction.....	15
B.	Serial Adaptive FEM Code.....	16
C.	Distributed Nonadaptive FEM Code.....	17
D.	Integrating the Serial and Distributed Codes.....	18
E.	Performance of the Integrated Code.....	19
F.	Acknowledgements.....	20
G.	References.....	21
II.	Node-Based Dynamic Adaptive Grid with Quadrilateral and Hexahedral Elements	
A.	1.Introduction.....	3
B.	2.Element Based vs. Node-based Data Structures.....	5
C.	3.Description of Data Structure and Operations.....	7
D.	4.2-D Node, Element and Grid.....	8
E.	5.Build and Traverse the 2-D Grid.....	11
F.	6.Grid Initialization.....	15
G.	7.2-D Element Refinement and Unrefinement.....	18
H.	8.3-D Node and Element.....	22
I.	10.3-D Grid Traversal	24
J.	11.3-D Element Refinement and Unrefinement.....	28
K.	12.Comparison of Memory Usage in Node Based vs. Element Based.....	30
	12-1. Element Based	
	12-2. Node Based	
	12-3. Node Based Savings	
L.	13.Interface to the Client Program.....	31
M.	14.Conclusion.....	33
N.	15.References.....	35
III.	A Third-State Correction Method for Non-Convex Scalar Conservation Laws	
A.	1.Introduction.....	3
B.	2.Godunov Method and Third-State Riemann Solvers.....	6
C.	3.Specific Two-Wave Methods and Third-States.....	13
	3-1 Proposed Methods	

D. Numerical Experiments.....	17
4-1 Riemann Problems	
4-2 An Initial Value Problem	
E. Conclusions.....	26
F. References.....	28

IV. A Finite Element Simulation System in Reservoir Engineering

A. 1. Reservoir Simulation Tools.....	1
1-1 BOAST	
1-2 BOAST's Known Limitations	
B. 2. FEM Well Region Model.....	5
2-1 FEM Fundamentals	
2-2 Interface to BOAST	
C. 3. A Real Time Distributed Graphics System.....	12
3-1 System Description	
3-2 Window Interface/Network	
3-3 System Requirement	
3-4 Design	
3-5 Prototype User's Manual	
D. 4. Test Results.....	31
4-1 Test One	
4-2 Test Two	
E. 5. Conclusions.....	42
F. References.....	44
G. Appendix.....	45

List of Figures

I. Minimizing Development Overhead with Partial Parallelization

- Figure 1 - Irregular 2D adaptive mesh
- Figure 2 - Flowchart of computations performed by serial, adaptive FEM code
- Figure 3 - Flowchart of computations performed by two distributed processes
- Figure 4 - Example problem domains partitioned into two element sets coupled by a set of common nodes
- Figure 5 - Integrated approach flowchart
- Figure 6 - Wall-clock times after one time step
- Figure A - Water coning at a production well
- Figure B - Physical model
- Figure C - Finite-element meshes
- Figure D - AVS output

II. Node-Based Dynamic Adaptive Grid with Quadrilateral and Hexahedral Elements

- Figure 1 - An example of irregular grid in 2-D
- Figure 2 - A 2-D node that has four directional pointers
- Figure 3 - 2-D element and the direction of assembly
- Figure 4 - Element traversal sequence and the resulting global node numbers
- Figure 5 - The stack content while traversing the grid in Figure 4
- Figure 6 - A quadrant of a grid for a well
- Figure 7 - Embedded sub-grid into larger problem
- Figure 8 - Refine 2 neighbors before refining element A
- Figure 9 - A 3-D node with six directional pointers
- Figure 10 - Numbering scheme of 3-D element
- Figure 11 - A quadrant of the 3-D grid of a well block
- Figure 12 - The global node number from traversing the grid
- Figure 13 - The stack content while traverse the grid in Figure 12
- Figure 14 - The neighbors of a 3-D element
- Figure 15 - The grid after element A in Figure 14 is refined

III. A Third-State Correction Method for Non-Convex Scalar Conservation Laws

- Figure 1 - Fractional flow for quadratic BL (5). Monotonicity (left), convexity (right).
- Figure 2 - Shock types for convex and non-convex flow Riemann problems
- Figure 3 - Relation between 'true' third state u_m , used in method 2G, and 'false' third state u_c , for method 2F.
- Figure 4 - Fractional flow for cubic BL 23 with $\alpha = 0$.
- Figure 5 - One-wave methods, Problem A, at times $t=0.13, 0.40$.
- Figure 6 - Two-wave methods, Problem A, at $t = 0.13, 0.40$
- Figure 7 - Two-wave methods, Problem B, at $t = 0.40$
- Figure 8 - Shock formation for the initial value problem (25), using method 2F.

IV. A Finite Element Simulation System in Reservoir Engineering

- Figure 1 - Refined mesh in well region
- Figure 2 - BOAST flow directions in a block
- Figure 3 - BOAST flow diagram
- Figure 4 - Flow diagram of system call of the socket
- Figure 5 - Main control window
- Figure 6 - Sample window plot
- Figure 7 - GIF file window
- Figure 8 - Print-out of variable
- Figure 9 - Print-out of plot of time vs. pressure
- Figure 10 - Test problem 1 grid layout
- Figure 11 - Pressure distribution versus distance from the well at day 20 of the simulation
- Figure 12 - Gas saturation's distribution at day 20
- Figure 13 - Oil saturation versus distance from the wellbore at day 20
- Figure 14 - Oil pressure versus distance
- Figure 15 - Gas saturation versus distance from wellbore at the top plane at day 24
- Figure 16 - Gas saturation distribution vs. distance from the well bore at the bottom plane at day 24
- Figure 17 - Test problem 2
- Figure 18 - Change of oil pressure in the wellbore over time
- Figure 19 - Oil pressure distribution vs. distance from well bore at 0.8 day for test problem 2

List of Tables

I. Minimizing Development Overhead with Partial Parallelization

- Table 1 - Wall-clock and CPU times for Cray Y-MP/T3D heterogeneous implementation - one level of refinement
- Table 2 - Wall-clock and CPU times for Cray Y-MP/T3D heterogeneous implementation - two levels of refinement
- Table 3 - Wall-clock and CPU times for Cray Y-MP/T3D heterogeneous implementation - three levels of refinement
- Table 4 - Wall-clock and CPU times for serial Cray Y-MP implementation

II. Node-Based Dynamic Adaptive Grid with Quadrilateral and Hexahedral Elements

- Table 1 - Grid interface functions

III. A Third-State Correction Method for Non-Convex Scalar Conservation Laws

- Table I - Per-step (non-rejected) and total time in *clawl*
- Table II - L2sq errors on two meshes for Problem A, $t = 0.13, 0.40$
- Table III - L2sq errors on two meshes for Problem B, $t = 0.13, 0.40$

This page left intentionally blank and should be substituted with “Minimizing Development Overhead with Partial Parallelization” by Donald J. Morton and John M. Tyler upon receiving permission to copy from publisher, IEEE Parallel & Distributed Technology.

LSU

Department of Computer Science

LOUISIANA STATE UNIVERSITY AND AGRICULTURAL AND MECHANICAL COLLEGE

BATON ROUGE · LOUISIANA · 70803-4020

(504) 388-1495 Fax: 388-1465
tyler@bit.csc.lsu.edu (Internet)

November 25, 1996

Mr. Bill Hagen
IEEE Permission Editor
445 Hoes Lane
Piscataway, New Jersey 008855-1331

Dear Mr. Bill Hagen:

This letter is to request permission to use "Minimizing Development Overhead with Partial Parallelization," by Donald J. Morton and John M. Tyler, IEEE Parallel & Distributed Technology, Fall 1966, Volume 4, Number 3, pages 15-24, in the final report to the Department of Energy for grant DE-FG22-95BC14802. Part of our support for the research work in this paper was provided from funds from this grant. We should include a copy of this paper in the final report if granted permission.

We have a deadline for our final report that requires expedience for our request. If you need any other information from me, my office phone is 504-388-2198. Thank you for your time in considering our request.

Sincerely yours,

John M. Tyler
Associate Professor

/unix

cc: William C. Kimbrell, Petroleum Engineering LSU ✓
Donald J. Morton, Assistant Professor, Cameron University

Node-Based Dynamic Adaptive Grid With Quadrilateral and Hexahedral Elements

G. H. Wang, J. M. Tyler, J. S. Weltman, and J. D. Callahan

Dept. of Computer Science, Louisiana State University

Baton Rouge, LA 70803

September 20, 1996

The support of this work under contract number DE-F622-95BC14802 from DOE is gratefully acknowledged

adaptive according to the 1-irregular requirement for refinement/unrefinement. Implemented in C++, the grid consists of Node objects linked with pointers to neighboring Node objects. The algorithm used to traverse the grid starts at the first node of the grid and constructs each FEM element by following the links of the nodes. This algorithm uses a stack to maintain the correct order of elements during grid traversals. Because the elements and their connectivity are not permanently stored, the computer storage requirements for the grid are reduced significantly in 3-D compared to an element based approach. Although the node based approach may increase the CPU time and complexity to access an element, it reduces the complexity and time required to refine and unrefine an element. This implementation is applicable to many FEM domains.

The paper by F. Guerinoni and J.M.Tyler, "A Third-State Correction Method for Non-Convex Scalar Conservation Laws" discusses an alternative approach to a solution to a Riemann Problem. The exact solution of a Riemann Problem in the classical Godunov method presents computational difficulties, especially when there is a sonic expansion. If the flow is non-convex, there is the additional complication of a third state whose speed governs a shock-fan expansion. The usual solution, the use of a Roe-type Riemann solver, is a popular method to overcome these difficulties, but, in some respects, its computation is cumbersome. Efficiency consideration leads, to averaging formulas which in the case of constant second derivatives perform well, but in the general case lead to schemes resulting in incorrect shock locations or fans. An alternative approach presented in this paper, is the use of what we called third state corrections, which can be constructed from these non-conservative approaches. Numerical experiments based on published results for the Buckley-Leverett equations compare very favorably with analytical solutions, and might be more efficient than current methods especially when there is a sharp change in convexity. All computations were made using CLAWPACK, the recently developed conservation laws software.

The thesis by X. Gu, "A Finite Element Simulation System in Reservoir Engineering", discusses the project that implemented a finite element simulation system. This system can simulate a general three dimensional three phase fluid flow problem in a porous media. Using a refined mesh in the well block, it can provide a more accurate solution of the physical variables in the well region than any other known reservoir simulator. Two test problems, from the National Institute for Petroleum and Energy Research, were used to test on this system. The test results are consistent with the fundamental principles of fluid flow.

Volume II
Task 5, Modify Publicly Available Simulators
(subtask 5.1, subtask 5.2, subtask 5.3)

ABSTRACT

The objective for this portion of the research involved the continuation of the modifications of the public domain simulators BOAST and MASTER. The modifications continued during this project are generic relative to both BOAST and MASTER. BOAST was the primary concern during the research, however, because MASTER is based upon BOAST all research performed during this project is applicable to MASTER as well. These research aspects have been presented in several papers and publications as follows:

- Morton, D.J., and Tyler J.M., "Minimizing Development Overhead with Partial Parallelization", Parallel and Distributed Technology, The Institute of Electrical and Electronics Engineers, Inc., Volume 4, Number 3, Fall, 1996, pps 15-24.
- Wang, G.H., Tyler J.M., Weltman, J.S, and Callahan, J.D., "Node-Based Dynamic Adaptive Grid with Quadrilateral and Hexahedral Elements"
- Guerinoni, F., and Tyler J.M., "A Third-State Correction Method for Non-Convex Scalar Conservation Laws"
- Gu, X., "A Finite Element Simulation System in Reservoir Engineering"

EXECUTIVE SUMMARY

The objective for this portion of the research involved the continuation of the modifications of the public domain simulators BOAST and MASTER. One task involved incorporating adaptive localized grid refinement capabilities using finite element methods. Modification also included additional enhancements to improve the accuracy, cost-effectiveness, and user-friendliness of the publicly available simulators. The modifications also included adapting the modified simulators for parallel processing and implementing this model on the available parallel platform.

The paper by D.J. Morton and J.M. Tyler, "Minimizing Development Overhead with Partial Parallelization" discusses balancing implementation effort with performance needs by focusing the parallelization of serial code on smaller components, rather than attempting to derive a completely parallel implementation. Although it is implemented in this paper in a Y-MP/T3D environment, it is a generic approach that will work in numerous heterogeneous computing environments.

The paper by G.H. Wang, J. M. Tyler, J.S. Weltman and J.D. Callahan, "Node-Based Dynamic Adaptive Grid with Quadrilateral and Hexahedral Elements" involves a node-based data structure that is developed for use in either 2-D or 3-D finite element method (FEM) problems. This grid uses quadrilateral (2-D) or hexahedral (3-D) elements and is dynamically

ABSTRACT

A node-based data structure is developed for use in either 2-D or 3-D finite element method (FEM) problems. This grid uses quadrilateral (2-D) or hexahedral (3-D) elements and is dynamically adaptive according to the 1-irregular requirement for refinement/unrefinement. Implemented in C++, the grid consists of Node objects linked with pointers to neighboring Node objects. The algorithm used to traverse the grid starts at the first node of the grid and constructs each FEM element by following the links of the nodes. This algorithm uses a stack to maintain the correct order of elements during the grid traversals. Because the elements and their connectivity are not permanently stored, the computer storage requirements for the grid are reduced significantly in 3-D compared to an element-based approach. Although the node-based approach may increase the CPU-time and complexity to access an element, it reduces the complexity and time required to refine and unrefine an element. This implementation is applicable to many FEM domains.

1. Introduction

The finite element method (FEM) is widely used for solving partial differential equations (PDE's) over complex domains [1]. The discretization of the domain is generally known as the mesh (or grid) generation, upon which the quality and accuracy of the numerical result of FEM depend. It is state-of-the-art to generate smaller elements in regions where the unknown PDE solutions vary the most and larger elements in smoother regions. This minimizes the demands for computer storage to store the mesh and related data and minimizes the CPU-time required for solutions without compromising the overall accuracy of these solutions. It is generally true that the initial mesh is not optimal for these unknown solutions being determined because we do not know a priori where the smaller elements are needed.

In a transient problem, such as those involving fluid flow, the regions of activity vary with time and a dynamic adaptive mesh is needed to capture this. If this fluid flow problem is an oil well driven by an aquifer, the original oil-water contact will change and be distorted after oil is produced. This type of transient flow problem requires that the grid adjust dynamically according to intermediate PDE answers and the answers from the previous time step. Adaptive grid refinement in FEM adds or rearranges the finer elements into the regions where the PDE solutions vary the most. For transient problems, being able to unrefine is almost as important as refinement. Unrefinement means that we will remove the finer elements in the smoother regions and release the computer memory that these elements occupied.

The implementation of the adaptive refinement method presented has not been trivial, especially in 3-D. The 3-D grids for transient FEM problems are very difficult to maintain. In these problems, one must store a very large and often growing number of nodes. To keep track of the elements that have been refined or unrefined, we must store and update the rapidly changing connectivity of the 3-D elements. The computation time and memory required to maintain this grid may be overwhelmingly large, and this can diminish the benefit derived from the use of adaptive grids. Remeshing and h-refinement are two of many adaptive refinement schemes. Their benefits and disadvantages are briefly discussed in the following.

Remeshing is used to describe the regeneration of the mesh over the whole domain with the knowledge of a new estimate for the unknown solution. To achieve remeshing, the capability of automatic mesh generation is required and examples of this usually use octree/Delaunay methods [2, 3]. The efficiency of these methods is compromised by its generality. Remeshing has several disadvantages, as observed by Connell et al. [4]:

"...adaptive remeshing has three disadvantages. The first is that the mesh generation algorithm needs to be robust and fast. If the scheme is not fast, the mesh generation time can quickly exceed the solution time, . . . The second disadvantage relates to the difficulty in computing transfer operators from unstructured mesh . . . Finally, unless the meshes are generated through solution adaption a large percentage of nodes on the finer meshes may serve no useful purpose."

On the other hand, h-refinement is a method that tries to add finer elements in the local regions. H-refinement separates the mesh generation from the mesh refinement processes and is usually fast because only part of the initial mesh is modified. During the adaptive process in h-refinement, more efficient storage schemes are possible for the mesh than were possible when the mesh was generated. For example, the data structure used in [4] to handle hexahedra in 3-D refinement resulted in "a reduction in CPU-time by up to a factor of 10 over the same case without multigrid."

2. Element Based vs. Node-based Data Structures

Grids (meshes) can be stored using different types of data structures. A node-based data structure stores the grid as a list of nodes in contrast with an element-based approach which stores the grid as a list of FEM elements. The element-based approach is proposed by Yerry[2], Schroeder[3], Devloo[5], and Hemker[6]. The node-based approach, as well as element-based approach, is discussed in Ruede[7], although implementation details are not given. In an element-based approach, each Element data structure has, at a minimum, a list of pointers to each of its nodes and a way to find its neighbors. In contrast, the node-based approach does not explicitly store elements. Instead, its Node data structures contain pointers to each of their neighboring nodes. An element can easily be constructed from a single node by starting with its root node (node at the southwest corner of the element) and finding its neighboring nodes (the vertices of the element). This construction procedure is referred to as "assembling" an element. Once assembled, the edges and nodes of

neighboring elements can be found in a similar manner. The memory required is significantly reduced because typically only one element is needed, and it does not need to be stored after use.

The element-based approach is more costly in required memory when compared to the node-based approach for the implementation of the 1-irregular h-refinement as outlined by [5]. The 1-irregular requirement means that the difference of refinement levels between neighboring elements cannot be greater than 1. To enforce this requirement, it is necessary to locate all the neighbors of an element with a pointer to each neighbor. Since a 3-D element has a maximum of 48 neighbors, the element approach requires 48 pointers per element. Although the quadtree/octree approach in [2,3] could eliminate the need for pointers to neighbors, it would become unwieldy and time consuming to search a large tree for neighbors without an extra neighbor data structure.

The node-based approach allows efficient access to neighbors with fewer pointers, and thus less memory, but the trade off for this savings is that elements must always be assembled before being accessed. We believe, as in [8], the sacrifice in CPU time, if this sacrifice is small, is better than using more memory. The minimal extra CPU time and software complexity needed to assemble an element in our approach is also offset by a reduction in time and complexity over the element-based approach when we refine/unrefine an element in 3-D. Refining an element in the element-based approach requires a complex and error-prone insertion of 8 new elements into the linked list of elements. Each new element requires the correct

placement in its linked list of 48 neighbor pointers as well as the correct placement of the neighbors pointing to it.

Because the use of less memory and reduced complexity were desired, we developed a node-based approach. Our decision was to only use quadrilateral/hexahedral elements. The node-based approach is not well suited to triangular elements [7].

3. Description of Data Structure and Operations

Our data structure stores the 1-irregular h-refinement grid of quadrilateral or hexahedral elements as a doubly linked list of Node-objects. Rather than maintaining different data structures for edges, faces, and elements, we chose to store only the Node from which all other grid relations are obtained. Our Node data structure records its 4 neighboring nodes in 2-D or 6 neighboring nodes in 3-D. It also stores data associated with the node such as X, Y, Z coordinates and additional numerical data values. The client specifies how many additional numerical data values are stored per node. When an element is refined, new nodes are inserted into the doubly linked list. When a Node is not used by any of the neighboring elements after unrefinement, it is removed from the doubly linked list. This grid software was implemented in C++ on a Unix platform. The software implementation of the grid in 2-D and 3-D will be discussed in separate sections. Although the 3-D grid follows a similar logic as 2-D, the coding in 3-D was much more complicated.

4. 2-D Node, Element and Grid

Figure 1. An example of 1-irregular grid in 2-D.

A two dimensional 1-irregular quadrilateral grid is shown on Figure 1. Each intersection of the line segments is a node. It is clear that there is a maximum of four line segments extending out from each node. An element is simply a group of nodes that are connected by these segments. This observation leads to the basic data structure that was used in object Node, that is, each of these connections of a node is represented by a Node pointer. Therefore, an object Node in 2-D has 4 pointers to other nodes. The only exceptions are the edge nodes or the nodes created during

subsequent refinement processes and these can have null pointers. This data structure enables the storage of a 2-D grid as a linked-list of Node objects. These Node objects also store any data needed for solving FEM problems, such as the coordinates of the nodes and physical properties of the FEM domain at the node location. The solutions of the PDEs are also stored with the Node objects.

A 2-D Node object is presented on

Figure 2. Left, Right, Up, and Down are used as the mnemonics for these 4 pointers because they are similar to the directions of the pointers. In the computer program, an array of 4 Node pointers is used to store these. Element 0 of this array stores the pointer to the right, and element 1 stores the up pointer, and so on. This enumerating

Figure 2. A 2-D node that has four directional pointers.

scheme allowed very efficient code to be written. One example of this is presented later when a finite element is assembled.

A 2-D element consists of 4-to-8 interconnected 2-D Nodes; see Figure 3. The local numbering scheme used for the nodes in an element is annotated on this figure. This annotation starts at the southwest node, and the 4 corners are numbered counterclockwise as node 0, 1, 2, 3. These 4 nodes exist for all 2-D quadrilateral elements. The 0 node was used as the root of the element because one and only one element can be assembled starting from this node. The node shown on Figure 3 be-

tween 0 and 1 was numbered as local node 4. Node 4 only exists if the neighboring element below had been refined. Similarly, nodes 5-7 exist only if there had been refinement of the neighboring elements in the corresponding direction. The object Element allocates an array of 8 Node pointers to store all the pointers to the nodes that belong to this element. When a middle node, 4-7, does not exist, the Node pointer is set to 0. With this local node numbering scheme, an element can be assembled efficiently given the pointer of node 0:

```
for (int I=0; I< 4; I++){
    if (nodes[i]->dir[i]->dir[(I+1) % 4] != 0){
        nodes[(I+1) % 4] = nodes[i]->dir[i];
        nodes[i+4] = 0;           // no middle node
    }
    else{
        nodes[i+4] = nodes[i]->dir[i];
        nodes[(I+1) % 4] = nodes[i+4]->dir[i];
    }
}
```

This code fragment is compact, and

this is a result of the numbering scheme for the Node and Element objects. It assembles each element without requiring any prior knowledge of the refinement status of the neighboring elements. Note, the actual code must check for null pointers in case the node is not the root of an element. An efficient element assembling method is es-

Figure 3. 2-D element and the direction of assembly.

essential to this new data structure because it is called frequently throughout a FEM calculation.

5. Build and Traverse The 2-D Grid

The program that uses the grid (the client) requires that all the elements be visited to assemble the stiffness matrix in a FEM calculation. Because the elements are not stored, there must be a grid traversing algorithm that ensures all the elements will be visited once and only once. To traverse the grid is equivalent to locating all the root nodes of the elements. This task is achieved by using a stack for the Node pointers. Once an element is assembled, the configurations of its nodes and the content of the stack uniquely determine the next element to be visited in this traversal. The order of traversing, also a priori, determines the global node numbering scheme.

Figure 4. Element traversal sequence and the resulting global node numbers.

The head of the grid was chosen to be the southwest node in the grid. While traversing the grid, the relative position and the configuration of the elements are used to determine the next element to be visited. Our program currently visits elements row-first. To add a column-first traversal would be a trivial task. An example of the row-first traversal is shown in Figure 4, where the order of elements traversed is annotated with underscored numbers. The global number of all the nodes is also shown in this figure.

Our traversal algorithm is also illustrated by the content of the stack shown on Figure 5. Starting from the head of the grid (global node 0 on Figure 4), as an element is assembled, its node 3 is examined. Using Figure 5, if the element is the leftmost element of the grid, e.g., element 1, 5, 12, or the southwest element of a group of refined elements, e.g., element 7, 14, 15, the pointer to its node 3 should be pushed onto the stack unless this element is located on the last row of the grid or the group. Once this information is stored, the terminal proceeds to the element at the right whose root node is node 1 of the current element. However, if the current element is the rightmost element of the grid, e.g., element 4, 9, 18, or the northeast element in a refined group, e.g., element 11, 20, there will not be an element to its right, and then the stack is popped. This pop provides the root pointer of the next element. This scheme assures visiting all the elements once no matter how complicated the refinement pattern becomes.

Because this traversal determines the global node numbering scheme, a counter is set to 0 before the traversal begins. Then any unnumbered node, indicated by a negative global number, is

Figure 5. The stack content while traversing the grid in Figure 4.

numbered according to their counter's current value, and the contents of the counter are increased by 1. The results of this numbering method are also shown in Figure 4. This numbering scheme yields a smaller bandwidth for the stiffness matrix when it's compared to a numbering of all of the refined nodes with the new nodes being the largest node number in the grid plus one. This reduced bandwidth represents an additional savings in storage for the FEM system.

Currently, our 2-D grid cannot contain a discontinuous boundary and the head of this grid must point to the first element of the first row. Because the coordinates of the node are stored in the Node objects, many different domains can use this same scheme without modification. One example is a quadrant of a grid that could be used with a well as shown in Figure 6. The connections of the nodes on the radius are represented with Node pointers 0 and 2, while the connections of the nodes with the same radius are implemented with pointers 1 and 3.

Figure 6. A quadrant of a grid for a well.

6. Grid Initialization

As already stated, many domains can be discretized with this grid structure. One example is the quadrant of a well block, in either 2-D (Figure 6) or 3-D (Figure 12). Reservoir domains often contain many wells and may require multiple well blocks. Domains which encompass a larger geographical region may also contain regions in which no nodes are needed. Different grid structures may require different initializations and traversal routines which are specifically designed for that domain.

For more complex domains (eg., reservoirs containing multiple wells), we have developed some additional guidelines which maintain a consistent and extensible grid. It is important to note that these guidelines are not always required for domains but are required for domains which have several different physical regions and are made up of a conglomeration of smaller sub-grids.

- *Consistency of the Pointers*

Figure 7. Embedded sub-grid into larger problem.

The consistency of Node pointers, see Figure 3, means that all subsequent links are made in the desired direction. All grids must be fully connected, see the 2-D radial domain in Figure 7. If we embed the grid in a larger grid, the problem shown in Figure 7 could occur. There are five directional Node pointers from node N, but there are only 4 Node pointers (0 right, 1 up, 2 left, 3 down) for eachNode object, and this is inconsistent. If

any of these pointers are removed then the desired connectivity is lost.

Mislinked or unlinked Node pointers destroy the structure of the grid. Each of the Node pointers must either be connected to a neighboring node or be a null pointer.

For simple domains, grid construction is not difficult. For more complex domains (those containing regions without nodes or with irregularly shaped domains), the grid construction is more difficult and requires additional work. A bound and scan method

was developed for these more complex grids. This bound and scan method finds the extent of all regions within the domain, creating nodes, and scanning the existing nodes to link the Node pointers to the correct neighboring nodes. A doubly linked list of Node pointers is created by this method. In 3-D, the nodes are ordered by their physical location, starting from the upper, left, front corner of the domain and ending with the lower, right, back corner of the domain. During the grid initialization, a temporary index is created for this list. With the temporary list, a binary search for a particular node requires only $O(\log n)$. This index is unallocated after the grid is constructed. The process involves these basic steps.

```
/* Proceed from upper, left, front corner of domain  
to lower, right, back corner of domain */
```

```
While ( < within range of domain > ) {  
    < determine location of prospective node >  
    if ( < node should be created at this location? > ) {  
        < Create a new node >  
        < search for upper neighbor and link to node >  
        < search for left neighbor and link to node >  
        < search for back neighbor and link to node >  
    }  
}
```

This initial grid construction requires the determination of whether a node needs to be created at each potential node location. This initial node location depends upon its location relative to the domain boundaries. The initial node locations are displayed on a computer screen to provide interactive intervention if desired later. When a node location (x, y, z) is final, the node structure is inserted into the doubly linked list and the Node pointers are linked to its neighboring nodes. The temporary list of nodes is scanned in descending order until the neighboring node is found, and it stores this node's location. The neighboring nodes to it are located at coordinates $(x-\Delta x, y, z)$, $(x, y-\Delta y, z)$ and $(x, y, z-\Delta z)$. The list is searched for these three neighbors and the Node pointers of the new node and neighboring nodes are reconciled. For N nodes, this is of order $N \log N$. This generates a grid which has nodes that are evenly spaced. Many implementations benefit from a post refinement generation phase in which additional nodes can be inserted interactively prior to the initial use of the grid.

The example problem, the 2D well grid (Figure 6) has 5×5 (25) initial nodes. The initial grid together with other information (e.g., X-Y coordinates) is stored in an ASCII file after being generated. A square and a well quadrant domain were used in this paper. If a more complex domain is required, then the traversing algorithm may have to be modified.

7. 2-D Element Refinement and Unrefinement

When the threshold criteria are met, an element needs to be refined or unrefined. The criteria will depend on the client using the grid and will be calculated for each element while traversing the grid. A method to traverse the grid and mark the

elements that need to be refined or unrefined was developed. If there is at least one element to be refined in the entire grid, the grid will be refined. If no element needs to be refined, then the grid will unrefine those elements that meet the unrefine requirements. The algorithm either refines or unrefines the grid but not both in a time step. The client may impose a maximum level of refinement, or alternately, let the grid refine as many levels as the resources allow.

When a 2-D element is refined, it is divided into 4 smaller elements. This division is accomplished by inserting 4 middle nodes on each link and then inserting a center node for these 4 middle nodes. The properties of this middle node, such as coordinates, are taken as the average of its corner nodes. In some cases, the middle node may already exist from a previous refinement of a neighbor, then the program only needs to update its pointers.

To preserve the 1-irregular structure of the elements, the neighboring element may have to be refined before the element under consideration can be refined. Such elements are easily recognized by checking the four corner nodes of the element. If any corner node has a null directional pointer not caused by being a boundary node of the grid, the neighbor in that direction must be refined first. The grids shown in Figure 8 illustrate how this refinement is done for element A. Node 1 and node 3 of element A have only 3 connections (non-null direction points). The null directional pointer of node 0 is pointer 3 of the node, which is pointing to the

Figure 8. Refine 2 neighbors before refining element A.

element below as shown on Figure 2. Therefore, the neighbor element below must be refined before element A can be refined. Depending on whether the southwest quadrant is refined or not, as illustrated by the dotted line segments, the root node of the neighbor element is 1 or 2 steps away from the node 0 of element A. This same procedure is used recursively to pre-refine this neighboring element. Similarly, the null pointer of node 3 in element A requires the neighboring element to the left to be refined first. Once the 1-irregular requirement is satisfied, the element is refined. Because the new nodes created are inserted into the grid permanently and may belong to several elements, an element must be assembled immediately before the refining. The result is shown on the right of Figure 8.

When an element is refined, 4 children elements are formed. The first element has the same root node as its parent. In fact, if we assemble this element after it is refined, its four corner nodes are the root nodes for the four children elements. This parent-child relationship must be kept in order to properly reverse the refinement process, i.e., unrefine the grid. The initial nodes must never be deleted from the linked-list.

An element is uniquely determined by its root node (node 0) because the element can be assembled from this node. The right-most and top nodes of the grid cannot be root nodes because no element can be assembled using them. Some middle nodes, such as node 4 and node 7 on Figure 3, cannot be root nodes. To distinguish such nodes and to also record the parent-child relationship, an extra pointer called Parent is stored in each Node object. It has 3 values: 0 if the node cannot be a root node for an element; it will point to the Node itself if it is not created by element refinement; otherwise, it will point to the root node of the element that created these nodes during refinement. Note that there is no need to have 4 pointers to the children because the 4 corner nodes of the parent element are the root nodes of the children. This saves more storage space than the comparable quadtree/octree.

Unrefinement is almost the reverse of refinement except the criteria are more conservative. When an element is marked for unrefinement, the number of nodes in the element is checked. If the element has any middle nodes, it cannot be unrefined because the 1-irregular requirement would be violated. No attempt to recursively unrefine the neighbors was made because it could force unrefinement of the elements

that should not be unrefined. A forced unrefinement results in oscillation. In addition, its 3 siblings have to be checked. An element can be refined only when all its siblings are eligible for unrefinement.

Unrefining an element does not always remove all the nodes of that element because some nodes are also shared by neighboring elements. When this is the case, only the node links are updated. If the node cannot be a root node any more, the parent pointer is set to 0. After the nodes are removed, the traversal stack is checked to remove any node pointers that no longer exist or that cannot be parent nodes.

8. 3-D Node and Element

The same procedure (a linked list) used for 2-D was also used to manage the 3-D grid. The node pointer stack was used for the 3-D grid without modification. The Node object needed an adaptation for the 3-D structure, instead of four links, the 3-D node has six directional pointers to link nodes.

With two pointers in each dimension, the traversal of nodes is more flexible. The

Figure 9. A 3-D node with six directional pointers.

numbering scheme of the 3-D Node object is shown in Figure 9. Comparing this with 2-D, the modification of the Node object is minor; however, the software that handles the Node objects becomes much more complex.

While a 2-D element may have a maximum of 8 nodes, a 3-D element may have from a minimum of 8 nodes to a maximum of 26 nodes depending upon the refinement status of its neighbors. The numbering scheme of all possible nodes is shown in Figure 10. The

sequence to assemble a 3-D

element is as follows: using the **Figure 10. Numbering scheme of 3-D element.**

existing code for the 2-D assembling function, the nodes on the smallest loop are allocated. If node 8 does exist, the loop would consist of only nodes 0, 4, 8 and 7. Otherwise, the loop will correctly encompass nodes 0, 1, 2, 3 and possibly nodes 4, 5, 6 and 7. Thus, when node 8 does exist, the output of the 2-D assemble function had to be adjusted to include all the nodes on the top face. With the knowledge of the structure of the nodes on the top face, it is possible to allocate node 9. Subsequently, the nodes on the bottom face are allocated and followed by the nodes on the middle plane. As in 2-D elements, an array of Node pointers are used to store the addresses of the nodes but the size of the array is now increased to 26. When a node does not exist, the pointer is set to 0; node 0 of the element is the root node of the element. The parent-child relationship is stored by the same pointer named 'Parent'.

10. 3-D Grid Traversal

The initial 3-D grid has an analogous structure to the 2-D grid. The initial grid of the 1-irregular 3-D grid in Figure 11 has 5 nodes on both the X and Y axes and 3 nodes on the Z axis. Using the pointer links to represent the axes, many different domains can be mapped with this grid structure. One useful example is a quadrant of a 3-D grid of a well block shown on Figure 11. Its initial grid, $5 \times 5 \times 4$ (100) nodes, are linked with the respective directional pointers.

Figure 11. A quadrant of the 3-D grid of a well block.

Figure 12. The global node number from traversing the grid.

pushed on the stack and no prior knowledge of the distribution of its neighbors is needed.

11. 3-D Element Refinement and Unrefinement

All the elements in a 3-D grid are also checked for refinement or unrefinement according to the user's criteria. If any of the elements need to be refined, the grid is refined. Similarly, all the neighbors of an element have to be checked to ensure

Figure 14. The neighbors of a 3-D element.

the 1-irregular requirement. A 3-D element has a maximum 24 face neighbors and a maximum 24 edge neighbors. Face neighbors share at least 4 common nodes on a face, denoted on Figure 15 by letter 'F'. Edge neighbors share at least two edge nodes, and they are denoted by 'E'. The number of neighbors depends on the refinement status of the neighbors. When there has been no refinement, each element has 6 face and 12 edge neighbors. A physical model was constructed and used during this development to help visualize the complicated 3-D neighboring elements.

The 3-D algorithm works as follows: first, the face neighbors are checked for pre-refinement. The 8 corner nodes of an element are examined, and if any of the corner nodes have a null directional pointer(s) and is not on the boundary, then the face neighbor in that direction

Figure 15. The grid after element A in Figure 14 is refined.

needs to be refined first. After the face neighbors have been pre-refined, the edge neighbors will be allocated and examined. To allocate edge neighbors on the top face is equivalent to finding the horizontal face neighbors of the element above the current element. The same logic applies to the edge neighbors of the lower face. We also use this method to locate part of the edge neighbors. Only the vertical edge neighbors required a new method.

When an element requires refinement, the element is divided into 8 smaller elements. This parent-child relationship needs to be recorded for use in unrefinement. This is recorded as seven child elements that point to the parent element, which is denoted the 0 node. The refining element A in Figure 14 is shown in Figure 15.

If unrefinement of a 3-D elements is compared to the similar 2-D problem, the main difference is more directional pointers need to be updated when a node is removed.

12. Comparison of Memory Usage in Node Based vs Element Based

The Fortran data structure Devloo[5] is a good basis for comparing the memory requirements of element based vs node based approaches. Although Fortran does not support structures and pointers directly, this scheme [5] can easily be analyzed in terms of any programming language.

12.1. Element Based

Each element must store pointers for each of its nodes and possible neighbors. Furthermore, [5] suggests a Group Array to record the nodes of refined elements.

In 2-D, there are 4 nodes at the vertices and 4 more nodes possible at interior points where a neighbor can be refined. There are 8 possible neighbors, and the Group Array requires 4 pointers, one to each of the sub-elements that are created during refinement. In total, there are 16 pointers per element in total.

In 3-D, there are 6 faces, 12 edges, and 8 vertices per element. There is a node at each of the 8 vertices, a possible midpoint node at each of the 12 edges, and a possible center node at each of the 6 faces for a total of 26 nodes. There are also face neighbors and edge neighbors. Each face can have 4 neighbors and each edge can have 2 neighbors, for a total of $(4 * 6 \text{ faces}) + (2 * 12 \text{ edges}) = 48$ neighbors. The group array needs an additional 8 pointers per element since each element is broken

into 8 sub-elements during refinement. There are a total of 82 pointers per 3-D element.

12.2 Node Based

In this node based approach, each node stores pointers to each of its possible neighboring nodes. There are 4 pointers per node in 2-D and 6 pointers per node in 3-D.

12.3 Node Based Savings

The node based and element based techniques store about the same number of basic data structures when a grid is large. For example, consider a cube where the number of nodes is $X*Y*Z$ and the number of elements is $(X-1)(Y-1)(Z-1)$. Consequently, the number of nodes and number of elements are not significantly different when the product of X , Y , and Z is large. Let N be the number of data structures in either approach. The node based approach requires $4*N$ pointers in 2-D and $6*N$ in 3-D. The element based approach requires $20*N$ pointers in 2-D and $82*N$ in 3-D. A pointer usually takes 4 bytes of memory, so the node based approach saves $16*N*4$ bytes in 2-D and $76*N*4$ in 3-D. The savings in 3-D of $304*N$ could be significant. Assuming half a million nodes in a large system [8], the node based approach would save more than 150 megabytes of memory.

13. Interface to the Client Program

The routines that are used for manipulating the nodes of the grid, assembling elements, and refining/unrefining the elements reside in separate files in a grid software library, which is compiled independently of the client (a program that uses the grid library). The grid library is designed to interface with a client written in any general purpose programming language. The current client program is written in Fortran and implements the shape functions for quadrilateral or hexahedral elements [9,10]. Since the grid library was implemented in C++ and uses object oriented principles [11], it could conceivably be used without any modification by a variety of clients.

Table 1. Grid Interface Functions

A client uses	<i>InitGrid:</i>	Initialize the grid with beginning values.
	<i>DataLoc:</i>	Specify the address where element data should be placed.
	<i>GetElement:</i>	Return the data for the next element in the grid.
	<i>UpdateVar:</i>	Store new node values in the grid.
	<i>Adapt:</i>	Perform one pass of refinement and unrefinement. (Client must provide <i>ElmEval</i> function.)
	<i>SaveResults:</i>	Create output file suitable for graphical display of grid.

several grid interface functions to establish the initial grid, access elements, change node values, refine the grid, and print the grid. The storage details of the nodes are isolated from the client, which only has access through interface functions (See Table 1). The client initializes the grid with an input text file that specifies the beginning node layout and data values. Once the grid has been initialized with a call to *InitGrid*, the client can traverse the grid to create the stiffness matrix by repeated calls to *GetElement*. *GetElement* returns the data from the first element of the grid. Each successive call to *GetElement* retrieves the data to be used with the next element in the grid. After creating and solving a step of an FEM problem, the client stores new node values into the entire grid with a call to *UpdateVar*. Then the client may refine/unrefine the grid by calling *Adapt*. *Adapt* calls a function *ElmEval*, which must be supplied by the client. *ElmEval* is used to evaluate the FEM answers to determine if the element should be refined/unrefined. Finally, at any time, a client can create an output file by calling *SaveResults*.

14. Conclusion

We have implemented a node based approach for dynamic adaptive 2-D and 3-D grids. This data structure efficiently stores the necessary data in Node objects, where the Node objects are linked to each of their neighboring nodes. While traversing the grid, finite elements can be dynamically assembled, refined and unrefined. The connectivity between elements is not stored, which saves computer memory. Bandwidth is also reduced because nodes are dynamically numbered. In the future,

more complex topologies could be combined with the node based approach. For example, an octree structure could be implemented where each octant points to a root node of an element instead of an entire Element data structure. This would provide the same memory savings and neighbor accessibility as the node based approach and allow the benefits of octree topology. In fact, the node based approach could be used as the underlying framework for many geometrical configurations, including configurations suitable for multi-processing.

REFERENCES

- [1] David S. Burnett, "Finite Element Analysis— From Concepts to Applications", Addison-Wesley Publishing Company, 1987.
- [2] Mark A. Yerry and Mark S. Shephard, "Automatic Three-Dimensional Mesh Generation by the Modified-Octree Technique", *International Journal for Numerical Methods in Engineering*, Vol, 20, 1984, p 1965-1990.
- [3] W.J. Schroeder, M.S. Shephard, "A Combined Octree/Delaunay Method for Fully Automatic 3-D Mesh Generation", *International Journal for Numerical Methods in Engineering*, Vol, 29, 1990, p 37-55.
- [4] Stuart D. Connel and D. Graham Holmes, "Three Dimensional Unstructured Adaptive Multigrid Scheme for the Euler Equations", *AIAA Journal*, Vol 32, No. 8, August 1994, p 1626-1632.
- [5] Devloo, P, Oden, J.T. and Strouboulis T." Implementation of an Adaptive Refinement Technique for the SUPG Algorithm", *Computer Methods in Applied Mechanics and Engineering*, vol 61, 1987, p 339- 358.
- [6] P. Hemker, H. Van Der Maarel, and C. Everaars, "BASIS: A data structure for adaptive multigrid computations, Report NM-R9014, Department of Numerical Mathematics, Centrum for Wiskunde en Informatica, August 1990.
- [7] U. Ruede, *Mathematical and Computational Techniques for Multilevel Adaptive Methods* (SIAM, Philadelphia, 1993).
- [8] R. Lohner, "Finite Elements in CFD: What Lies Ahead", *International Journal for Numerical Methods in Engineering*, Vol, 24, 1987, p 1741-1756.
- [9] Morton, D.J, Tyler, J.M. and Dorroh, J.R.: "A New 3D Finite Element For Adaptive h-Refinement in 1-regular Meshes", *International Journal for Numerical Methods in Engineering*, accepted for publication 1995.
- [10] D.J. Morton, J.M. Tyler, A.T. Bourgoyne and P.A. Schenewerk, "An Adaptive Finite Element Methodology For 2D Simulation of Two-Phase Flow Through Porous Media", *ACM Proceedings of 1994 Symposium of Applied Computing*, pp. 357-362.

[11] Stanley B. Lippman, C++ Primer, 2nd Edition (Addison Wesley, Reading, MA, 1993)

A Third-state Correction Method for for Non-Convex Scalar Conservation Laws

Fabio Guerinoni and John M. Tyler

April 2, 1996

Abstract

The exact solution of a Riemann Problem in the classical Godunov method presents computational difficulties, especially when there is a sonic expansion. If the flow is non-convex, there is the additional complication of a third state whose speed governs a shock-fan expansion (in the simplest case). The usual solution, the use of a Roe-type Riemann solver, is a popular method to overcome these difficulties, but, in some respects, its computation is cumbersome. Efficiency consideration leads, to averaging formulas which in the case of constant second derivatives perform well, but in the general case lead to schemes resulting in incorrect shock locations or fans. An alternative approach presented here, is the use of what we called third-state corrections, which can be constructed from these non-conservative approaches. Numerical experiments based on published results for the Buckley-Leverett equations compare very favorably with analytical solutions, and might be more efficient than current methods especially when there is a sharp change in convexity. All computations were made using CLAWPACK, the recently developed conservation laws software package.

1 Introduction

The problem that concerns us had its origin in attempts to numerically solve the one-dimensional Buckley-Leverett (BL) equations in the 1970's. The BL equations have been used by researchers to model non-compressible flow of two substances which do not mix (water and oil) and in general there is little agreement as to what should be called the Buckley-Leverett equation(s). Depending on the author, they may represent an elliptic and first order hyperbolic type system, a convection-diffusion equation, a hyperbolic equation, or a conservation law.

In the usual derivation from fundamental conservation laws, stated in the work by Peaceman[13], one obtains the pressure (p) and saturation (S) equations,

$$\nabla \cdot K(\lambda \nabla p + (\lambda_w \rho_w + \lambda_o \rho_o) g \nabla Z) = 0 \quad (1)$$

$$\phi \frac{\partial S}{\partial t} + \nabla \cdot (\lambda_w / \lambda) (v - K \lambda_o (\rho_w - \rho_o) g \nabla Z) = 0. \quad (2)$$

The subscripted λ 's are relative mobilities of water and oil and are assumed to be known functions of S . The variable ϕ , the porosity, depends only on x . The densities ρ , absolute permeability K , and gravity term $g \nabla Z$ are usually assumed constant. In one dimension, (1) states that the volumetric flow v (the argument of the divergence) depends only on time. Thus, if boundary conditions are given independent of time, the volumetric flow must be a constant v_0 . By appropriately factoring (2), we get the "original" hyperbolic Buckley-Leverett equation.

$$\frac{\partial S}{\partial t} + \frac{v_0}{\phi} \frac{\partial}{\partial x} \frac{\lambda_w}{\lambda} \left(1 - \frac{K \lambda_o}{v_0} (\rho_w - \rho_o) g \nabla Z \right) = 0, \quad (3)$$

which with other simplifying assumptions plus setting $u = S$, yields the scalar con-

servation law

$$u_t + (f(u)(1 - g(u)))_x = 0. \quad (4)$$

In the most common version of (4), based on empirical quadratic formulations of relative permeabilities for water and oil, we use

$$f(u) = \frac{u^2}{u^2 + \sigma(1 - u)^2}, \quad g(u) = \alpha(1 - u)^2. \quad (5)$$

The convective term in (4), is often called the fractional flow. The parameter σ is the viscosity ratio of water to oil typically taken to be $1/4$; α is a dimensionless parameter involving gravity, densities, and volumetric flow rates, all of which are irrelevant in this discussion.

The fractional flow and some properties are shown in Figure 1. From it, it is easy to identify two regimes: a *limited-gravity* regime and a *gravity* regime. In the first case, $0 \leq \alpha < 1$, the flow is monotonic increasing, which results in signals that will always propagate to the right. Furthermore, the flow function has *exactly* one change in convexity and goes from being convex at low saturation values to being concave at high saturations. The gravity regime, corresponding to $1 \leq \alpha$, has two inflexion points. As we will discuss later in the paper, this property was believed to be the cause of early numerical difficulties, but the change in sign of the first derivative may have played a more important role.

In this paper we address only the limited-gravity regime. Within the context of modern shock capturing methods, this is one of the first non-convex problems addressed. As we shall see, sonic expansions are never developed and the exact Godunov method becomes simple upwinding. It also has the advantage that the

initial boundary value problem (4,5) and $u(0, t) = 1, u(x, 0) = 0$, is well posed and can be considered a Riemann Problem, abbreviated henceforth $RP(u_l, u_r)$.

The early work [2, 14], was followed by many papers by Glimm, Collela and others [12, 4, 1] and were attempts to solve these type of equations in two dimensional problems. Unfortunately, the equations do not preserve the hyperbolic form (3) or (4,5). The pressure (1) and saturation (2) equations are of elliptic-like and first-order hyperbolic type. To solve them, hybrid methods have been devised like IMPES [17], which are used in large reservoir simulations in use today. A renewed interest in the conservation form of the BL equations occurred after the inclusion of a capillary pressure empirical function. After this inclusion, the equation becomes of convection-diffusion type. Nowadays, for the most part, the conservation law BL has remained of only theoretical interest, and in this paper we intend to use it as such, as an example of a non-convex conservation law, using the relatively abundant published numerical solutions.

In Section 2, we outline the Godunov and Roe's method and introduce the *third-state solvers*. Third-state solvers can be described simply as an inexact, two-wave, Roe matrix. Section 3 begins by describing the solution to $RP(u_l, u_r)$ in a specific non-convex setting, which makes it easy to make a case for a particular type of third-state solvers (which are correctors to fast but inexact approximations). The methods presented are classified in as 1-wave or 2-wave. Third-state solvers are used exclusively on the latter.

In the next section, the ideas are cast into the framework of CLAWPACK. As

Figure 1: Fractional flow for quadratic BL (5). Monotonicity (left), convexity (right). The effect of increasing gravity $\alpha = 0, 1 \dots 4$ is shown.

test problems, we have chosen cases often found in the literature. Some of the strong points of the third-state methods presented is their efficiency, thus we perform some timings of the algorithms, while acknowledging limitations. In addition, carefully measured errors and the solution of an initial value problem are presented.

2 Godunov method and third-state Riemann solvers

There are many available numerical methods that capture shocks properly. Many can be viewed as derivations of the Godunov method, based on local solution of Riemann problems.

For this discussion, we denote the flux (fractional flow) f . In the scalar case

Figure 2: Shock types for convex and non-convex flow Riemann problems ($u_l > u_r$).

there is a natural relation between convexity and the formation of shocks/expansions. Consider a monotonically decreasing initial function. Locally we solve a $RP(u_l, u_r)$ with $u_l > u_r$. Assuming that $f'' > 0$, since the characteristics have speeds $f'(u)$ we have $f'(u_r) < f'(u_l)$ and a shock will develop. This is illustrated in Figure 2, left. A similar argument, when applied to a monotonic increasing initial value, will result in a rarefaction zone since the solution of $RP(u_l, u_r)$ for $u_r > u_l$ results in the development of an expansion fan. An arbitrary initial value, may develop a shock at some finite time.

In either case, shocks and fans may propagate forward or backward, their direction depends exclusively on the sign of $f'(u)$. However, there are situations where we have a *sonic expansion* where the local expansion fan straddles $x/t = 0$. The methods under discussion confronting such situations are particularly prone to develop physically incorrect solutions.

The classical Godunov Method is given by

$$u_j^n = u_j^n + \frac{\Delta t}{\Delta x} (F(u_{j+1}^n, u_j^n) - F(u_j^n, u_{j-1}^n)), \quad (6)$$

where each F is called a numerical flux. In the case of Godunov

$$F(u_{j+1}^n, u_j^n) = \frac{1}{\Delta t} \int_{\Delta t} f(w(0)) dt \quad (7)$$

where the function $w = w(x/t)$ is an *exact* solution for $RP(u_j, u_{j+1})$. When the time step is kept small enough, so that waves do not have the opportunity to interact, then the integrand in (7) remains constant and the numerical flux is evaluated to

$$F(u_{j+1}^n, u_j^n) = f(w(0)) = \begin{cases} f(u_j), & \text{if waves travel left} \\ f(u_{j+1}), & \text{if waves travel right} \\ f(u_*), & \text{if sonic expansion.} \end{cases} \quad (8)$$

Godunov scheme is *conservative* by having the form (6) and *upwind* by satisfying the first two cases in (8). The third case is rather special and involves the intermediate state u_* .

In general is difficult to compute w exactly (and certainly not necessary since we need only $w(0)$), at least for vector equations. Since such computation has to be done at each mesh point, it is important that w be easy to calculate.

The next step in the development was based on the solution of a $RP(u_l, u_r)$ of typical one-dimensional constant coefficient n -system $u_t + Au_x = 0$. If A is diagonalizable, the solution consists of at most n -waves defining $n-1$ states connecting states u_r with u_l . If we rewrite the hyperbolic system in terms of characteristic variables, we see that the solution consists of states q_i .

$$u_r - u_l = \sum_{i=0}^{n-1} W_i, \quad W_i = q_i - q_{i-1}, \quad i = 0, \dots, n-1, \quad (9)$$

with $q_0 = u_r$ and $q_n = u_l$. Each state q_i is constant and for each “wave” W_i there is a speed α_i defined by the Rankine-Hugoniot relations on the corresponding characteristic variable. In the case of non-linear fluxes, such states may no longer be constant but they adopt the form of fans, that is, smooth transitions between states.

One way to specify a particular Riemann solver for a general non-linear flux is by the specification of a diagonalizable matrix $A(u_l, u_r)$, which satisfies the following properties, first proposed by Roe [16]

$$A(u_l, u_r)(u_r - u_l) = f(u_r) - f(u_l), \quad (10)$$

$$A(u_l, u_r) = \frac{\partial f}{\partial u}, \quad u_r, u_l \rightarrow u. \quad (11)$$

By analogy with the constant advection equation, the matrix $A(u_l, u_r)$, by means of its real eigenvalues, approximates the shock speeds. This approach assumes that *all* waves are “shocks.” Numerical problems associated with this approach have been well studied and are related to the *entropy conditions*, namely, certain requirements on the speed of decay of fans connecting two states. In Godunov-type methods (those based on exact or approximate Riemann solvers), only the “centered value,” as seen in (8), is involved in the numerical computation. Thus, in general, we do not need to worry about schemes violating entropy, if there are no sonic expansions. The standard reference is [7, 5], but an excellent discussion is found in [9].

For a scalar problem, the only choice is to set

$$A(u_l, u_r) = \frac{f(u_r) - f(u_l)}{u_r - u_l} \quad (12)$$

which defines a shock speed. The shock may not exist at all, since it depends on

whether $u_l < u_r$ or $u_r < u_l$ and on the sign of f'' , as discussed. We use the common terminology which is to call (12) the Roe matrix.

This relation cannot be universally used to define A ; it might be entropy violating. We give an example of this situation later. Another reason has to do with the implementation; for nearly equal values of u in (12) the subtraction in the denominator may result in cancelation of significant digits, in which case relation (11) must be used. This switching may render the solver procedure inefficient.

An alternative expression for $A(u_l, u_r)$ might be constructed by connecting the two states u_l and u_r by a straight line and integrating over its path. Defining

$$h(\theta) = u_r\theta + (1 - \theta)u_l,$$

the desired property is obtained by a change of variables over the integral

$$f(u_l) - f(u_r) = \int_{u_l}^{u_r} f'(u) du = (u_l - u_r) \int_0^1 f'(h(\theta)) d\theta,$$

thus

$$A(u_l, u_r) = \int_0^1 f'(h(\theta)) d\theta.$$

Notice that (10,11) are automatically satisfied.

However, we still need to compute the integral. Consider, for example, the quadrature formulas of Newton-Cotes type. Some of the simplest are:

$$\begin{aligned} \bar{A}_1(u_l, u_r) &= f'\left(\frac{u_r + u_l}{2}\right), \\ A_1(u_l, u_r) &= \frac{1}{2}(f'((u_r + 2u_l)/3) + f'((2u_r + u_l)/3)), \\ A_2(u_l, u_r) &= \frac{1}{6}(f'(u_r) + 4f'\left(\frac{u_r + u_l}{2}\right) + f'(u_l)). \end{aligned} \tag{13}$$

The i -th formula requires $i + 1$ evaluations of f' . All such formulas have errors that can be expressed

$$A(u_l, u_r) - A_n(u_l, u_r) = k f^{(n+2)}(u^*), \quad (14)$$

where k is a constant that depends only on the quadrature used. These are examples of (one-wave) *approximate* Roe matrices.

In the case when f'' has constant *sign*, these error terms can be thought of as “departure from conservation.” The condition (12) can be thought as the Rankine-Hugoniot relation, which, depending on the sign of f'' must hold for either $u_l < u_r$ or $u_r < u_l$. The replacement of A by A_* results in a spurious shock speed, if the states are in the proper order. In this paper, by *non-conservative* we refer to Godunov-type schemes that use these inexact solvers, rather than the more standard definition using the numerical flux as in (6).

For fluxes of constant convexity, the Roe matrix coincides with its approximations $A = \bar{A}_1 = A_1$, and we get the extra accuracy at no cost. This explains why the simple Riemann solver $\bar{A}_1 = (u_l + u_r)/2$ produces such a good solution in a Burgers' equation, provided that sonic expansions are avoided. Following the discussion of fans and shocks in this section, if $u_r = -u_l$, $A = \bar{A}_1 = 0$, implying a correct steady shock if $u_l > 0$, but an entropy violating “shock” if $u_l < 0$. Hence (12) can potentially result in unphysical solutions.

A formality for third state solvers

Let $U = U(u_l, u_r)$ be a “third state”. In general, we want U to be between u_l and u_r ,

but this is not required. We define the the matrix

$$\mathcal{A} = \begin{pmatrix} A_I & B \\ B & A_{II} \end{pmatrix}, \quad (15)$$

and the vector

$$v = \begin{pmatrix} u_r - U \\ U - u_l \end{pmatrix}, \quad (16)$$

where $A_I(U, u_r)$, $A_{II}(u_l, U)$ are exact or approximate Roe matrices (12,13). One can think of A_I and A_{II} as carrying the principal waves and B as a correction term when approximations are used. Since we have to revert to the scalar space, a linear functional $1 \cdot$ is used to define the analogous of Roe's condition (10)

$$1 \cdot \mathcal{A}v = f(u_r) - f(u_l), \quad \text{if } 1 \cdot v = u_r - u_l. \quad (17)$$

In the same way that we have used *approximations* to the Roe matrix of the type (13), we will also use approximations $\mathcal{A}_i$ to $\mathcal{A}$ satisfying (17). We will call these *two-wave* approximate Roe matrices.

The formal error $B(u_l, u_r)$ can be considered as departure from conservation in the sense discussed above. In some cases, a simple expression for the formal error $B(u_l, u_r)$ can be found. For example if

$$A_I = A_{II} = A_n,$$

where we assume that $f^{(n+2)}$ does not change sign, the correction term is

$$B(u_l, u_r) = k f^{(n+2)}(u^*).$$

Using (15,14)

$$1 \cdot \mathcal{A}v = (A_I(U, u_r) - k f^{(n+2)}(u_I))(u_r - U)$$

Figure 3: Relation between ‘true’ third state u_m , used in method 2G, and ‘false’ third state u_c , for method 2F.

$$\begin{aligned}
 &+ (A_{II}(u_l, U) - k f^{(n+2)}(u_{II}))(U - u_l) + B(u_l, u_r)(u_r - u_l) \\
 &= f(u_r) - f(u_l).
 \end{aligned}$$

Thus, u^* must be an intermediate value such that

$$f^{(n+2)}(u^*) = (f^{(n+2)}(u_l)(u_r - U) + f^{(n+2)}(u_{II})(U - u_l))(u_r - u_l).$$

3 Specific two-wave methods and third-states

Having defined the two-wave Riemann solver, we need to choose an appropriate third state. The solution to non-convex Riemann problems, provides some criteria on how to chose the third state.

Analytical solutions for $RP(u_l, u_r)$ for non-convex fluxes

We consider a flow f and two states $u_r < u_l$, with f being convex-concave in the

interval $[u_r, u_l]$. In other words, f satisfies

$$f''(u) \text{ is } \begin{cases} > 0, & u_r \leq u < u_c, \\ = 0, & u = u_c, \\ < 0, & u_c < u \leq u_l. \end{cases}$$

It is well known that the unique, entropy-satisfying weak solution of $RP(u_l, u_r)$ with this f is given by

$$u(x, t) = w\left(\frac{x}{t}\right) = \begin{cases} u_r, & f'(u_m) \leq \frac{x}{t} \\ v\left(\frac{x}{t}\right), & f'(u_l) \leq \frac{x}{t} < f'(u_m) \\ u_l, & \frac{x}{t} < f'(u_l) \end{cases} \quad (18)$$

with v satisfying

$$f'(v(\xi)) = \xi, \quad (19)$$

and $u_m = u_m(u_l, u_r)$ is a third state defining the *convex hull* of the graph of f on the interval $[u_r, u_l]$, Figure 3. By construction $f'(u_m)$ is the shock speed given by the Rankine-Hugoniot relations. Also note that $u_c \leq u_m \leq u_l$ and that in the interval $[u_c, u_l]$ f' is monotonic. From this, it follows that v satisfying (19) is unique. It can be shown that this construction is, in addition, entropy satisfying and thus the unique solution to the Riemann problem. With this type of flow, in contrast to the convex case, a decreasing initial function and will have next to the rarefaction part (the concave region of f) a shock starting at $t = 0$ and delimited by the fastest characteristic with speed $f'(u_c)$.

The solutions to the $RP(u_l, u_r)$, ($u_r < u_l$) at any given time $t > 0$, consist of two waves. One wave is a shock $u_m - u_r$ traveling with speed $f'(u_m)$ and the other an expansion fan v immediately following the shock. This is also illustrated in the right-hand side of Figure 2. In the case $u_r > u_l$, a dual construction is necessary. The dual solution is similar, consisting of a third state defining a *concave hull* to connect

it via shock wave to the right state and an expansion to connect it to the smaller left state.

It is apparent that solving this problem using a Godunov-type method, would not only involve a more elaborate Riemann solver, but in addition, requires the determination of $u_m(u_l, u_r)$. For a flux of type defined in (3), the method described in [2] (see also later note) explicitly finds u_m and then solves the Riemann Problem. [14] suggests a general geometric way to deal with more than one change in convexity, and presents a Buckley- Leverett solution in the gravity regime for $\alpha = 10$.

We sought a “universal” third state U which will not require re-computation every time a Riemann problem is approximately or exactly solved. U might be initial value problem dependent, but should remain the same for every local $RP(u_l, u_r)$ arising during the computation. Moreover, we required that the third state U and associated two-wave solver be used in the following fashion. For the case $u_l > u_r$, let the solver be

1. two-wave $\mathcal{A}_*$ only if $u_l > U > u_r$, and
2. one-wave otherwise.

Both of these conditions can be satisfied using the two-wave Roe matrix (15), in which $U = u_l$ if $u_l < U$ and $U = u_r$ if $u_r > U$. Then, (15,16) become

$$1 \cdot \mathcal{A}v = \begin{cases} (A_I + B)(u_r - u_l), & u_l < U \\ 1 \cdot \mathcal{A}v, & u_l > U > u_r \\ (A_{II} + B)(u_r - u_l), & u_r > U \end{cases} \quad (20)$$

If $u_l < u_r$, a dual construction should be used possibly involving another universal

third-state.

Proposed Methods

The methods we tested can be classified in two groups: methods that do not use specific properties of the flux like convexity, inflexion points etc., and methods that do use that type of information. Of the first type are all the non-conservative approximations (13). Note that in this case, the meaning of non-conservative as violating Rankine-Hugoniot no longer has any meaning since as we have seen in (18) there might not be any shock directly involving u_l and u_r . Solutions using these techniques are named methods 1A0, 1A1, 1A2 for approximations $\bar{A}_1, A_1, A_2$, respectively. We refer to them as one-wave methods. They are progressively more expensive to compute, requiring respectively 1, 2 and 3 evaluations at every RP solved.

On the other hand, the two-wave methods use a universal third state and two-wave approximate Roe matrices (15). The properties of the exact solution, suggests that taking $U = u_m$ is a natural way to choose the third state. The values behind the shock are slightly above u_m according to (18). Viceversa, values ahead of the shock will be less than u_m . Thus, if we assume that the numerical solution, as does the exact solution, preserves monotonicity (as is in fact the case), it is apparent that all we need is a one-sided solver and a single third state defining the convex hull.

The two-wave Roe matrix approximation, is constructed using the least expensive possibilities, namely the diagonal

$$\mathcal{A}_G = \begin{pmatrix} A & 0 \\ 0 & \bar{A}_1 \end{pmatrix} \quad (21)$$

where A is the unique Roe matrix given in (12). We call this method 2G.

Two other two-wave methods are tested using two-wave Roe matrices

$$\mathcal{A}_F = \begin{pmatrix} A & 0 \\ 0 & \bar{A}_1 \end{pmatrix}, \quad \mathcal{A}_{F0} = \begin{pmatrix} \bar{A}_1 & 0 \\ 0 & \bar{A}_1 \end{pmatrix}. \quad (22)$$

Notice that the first solver would seem to be identical to that of $2G$; however at this time, we have chosen the third state $U = u_c$, the inflexion point. In the next section we will see that this is an important difference. The methods described by (22) are called, respectively, $2F$ and $2F0$.

Methods $2F$, $2G$ call for the exact Roe matrix (12) when reverting to one-wave in the case $u_l < U$. We prefer the use of $\bar{A}_1$ rather than the switching from secant to derivative. The latter procedure also provides certain numerical advantages.

The number of evaluations in these two-wave methods is variable, being more expensive when the third-state is needed. In the next section, we shall see that in practice, these are approximately as expensive as the simplest one-wave methods but much more accurate.

4 Numerical Experiments

CLAWPACK: The Conservation Laws Package

All computations were made using CLAWPACK, a Fortran software package for conservation laws using state-of-the-art generalized first order and second order Godunov methods. The package was developed by R. LeVeque and recently has undergone a major revision [10], with updated documentation available on-line [11]. Currently, it supports the solution of (first-order) hyperbolic problems in two dimensions, and three dimensions will be incorporated into the package in the future.

In one dimension, the core of the program consists of a routine *claw1* and a user supplied Riemann Solver *rp1* (note, that this routine has been changed in format from that described in [10]), which provides the (approximate) Roe matrices. In addition, the package includes a driver routine that does the input/output and provides *claw1* with the data. In the variable time step option, the user may suggest maximum Courant numbers to be used. However, the actual value used depends upon run time data, unless it exceeds the user value. If this is the case, the time-step is retaken using a smaller Courant number.

In addition, CLAWPACK allows the specification of more than one wave associated to each scalar component, each of which may use a different Riemann solver. Typically, as in the case for the all-shock simplification of the Roe matrix, each wave W_i is a difference between two states as in (9), although more general waves can be specified. The two-wave methods simulate such waves by superposition of these “shock-like” waves, automatically done by the routines of the program.

Riemann problems

As test problems, we have chosen problems that have been encountered in the literature for the conservation form of the BL equations. They are of two different types: purely $(1, 0)$ Riemann problems and initial value problems.

The first test problem is the classical $RP(1, 0)$ of the quadratic Buckley-Leverett with no gravity, described earlier. This problem has been used as a test in [8, 6, 3], and with variations of σ in (5) by [1]. In some of those cases moving mesh techniques have been used. We refer to this basic problem as *Problem A*.

In [3] are incorporated moving mesh techniques, a Galerkin method, and a centered finite difference. Unfortunately, an explicit dissipative term is added, which smears the theoretically sharp shock and fan, presented at $t = 0.3$. [8] shows a number of explicit and implicit methods for Problem A, but again dissipation has been added in the form of empirically determined capillary pressure and the same type of problems appear in [3].

The results for Problem A, that seem better are those presented by Bell and Shubin, [6]. No approximate solvers are used. Exact Godunov, equivalent to upwinding in this particular case, results in sharp numerical shocks, correctly positioned. Similar results (for $\sigma = 1/2$) are reported in [1]. However, both computations show an error by defect over the expansion fan. This error is especially noticeable at an early stage (small t) of the computations, and tends to fade away as the computation progresses. An explanation is provided later.

Collela and co-authors, succeeded in reducing these upwinding effects by using the idea of limiters [1]. Bell and Shubin achieve the desired effect by mesh moving [6]. We point this out because the third-state methods presented in this paper have been good at providing correct fans especially at *early stages*, without limiters or mesh moving.

It is interesting that, as used in the above references, Godunov method does not call for any specific special treatment with respect to non-convexity. This follows from the monotonicity of f' , which causes the numerical flux in (8) to be evaluated always upwind. Should a sonic fan be present, that information is needed.

The second of the Riemann problems tested, *Problem B*, is derived from another

Figure 4: Fractional flow for cubic BL 23 with $\alpha = 0$. Monotonicity (left), convexity (right).

model for permeabilities [18]. The resulting flow uses

$$f(u) = \frac{u^3}{u^3 + (2/3)(1-u)^3}, \quad g(u) = \alpha(1-u)^3. \quad (23)$$

Figure 4 shows this flow and its properties. This problem exhibits a single very sharp change in convexity for $\alpha = 0$, and at the same time, the solution has a very steep fan at early stages. This is due to the solution in (18) $v' \approx 1/f'$ near u_l , and we have that in this region $f' \approx 0$. Another reason why this is a good test problem is that the ‘correct’ expansion fan speed for the Riemann problem $f'(u_c)$ in $2G$ differs most from the ‘false’ shock speed in method $2F$. Thus (23) might be useful to discriminate between third states.

For timings of test problems, a single computation was performed at $t = 0.4$, with

a single call to the main routine *claw1*. The maximum Courant number allowed was 0.9. Table I presents the number of function evaluations for one-wave methods, and timings in seconds. The times shown were best group-of-ten-average each over thirty runs on an IBM RS6000/590. The performance figures, varied about ten percent depending on the time of day the group-of-ten tests were run. The values in each column represent the average time per step and the total time on the routine *claw1*, which calls the solver and performs the time update. Sometimes a step is *attempted* but not actually executed due to Courant number restrictions. In consequence, the actual time per step may be less than reported in the table. This effect was particularly emphasized with the Problem B, since more steps were rejected, especially with the low order one-wave method. Thus relative timings between the 1A0 and 1A2 methods (and possibly between these same methods and the two-wave methods), might be enhanced by changing the Courant number.

The values in the Table I are for comparison purposes. No attempt was made to optimize the computations. Problem B evaluations of f' are significantly more expensive to compute since more powers are computed. The significant result is that two-wave methods take, in worst case, the same time as the simplest method with one evaluation, but producing more accurate solutions, provided the third-state is correctly chosen.

Method	calls	Problem A (sec.)		Problem B (sec.)	
1A0	1	0.00159	0.310	0.00216	0.501
1A1	2	0.00172	0.324	0.00250	0.581
1A2	3	0.00208	0.371	0.00260	0.556
2F0	var.	0.00160	0.325	0.00190	0.470
2F	var.	0.00158	0.321	0.00189	0.470
2G	var.	0.00163	0.320	0.00162	0.406

Table I: Per-step (non rejected) and total time in *claw1*.

Meth.	Global	Fan	Meth.	Global	Fan
1A0	.0127343	.0001207	1A0	.0176723	.0000745
	.0067169	.0005001		.0133424	.0001603
1A1	.0108729	.0002303	1A1	.0134316	.0001009
	.0046633	.0007722		.0090857	.0002525
1A2	.0035801	.0000310	1A2	.00343597	.0000301
	.0006184	.0002024		.0003703	.0000688
2F0	.0047277	.0000662	2F0	.0061992	.0000670
	.0010862	.0000293		.0037635	.0000275
2F	.0036914	.0000716	2F	.0035202	.0000742
	.0003098	.0000326		.0012257	.0000340
2G	.008034	.0000494	2G	.0139403	.0000552
	.0051169	.0000310		.0119237	.0000236

Table II: L_2^2 errors on two meshes for Problem A, $t = 0.13, 0.40$.

We present the results of the computations by showing data obtained by two types of meshes, coarse – 50 cells – and fine –100 cells. Most of the graphs are shown for the fine mesh, but errors are tabulated for both meshes.

Figure (5) illustrates graphically the behavior of one-wave methods (13) at two different times in solving Problem A. Also plotted is the exact solution shown by a solid line. The shock position is shown by the dashed line. Although apparently qualitatively correct, the main source of error is at early times ($t = 0.13$) and occurs at the fan. The typical conservation error is manifested by what appears as different

Figure 5: One-wave methods, Problem A, at times $t = 0.13, 0.40$.

shock speeds.

Meth.	Global	Fan	Meth.	Global	Fan
1A0	.0231095	.0005866	1A0	.0320679	.0001885
	.0146064	.0001470		.0184460	.0000417
1A1	.0105183	.0000110	1A1	.0226466	.0000195
	.0109267	.0001126		.0182401	.0000306
1A2	.0142835	.0012906	1A2	.0143867	.0005049
	.0050992	.0001511		.0013507	.0000480
2F0	.0044701	.0002333	2F0	.0135326	.0055974
	.0048356	.0009189		.0182067	.0142653
2F	.0005029	.0001961	2F	.0003691	.0001464
	.0004402	.0000519		.0009637	.0000615
2G	.0056229	.0000899	2G	.0142502	.0000812
	.0039961	.0000404		.0093260	.0000322

Table III: L_2^2 errors on two meshes for Problem B, $t = 0.13, 0.40$.

At later times ($t = 0.4$), the error is magnified for low order approximations. From right to left, shock positionings are 1A0, 1A1 and 1A2. Nonetheless, it seems that

Figure 6: Two-wave methods, Problem A, at $t = 0.13, 0.40$.

the fan errors have diminished for each scheme. This is expected; if we think of the numerical solution as an exact solution to another conservation law, *any* expansion fan will uniformly converge to the constant state u_l as required by the entropy law, and thus the numerical fan will converge to the exact fan over a fixed spatial interval.

The high-order, non-conservative, one-wave approximations seem to propagate the shock much more closely to the correct speed. Analogous observations can be made for Problem B.

Figures 6 and 7 present the results at the same two times, for Problems A and B, using the two third-state methods. We first notice the accuracy of both early and fully expanded fans. This contrasts sharply with the one-wave methods.

The third-state method 2G, constructed on the physical Riemann problem so-

Figure 7: Two-wave methods, Problem B, at $t = 0.40$.

lution, introduces a conservation error possibly generated in a situation during the solution which result in RP with states fully within a concave zone $[u_c, u_m]$ and being given the wrong treatment instead, or viceversa. A similar explanation may hold valid for the $2F0$ (computations not shown) but which exhibits a similar problem.

To verify our assertions, we computed two types of L^2_2 discrete errors, one the *global error* E_g , and the other the *fan error* E_f defined

$$E_g = \Delta x \sum_{i=0}^{n-1} ((u(x_i) - u_i)^2, \quad E_f = \Delta x \sum_{i=0}^{p-1} ((u(x_i) - u_i)^2. \quad (24)$$

The upper index for the fan error computation p is computed by rounding the exact speed, i.e. $p = \text{round}(f'(u_m)t/\Delta x)$. The similarity property of v in (18) could have also been used. The numerical values results are shown in Tables II and III

Note that the construction using the third state $U = u_c$, method $2F$, proves

very successful in solving both problems. In Table II, 1A2 appears to have a slight advantage over 2F, but this can be due to the rounding of the shock position. For Problem B, at early stages, all of the one-wave methods have errors significantly larger than Problem A. In this situation, the two-wave methods (in particular 2F), perform significantly better, at minimum cost.

An Initial Value Problem

A smooth initial boundary value problem is not very useful as a test of a Riemann solver, unless it quickly develops a discontinuity where possible improvements are made patent. No such appropriate problem in the surveyed literature seemed suitable. Nonetheless, we show an initial value computation using method 2F.

The problem chosen, originally proposed by [2], is the most widely used. Recently, this was a test problem, Ren and Russell [15], in a mesh moving technique. The initial and boundary conditions are

$$u_0(x) = \frac{0.1}{(0.1 + x)}, \quad u(0, t) = 1. \quad (25)$$

[15] uses the quadratic model (5) but takes the viscosity ratio $\sigma = 1/2$ in (5), making it slightly different from that used in Problem A. Figure 8 shows the results on a coarse mesh with 40 nodes, showing initial value (25) and shock formation at 0.1 second intervals.

5 Conclusions

This paper has given some steps toward answering the general question: How much a priori information needs to be supplied to the approximate Riemann solver in order

Figure 8: Shock formation for the initial value problem (25), using method 2F.

to obtain reasonable solutions for the non-convex problem? From the computations made, it seems that the answer is: very little. Experiments suggests that ‘high order’ one-wave methods like 1A2 have in general performed fairly well, in terms of shock resolution and global error. The two-wave method, which requires only the inflexion point location, improves the simplest one-wave methods at no cost.

However, one must be cautious here, since the issue has to do more with the *monotonicity* of the flow more than with the *convexity*. In this paper, as in most literature on the subject, only problems in the limited-gravity regime have been addressed. Each case represents monotonic flow, in which the special situation of sonic expansion does not occur.

The much more difficult problem of the BL in the gravity regime, is, to our

knowledge, underrepresented in the literature. Part of the problem may be attributed to the relevant theory (i.e. entropy corrections) which was not fully developed until the mid-eighties. The interest in the conservative Buckley-Leverett had diminished by this time.

The analytical solution of the gravity problem is well known [14]. This same paper presents two numerical computations using the random choice method coupled with an exact Riemann solver. Unfortunately, for the Riemann problems presented only limited information about the numerical solution is provided.

The potential utility of two-wave methods, within this limited type of non-convex flow, has been established as an alternative to the more expensive approximate Roe matrices. The use of the inflexion point u_c as opposed to the point defining the convex hull u_m yields much better results with respect to shock positioning and fan accuracy. Limited experiments have shown that the price to pay for these advantages is round-off errors, as they occur earlier than with the one-wave methods.

The natural next step is to investigate the methods in sonic expansion situations. Solutions of the type (18) are still valid, as long as there are no convexity changes. Overall, there are still grounds for generalizations and extensions of these ideas as well as for development of more theoretical results.

References

- [1] Ph. Collela, P. Concus, and J. Sethian. Some numerical methods for discontinuous flow in porous media. In R.E. Ewing, editor, *Mathematics of Reservoir*

Simulation. SIAM, Philadelphia, 1983.

- [2] P. Concus and W. Prokurowski. Numerical solution of a nonlinear hyperbolic equation by the random chooice method. *J. Comput. Phys.*, 30:153–166, 1979.
See Correction in JCP 73, 490 (1987).
- [3] J. Douglas, M.F. Darlow, B.L. and Wheeler, and R.P. Kendall. Self-adaptive finite element and finite difference methods for one-dimensional two phase immiscible flow. *Comp. Meth. Appl. Mech. Eng.*, 47:119–130, 1984.
- [4] J. Glimm, Lindquist, B. McBryan O., and L. Padmanabhan. A front tracking reservoir simulator , five-spot validation studies and the water coning problem. In R.E. Ewing, editor, *Mathematics of Reservoir Simulation*. SIAM, Philadelphia, 1983.
- [5] A. Harten, P. Lax, and B. van Leer. On upstream differencing and Godunov type schemes for hyperbolic conservation laws. *SIAM Review*, 25:35–61, 1983.
- [6] Bell. J.B. and G.R Shubin. An adaptive finite difference method for conservation laws. *J. Comput. Phys.*, 52:569–591, 1983.
- [7] P.D. Lax. *Hyperbolic Systems of Conservation Laws and the Mathematical Theory of Shock Waves*, volume 11 of *Regional Conference Series in Applied Mathematics*. 1973.
- [8] S.H Leventhal. The operator compact implicit method of reservoir simulation. In A.E. McDonald, editor, *Numerical Simulation II*. Society of Petroleum Engineering, Richardson, Texas, 1986.

- [9] R.J. LeVeque. *Numerical Methods for Conservation Laws*. Lectures in Mathematics, ETH Zürich. Birkhäuser Verlag, 1992.
- [10] R.J. LeVeque. CLAWPACK – a software package for solving multi-dimensional conservation laws. In *Proceedings of 5th Intl. Conf. on Hyperbolic Problems*. Stony Brook, 1994.
- [11] R.J. LeVeque. *CLAWPACK notes (notes??ps, Changes.V2.1)*. U. of Washington, November 1995.
- [12] P. Lotstedt. A front tracking method applied to Burgers' equation and two-phase porous flow. *J. Comput. Phys.*, 47:211–228, 1982.
- [13] D.W. Peaceman. *Fundamentals of numerical reservoir simulation*. Elsevier, 1977.
- [14] W. Proskurowski. A note on solving the Buckley-Leverett equation in the presence of gravity. *J. Comput. Phys.*, 41:136–141, 1981.
- [15] Y Ren and R.D. Russel. Moving mesh techniques based upon equidistribution and their stability. *SIAM Journal of Sci. Stat. Comp.*, 13(6):1265–1286, November 92.
- [16] P.L. Roe. Approximate Riemann solvers, parameter vectors, and difference schemes. *Jour. Comp. Phys.*, 26:649–686, 1981.
- [17] T.F. Russell and M.F. Wheeler. Finite element and finite difference methods for continuous flows in porous media. In R.E. Ewing, editor, *Mathematics of Reservoir Simulation*. SIAM, Philadelphia, 1983.

- [18] L.C. Young. Spatial approximation for simulating fluid displacements in petroleum reservoirs. *Comp. Meth. Appl. Mech. Eng.*, 47:3-46, 1984.

A Finite Element Simulation System in Reservoir Engineering

A Project

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
Master of Science in System Science

in

The Department of Computer Science

by

Xiaozhong Gu

March 1996

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

ABSTRACT

1. RESERVOIR SIMULATION TOOLS

1.1 BOAST

1.2 BOAST'S KNOWN LIMITATIONS

2. FEM WELL REGION MODEL

2.1 FEM FOUNDEMENTALS

2.2 INTERFACE TO BOAST

3. A REAL TIME DISTRIBUTED GRAPHICS SYSTEM

3.1 SYSTEM DESCRIPTION

3.2 WINDOW INTERFACE/NETWORK

3.3 SYSTEM REQUIREMENT

3.4 DESIGN

3.5 PROTOTYPE USER'S MANUAL

4. TEST RESULTS

4.1 TEST ONE

4.2 TEST TWO

5. CONCLUSIONS

REFERENCES

APPENDIX

ACKNOWLEDGEMENTS

Above all others, I would like to express my sincere gratitude to my major advisor, Dr. John Tyler, for his consistent direction and encouragement through this project.

I am always grateful to Dr. Jianhua Chen and Dr. Si-Qing Zheng for serving on my committee and extending their suggestions and advice.

Dr. Ming-Ming Chang, a former senior research scientist from National Institute for Petroleum and Energy Research, spent a week in August, 1995 on providing me useful suggestions related to this project.

I would like to thank Hui Ning, Kunyang Wang, Ce Xu, Jiangchen Yu, Jianhui Tao and Rui Shen and many others for all the helps they have provided through the project. Special thanks are given to Husam Yaghi, who not only offered many useful discussions, but also helped me to type the Appendix.

I am also grateful to Elias Khalaf for his work of setting up the computing facility related to this project.

Finally, the financial support from Department of Energy is greatly appreciated.

ABSTRACT

This project implemented a finite element simulation system used in Petroleum Engineering. This system can simulate a general three dimension three phase fluid flow problem in a porous media. Using a refined mesh in the well block, it can provide a more accurate solution of the physical variables in the well region than any other known reservoir simulator. Two test problems, from National Institute for Petroleum and Energy Research, was used to test on this system. The test results are consistent with the fundamental principles of fluid flow and Petroleum Engineering.

A prototype real time graphics system has been developed to visualize the simulation results. Xlib was used to provide the basic elements of drawing an object. Motif 1.2 was used to provide the user interface. The UNIX version 4.3 BSD socket was used to transfer data between the two required processes, one for the simulation running in an IBM RS 6000 machine and the other for the graphics running in a Dec Alpha Station 200.

REFERENCES

1. John M. Tyler, "3D Three Phase Finite Element Method in Porous Media", Department of Computer Science LSU, 1994
2. Ming-Ming Chang, "BOAST Technical Manual", NIPER 1991
3. Husam M. Yaghi, John M. Tyler and Xiaozhong Gu, "3-D Finite Element IMPES Formulation for Oil Reservoir Simulation", 5th International Conference and Exhibition on Multi-lingual Computing, Cambridge, Apr 11-13, 1996
4. Husam M. Yaghi, John M. Tyler and Xiaozhong Gu, "Triphasic Fluid Flow simulation: IMPES vs. Fully-Implicit 3D FEM", Parallel CFD '96, Italy, May 20-23, 1996
5. D.W.Peaceman, "Fundamentals of Numerical Reservoir Simulation." Elsevier, 1977
6. W. Richard Stevens, "Unix Network Programming", Prentice Hall, 1990
7. Adrian Nye, "Xlib Programming Manual Vol.1", O'Reilly & Associates, Inc., 1992
8. Adrian Nye, "Toolkit Intrinsics Programming Manual Vol.4", O'Reilly & Associates, Inc., 1992
9. Dan Huller and Paula M. Ferguson, "Motif Programming Manual Vol. 6A", O'Reilly & Associates, Inc., 1993

1. RESERVOIR SIMULATION TOOLS

1.1 BOAST

Reservoir simulation is performed to predict the future performance of a reservoir based on its current state and past performance and to explore other methods for increasing the recovery of hydrocarbons from a reservoir. Reservoir simulators are routinely used for these purposes. A reservoir simulator is a sophisticated computer program which solves a system of partial differential equations describing multiphase fluid flow (oil, water and gas) in a porous reservoir rock.

Reservoir simulators are usually based on:

- (1) reservoir engineering principles,
- (2) a set of partial differential equations that describe the flow of fluids through porous media,
- (3) a numerical technique (Finite Difference or Finite Element Method) to solve the set of differential equations,
- (4) and a computer program which implements the computing model. It usually requires a high performance computer and a graphics workstation to display the computed results.

BOAST is one of many current reservoir simulators in use today. The

version of BOAST used in this research project was developed by NIPER (National Institute for Petroleum and Energy Research) in July, 1991. It can be used to simulate primary depletion, pressure maintenance by water and/or gas injection, and basic secondary recovery operations (such as waterflooding) in a black-oil reservoir. It is competitive with commercially available simulators, most of which are developed by oil companies such as Amoco Production Co. and Exxon Production Research Co. Since BOAST uses an IMPES (Implicit pressure, explicit saturation) method to solve the differential equations, it is faster than most commercial simulators, but probably not as accurate.

1.2 BOAST'S KNOWN LIMITATIONS

BOAST has some limitations that do not allow accurate solutions in certain cases. These known cases are:

(1) It can only provide an average value of the pressure and phase saturations in the well region. A finer distribution of the pressure and the phase saturations in the well region are impossible. These distributions are very important for the prediction of the performance of a well.

(2) The well model used in BOAST, as well as with many commercial simulators, is not accurate enough to solve for pressure and saturations in the

well region. Relatively little information has appeared in the literature about detailed representations of wells in numerical simulators. Some discussions of well regions was published by Peaceman(1977). These state that when we want to represent an oil production rate:

$$Q_o = PI \left(\frac{\lambda_o}{B_o} \right) (p_e - p_{wf}) \text{ where } p_e > p_{wf}$$

where p_{wf} is the oil pressure in the wellbore and Q_o is the amount of oil produced from the well. PI is a constant, representing the rock and wellbore properties. This well model assumes the fluid flow is at steady-state and the oil is incompressible. These assumptions are not accurate. The fluid flow is not at steady state and the fluids are compressible in every stage of a simulation.

(3) BOAST uses the IMPES method, which solves the pressure equation first, then it solves for the phase saturations. This method is only conditionally stable. The stability limit for any explicit method is directly proportional to the grid size that is used. This is a major concern when small grid cells and very high flux rates are used that produce rapid saturation changes, which is the environment in the well region. Thus, to get reasonable solutions, either a large grid or a small time step must be used.

(4) Most commercial reservoir simulators today are fully implicit and BOAST is not.

The fundamental concept of the finite element method is that any continuous quantity, such as saturation and pressure, can be approximated by a discrete model composed of a set of piecewise continuous functions defined over a finite number of subdomains. The piecewise continuous functions use the values of the continuous quantity at a finite number of points in its domain.

The discrete model is constructed as follows:

1. A finite number of points in the domain are defined. These are called nodes.
2. The value of the continuous quantity at each nodal point is denoted as a variable which is to be determined.
3. The domain is divided into a finite number of subdomains called elements. These elements are connected at common nodes and collectively approximate the shape of the domain.
4. The continuous quantity is approximated over each element by a polynomial that is defined using the nodal values of the continuous quantity.

2.2 INTERFACE TO BOAST

To interface our FEM well region model with the block containing the well in BOAST, the movement of the three fluid phases (oil, gas and water) be-

tween neighbouring blocks in BOAST is determined by the transmissibilities. Assuming that the conditions needed for Darcy flow are satisfied, then an average phase volumetric flow-rate can be determined at each face of a block.

After the transmissibilities are computed in BOAST, they are used to determine the phase volumetric flow at each face of all of the blocks. This volumetric data is used in our well block after it is computed by BOAST.

The flow in BOAST is given in the direction of west, east, south, north, top and bottom of a block (Figure 2). Figure 2 presents these directions and what we named each phase variable for our well region model.

Figure 2

OW,OE,OS,ON,OT and OB arrays are the variable names for the phase

transmissibilities for oil phase. WW,WE,WS,WN,WT and WB are the phase transmissibilities for water. GW,GE,GS,GN,GT and GB are the phase transmissibilities for gas. DPi (i=1,6) are the pressure differences in the direction of two neighbouring blocks and these are needed to compute the volumes of each phase that cross each boundary of the well region block.

OIL:

$$QOW(k)=DP1*OW(i,j,k)$$

$$QOE(k)=DP2*OE(i,j,k)$$

$$QOS(k)=DP3*OS(i,j,k)$$

$$QON(k)=DP4*ON(i,j,k)$$

$$QOT(K)=DP5*OT(i,j,k)$$

$$QOB(K)=DP6*OB(i,j,k)$$

Water:

$$QWW(K)=DP1*WW(i,j,k)$$

$$QWE(k)=DP2*WE(i,j,k)$$

$$QWS(K)=DP3*WS(i,j,k)$$

$$QWN(k)=DP4*WN(i,j,k)$$

$$QWT(k)=DP5*WT(i,j,k)$$

$$QWB(k)=DP6*WB(i,j,k)$$

Free Gas:

$$QDW(k)=DP1*GW(i,j,k)$$

$$QGE(k)=DP2*GE(i,j,k)$$

$$QGS(k)=DP3*GS(i,j,k)$$

$$QGN(k)=DP4*GN(i,j,k)$$

$$QGT(k)=DP5*GT(i,j,k)$$

$$QGB(k)=DP6*GB(i,j,k)$$

The inner boundary condition in the wellbore is implemented by the total fluid flow (oil, water and gas) being equally distributed in each node at the wellbore. That is, the total fluid withdrawn during a single time step is divided by the number of nodes in the wellbore and this quantity is then removed from each node. The total fluid flow is specified in the BOAST input file. It is controlled by the well production conditions.

The flowchart of FEM model with the interface to BOAST that obtains these volumetric amounts is presented :

Figure 3

The reason that we reduced the time step in the FEM part was because of the highly non-linear effect of fluid flow in the well region.

3. A REAL TIME DISTRIBUTED GRAPHICS SYSTEM

3.1 SYSTEM DESCRIPTION

Traditionally, the output of a simulator is saved in a disk file for further analysis. It is both tedious and time intensive to read and understand the output text file. Another limitation of the existing BOAST reservoir simulator is that the results are available after the entire simulation is over. But the whole simulation may take a few hours or even a few days, depending on the size and accuracy of the real problem.

To better understand and subsequently use the simulation results, we propose a real time distributed graphics system. This system will visualize the output data even while the simulation is running. Using the new computing technology of X-Windows, Motif and a TCP/IP network, a real time graphics system has been implemented using both an IBM AIX RS 6000 and a DEC Alpha workstation (UNIX). The UNIX 4.3 BSD socket was used to transfer data between the two computer systems, that are used in the simulation, one for the simulation and the other for the real time graphics system.

This graphics system provides a great flexibility for the petroleum engineer to visualize the simulation results. This graphics interface is easy to

understand. The user can use as many of the window widgets as desired, such as pulldown menus and pushdown buttons to make selections, and plot the desired graph in the real time.

The main software used in the design and implementation of this real time graphics system were:

1. Graphics routines from X-Windows (Xlib),
2. User interface design (Motif 1.2),
3. and Network Communications (4.3 BSD Sockets).

3.2. WINDOW INTERFACE/NETWORK

3.2.1 X Window

The X Window System is a hardware and operating system-independent windowing system. It was developed jointly by MIT and Digital Equipment Corporation, and has been adopted by the computer industry as a standard for graphics applications.

Like other windowing systems, the X-Window System divides the screen into multiple input and output areas called windows. The X-Window System takes user input from a pointer, usually by clicking a mouse. This method of using programs is often easier to learn than traditional keyboard control because it is more intuitive.

Programming a graphically-based window system is fundamentally different from standard procedural programming. In traditional character-based interfaces, once the application starts execution, it is always in control. The executing application knows what kind of input will it allow, and may define exclusive modes to limit that input. Only "keyboard" input is expected. In a windowing system, by contrast, multiple graphic applications may be running simultaneously. In addition to the "keyboard" input, the user can use the pointer to select data. Window applications are almost modeless, that

is, the user can suddenly switch from the keyboard to the mouse, or from one application area to another.

Event-driven window programming reduces exclusive modes to a minimum , so that the user doesn't need to navigate a deep menu structure and can perform any action at any time. The user, not the application in execution, is in control. The application simply performs some setup tasks and then goes into a loop from which application functions may be invoked in any order as events arrive from the user interactively.

3.2.2 Xt/Motif

The Xt Intrinsics are based upon Xlib, the lowest-level C-Language interface to X windows. Xlib provides full access to the capabilities of the X Protocol, but does little to make programming easier.

It is quite difficult to build applications that have a graphics user interface using a low level programming library such as Xlib. To simplify development, Xt Intrinsics is designed to contain each of the user-interface elements of a graphical application such as scrollbars, command buttons, dialog boxes, popup or pulldown menus. The purpose of Xt Intrinsics is to provide an object-oriented layer that supports the user-interface abstraction called widget. A widget is a reusable, configurable piece of code that operates independently from the application.

Motif is based on Xt Intrinsics. Motif is a set of guidelines that specify how a user interface of graphical computers could "look and feel." This term, "look and feel", describes how an application appears on the screen (the look) and how the user interacts with it (the feel).

3.2.3 TCP/IP and Socket

3.2.3.1 TCP/IP - the Internet Protocols

The Advanced Research Projects Agency (APRA) of the Department of Defense (DoD) sponsored the development of the APRANET during the late 1960s and the early 1970s. The ARPANET included military, university, and research sites, and was used to support computer science and military research projects. Although the accurate name for this family of protocols is the "DARPA Internet protocol suite," it is commonly referred to as TCP/IP protocol suite, or just TCP/IP.

TCP (Transmission Control Protocol) is a connection-oriented protocol that provides a reliable, full-duplex, byte stream for the user process. It is used by most Internet application programs such FTP and TELNET protocols. IP (Internet Protocol) is the protocol that provides the packet delivery service for TCP. Since TCP uses IP, the entire Internet protocol suite is often called the TCP/IP protocol family.

There are several interesting properties for TCP/IP:

1. It is not vendor-specific.
2. It has been implemented on almost every computer from personal computers to the largest supercomputers.

3. It is used for both LANs and WANs.

4. It is widely used by many different government agencies, Universities and commercial sites today and not just DARPA-funded research projects.

One reason for the increased use of the TCP/IP protocols during the 1980s was their inclusion in the BSD Unix system around 1982. This, along with the use of BSD Unix in technical workstations, allowed many organizations and university departments to establish their own LANs.

3.2.3.2 Berkeley Sockets

The API (Application Program Interface) is the interface available to an application programmer. The availability of an API depends on both the operating system and the programming language. The two most prevalent communication APIs for Unix systems are Berkeley sockets and the System V Transport Layer Interface (TLI). Both of these interfaces were developed for the C language.

The new release of 4.3 BSD Unix socket supports communication protocols in the Unix domain and Internet domain (TCP/IP). Unix domain sockets provide a data communication between two processes in the same Unix machine, while Internet domain sockets provide data communication between two processes on physically different machines.

3.3. SYSTEM REQUIREMENTS

Before the design and implementation of a system, one needs to study the requirements of the user. Since the user probably doesn't know much about our programming language, they are probably not interested in how the functions associated with a window are implemented. What they usually want is a simple and working graphics interface.

The user may specify many of the function requirements. In this system they are :

1. The interface system should be easily understood. It should provide a certain kind of on-line help.
2. The system should provide a set of window widgets such as the pull-down menu and the pushdown button to assist the user's selection.
3. The system should be able to plot 2D graphics and 3D color images to visualize the data.
4. The system should be real time and support network connection.

3.4. DESIGN

There were three main issues related to the design of this system. Xlib is used to provide the basic draw routines. An internal data structure is built to retrieve data from an input file or socket. If the input is from a socket, the window system should receive the data as soon as it arrives from the network and then do the required computing.

3.4.1 Xlib plotting routines

Xlib provides a set of plotting functions useful in implementing this system, the functions we used from Xlib are XDrawRectangle, XDrawLine, XDrawImageString and XDrawPoint.

The descriptions of these functions follow:

1. XDrawRectangle(Display *display, Drawable drawable, GC gc, int x, int y, unsigned width, unsigned height). Display is associated with the physical device to display the plots. Drawable is an object to make plots on. GC is defined as graphics context, which is a data structure specifying the setting up of a window object, for example foreground color and background color. XDrawRectangle draws the outline of a rectangle by using ordinate and abscissa as coordinates.

2. XDrawLine(Display *display, Drawable drawable, GC gc, int x, int

y, unsined width, unsigned height). XDrawLine draws a line between two points on the screen.

3. XDrawString(Display *display, Drawable drawable,GC gc, int x, int y,char *string, int length). XDrawString draws a string on the screen.

4. XDrawPoint(Display *display, Drawable drawable,GC gc, int x, int y). XDrawPoint draws a pixel on the screen. The color of the pixel can be specified by the value of gc.

It is interesting that any complicated computer presentation graphics can be developed based on these four basic routines.

3.4.2 System Input

This system accepts input data from three different sources. These input data sources are: 1) a disk file, 2) the Unix domain socket, 3) and the TCP/IP socket. Each of these input sources are virtual files since Unix treats its input and output devices as files. All of these input sources need to be opened as a file descriptor before reading or writing to them. The code to handle the reading from or writing to the file descriptor is the same.

In order to capture the event that the data from socket is ready, the concept of event handler in X windows is used. `XtAppAddInput()` can be used to add an event handler to a window object. It registers a procedure to be called when input is available from a file or socket. The registered procedure will perform the reading and plot in the window.

The input file records the information about the three variable distributions in the real space associated with coordinates x, y, z over time. In the C program, a data structure is written in a four dimension array and has the following format:

```
typedef struct {  
    float variable[nθ][nr][nz][3];  
    float day;
```

... other variables ...

} DATA

A dynamic array of the type DATA is allocated whenever the window receives data from the input. The C function call for a dynamic allocation in Unix is `realloc()` and this is used in this system.

3.4.3 Network Communications

This system uses a client/server model to transfer data between two processes in the network. This graphics system acts as a server on a Dec Alpha Station 200 (233MHZ). The reservoir simulator runs as a client on an IBM RS 6000 workstation located in another building. The Dec Alpha Station 200 and the IBM RS6000 communicates through the local Ethernet. 4.3 BSD socket is used for the data and communication control.

The scenario of the system call of the socket is presented in in Figure 4:

Figure 4

3.5. PROTOTYPE USER'S MANUAL

There are three main windows in this system. This section describes each.

3.5.1 Main control window

Figure 5

This Real Time Graphics System has File, View Data, Control and Help in its pulldown menu. The explanation for each of these follows:

1. File has "quit" submenu that can terminate the program.
2. View Data has "First Step", "Next Step" and "Prev Step" in its submenu. It can be used to control the time of the plots.
3. Control has the "Step Size" submenu. It controls the time interval for the real time plots.
4. Help has the "Help" submenu. It pops up a dialog window which contains a help manual and a graphics illustration of the problem.

3.5.2 Variable's space distribution window

Figure 6

In Figure 6, a sample window plot is presented and the variables that are plotted are pressure vs distance. This was plotted at a simulation time of

0.8 day. It is named the pressure distribution. This window has File, Graph, Image and Data for pulldown menus.

Each of these icons for the pressure distribution has additional features.

These are:

1. File has a "save" submenu. This saves the graphics in the window as a GIF file for future reference as shown in Figure 7.

Figure 7

2. Graph has "by Radical " and "Direction" submenus. They provide two

different ways of visualizing the data. One can choose to plot the variable's distribution against either radial distance or linear direction.

3. Image has the "Image" submenu. This can be used to produce a color image of the distribution over a two dimensional plane.

4. Data has the "Data" submenu. It "prints" out the value of the variable as shown in Figure 8.

Distance from well (ft)	Pressure (psi)
0.250	3618.78
100.0	4274.35
200.0	4472.65
300.0	4574.41
400.0	4625.04

Figure 8

Besides the pulldown menus, the window presented in Figure 9 also has

five pushdown buttons implemented. The two on the left provide a layer selection. The three on the right provide a selection of the different kinds of variables that can be chosen to visualize.

3.5.3 Variable's change over time

Figure 9

The above window is similar to the previous window except that in this plot the abscissa is time. The variable's real time behavior is displayed in this window.

4. TEST RESULTS

Two sets of test data for our FEM well block model were selected by Dr. Ming-Ming Chang, senior research engineer from NIPER. Each of these two sets of data has been used in testing our simulation system. Both sets of test results obtained were consistent with the fundamental principles of Petroleum Engineering and exhibited expected fluid flow characteristics.

4.1 Test Problem 1

Test Problem 1 is shown in Figure 10. It has a single oil production well located at block (6,5,1) in a reservoir. Initially, there were two phases (oil and water) present in the reservoir. All the gas was dissolved in the oil phase. The water phase was immobile since its concentration was always below the critical saturation of water present to cause mobility. When the oil pressure decreased as oil was being produced, the gas came out of the oil phase (this occurred at the bubble point) and this produces free gas. We expected the oil pressure around the well bore in the well block to drop below bubble point and to observe an increase in gas saturation.

This was in fact, what our calculation using the output data for test problem 1, produced.

Figure 10

Our reservoir simulation calculated the oil pressure, water saturation and gas saturation implicitly in our FEM grid over the "6,5,1" well block over time. Some example results from this calculation are shown in the following figures.

Figure 11 shows the pressure distribution versus distance from the well at day 20 in our simulation. Our simulation was started at day 0. The upper line is the pressure at the bottom plane of the well block at each distance from the well bore. The gravity effect is pronounced, the pressure at the bottom is more than the pressure at the top. This pressure difference is due completely to the force of gravity on the fluid phases.

Figure 12 shows the gas saturation's distribution also at day 20. The upper line is the gas saturation at the top of the well block. The lower line is the gas saturation at the bottom of the well block. Since gas is less dense, with respect to oil, it tends to gather at the higher plane. The distance is measured out from the wellbore. As seen in Figure 12, the gas saturations decreases as the distance from the wellbore increases. This is the expected result since the reservoir pressure is increasing as the distance from the wellbore increases.

Figure 13 shows the oil saturation versus distance from the wellbore also at day 20. The upper line is the bottom plane's oil saturation and the top plane is the lower line. This was the expected result.

Figure 14 shows the oil pressure versus distance from the wellbore at

the top plane. The results became unstable at day 24 and the pressure was slightly different vs distance from the wellbore depending upon which grids were traversed from the wellbore.

Figure 15 shows the gas saturation versus distance from the wellbore at the top plane at day 24. The unstable result is very apparent here. All of the gas saturations should be congruent but they are not. The gas saturation was not well behaved and this will be discussed on the next page.

Figure 16 shows the gas saturation distribution vs. distance from the wellbore at the bottom plane at day 24. These vary depending upon the radius traversed from the wellbore.

As shown in Figure 15 and 16, the gas saturation in our solution has an instability. This may be due to two possible reasons (1) the FEM stiffness matrix may be ill-conditioned and not stable or (2) the boundary condition at the wellbore may not be correctly stated and implemented. The boundary condition being addressed in another research project. The final discretized FEM equation is a first order difference equation with respect to time. For first order linear system, the solution is stable if all of the eigenvalues are less than 1. The eigenvalues of the final FEM stiffness matrix are less than

1. So we believe that the boundary condition at the wellbore is the problem because we only know the total fluid flow being removed from the well and not how it is distributed among the nodes at the wellbore.

4.2 Test Problem 2

Figure 17

The reservoir mesh for this problem is $10 \times 10 \times 3$. It has 3 layers, and a grid that has 100 blocks that are 1000 feet on each side. The model has a gas injection well on one corner and an oil production well at the opposite corner as depicted in figure 17 (A). The gas is injected into the top layer of the 3 layers and the oil is to be produced from the lowest layer of the 3 layers as shown in figure 17 (B).

The injected gas drives the oil from the reservoir into the oil production well. The oil well in Figure 17 (A) is located in block (10,10,3). This oil well was completed in only one layer and this was the lowest of the 3 layers.

Figure 18 shows the change of oil pressure in the wellbore over time. It also shows a graphics presentation that was developed to present these results as a part of this research.

Figure 18

Figure 19 shows the oil pressure distribution vs distance from the wellbore at 0.8 day for test problem 2.

Figure 19

The instability that occurred with test data 1 also occurred with test data 2. The reason for this is believed to be the same as discussed in Test Problem One.

5. CONCLUSIONS

This finite element reservoir simulation system provides an accurate solution in the well block region for general three dimension and three phase problems in reservoir engineering. Its formulation and coding have been extensively tested on an IBM RS 6000 Unix workstation. This simulation is consistent with the fundamental principles of Petroleum Engineering and fluid flow in a porous media.

The limitation of this system is that it exhibits an instability during the simulation. We believe that this instability is caused after the breakthrough of a second phase and how this is to be distributed at the inner wellbore boundary. Therefore additional knowledge is required from experimental reservoir engineering and research before this system can be completed and used in actual field conditions. For certain well test problems known, this system is a working tool .

Although this system has limitations, it can still provide an accurate solution to many reservoir well simulations in their initial stages. Most reservoir simulators are not accurate in their initial solution stages.

A real time distributed graphics system has been developed to be used

with this simulation system. This graphics system is able to visualize the simulation results while the simulation is running. It is intended to reduce the cycle of development in a well block region simulator.

Three Phase 3-D Finite Element Method in Reservoir Simulation

The flow of fluid into and out of a single reservoir cube:

Assume fluid flows into this cube at $x(\tau_x)$ and out of the cube at $x+\Delta x (\tau_{x+\Delta x})$. τ denotes the fluid flux and is defined as the rate of flow of mass per unit cross-sectional area normal to the direction of flow. Conservation of mass yields:

1.2 Flow Equations

A flux in a given direction can be written as the density of the fluid times its velocity in that given direction:

$$(t)_o = \frac{\rho_o \vec{V}_o}{B_o} \quad (1.2.1)$$

$$(t)_w = \frac{\rho_w \vec{V}_w}{B_w} \quad (1.2.2)$$

$$(t)_g = \frac{\rho_g \vec{V}_g}{B_g} + \frac{R_{so} \rho_g \vec{V}_o}{B_o} + \frac{R_{sw} \rho_g \vec{V}_w}{B_w} \quad (1.2.3)$$

where R_{so} and R_{sw} are gas solubilities in SCF/STB. B_o , B_w and B_g are formation volume factors in reservoir volume/standard volume and the ρ 's are fluid densities at standard conditions (60°F and 14.7 psia).

The velocities are assumed to be Darcy velocities and their X-components are:

$$(v_o)_x = -K_x \lambda_o \frac{\partial}{\partial x} \left[P_o - \frac{\rho'_o g z}{144 g_c} \right] \quad (1.2.4)$$

$$(v_w)_x = -K_x \lambda_w \frac{\partial}{\partial x} \left[P_w - \frac{\rho'_w g z}{144 g_c} \right] \quad (1.2.5)$$

$$(v_g)_x = -K_x \lambda_g \frac{\partial}{\partial x} \left[P_g - \frac{\rho'_g g z}{144 g_c} \right] \quad (1.2.6)$$

Assume $g = g_c$

Similar expressions can be written for the y and z components. Each phase mobility λ_p is defined as the ratio of the

relative permeability to flow of the phase divided by its viscosity, $\lambda_p = \frac{K_{rp}}{\mu_p}$.

The phase densities are related to the formation volume factors and gas solubilities by

$$\rho'_o = \frac{1}{B_o} [P_o + R_{so} \rho_g] \quad (1.2.7)$$

$$\rho'_w = \frac{1}{B_w} [\rho_w + R_{sw} \rho_g] \quad (1.2.8)$$

$$\rho'_g = \frac{\rho_g}{B_g} \quad (1.2.9)$$

and the concentrations are given by

$$C_o = \frac{\varphi \rho_o S_o}{B_o} \quad (1.2.10)$$

$$C_w = \frac{\varphi \rho_w S_w}{B_w} \quad (1.2.11)$$

$$C_g = \varphi \rho_g \left[\frac{S_g}{B_g} + \frac{R_{so} S_o}{B_o} + \frac{R_{sw} S_w}{B_w} \right] \quad (1.2.12)$$

where φ is the porosity and S_p is the saturation of phase P . The saturations satisfy:

$$S_o + S_w + S_g = 1 \left(\text{and } 0 \leq S_o, S_w, S_g \leq 1 \right) \quad (1.2.13)$$

combining these equations¹:

Oil

$$-\left[\frac{\partial}{\partial x} \left(\frac{\rho_o}{B_o} V_{xo} \right) + \frac{\partial}{\partial y} \left(\frac{\rho_o}{B_o} V_{yo} \right) + \frac{\partial}{\partial z} \left(\frac{\rho_o}{B_o} V_{zo} \right) \right] - q_o = \frac{\partial}{\partial t} \left(\frac{\varphi \rho_o S_o}{B_o} \right) \quad (1.2.14)$$

Water

$$-\left[\frac{\partial}{\partial x} \left(\frac{\rho_w}{B_w} V_{xw} \right) + \frac{\partial}{\partial y} \left(\frac{\rho_w}{B_w} V_{yw} \right) + \frac{\partial}{\partial z} \left(\frac{\rho_w}{B_w} V_{zw} \right) \right] - q_w = \frac{\partial}{\partial t} \left(\frac{\varphi \rho_w S_w}{B_w} \right) \quad (1.2.15)$$

Gas

1. All densities are at standard conditions.

$$\begin{aligned}
& -\frac{\partial}{\partial x} \left[\frac{\rho_g}{B_g} V_{xg} + \frac{R_{so} \rho_g}{B_o} V_{xo} + \frac{R_{sw} \rho_g}{B_w} V_{xw} \right] - \frac{\partial}{\partial y} \left[\frac{\rho_g}{B_g} V_{yg} + \frac{R_{so} \rho_g}{B_o} V_{yo} + \frac{R_{sw} \rho_g}{B_w} V_{yw} \right] \\
& - \frac{\partial}{\partial z} \left[\frac{\rho_g}{B_g} V_{zg} + \frac{R_{so} \rho_g}{B_o} V_{zo} + \frac{R_{sw} \rho_g}{B_w} V_{zw} \right] - q_g = -\frac{\partial}{\partial t} \left[\phi \rho_g \left(\frac{S_g}{B_g} + \frac{R_{so} S_o}{B_o} + \frac{R_{sw} S_w}{B_w} \right) \right]
\end{aligned} \tag{1.2.16}$$

The densities at standard conditions are constants, and these equations can be reduced as shown below:

Oil:

$$-\left[\frac{\partial}{\partial x} \left(\frac{1}{B_o} V_{xo} \right) + \frac{\partial}{\partial y} \left(\frac{1}{B_o} V_{yo} \right) + \frac{\partial}{\partial z} \left(\frac{1}{B_o} V_{zo} \right) \right] - \frac{q_o}{\rho_o} = \frac{\partial}{\partial t} \left(\frac{\phi S_o}{B_o} \right) \tag{1.2.17}$$

Water:

$$-\left[\frac{\partial}{\partial x} \left(\frac{1}{B_w} V_{xw} \right) + \frac{\partial}{\partial y} \left(\frac{1}{B_w} V_{yw} \right) + \frac{\partial}{\partial z} \left(\frac{1}{B_w} V_{zw} \right) \right] - \frac{q_w}{\rho_w} = \frac{\partial}{\partial t} \left(\frac{\phi S_w}{B_w} \right) \tag{1.2.18}$$

Gas:

$$\begin{aligned}
& -\frac{\partial}{\partial x} \left(\frac{V_{xg}}{B_g} + \frac{R_{so} V_{xo}}{B_o} + \frac{R_{sw} V_{xw}}{B_w} \right) - \frac{\partial}{\partial y} \left(\frac{V_{yg}}{B_g} + \frac{R_{so} V_{yo}}{B_o} + \frac{R_{sw} V_{yw}}{B_w} \right) - \frac{\partial}{\partial z} \left(\frac{V_{zg}}{B_g} + \frac{R_{so} V_{zo}}{B_o} + \frac{R_{sw} V_{zw}}{B_w} \right) - \frac{q_g}{\rho_g} \\
& = \frac{\partial}{\partial t} \left[\phi \left(\frac{S_g}{B_g} + \frac{R_{so} S_o}{B_o} + \frac{R_{sw} S_w}{B_w} \right) \right]
\end{aligned} \tag{1.2.19}$$

These equations can be written as:

$$-\nabla \cdot \frac{\vec{v}_o}{B_o} - \frac{q_o}{\rho_o} = \frac{\partial}{\partial t} \left(\frac{\phi S_o}{B_o} \right) \tag{1.2.20}$$

$$-\nabla \cdot \frac{\vec{v}_w}{B_w} - \frac{q_w}{\rho_w} = \frac{\partial}{\partial t} \left(\frac{\varphi S_w}{B_w} \right) \quad (1.2.21)$$

$$-\nabla \cdot \left\{ \frac{\vec{v}_g}{B_g} + \frac{R_{so}}{B_o} \vec{v}_o + \frac{R_{sw}}{B_w} \vec{v}_w \right\} - \frac{q_g}{\rho_g} = \frac{\partial}{\partial t} \left(\varphi \left[\frac{S_g}{B_g} + \frac{R_{so} S_o}{B_o} + \frac{R_{sw} S_w}{B_w} \right] \right) \quad (1.2.22)$$

where

$$\nabla \cdot \vec{v} = \frac{\partial}{\partial x} v_x + \frac{\partial}{\partial y} v_y + \frac{\partial}{\partial z} v_z \quad (1.2.23)$$

Define the potential¹ Ψ_p of phase p as:

$$\Psi_p = P_p - \frac{\rho'_p Z}{144} \quad (1.2.24)$$

Also introduce the dyadic notation $\hat{M}_o, \hat{M}_w$ and $\hat{M}_g$ to signify that mobility is a tensor of rank two.

$\hat{M}_o = \frac{1}{\mu_o} (\vec{k} \vec{k}_{ro})$ where $(\vec{k} \vec{k}_{ro})$ is the permeability matrix whose terms are given individually by the product $k_{ijk} k_{ro}$, with k_{ijk} the local permeability and k_{ro} the permeability to oil, and μ_o is the oil viscosity.

1.3 Finite Elements

A finite element method will be used to obtain simultaneous solution of the partial differential equations for the flow of oil, water, and gas. The following fluids were considered to be compressible and immiscible which justifies the treatment of relative permeability differences, complex fluid properties, and the effects of capillary pressure.

The continuity equation for each flowing phase can be found in "Modern Reservoir Engineering - A Simulation Approach" by H.B. Crichtlow, Prentice Hall, Englewood Cliffs, NJ (1977) or "Petroleum Reservoir Simulation" by K. Aziz and A. Settari, Applied Science Publications, New York (1979). Using Darcy's flow law with these continuity

1. In BOAST, the density is in $lb(mass)/ft^3$ and the division by 144 converts this into psi. A density of $1gm/cm^3 = 62.4lb/ft^3$.

equations the following expressions can be applied for each flowing phase at standard conditions, the formula that follows was taken from "A Finite Element Solution of a Fully Coupled Implicit Formulation for Reservoir Simulation," by Y. Sukirman and R.W. Lewis, International Journal for Numerical and Analytical Methods in Geomechanics, Vol. 17, 677-698 (1993).

$$-\Delta(\hat{M}_l \Delta(P_l + \rho_l g h)) + (\text{rate of fluid accumulation}) = Q_{\text{net}} \quad (1.3.1)$$

$$\hat{M}_l = \frac{K k_{rl}}{\mu_l B_l} \quad (1.3.2)$$

$$l = o, g, w \quad (1.3.3)$$

where P_l is the pressure, ρ_l is the fluid density, B_l is the formation volume factor, K the absolute permeability matrix, μ_l is the dynamic fluid viscosity and h is the height above an arbitrary datum. The subscripts o , g , and w refer to oil, gas, and water phases, respectively.

$$\hat{M}_w = \frac{1}{\mu_w} (\vec{K} K_{rw}) \quad (1.3.4)$$

$$\hat{M}_g = \frac{1}{\mu_g} (\vec{K} K_{rg}) \quad (1.3.5)$$

let

$$\hat{\lambda}_o = \frac{\hat{M}_o}{B_o} = \frac{K_o}{B_o \mu_o} \quad (1.3.6)$$

$$\hat{\lambda}_w = \frac{\hat{M}_w}{B_w} = \frac{K_w}{B_w \mu_w} \quad (1.3.7)$$

$$\hat{\lambda}_g = \frac{\hat{M}_g}{B_g} = \frac{K_g}{B_g \mu_g} \quad (1.3.8)$$

then

$$\hat{\lambda}_o = -\hat{M}_o \cdot \nabla \Psi_o \quad (1.3.9)$$

$$\hat{\lambda}_w = -\hat{M}_w \cdot \nabla \Psi_w \quad (1.3.10)$$

$$\vec{V}_g = -M_g \cdot \nabla \Psi_g \quad (1.3.11)$$

and,

$$\nabla \cdot (\hat{\lambda}_o \cdot \nabla \Psi_o) - \frac{q_o}{\rho_o} = \frac{\partial}{\partial t} \left(\frac{\Phi S_o}{B_o} \right) \quad (1.3.12)$$

$$\nabla \cdot (\hat{\lambda}_w \cdot \nabla \Psi_w) - \frac{q_w}{\rho_w} = \frac{\partial}{\partial t} \left(\frac{\Phi S_w}{B_w} \right) \quad (1.3.13)$$

$$\nabla \cdot [\hat{\lambda}_g \cdot \nabla \Psi_g + R_{so} \hat{\lambda}_o \cdot \nabla \Psi_o + R_{sw} \hat{\lambda}_w \cdot \nabla \Psi_w] - \frac{q_g}{\rho_g} = \frac{\partial}{\partial t} \left[\Phi \left(\frac{S_g}{B_g} + \frac{R_{so} S_o}{B_o} + \frac{R_{sw} S_w}{B_w} \right) \right] \quad (1.3.14)$$

can be found in Peaceman (1977) equations (1-121) through (1-123). Equivalent Ψ_o, Ψ_w, Ψ_g are the phase potentials, to simplify the handling of these the phase pressures (P_o, P_w, P_g) can be related as capillary pressures:

$$P_{cow} = P_o - P_w \quad (1.3.15)$$

$$P_{cgo} = P_g - P_o \quad (1.3.16)$$

These differences P_{cow} (capillary pressure of oil-to-water) and P_{cgo} (capillary pressure of gas-to-oil). Experimentally P_{cow} and P_{cgo} can be measured and are assumed to be functions of water and gas saturations only. Using these equations, the water and gas phase potentials are:

$$\Psi_w = P_o - P_{cow} - \frac{\rho_w' Z}{144} \quad (1.3.17)$$

$$\Psi_g = P_o - P_{cgo} - \frac{\rho_g' Z}{144} \quad (1.3.18)$$

combining these equations with our previous equations:

Oil:

$$\nabla \cdot (\hat{\lambda}_o \cdot \nabla P_o) - \nabla \cdot \left(\hat{\lambda}_o \cdot \nabla \left(\frac{\rho_o' Z}{144} \right) \right) - \frac{q_o}{\rho_o} = \frac{\partial}{\partial t} \left(\frac{\Phi S_o}{B_o} \right) \quad (1.3.19)$$

Water:

$$\nabla \cdot (\hat{\lambda}_w \cdot \nabla P_o) - \nabla \cdot \left(\hat{\lambda}_w \cdot \nabla \left(\frac{\rho'_w Z}{144} + P_{cow} \right) \right) - \frac{q_w}{\rho_w} = \frac{\partial}{\partial t} \left(\frac{S_w}{B_w} \right) \quad (1.3.20)$$

Gas:

$$\begin{aligned} \nabla \cdot \left[\hat{\lambda}_g \cdot \nabla \left(P_o + P_{cgo} - \frac{\rho'_g Z}{144} \right) + R_{so} \hat{\lambda}_o \cdot \nabla \left(P_o - \frac{\rho'_o Z}{144} \right) \right. \\ \left. + R_{sw} \hat{\lambda}_w \cdot \nabla \left(P_o - P_{cow} - \frac{\rho'_w Z}{144} \right) \right] - \frac{q_g}{\rho_g} = \frac{\partial}{\partial t} \left[\frac{S_g}{B_g} + \frac{R_{so} S_o}{B_o} + \frac{R_{sw} S_w}{B_w} \right] \end{aligned} \quad (1.3.21)$$

$$S_o = 1 - S_w - S_g \quad (1.3.22)$$

$$\frac{dS_o}{dt} = -\frac{dS_w}{dt} - \frac{dS_g}{dt} \quad (1.3.23)$$

$$\frac{\partial}{\partial t} \left(\frac{\varphi S_o}{B_o} \right) = \frac{\varphi}{B_o} \cdot \frac{\partial S_o}{\partial t} + \frac{S_o}{B_o} \cdot \frac{\partial \varphi}{\partial t} + \varphi S_o \left(-\frac{1}{B_o^2} \cdot \frac{\partial B_o}{\partial t} \right) \quad (1.3.24)$$

$$\frac{\partial \varphi}{\partial t} = \frac{d\varphi}{dP_o} \cdot \frac{\partial P_o}{\partial t} \quad (1.3.25)$$

$$\frac{\partial B_o}{\partial t} = \frac{dB_o}{dP_o} \cdot \frac{\partial P_o}{\partial t} \quad (1.3.26)$$

so that,

$$\frac{\partial}{\partial t} \left(\frac{\varphi S_o}{B_o} \right) = \frac{\varphi}{B_o} \cdot \frac{\partial S_o}{\partial t} + \frac{S_o}{B_o} \left(\frac{d\varphi}{dP_o} \right) \frac{\partial P_o}{\partial t} - \frac{\varphi S_o}{B_o^2} \left(\frac{dB_o}{dP_o} \right) \frac{\partial P_o}{\partial t} = \frac{\varphi}{B_o} \cdot \frac{\partial S_o}{\partial t} + \frac{\varphi S_o}{B_o} \left[\frac{1}{\varphi} \frac{d\varphi}{dP_o} - \frac{1}{B_o} \frac{dB_o}{dP_o} \right] \frac{\partial P_o}{\partial t} \quad (1.3.27)$$

$$= \frac{\varphi}{B_o} \left[\frac{\partial S_o}{\partial t} + S_o \left(\frac{1}{\varphi} \frac{d\varphi}{dP_o} - \frac{1}{B_o} \frac{dB_o}{dP_o} \right) \frac{\partial P_o}{\partial t} \right] \quad (1.3.28)$$

$$\frac{\partial}{\partial t} \left(\frac{\varphi S_w}{B_w} \right) = \frac{\varphi}{B_w} \cdot \frac{\partial S_w}{\partial t} + \frac{S_w}{B_w} \frac{\partial \varphi}{\partial t} - \frac{\varphi S_w}{B_w^2} \cdot \frac{\partial B_w}{\partial t} \quad (1.3.29)$$

$$\frac{\partial \varphi}{\partial t} = \frac{d\varphi}{dP_o} \cdot \frac{\partial P_o}{\partial t} \quad (1.3.30)$$

$$\frac{\partial B_w}{\partial t} = \frac{dB_w}{dP_o} \cdot \frac{\partial P_o}{\partial t} \quad (1.3.31)$$

$$\frac{\partial}{\partial t} \left(\frac{\varphi S_w}{B_w} \right) = \frac{\varphi}{B_w} \cdot \frac{\partial S_w}{\partial t} + \frac{S_w}{B_w} \left[\frac{\partial \varphi}{\partial t} \right] \frac{\partial P_o}{\partial t} \frac{\varphi S_w}{B_w^2} \cdot \left[\frac{dB_w}{dt} \right] \frac{\partial P_o}{\partial t} = \frac{\varphi}{B_w} \left[\frac{\partial S_w}{\partial t} + S_w \left(\frac{1}{\varphi} \frac{d\varphi}{dP_o} - \frac{1}{B_w} \frac{dB_w}{dP_o} \right) \frac{\partial P_o}{\partial t} \right] \quad (1.3.32)$$

$$\frac{\partial}{\partial t} \left[\varphi \left(\frac{S_g}{B_g} + \frac{R_{so} S_o}{B_o} + \frac{R_{sw} S_w}{B_w} \right) \right] = L_1 + L_2 + L_3 \quad (1.3.33)$$

$$L_1 = \frac{\partial}{\partial t} \left(\frac{\varphi S_g}{B_g} \right) = \frac{S_g}{B_g} \frac{\partial \varphi}{\partial t} + \frac{\varphi}{B_g} \frac{\partial S_g}{\partial t} - \frac{\varphi S_g}{B_g^2} \frac{\partial B_g}{\partial t} \quad (1.3.34)$$

$$\frac{\partial \varphi}{\partial t} = \frac{d\varphi}{dP_o} \frac{\partial P_o}{\partial t} \quad (1.3.35)$$

$$\frac{\partial B_g}{\partial t} = \frac{dB_g}{dP_o} \frac{\partial P_o}{\partial t} \quad (1.3.36)$$

$$L_1 = \frac{S_g}{B_g} \left[\frac{d\varphi}{dP_o} \right] \frac{\partial P_o}{\partial t} + \frac{\varphi}{B_g} \frac{\partial S_g}{\partial t} - \frac{\varphi S_g}{B_g^2} \left[\frac{dB_g}{dP_o} \right] \frac{\partial P_o}{\partial t} \quad (1.3.37)$$

$$L_1 = \frac{\partial S_g}{\partial t} + \frac{\varphi S_g}{B_g} \left[\frac{1}{\varphi} \frac{d\varphi}{dP_o} - \frac{1}{B_g} \frac{dB_g}{dP_o} \right] \frac{\partial P_o}{\partial t} \quad (1.3.38)$$

$$L_2 = \frac{\partial}{\partial t} \left[\frac{\varphi R_{so} S_o}{B_o} \right] \quad (1.3.39)$$

$$L_2 = \frac{\partial R_{so} \varphi S_o}{\partial t B_o} + R_{so} \frac{\partial}{\partial t} \left(\frac{\varphi S_o}{B_o} \right) \quad (1.3.40)$$

$$= \frac{dR_{so}}{dP_o} \left(\frac{\varphi S_o}{B_o} \right) \frac{dP_o}{dt} + R_{so} \frac{\partial}{\partial t} \left(\frac{\varphi S_o}{B_o} \right) \quad (1.3.41)$$

$$L_3 = \frac{\partial}{\partial t} \left[\frac{\varphi R_{sw} S_w}{B_w} \right] = \frac{dR_{sw}}{dP_o} \frac{\varphi S_w}{B_w} \frac{dP_o}{dt} + R_{sw} \frac{\partial}{\partial t} \left(\frac{\varphi S_w}{B_w} \right) \quad (1.3.42)$$

$$\frac{\partial \varphi}{\partial t} = \frac{d\varphi}{dP_o} \cdot \frac{\partial P_o}{\partial t} \quad (1.3.43)$$

$$L_1 + L_2 + L_3 = \frac{\varphi}{B_g} \frac{\partial S_g}{\partial t} + \frac{\varphi S_g}{B_g} \left(\frac{1}{\varphi} \frac{d\varphi}{dP_o} - \frac{1}{B_g} \frac{dB_g}{dP_o} \right) \frac{\partial P_o}{\partial t} + \frac{dR_{so}}{dP_o} \frac{\varphi S_o}{B_o} \frac{\partial P_o}{\partial t} + \frac{dR_{sw}}{dP_o} \frac{\varphi S_w}{B_w} \frac{\partial P_o}{\partial t} + R_{so} \frac{\partial}{\partial t} \left(\frac{\varphi S_o}{B_o} \right) + R_{sw} \frac{\partial}{\partial t} \left(\frac{\varphi S_w}{B_w} \right) \quad (1.3.44)$$

Compressibilities of oil, water, gas, and rock:

$$C_o = -\frac{1}{B_o} \frac{\partial B_o}{\partial P_o} + \frac{B_g}{B_o} \frac{\partial R_{so}}{\partial P_o} \quad (1.3.45)$$

$$C_w = -\frac{1}{B_w} \frac{\partial B_w}{\partial P_o} + \frac{B_g}{B_w} \frac{\partial R_{sw}}{\partial P_o} \quad (1.3.46)$$

$$C_g = \frac{1}{B_g} \frac{\partial B_g}{\partial P_o} \quad (1.3.47)$$

$$C_r = \frac{1}{\varphi} \frac{\partial \varphi}{\partial P_o} \quad (1.3.48)$$

and the compressibility total is:

$$C_t = C_r + S_o C_o + S_w C_w + S_g C_g \quad (1.3.49)$$

Equation (1):

$$\frac{\partial}{\partial t} \left(\frac{\varphi S_o}{B_o} \right) = \frac{\varphi}{B_o} \left[\frac{\partial S_o}{\partial t} + S_o \left(\frac{1}{\varphi} \frac{\partial \varphi}{\partial P_o} - \frac{1}{B_o} \frac{dB_o}{dP_o} \right) \frac{\partial P_o}{\partial t} \right] \quad (1.3.50)$$

$$= \frac{\varphi}{B_o} \left[\frac{\partial S_o}{\partial t} \right] + \frac{\varphi S_o}{B_o} \left(C_r - \frac{1}{B_o} \frac{dB_o}{dP_o} \right) \frac{\partial P_o}{\partial t} \quad (1.3.51)$$

Equation (2):

$$\frac{\partial}{\partial t} \left(\frac{\varphi S_w}{B_w} \right) = \frac{\varphi}{B_w} \left[\frac{\partial S_w}{\partial t} + S_w \left(\frac{1}{\varphi} \frac{\partial \varphi}{\partial P_o} - \frac{1}{B_w} \frac{dB_w}{dP_o} \right) \frac{\partial P_o}{\partial t} \right] \quad (1.3.52)$$

$$= \frac{\varphi}{B_w} \left[\frac{\partial S_w}{\partial t} \right] + \frac{\varphi S_w}{B_w} \left(C_r - \frac{1}{B_w} \frac{dB_w}{dP_o} \right) \frac{\partial P_o}{\partial t} \quad (1.3.53)$$

Equation (3):

$$L_1 + L_2 + L_3 = \frac{\varphi}{B_g} \cdot \frac{\partial S_g}{\partial t} + \varphi \left[\frac{1}{B_g} \left(S_g C_r + S_g C_g \right) + \frac{S_o}{B_o} \frac{dR_{so}}{dP_o} + \frac{S_w}{B_w} \frac{dR_{sw}}{dP_o} \right] \cdot \frac{\partial P_o}{\partial t} + R_{so} \frac{\partial}{\partial t} \left(\frac{\varphi S_o}{B_o} \right) + R_{sw} \frac{\partial}{\partial t} \left(\frac{\varphi S_w}{B_w} \right) \quad (1.3.54)$$

Let,

$$C_\alpha = \frac{B_g}{B_o} \frac{dR_{so}}{dP_o} \quad (1.3.55)$$

Oil:

$$\nabla \cdot (\hat{\lambda}_o \cdot \nabla P_o) - \nabla \cdot \left(\hat{\lambda}_o \cdot \nabla \left(\frac{\rho_o' z}{144} \right) \right) - \frac{q_o}{\rho_o} = -\frac{\varphi}{B_o} \left[\frac{\partial S_w}{\partial t} + \frac{\partial S_g}{\partial t} \right] + \frac{\varphi}{B_o} \left[C_r - \frac{1}{B_o} \frac{dB_o}{dP_o} \right] \left[\frac{\partial P_o}{\partial t} - S_w \frac{\partial P_o}{\partial t} - S_g \frac{\partial P_o}{\partial t} \right] \quad (1.3.56)$$

Water:

$$\nabla \cdot (\hat{\lambda}_w \cdot \nabla P_o) - \nabla \cdot \left(\hat{\lambda}_w \cdot \nabla \left[\frac{\rho_w' z}{144} + P_{cow} \right] \right) - \frac{q_w}{\rho_w} = \frac{\varphi}{B_w} \frac{\partial S_w}{\partial t} + \frac{\varphi}{B_w} \left[C_r - \frac{1}{B_w} \frac{dB_w}{dP_o} \right] S_w \frac{\partial P_o}{\partial t} \quad (1.3.57)$$

Gas:

$$\begin{aligned} \nabla \cdot \left(\hat{\lambda}_g \cdot \nabla \left[P_o + P_{cgo} - \frac{\rho'_g Z}{144} \right] + R_{so} \hat{\lambda}_o \cdot \nabla \left[P_o - \frac{\rho'_o Z}{144} \right] + R_{sw} \hat{\lambda}_w \cdot \nabla \left[P_o - P_{cow} - \frac{\rho'_w Z}{144} \right] \right) - \frac{q_g}{\rho_g} = \\ = \frac{\Phi}{B_g} \cdot \frac{\partial S_g}{\partial t} + \Phi \left[\frac{1}{B_g} (S_g C_r + S_g C_g) + \frac{S_o dR_{so}}{B_o dP_o} + \frac{S_w dR_{sw}}{B_w dP_o} \right] \cdot \frac{\partial P_o}{\partial t} + R_{so} \frac{\partial}{\partial t} \left(\frac{\Phi S_o}{B_o} \right) + R_{sw} \frac{\partial}{\partial t} \left(\frac{\Phi S_w}{B_w} \right) \end{aligned} \quad (1.3.58)$$

Now perform the following operation: Equation (1.3.58) - R_{so} times Equation (1.3.19) - R_{sw} times Equation (1.3.20)

Then we would have:

Gas:

$$\begin{aligned} \nabla \cdot \left(\hat{\lambda}_g \cdot \nabla \left[P_o + P_{cgo} - \frac{\rho'_g Z}{144} \right] \right) + \nabla \cdot \left(R_{so} \hat{\lambda}_o \cdot \nabla \left[P_o - \frac{\rho'_o Z}{144} \right] \right) - R_{so} \nabla \cdot \left(\hat{\lambda}_o \cdot \nabla \left[P_o - \frac{\rho'_o Z}{144} \right] \right) \\ + \nabla \cdot \left(R_{sw} \hat{\lambda}_w \cdot \nabla \left[P_o - P_{cow} - \frac{\rho'_w Z}{144} \right] \right) - R_{sw} \nabla \cdot \left(\hat{\lambda}_w \cdot \nabla \left[P_o - P_{cow} - \frac{\rho'_w Z}{144} \right] \right) - \left(\frac{q_g}{\rho_g} - \frac{R_{so} q_o}{\rho_o} - \frac{R_{sw} q_w}{\rho_w} \right) \end{aligned} \quad (1.3.59)$$

$$= \frac{\Phi}{B_g} \cdot \frac{\partial S_g}{\partial t} + \Phi \left[\frac{1}{B_g} (S_g C_r + S_g C_g) + \frac{S_o dR_{so}}{B_o dP_o} + \frac{S_w dR_{sw}}{B_w dP_o} \right] \cdot \frac{\partial P_o}{\partial t} \quad (1.3.60)$$

An approach to this problem is to find an average value in each element for R_{sw} and R_{so} . Then treat these as constants in the elemental equation for gas in each element. That is to find the average pressure in the element and find the value of R_{sw} and R_{so} for this average element pressure. For 8 nodes this average pressure would be:

$$P_{ave} = \sum_{i=1}^8 P_i N_i$$

We use $\bar{R}_{so}$ as the average for R_{so} and $\bar{R}_{sw}$ as the average for R_{sw} in each element. Then the left side of the Gas Equation becomes:

$$\begin{aligned} \nabla \cdot \left(\hat{\lambda}_g \cdot \nabla P_o \right) + \nabla \cdot \left(\hat{\lambda}_g \cdot \nabla P_{cgo} \right) - \nabla \cdot \left(\hat{\lambda}_g \cdot \nabla \left(\frac{\rho'_g Z}{144} \right) \right) + \nabla \cdot \left(\bar{R}_{so} \hat{\lambda}_o \cdot \nabla P_o \right) - \nabla \cdot \left(\bar{R}_{so} \hat{\lambda}_o \cdot \nabla \left(\frac{\rho'_o Z}{144} \right) \right) + \nabla \cdot \left(\bar{R}_{sw} \hat{\lambda}_w \cdot \nabla P_o \right) \\ - \nabla \cdot \left(\bar{R}_{sw} \hat{\lambda}_w \cdot \nabla P_{cow} \right) - \nabla \cdot \left(\bar{R}_{sw} \hat{\lambda}_w \cdot \nabla \left(\frac{\rho'_w Z}{144} \right) \right) - \bar{R}_{so} \left(\nabla \cdot \left(\hat{\lambda}_o \cdot \nabla P_o \right) - \nabla \cdot \left(\hat{\lambda}_o \cdot \nabla \left(\frac{\rho'_o Z}{144} \right) \right) \right) \\ - \bar{R}_{sw} \left(\nabla \cdot \left(\hat{\lambda}_w \cdot \nabla P_o \right) - \nabla \cdot \left(\hat{\lambda}_w \cdot \nabla \left(\frac{\rho'_w Z}{144} + P_{cow} \right) \right) \right) - \left(\frac{q_g}{\rho_g} - \frac{R_{so} q_o}{\rho_o} - \frac{R_{sw} q_w}{\rho_w} \right) \\ = \nabla \cdot \left(\hat{\lambda}_g \cdot \nabla P_o \right) + \nabla \cdot \left(\hat{\lambda}_g \cdot \nabla P_{cgo} \right) - \left(\nabla \cdot \left(\hat{\lambda}_g \cdot \nabla \left(\frac{\rho'_g Z}{144} \right) \right) \right) - \left(\frac{q_g}{\rho_g} - \frac{R_{so} q_o}{\rho_o} - \frac{R_{sw} q_w}{\rho_w} \right) \end{aligned} \quad (1.3.61)$$

So the Gas equation becomes:

$$\begin{aligned} \nabla \cdot \left(\hat{\lambda}_g \cdot \nabla P_o \right) + \nabla \cdot \left(\hat{\lambda}_g \cdot \nabla P_{cgo} \right) - \nabla \cdot \left(\hat{\lambda}_g \cdot \nabla \left(\frac{\rho'_g Z}{144} \right) \right) - \left(\frac{q_g}{\rho_g} - \frac{R_{so} q_o}{\rho_o} - \frac{R_{sw} q_w}{\rho_w} \right) \\ = \frac{\Phi}{B_g} \cdot \frac{\partial S_g}{\partial t} + \Phi \left[\frac{1}{B_g} (S_g C_r + S_g C_g) + \frac{S_o dR_{so}}{B_o dP_o} + \frac{S_w dR_{sw}}{B_w dP_o} \right] \cdot \frac{\partial P_o}{\partial t} \end{aligned}$$

Define,

$$\frac{q_g}{\rho_g} = \frac{q_g}{\rho_g} - R_{so} \frac{q_o}{\rho_o} - R_{sw} \frac{q_w}{\rho_w} \quad (1.3.62)$$

which is the free gas production rate at surface conditions from the source (it is zero if the closed region does not contain a well).

(1.3.63)

We assume that the capillary pressure for oil-to-water (P_{cow}) is a function of the water saturation only and the capillary for gas-to-oil (P_{cgo}) is a function of the gas saturation only. Then

$$\nabla P_{cow} = i \frac{\partial P_{cow}}{\partial x} + j \frac{\partial P_{cow}}{\partial y} + k \frac{\partial P_{cow}}{\partial z} = \left(i \frac{\partial S_w}{\partial x} + j \frac{\partial S_w}{\partial y} + k \frac{\partial S_w}{\partial z} \right) \frac{dP_{cow}}{dS_w} \quad (1.3.64)$$

$$\nabla P_{cow} = \frac{dP_{cow}}{dS_w} \nabla S_w \quad (1.3.65)$$

and,

$$\nabla P_{cgo} = i \frac{\partial P_{cgo}}{\partial x} + j \frac{\partial P_{cgo}}{\partial y} + k \frac{\partial P_{cgo}}{\partial z} = \left(i \frac{\partial S_g}{\partial x} + j \frac{\partial S_g}{\partial y} + k \frac{\partial S_g}{\partial z} \right) \frac{dP_{cgo}}{dS_g} \quad (1.3.66)$$

$$\nabla P_{cgo} = \frac{dP_{cgo}}{dS_g} \nabla S_g \quad (1.3.67)$$

Using these two equations:

Oil:

$$\nabla \cdot (\hat{\lambda}_o \cdot \nabla P_o) - \nabla \cdot \left(\hat{\lambda}_o \cdot \nabla \left(\frac{\rho_o' z}{144} \right) \right) - \frac{q_o}{\rho_o} = -\frac{\Phi}{B_o} \left[\frac{\partial S_w}{\partial t} + \frac{\partial S_g}{\partial t} \right] + \frac{\Phi}{B_o} \left[C_r - \frac{1}{B_o} \frac{\partial B_o}{\partial P_o} \right] \left[\frac{\partial P_o}{\partial t} - S_w \frac{\partial P_o}{\partial t} - S_g \frac{\partial P_o}{\partial t} \right] \quad (1.3.68)$$

Water:

$$\nabla \cdot (\hat{\lambda}_w \cdot \nabla P_o) - \nabla \cdot \left(\hat{\lambda}_w \cdot \nabla \left(\frac{\rho'_w z}{144} \right) \right) - \frac{dP}{dS_w} \nabla \cdot (\hat{\lambda}_w \cdot \nabla S_w) - \frac{q_w}{\rho_w} = \frac{\Phi}{B_w} \frac{\partial S_w}{\partial t} + \frac{\Phi}{B_w} \left[C_r - \frac{1}{B_w} \frac{\partial B_w}{\partial P_o} \right] S_w \frac{\partial P_o}{\partial t} \quad (1.3.69)$$

Gas:

$$\nabla \cdot \hat{\lambda}_g \cdot \nabla \left[P_o - \frac{\rho'_g z}{144} \right] + \frac{dP}{dP_o} \nabla \cdot (\hat{\lambda}_g \cdot \nabla S_g) - \frac{q_g}{\rho_g} = \frac{\Phi}{B_g} \frac{\partial S_g}{\partial t} + \Phi \left[\frac{1}{B_g} (S_g C_r + S_g C_g) + \frac{S_o}{B_o} \frac{dR_{so}}{dP_o} + \frac{S_w}{B_w} \frac{dR_{sw}}{dP_o} \right] \cdot \frac{\partial P_o}{\partial t} \quad (1.3.70)$$

The "mobility" tensor for phase l ,

$$\hat{\lambda}_l = \frac{1}{B_l \mu_l} \begin{bmatrix} k_x \\ k_y \\ k_z \end{bmatrix} \begin{bmatrix} k_{rlx} & k_{rly} & k_{rlz} \\ k_{rlx} & k_{rly} & k_{rlz} \\ k_{rlx} & k_{rly} & k_{rlz} \end{bmatrix} \quad (1.3.71)$$

which, for problems in which we have permeabilities that are assumed to be parallel to the coordinates in the x, y, and z directions; that is, K_x, K_y, K_z are non-zero and all other permeabilities are zero:

$$\hat{\lambda}_l = \frac{k_{rl}}{B_l \mu_l} \begin{bmatrix} K_x & 0 & 0 \\ 0 & K_y & 0 \\ 0 & 0 & K_z \end{bmatrix} \quad (1.3.72)$$

We will use these assumptions in later development to simplify the discretized equations, letting ϵ denote the transmissibility of phase l :

$$\hat{\epsilon}_l = \begin{bmatrix} \epsilon_{zl} \\ \epsilon_{yl} \\ \epsilon_{xl} \end{bmatrix} = \frac{1}{\mu_l} \begin{bmatrix} K_x k_{rlx} \\ K_y k_{rly} \\ K_z k_{rlz} \end{bmatrix}$$

1.4 Finite Element Formulation:

Finite Element formulation of the 3 phase, 3-D reservoir problem follows:

The element trial solution for the oil pressure:

$$P_o^{(e)} = \sum_{j=1}^n P_j^{(e)}(t) N_j^{(e)}(x, y, z) = \sum_{j=1}^n P_j N_j \quad (1.4.1)$$

The element trial solution for the water saturation:

$$S_w^{(e)} = \sum_{j=1}^n S_{wj}^{(e)}(t) N_j^{(e)}(x, y, z) = \sum_{j=1}^n S_{wj} N_j \quad (1.4.2)$$

The element trial solution for the gas saturation:

$$S_g^{(e)} = \sum_{j=1}^n S_{gj}^{(e)}(t) N_j^{(e)}(x, y, z) = \sum_{j=1}^n S_{gj} N_j \quad (1.4.3)$$

The element trial solution for the density term:

$$\rho_o = \sum_{j=1}^n (\rho_o)_j N_j^{(e)}(x, y, z)$$

where the $N_j^{(e)}(x, y, z)$ are the shape functions and n is the number of degrees of freedom (DOF) in the element.

Applying Galerkin's Method to the equations that have been developed yields the following set of weighted residuals equations over each finite element e :

Oil:

$$\int_V \left(\nabla \cdot (\bar{\lambda}_o \cdot \nabla P_o) - \nabla \cdot \bar{\lambda}_o \cdot \nabla \left(\frac{\rho_o z}{144} \right) - \frac{q_o}{\rho_o} + \frac{\Phi}{B_o} \left[\frac{\partial S_w}{\partial t} + \frac{\partial S_g}{\partial t} + \frac{\partial S_o}{\partial t} \right] \right. \\ \left. - \frac{\Phi}{B_o} \left[C_r - \frac{1}{B_o} \frac{\partial B_o}{\partial P_o} \right] \left[\frac{\partial P_o}{\partial t} - S_w \frac{\partial P_o}{\partial t} - S_g \frac{\partial P_o}{\partial t} \right] \right) N_i dV = 0 \quad (1.4.4)$$

Water:

$$\begin{aligned} & \int_V^{(e)} \left\{ \nabla \cdot (\hat{\lambda}_w \cdot \nabla P_o) - \nabla \cdot \left(\hat{\lambda}_w \cdot \nabla \left(\frac{\rho'_w z}{144} \right) \right) - \frac{dP_{cow}}{dS_w} \nabla \cdot (\hat{\lambda}_w \cdot \nabla S_w) - \frac{q}{\rho_w} \right. \\ & \left. - \frac{\phi}{B_w} \frac{\partial S_w}{\partial t} - \frac{\phi}{B_w} \left[C_r - \frac{1}{B_w} \frac{\partial B_w}{\partial P_o} \right] S_w \frac{\partial P_o}{\partial t} \right\} N_i dV = 0 \end{aligned} \quad (1.4.5)$$

Gas:

$$\int_V^{(e)} \left(\nabla \cdot \hat{\lambda}_g \cdot \nabla \left(P_o - \frac{\rho'_g z}{144} \right) + \frac{dP_{cgo}}{dS_g} \nabla \cdot (\hat{\lambda}_g \cdot \nabla S_g) - \frac{q_{fg}}{\rho_g} - \frac{\phi}{B_g} \frac{\partial S_g}{\partial t} - \phi \left[\frac{1}{B_g} (S_g C_r + S_g C_g) + \left(\frac{S_o}{B_o} \frac{dR_{so}}{dP_o} + \frac{S_w}{B_w} \frac{dR_{sw}}{dP_o} \right) \right] \right) \frac{\partial P_o}{\partial t} N_i dV = 0 \quad (1.4.6)$$

1.5 Oil FEM:

We now proceed to produce the formulation, one term at a time.

Conductance Term

$$\int_V^{(e)} \left(\nabla \cdot (\hat{\lambda}_o \cdot \nabla P_o) \right) N_i dV \quad (1.5.1)$$

where

$$\nabla = i \frac{\partial}{\partial x} + j \frac{\partial}{\partial y} + k \frac{\partial}{\partial z} \quad (1.5.2)$$

$$\nabla \cdot P_o = \sum_{j=1}^n P_j^{(e)} \nabla \cdot N_j \quad (1.5.3)$$

Using Green's Theorem:

$$\iiint_V (\nabla^2 g + \nabla f \cdot \nabla g) dV = \iint_S f \frac{\partial g}{\partial n} dA \quad (1.5.4)$$

$$\iiint_V (\nabla^2 g) dV = -\iiint_V (\nabla f \cdot \nabla g) dV + \iint_S f \frac{\partial g}{\partial n} dA \quad (1.5.5)$$

Green's Theorem becomes:

$$\int_V \nabla \cdot (\hat{\lambda}_o \cdot \nabla P_o) N_i dV = - \int_V (\hat{\lambda}_o \cdot \nabla P_o) \cdot \nabla N_i dV + \int_S (\hat{\lambda}_o \cdot \nabla P_o) \cdot \hat{n} N_i ds \quad (1.5.6)$$

where $\hat{n}$ is the unit outer normal vector to the surface S . We can observe that $\hat{\lambda}_o \cdot \nabla P_o$ represents the oil flux, so we may represent this oil flux normal to the surface S as:

$$\tau_o^{(e)} = (\hat{\lambda}_o \cdot \nabla P_o) \cdot \hat{n} \quad (1.5.7)$$

$$\int_S \tau_o^{(e)} N_i ds = \tau_o^{(e)} \int_S N_i ds \quad (1.5.8)$$

for sufficiently small time steps and elements, we may assume that the flux on each face in an element e is a constant.

For a quadrilateral face:

$$\tau_o^{(e)} \int_S N_i ds = \frac{\tau_o^{(e)} A}{4} \quad (1.5.9)$$

(1/4 of the flux is assigned to each node).

Assuming a medium with only K_x, K_y and K_z , then :

$$-\int_V (\hat{\lambda}_o \cdot \nabla P_o) \cdot \nabla N_i dV = -\int_V \sum_{j=1}^n (\hat{\lambda}_o P_j \cdot \nabla N_j) \cdot \nabla N_i dV \quad (1.5.10)$$

$$= -\sum_{j=1}^n \left\{ \int_V [(\hat{\lambda}_o \cdot \nabla N_j) \cdot \nabla N_i] dV \right\} P_j \quad (1.5.11)$$

$$= -\frac{1}{B_o \mu_o} \sum_{j=1}^n \int_V \left[\left((K_{ro})_x \frac{\partial N_j}{\partial x} + (K_{ro})_y \frac{\partial N_j}{\partial y} + (K_{ro})_z \frac{\partial N_j}{\partial z} \right) \cdot \left(\frac{\partial N_i}{\partial x} + \frac{\partial N_i}{\partial y} + \frac{\partial N_i}{\partial z} \right) \right] dV P_j \quad (1.5.12)$$

$$-\int_V (\hat{\lambda}_o \cdot \nabla P_o) \cdot \nabla N_i dV = -\frac{1}{B_o \mu_o} \sum_{j=1}^n P_j \int_V \left[\left((K_{ro})_x \frac{\partial N_j}{\partial x} \frac{\partial N_i}{\partial x} + (K_{ro})_y \frac{\partial N_j}{\partial y} \frac{\partial N_i}{\partial y} + (K_{ro})_z \frac{\partial N_j}{\partial z} \frac{\partial N_i}{\partial z} \right) \right] dV \quad (1.5.13)$$

$$= -\frac{1}{B_o \mu_o} \sum_{j=1}^n P_j \int_V \left[\left((K_{ro})_x \frac{\partial N_j}{\partial x} \frac{\partial N_i}{\partial x} + (K_{ro})_y \frac{\partial N_j}{\partial y} \frac{\partial N_i}{\partial y} + (K_{ro})_z \frac{\partial N_j}{\partial z} \frac{\partial N_i}{\partial z} \right) \right] dV \quad (1.5.14)$$

let,

$$A_{ij}^{(e)} = \int_V \frac{1}{B_o \mu_o} \left[\left((K_{ro})_x \frac{\partial N_j}{\partial x} \frac{\partial N_i}{\partial x} + (K_{ro})_y \frac{\partial N_j}{\partial y} \frac{\partial N_i}{\partial y} + (K_{ro})_z \frac{\partial N_j}{\partial z} \frac{\partial N_i}{\partial z} \right) \right] dV \quad (1.5.15)$$

Gravity Term

$$-\int_V \left[\nabla \cdot \left(\hat{\lambda}_o \cdot \nabla \left(\sum_{j=1}^n \frac{(\rho'_o z_j) N_j}{144} \right) \right) \right] N_i dV = -\sum_{j=1}^n A_{ij}^{(e)} \left[\frac{\rho'_o z_j}{144} \right]_j = D_{oi} \quad (1.5.16)$$

$$D_{oi} = -\sum_{j=1}^n \frac{A_{ij}^{(e)} [\rho'_o z_j]_j}{144}$$

Injection/Production Term

$$-\int_V \frac{q_o}{\rho_o} N_i dV = Q_{oi}^{(e)} \quad (1.5.17)$$

We can probably ignore this term and handle all injection/production with the flux terms. We could also choose to use this term and then integrate it as shown.

Saturation Capacity Terms

$$\int_v \frac{\Phi}{B_o} \left(\frac{\partial S_w}{\partial t} \right) N_i dV = \frac{\Phi}{B_o} \sum_{j=1}^n \left(\frac{\partial S_{wj}}{\partial t} N_j \right) N_i dV = \frac{\Phi}{B_o} \sum_{j=1}^n \left[\left(\frac{\partial S_{wj}}{\partial t} \right) \int_v N_j N_i dV \right] \quad (1.5.18)$$

$$C_{ij}^o = \frac{\Phi}{B_o} \int_v N_i N_j dV \quad (1.4.4.1)$$

$$\int_v \frac{\Phi}{B_o} \left(\frac{\partial S_g}{\partial t} \right) N_i dV = \frac{\Phi}{B_o} \sum_{j=1}^n \left[\left(\frac{\partial S_{gj}}{\partial t} \right) \int_v N_j N_i dV \right] \quad (1.5.5)$$

Compressibility Term

$$\int_v \frac{\Phi}{B_o} \left[C_r - \frac{1}{B_o} \frac{\partial B_o}{\partial P_o} \right] \left[\frac{\partial P_o}{\partial t} - S_w \frac{\partial P_o}{\partial t} - S_g \frac{\partial P_o}{\partial t} \right] N_i dV \quad (1.5.6)$$

let

$$\alpha_o = \left[C_r - \frac{1}{B_o} \frac{\partial B_o}{\partial P_o} \right] \quad (1.5.7)$$

(these saturations are fixed at the k-timestep or the last timestep represented by the k superscript)

$$S_w^{(e)} = \sum_{l=1}^n (S_w^k) N_l \quad (1.5.8)$$

$$S_g^{(e)} = \sum_{l=1}^n (S_g^k) N_l \quad (1.5.9)$$

$$\frac{\partial P_o}{\partial t} = \sum_{j=1}^n \left(\frac{\partial P}{\partial t} \right) N_j \quad (1.5.10)$$

$$\begin{aligned}
& -\frac{\varphi \alpha_o}{B_o} \int_v \left[\frac{\partial P_o}{\partial t} - S_w \frac{\partial P_o}{\partial t} - S_g \frac{\partial P_o}{\partial t} \right] N_i dV - \frac{\varphi \alpha_o}{B_o} \left[\int_v \left\{ \sum_{j=1}^n \left(\frac{\partial P}{\partial t} \right)_j N_j N_i dV \right\} - \int_v \left\{ \sum_{l=1}^n (S_w^k)_l N_l \right\} \left\{ \sum_{j=1}^n \left(\frac{\partial P}{\partial t} \right)_j N_j \right\} N_i dV \right. \\
& \left. - \int_v \left\{ \sum_{l=1}^n (S_g^k)_l N_l \right\} \left\{ \sum_{j=1}^n \left(\frac{\partial P}{\partial t} \right)_j N_j \right\} N_i dV \right] \quad (1.5.11)
\end{aligned}$$

$$-\frac{\varphi \alpha_o}{B_o} \left\{ \sum_{j=1}^n \left(\frac{\partial P}{\partial t} \right)_j \int_v N_i N_j dV - \sum_{j=1}^n \left(\frac{\partial P}{\partial t} \right)_j \left[\sum_{l=1}^n \left[(S_w^k)_l \int_v N_i N_j N_l dV \right] - \sum_{l=1}^n \left(\frac{\partial P}{\partial t} \right)_l \left[\sum_{l=1}^n (S_g^k)_l \int_v N_i N_j N_l dV \right] \right] \right\} \quad (1.5.12)$$

Define:

$$f_2(i, j) = \int_v N_i N_j dV \quad (1.5.13)$$

and

$$f_3(i, j, l) = \int_v N_i N_j N_l dV \quad (1.5.14)$$

then:

$$\begin{aligned}
& -\frac{\varphi \alpha_o}{B_o} \int_v \left[\frac{\partial P_o}{\partial t} - S_w \frac{\partial P_o}{\partial t} - S_g \frac{\partial P_o}{\partial t} \right] N_i dV - \frac{\varphi \alpha_o}{B_o} \left\{ \sum_{j=1}^n \left(\frac{\partial P}{\partial t} \right)_j \left[\sum_{l=1}^n \left[(S_w^k)_l + (S_g^k)_l \right] f_3(i, j, l) \right] - \sum_{j=1}^n \left(\frac{\partial P}{\partial t} \right)_j f_2(i, j) \right\} \\
& = \frac{\varphi \alpha_o}{B_o} \sum_{j=1}^n \left[\sum_{l=1}^n \left\{ \left[(S_w^k)_l + (S_g^k)_l \right] f_3(i, j, l) \right\} - f_2(i, j) \right] \left(\frac{\partial P}{\partial t} \right)_j \quad (1.5.15)
\end{aligned}$$

$$CO_{ij}^{(\epsilon)} = \frac{\varphi \alpha_o}{B_o} \left[\sum_{l=1}^n \left\{ \left[(S_w^k)_l + (S_g^k)_l \right] f_3(i, j, l) \right\} - f_2(i, j) \right] \quad (1.5.16)$$

The matrix representation for oil becomes:

$$A_{ij}^{(\epsilon)} p_j^{k+1} + CO_{ij}^{(\epsilon)} p_j + C_{ij}^{(\epsilon)} (S_w + S_g)_j + Q_{oi}^{(\epsilon)} + D_{oi}^{(\epsilon)} = 0 \quad (1.5.17)$$

1.6 Water FEM

Conductance Term

$$\int_V \nabla \cdot (\hat{\lambda}_w \cdot \nabla P_o) N_i dV = - \int_V (\hat{\lambda}_w \cdot \nabla P_o) \cdot \nabla N_i dV + \int_S (\hat{\lambda}_w \cdot \nabla P_o) \cdot \hat{n} N_i ds \quad (1.6.1)$$

$$\tau_w^{(e)} = (\hat{\lambda}_w \cdot \nabla P_o) \cdot \hat{n} \quad (1.6.2)$$

$$\int_S \tau_w^{(e)} N_i ds = \tau_w^{(e)} \int_S N_i ds = \frac{\tau_w^{(e)} A}{4} \quad (1.6.3)$$

$$- \int_V (\hat{\lambda}_w \cdot \nabla P_o) \cdot \nabla N_i dV = - \frac{1}{B_w \mu_w} \sum_{j=1}^n P_j \int_V \left[(K_{rw})_x \frac{\partial N_i}{\partial x} \frac{\partial N_j}{\partial x} + (K_{rw})_y \frac{\partial N_i}{\partial y} \frac{\partial N_j}{\partial y} + (K_{rw})_z \frac{\partial N_i}{\partial z} \frac{\partial N_j}{\partial z} \right] dV \quad (1.6.4)$$

$$H_{ij}^{(e)} = - \frac{1}{B_w \mu_w} \int_V \left[(K_{rw})_x \frac{\partial N_i}{\partial x} \frac{\partial N_j}{\partial x} + (K_{rw})_y \frac{\partial N_i}{\partial y} \frac{\partial N_j}{\partial y} + (K_{rw})_z \frac{\partial N_i}{\partial z} \frac{\partial N_j}{\partial z} \right] dV \quad (1.6.5)$$

Gravity Term

$$- \int_V \nabla \cdot \left(\hat{\lambda}_w \cdot \nabla \left(\frac{\rho'_w z}{144} \right) \right) N_i dV = \sum_{j=1}^n H_{ij}^{(e)} \left[\frac{\rho'_w z}{144} \right]_j \quad (1.6.6)$$

$$= D_{wi}^{(e)} \quad (1.6.7)$$

Injection/Production Term

$$- \int_V \frac{q_w}{P_w} N_i dV = Q_{wi}^{(e)} \quad (1.6.8)$$

Capillary Term

$$-\int_V \frac{dP_{cow}}{dS_w} [\nabla \cdot (\hat{\lambda}_w \cdot \nabla S_w)] N_i dV = -\frac{dP_{cow}}{dS_w} \int_V [\nabla \cdot (\hat{\lambda}_w \cdot \nabla S_w)] N_i dV \quad (1.6.9)$$

$$= \frac{dP_{cow}}{dS_w} \int_V (\hat{\lambda}_w \cdot \nabla S_w) \cdot \nabla N_i dV - \frac{dP_{cow}}{dS_w} \int_S (\hat{\lambda}_w \cdot \nabla S_w) \cdot \hat{n} N_i dS \quad (1.6.10)$$

The value of $\frac{dP_{cow}}{dS_w}$ is determined empirically by observing a table of relationships between P_{cow} and S_w , and may be considered a constant within a given time step. The flux term as before, "fluxes" at interior boundaries will cancel. For external boundaries, if there is no change in saturation then $(\hat{\lambda}_w \cdot \nabla S_w) \cdot \hat{n} = 0$, this will be modified as required at the production nodes.

$$S_w^{(c)} = \sum_{j=1}^n (S_w)_j N_j \quad (1.6.11)$$

and,

$$\nabla S_w^{(c)} = \sum_{j=1}^n (S_w)_j \nabla N_j \quad (1.6.12)$$

$$\frac{dP_{cow}}{dS_w} \int_V (\hat{\lambda}_w \cdot \nabla S_w) \cdot \nabla N_i dV = \frac{dP_{cow}}{dS_w} \int_V (\hat{\lambda}_w \cdot \left(\sum_{j=1}^n (S_w)_j \nabla N_j \right) \cdot \nabla N_i) dV \quad (1.6.13)$$

$$= \frac{dP_{cow}}{dS_w} \left(\sum_{j=1}^n (S_w)_j \left\{ \int_V (\hat{\lambda}_w \cdot \nabla N_j) \cdot \nabla N_i dV \right\} \right) \quad (1.6.14)$$

$$= \frac{dP_{cow}}{dS_w} \frac{1}{B_w \mu_w} \left(\sum_{j=1}^n (S_w)_j \int_V \left[(K_{rw})_x \frac{\partial N_j}{\partial x} \frac{\partial N_i}{\partial x} + (K_{rw})_y \frac{\partial N_j}{\partial y} \frac{\partial N_i}{\partial y} + (K_{rw})_z \frac{\partial N_j}{\partial z} \frac{\partial N_i}{\partial z} \right] \right) \quad (1.6.15)$$

$$B_{ij}^{(\epsilon)} = \frac{dP_{cow}}{dS_w} \frac{1}{B_w \mu_w} \left\{ \int_v \left[\left(K_{rw} \right)_x \frac{\partial N_i}{\partial x} \frac{\partial N_j}{\partial x} + \left(K_{rw} \right)_y \frac{\partial N_i}{\partial y} \frac{\partial N_j}{\partial y} + \left(K_{rw} \right)_z \frac{\partial N_i}{\partial z} \frac{\partial N_j}{\partial z} \right] dV \right\} \quad (1.6.16)$$

Saturation Capacity Term

$$- \int_v \left(\frac{\Phi}{B_w} \frac{\partial S_w}{\partial t} \right) N_i dV = - \frac{\Phi}{B_w} \int_v \frac{\partial S_w}{\partial t} N_i dV = - \frac{\Phi}{B_w} \sum_{j=1}^n \left\{ \frac{\partial S_w}{\partial t} \int_v N_j N_i dV \right\} \quad (1.6.17)$$

$$C_{ij}^w = - \frac{\Phi}{B_w} \int_v N_j N_i dV \quad (1.6.18)$$

Compressibility Term

$$\int_v \frac{\Phi}{B_w} \left[C_r - \frac{1}{B_w} \frac{\partial B_w}{\partial P_o} \right] S_w \frac{\partial P_o}{\partial t} N_i dV = - \frac{\Phi}{B_w} \left[C_r - \frac{1}{B_w} \frac{\partial B_w}{\partial P_o} \right] \int_v \left(S_w \frac{\partial P_o}{\partial t} \right) N_i dV \quad (1.6.19)$$

$$- \frac{\Phi}{B_w} \left[C_r - \frac{1}{B_w} \frac{\partial B_w}{\partial P_o} \right] \left\{ \int_v \left[\sum_{l=1}^n (S_w^k)_l N_l \right] \left[\sum_{j=1}^n \left(\frac{\partial P_o}{\partial t} \right)_j N_j \right] N_i dV \right\} \quad (1.6.20)$$

$$- \frac{\Phi}{B_w} \left[C_r - \frac{1}{B_w} \frac{\partial B_w}{\partial P_o} \right] \left\{ \sum_{j=1}^n \left(\frac{\partial P_o}{\partial t} \right)_j \left[\sum_{l=1}^n (S_w^k)_l \int_v N_l N_j N_i dV \right] \right\} \quad (1.6.21)$$

$$f_3(i, j, l) = \int_v N_i N_j N_l dV \quad (1.6.22)$$

$$- \frac{\Phi}{B_w} \left[C_r - \frac{1}{B_w} \frac{\partial B_w}{\partial P_o} \right] \left\{ \sum_{j=1}^n \left(\frac{\partial P_o}{\partial t} \right)_j \left[\sum_{l=1}^n (S_w^k)_l f_3(i, j, l) \right] \right\} \quad (1.6.23)$$

$$CW_{ij}^{(\epsilon)} = - \frac{\Phi}{B_w} \left[C_r - \frac{1}{B_w} \frac{\partial B_w}{\partial P_o} \right] \left\{ \sum_{l=1}^n (S_w^k)_l f_3(i, j, l) \right\} \quad (1.6.24)$$

The matrix representation for water becomes:

$$H_{ij}^{(e)} P_j^{k+1} + C W_{ij}^{(e)} \frac{\partial P_j}{\partial t} + C W_{ij}^w \frac{\partial S_{wj}}{\partial t} + B_{ij}^{(e)} S_{wj}^{k+1} + Q_{wi}^{(e)} + D_{wi}^{(e)} = 0 \quad (1.6.25)$$

1.7 Gas FEM

$$\int_v \nabla \cdot (\hat{\lambda}_g \cdot \nabla P_o) N_i dV = - \int_v (\hat{\lambda}_g \cdot \nabla P_o) \cdot \nabla N_i dV + \int_s (\hat{\lambda}_g \cdot \nabla P_o) \cdot \hat{n} N_i dS \quad (1.7.1)$$

$$\tau_g^{(e)} = (\hat{\lambda}_g \cdot \nabla P_o) \cdot \hat{n} \quad (1.7.2)$$

$$\tau_g^{(e)} \int_s N_i dS = \frac{\tau_g^{(e)} A}{4} \quad (1.7.3)$$

$$- \int_v (\hat{\lambda}_g \cdot \nabla P_o) \cdot \nabla N_i dV = - \sum_{j=1}^n \left\{ \int_v [(\hat{\lambda}_g \cdot \nabla N_j) \cdot \nabla N_i] dV \right\} P_j \quad (1.7.4)$$

$$= - \frac{1}{B_g \mu_g} \sum_{j=1}^n P_j \int_v \left[(K_{rg})_x \frac{\partial N_j}{\partial x} \frac{\partial N_i}{\partial x} + (K_{rg})_y \frac{\partial N_j}{\partial y} \frac{\partial N_i}{\partial y} + (K_{rg})_z \frac{\partial N_j}{\partial z} \frac{\partial N_i}{\partial z} \right] dV \quad (1.7.5)$$

let

$$E_{ij}^{(e)} = \int_v \left[- \frac{1}{B_g \mu_g} \left((K_{rg})_x \frac{\partial N_j}{\partial x} \frac{\partial N_i}{\partial x} + (K_{rg})_y \frac{\partial N_j}{\partial y} \frac{\partial N_i}{\partial y} + (K_{rg})_z \frac{\partial N_j}{\partial z} \frac{\partial N_i}{\partial z} \right) \right] dV \quad (1.7.6)$$

Capillary Term

$$\int_v \frac{dP}{dS_g} [\nabla \cdot (\hat{\lambda}_g \cdot \nabla S_g)] N_i dV = \frac{dP}{dS_g} \int_v (\nabla \cdot (\hat{\lambda}_g \cdot \nabla S_g)) N_i dV \quad (1.7.7)$$

$$= \frac{dP}{dS_g} \int_v (\hat{\lambda}_g \cdot \nabla S_g) \cdot \nabla N_i dV + \frac{dP}{dS_g} \int_s (\hat{\lambda}_g \cdot \nabla S_g) \cdot \hat{n} dS \quad (1.7.8)$$

as before,

$$\frac{dP_{cgo}}{dS_g} \int_v (\hat{\lambda}_g \cdot \nabla S_g) \cdot \nabla N_i dV = \frac{dP_{cgo}}{dS_g} \sum_{j=1}^n \frac{(S_g)_j}{B_g \mu_g} \int_v \left[(K_{rg})_x \frac{\partial N_i}{\partial x} \frac{\partial N_j}{\partial x} + (K_{rg})_y \frac{\partial N_i}{\partial y} \frac{\partial N_j}{\partial y} + (K_{rg})_z \frac{\partial N_i}{\partial z} \frac{\partial N_j}{\partial z} \right] dV \quad (1.7.9)$$

$$G_{ij}^{(\epsilon)} = \int_v \frac{dP_{cgo}}{dS_g} \frac{\Phi}{B_g \mu_g} \left[(K_{rg})_x \frac{\partial N_i}{\partial x} \frac{\partial N_j}{\partial x} + (K_{rg})_y \frac{\partial N_i}{\partial y} \frac{\partial N_j}{\partial y} + (K_{rg})_z \frac{\partial N_i}{\partial z} \frac{\partial N_j}{\partial z} \right] dV \quad (1.7.10)$$

Saturation Capacity Term

$$-\int_v \left(\frac{\Phi}{B_g} \frac{\partial S_g}{\partial t} \right) N_i dV = - \sum_{j=1}^n \int_v \frac{\Phi}{B_g} \left(\frac{\partial S_g}{\partial t} \right)_{ij} N_j N_i dV \quad (1.7.11)$$

Let,

$$f_g^{(\epsilon)} = - \int_v \frac{\Phi}{B_g} N_j N_i dV \quad (1.7.12)$$

The Gravity Term

$$-\int_v \nabla \cdot \left(\hat{\lambda}_g \cdot \nabla \left(\frac{\rho_g' z}{144} \right) \right) N_i dV = - \sum_{j=1}^n E_{ij}^{(\epsilon)} \left[\frac{\rho_g' z}{144} \right]_j = D_{gi}^{(\epsilon)} \quad (1.7.13)$$

$$(1.7.14)$$

Injection/Production Term

$$-\int_v \frac{q_{fg}}{\rho_g} N_i dV = Q_{gi}^{(\epsilon)} \quad (1.7.15)$$

let

$$\int_v \frac{\Phi}{B_g} (S_g C_r + S_g C_g + S_o C_\alpha) \frac{\partial P_o}{\partial t} N_i dV = \int_v \frac{\Phi}{B_g} (S_g C_r + S_g C_g + (1 - S_w - S_g) C_\alpha) \frac{\partial P}{\partial t} N_i dV \quad (1.7.16)$$

$$= \int_v \frac{\Phi}{B_g} (S_g C_r + S_g C_g + (1 - S_w - S_g) C_\alpha) \sum_{j=1}^n \left(\frac{\partial P_o}{\partial t} \right)_j N_j N_i dV \quad (1.7.17)$$

Let,

$$GX_{ij} = - \int_v \frac{\Phi}{B_g} (S_g C_r + S_g C_g + (1 - S_w - S_g) C_\alpha) N_j N_i dV \quad (1.7.18)$$

$$(1.7.19)$$

$$\int_v \frac{\Phi}{B_w} \frac{\partial R_{sw}}{\partial P_o} S_w \frac{\partial P_o}{\partial t} N_i dV = \sum_{j=1}^n \int_v \frac{\Phi}{B_w} \frac{\partial R_{sw}}{\partial P_o} S_w \left(\frac{\partial P_o}{\partial t} \right)_j N_j N_i dV$$

let

$$WX_{ij} = \int_v \frac{\Phi}{B_w} \frac{\partial R_{sw}}{\partial P_o} S_w N_j N_i dV \quad (1.7.20)$$

$$[E_{ij}^{(\epsilon)}] P_j^{k+1} + [G_{ij}^{(\epsilon)}] S_{gj}^{k+1} + D_{gi}^{(\epsilon)} + Q_{gi}^{(\epsilon)} + f_g \frac{\partial S_g}{\partial t} + [GX_{ij}] \frac{\partial P_j}{\partial t} + [WX_{ij}] \frac{\partial P_j}{\partial t} = 0 \quad (1.7.21)$$

Using the finite element equations that we have developed for 3-D 3-phase flow in a porous media in matrix-vector form and replacing the time derivatives with a finite-difference approximations and by indicating with superscripts k and $k+1$ quantities at the beginning and at the end of a generic iteration step, then:

(1)

$$[A] P^{k+1} + [CO] \frac{\partial P}{\partial t} + [C^o] \left(\frac{\partial S_w}{\partial t} + \frac{\partial S_g}{\partial t} \right) + Q_o + D_o = 0 \quad (1.7.22)$$

(2)

$$[H] P^{k+1} + [CW] \frac{\partial P}{\partial t} + [C^w] \frac{\partial S_w}{\partial t} + [B] S_w^{k+1} + Q_w + D_w = 0 \quad (1.7.23)$$

(3)

$$[E] P^{k+1} + [G] S_g^{k+1} + D_g + Q_g + f_g \frac{\partial S}{\partial t} + [GX] \frac{\partial P}{\partial t} + [WX] \frac{\partial P}{\partial t} = 0 \quad (1.7.24)$$

With a time step of Δt and doing an "implicit" solution:

(1)

$$[A] P^{k+1} + [CO] \left(\frac{P^{k+1} - P^k}{\Delta t} \right) + [C^O] \left(\frac{S_w^{k+1} - S_w^k}{\Delta t} + \frac{S_g^{k+1} - S_g^k}{\Delta t} \right) + Q_o + D_o = 0 \quad (1.7.25)$$

$$\left([A] + \frac{1}{\Delta t} [CO] \right) P^{k+1} + \frac{1}{\Delta t} [C^O] (S_w^{k+1} + S_g^{k+1}) = -Q_o - D_o + \frac{1}{\Delta t} [CO] P^k + \frac{1}{\Delta t} [C^O] (S_w^k + S_g^k) \quad (1.7.26)$$

(2)

$$[H] P^{k+1} + [CW] \left(\frac{P^{k+1} - P^k}{\Delta t} \right) + [C^W] \left(\frac{S_w^{k+1} - S_w^k}{\Delta t} \right) + [B] S_w^{k+1} + Q_w + D_w = 0 \quad (1.7.27)$$

$$\left([H] + \frac{1}{\Delta t} [CW] \right) P^{k+1} + \left([B] + \frac{1}{\Delta t} [C^W] \right) S_w^{k+1} = -Q_w - D_w + \frac{1}{\Delta t} [CW] P^k + \frac{1}{\Delta t} [C^W] S_w^k \quad (1.7.28)$$

(3)

$$[E] P^{k+1} + [G] S_g^{k+1} + D_g + Q_g + f_g \frac{\partial S}{\partial t} + [GX] \frac{\partial P}{\partial t} + [WX] \frac{\partial P}{\partial t} = 0 \quad (1.7.29)$$

and,

$$\begin{aligned} [E] P^{k+1} + [G] S_g^{k+1} + [GX] \left(\frac{P^{k+1} - P^k}{\Delta t} \right) + [WX] \left(\frac{P^{k+1} - P^k}{\Delta t} \right) + f_g \left(\frac{S_g^{k+1} - S_g^k}{\Delta t} \right) \\ = -D_g - Q_g \end{aligned} \quad (1.7.30)$$

and,

$$\left([E] + \frac{1}{\Delta t} [GX] + \frac{1}{\Delta t} [WX] \right) P^{k+1} + \left([G] + \frac{1}{\Delta t} f_g \right) S_g^{k+1} = -D_g - Q_g + \frac{1}{\Delta t} ([C^W] + [GX] + [WX]) P^k + \frac{[f_g]}{\Delta t} S_g^k \quad (1.7.31)$$

To write this in a more compact form, let:

$$[Y] = [GX] + [WX] \quad (1.7.32)$$

$$F_i = -D_g - Q_g + \frac{1}{\Delta t} [\gamma] p^k + \frac{1}{\Delta t} [f_g] S_g^k \quad (1.7.33)$$

then:

$$\left([E] + \frac{1}{\Delta t} [\gamma] \right) p^{k+1} + \left([G] + \frac{1}{\Delta t} [F_g] \right) S_g^{k+1} = -D_g - Q_g + \frac{1}{\Delta t} [\gamma] p^k + \frac{1}{\Delta t} [f_g] S_g^k \quad (1.7.34)$$

These three equations can be solved using a fully implicit scheme:

$$\begin{bmatrix} [A] + \frac{1}{\Delta t} [CO] & \frac{1}{\Delta t} [C^o] & \vdots & -\frac{1}{\Delta t} [C^o] \\ [H] + \frac{1}{\Delta t} [C_w] & [B] + \frac{1}{\Delta t} [C^w] & \vdots & 0 \\ [E] + \frac{1}{\Delta t} [\gamma] & 0 & \vdots & [G] + \frac{1}{\Delta t} [F_g] \end{bmatrix} \begin{bmatrix} p^{k+1} \\ S_w^{k+1} \\ S_g^{k+1} \end{bmatrix} = \begin{bmatrix} -Q_o - D_o + \frac{1}{\Delta t} [CO] p^k + \frac{1}{\Delta t} [C^o] (S_w^k + S_g^k) \\ -Q_w - D_w + \frac{1}{\Delta t} [C_w] p^k + \frac{1}{\Delta t} [C^w] S_w^k \\ F_i \end{bmatrix} \quad (1.7.35)$$

The simulation of 3-phase, 3-D flow in a porous medium requires solving this coupled system of nonlinear approximation to the partial differential equations. These approximations arise from the conservation of mass principle for an oil-water-gas system.