DEVELOPMENT OF A THREE DIMENSIONAL TERRAIN DISPLAY FOR A LIGHT INFANTRY COMBAT MODEL

by

Thomas G. Dodd

June 1990

Thesis Advisor: Samuel H. Parry

Approved for public release; distribution is unlimited.
As an augmentation to field training, the author identifies a need for an easily available light infantry platoon combat model that presents a realistic view of the battlefield environment. To meet this need, the author examines the feasibility of developing a realistic three-dimensional display of a terrain representation on a personal computer. The target computer provides only limited graphics support with an Enhanced Graphics Adapter and all graphics routines are implemented in software. Three methods of terrain representation are examined, and the Dynamic Tactical Simulation (DYNAC) terrain model is chosen for implementation. The DYNAC representation uses a specialized triangle drowing procedure written in assembly language, the painter's algorithm for hidden surface removal, and Defense Mapping Agency Digital Terrain Elevation Data. The implementation obtains a display rate between 1.2 and 1.5 seconds on a 80386 based 25 MHz computer. The author concludes that with the addition of enhancements that provide the capability to display cultural features, and model the target acquisition process, the program could be developed into a light infantry platoon combat model or a research tool for examining the effects of human factors on tactical decision making.
Development of a Three Dimensional
Terrain Display for a
Light Infantry Platoon Combat Model

by

Thomas G. Dodd
Captain, United States Army
B.S., United States Military Academy, 1981

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS TECHNOLOGY
(Command, Control, and Communications)

from the

NAVAL POSTGRADUATE SCHOOL
June 1990

Author: Thomas G. Dodd

Approved by: Samuel H. Parry, Thesis Advisor
James C. Hoffman, Second Reader
Carl R. Jones, Chairman,
Command, Control, and Communications
Academic Group
ABSTRACT

As an augmentation to field training, the author identifies a need for an easily available light infantry platoon combat model that presents a realistic view of the battlefield environment. To meet this need, the author examines the feasibility of developing a realistic three dimensional display of a terrain representation on a personal computer. The target computer provides only limited graphics support with an Enhanced Graphics Adapter and all graphics routines are implemented in software. Three methods of terrain representation are examined, and the Dynamic Tactical Simulation (DYNTACS) terrain model is chosen for implementation. The DYNTACS representation uses a specialized triangle drawing procedure written in assembly language, the painter's algorithm for hidden surface removal, and Defense Mapping Agency Digital Terrain Elevation Data. The implementation obtains a display rate between 1.2 and 1.5 seconds on a 80386 based 25 MHz computer. The author concludes that with the addition of enhancements that provide the capability to display cultural features, and model the target acquisition process, the program could be developed into a light infantry platoon combat model or a research tool for examining effects of human factors effects on tactical decision making.
# TABLE OF CONTENTS

I. INTRODUCTION ............................................. 1  
   A. BACKGROUND ........................................... 1  
   B. SUMMARY OF SUBSEQUENT CHAPTERS ..................... 3  

II. THE NEED AND THE REQUIREMENTS .......................... 5  
    A. THE NEED ............................................. 5  
       1. Command and Control - The Unifying Thread ...... 7  
       2. Bounding the Problem .............................. 7  
    B. THE DESIGN CHOICES .................................. 10  
       1. The Microcomputer ................................. 11  
       2. Programming Language Software .................... 12  
    C. REQUIREMENTS OF THE BATTLEFIELD ENVIRONMENT MODEL 13  

III. TERRAIN MODELING METHODOLOGY .......................... 15  
     A. SELECTION OF A TERRAIN REPRESENTATION .......... 15  
        1. The DYNTACS Terrain Model ....................... 16  
        2. The IUA Terrain Model ............................ 19  
        3. The STAR Terrain Model ........................... 20  
        4. The Macro Terrain Model of Choice ................. 21  
        5. Simulating the Micro Terrain ...................... 22  
        6. Representing Forest and Other Terrain Features ... 22
B. LINE OF SIGHT CALCULATIONS FOR THE DYNTACS TERRAIN
MODEL ............................................. 24
1. Determining Elevation at a Location on the Terrain
   Model ............................................. 24
2. Line of Sight Routine ............................................. 28
C. MOVEMENT MODELING ............................................. 30
D. MODELING TARGET ACQUISITION ............................................. 31

IV. DISPLAYING THE DYNTACS REPRESENTATION ............................................. 33
A. PROGRAM OVERVIEW ............................................. 34
B. GRAPHICS IMPLEMENTATION ISSUES ............................................. 35
   1. Providing Realism ............................................. 39
      a. Perspective Projection ............................................. 39
      b. Filled Polygons ............................................. 40
      c. Shades of Color Dependent on Light Conditions 40
   2. Providing Speed ............................................. 42
      a. The Specialized Triangle Drawing Procedure ... 42
        (1) The Need ............................................. 42
        (2) The Specialized Algorithm ......................... 44
        (3) Implementing the Special Algorithm .......... 45
        (4) The Results ............................................. 47
      b. The Soldier Sorting Algorithm ......................... 47
        (1) Nature of the Problem ......................... 47
        (2) Binary Search Trees ......................... 48
        (3) The Results ............................................. 49
      c. The Integrated Display Algorithm ......................... 49
MISSING PAGE NOT ATTAINABLE
I. INTRODUCTION

The purpose of this thesis is to develop a three dimensional display of a terrain model on a personal computer. Such a model can be utilized as a component of a light infantry platoon combat model for training platoon leaders or as a tool for conducting experiments to measure human factors effects on Command and Control (C2) decisions. Before discussing the development of the terrain model, some background material is necessary and is presented in this chapter.

A. BACKGROUND

The U. S. Army has identified five strategic roles for itself, one of which is to maintain contingency forces for immediate combat worldwide across the spectrum of conflict [Ref 1:p. 6]. In order to prepare units for this role, not to mention as a deterrent to war, the Army conducts deployments to many countries (e.g., Thailand, South Korea, West Germany, Honduras, etc.). These deployments provide training experiences that cannot be gained in the United States and are thus necessary to insure the Army is capable of performing its wartime missions.

To insure a trained and ready force, the U. S. Army has identified several fundamental imperatives. Two of these imperatives are of interest in this thesis: conduct tough and realistic training and develop competent, confident leaders [Ref. 2:p. II-5]. In order to develop competent, confident leaders, the Army advocates leader training and unit training. Leader training insures a technically competent leader, while unit training assists
in developing leaders who are confident in executing their function. In order to conduct realistic training, the Army uses field training exercises that are planned and conducted as realistically as possible within safety constraints. The most realistic peacetime training available to the Army occurs at Combat Training Centers (CTC). These centers and their roles are:

- The National Training Center (NTC) -- provide realistic combat training for Battalion and Brigades in mid to high intensity scenarios.
- The Joint Readiness Training Center (JRTC) -- provide realistic combat training for non-mechanized battalion task forces in low to mid intensity scenarios.
- The Combat Maneuver Training Center (CMTC) -- provide a NTC type experience for units in the Federal Republic of Germany.
- Battle Command Training Program (BCTP) -- provide realistic combat training for Corps and Division commanders and their staff. [Ref. 2:pp. VI-1 - VI-2]

These centers provide for realistic training to units that participate, but no matter how realistic the training, several ingredients are missing that are present in combat. Firstly, in combat people die. In training, except for training accidents, people do not really die. Secondly, because people do not get killed, the psychological stresses and fears do not manifest themselves the same way they do in combat. Thirdly, due to resource limitations, the representation of the battlefield environment is limited. The terrain and the enemy are limited to that of the training center. There is not a significant number of noncombatants represented at these training centers as they are present on the battlefield. For example, consider how many civilians were in Panama during Operation Just Cause. The units involved in that operation had to deal with Panamanian
civilians in addition to the Panamanian Defense Force. U. S. forces were prepared for combat, however results indicate they were not prepared for the Panamanian civilians and the impact they would have on operations (e.g., looting, firing on noncombatants, etc.). The absence of these ingredients in unit field training results in a semi-sterile environment that does not completely represent the environment of the battlefield.

One solution to this deficiency in training is to increase the amount of resources involved in the training exercise (i.e., make additional soldiers play the role of noncombatants and build more training facilities). Given the trend in today's defense budget discussions in Congress, this may not be a feasible alternative. Another solution would be to make use of available resources, such as personal computers, and develop a computer simulation to round out the experience of unit leaders. In order to develop such a simulation, it must first be determined if a realistic terrain display can be developed for a computer with limited capability.

B. SUMMARY OF SUBSEQUENT CHAPTERS

The remainder of this thesis discusses the development of a three dimensional display of a terrain model on a personal computer. It consists of five more chapters that address the following:

- Chapter II addresses three areas. Firstly, it translates the need for training in a realistic battlefield environment into a need for a combat model which is in turn translated into a need for a realistic terrain display. Secondly, it discusses some design considerations and why they were chosen. Thirdly, it address the requirements that are derived from the need and the design considerations.

- Chapter III addresses terrain modeling methodology in four areas. Firstly, it discusses the selection of a terrain representation. Secondly, it discusses the Line of Sight calculations for the selected
representation. Thirdly, it discusses movement modeling on the selected representation. Lastly, it discusses detection modeling in the battlefield environment.

- Chapter IV discusses the display program that was developed in order to implement the three dimensional display of the selected terrain representation. It provides an overview of the program and addresses some of the implementation issues and resulting algorithms that solved some of the problems.

- Chapter V discusses enhancements to the program for the displaying of the terrain model that will fully implement the areas discussed in Chapter II that have not been implemented.

- Chapter VI provides conclusions that are obtained from implementation of the terrain model.
II. THE NEED AND THE REQUIREMENTS

A. THE NEED

The last chapter addressed the lack of a complete representation of the combat environment in training. Even though it is not possible to entirely replicate the battlefield environment for training, it is possible to simulate some of its qualities through computer simulation. A computer simulation could theoretically simulate the battlefield environment more robustly than in training exercises. A simulation can use different terrain by changing its database. The enemy can also be changed in the same manner. Computer representations of civilians can be integrated into the simulation to provide a more realistic battlefield. Such a simulation could be a surrogate for experience and augment a leader's understanding of the battlefield in conjunction with the CTC.

The Army currently has a Family of Simulations (FAMSIM) that develops and sustains skills for commanders and their staffs at Battalion level and higher [Ref. 2:p. VI-4]. One problem with these types of simulations is the modeling of the information flow. They do not adequately model bad information and the impact it has on decisions. One solution to exposing leaders to the problems of dealing with bad information is to improve the quality and quantity of battle simulations for commanders and leaders at all levels [Ref. 3:p. 52]. Smith states:

The only real way to learn at the tactical level is to practice continually in a brutal environment, make mistakes (which often mean getting you ego bruised), get good counsel, and get back in the ring for another go. We can no longer afford to teach leaders the critical
art of fighting with poor information during one or two high reputation events a year. They must be repeatedly immersed in a learning environment (like the combat training centers at Fort Irwin and Hohenfels or BABAS ... exercises) and be allowed to make mistakes without a reputation cost [Ref. 3:p. 53].

A realistic combat simulation that incorporates the modeling of bad information flow would meet this need.

As mentioned, the Army has simulations that are structured toward the battalion level and above. According to the Operations Field Manual 100-5,

... modern combat requires greater dispersal of units, the quality and effectiveness of junior leaders has a proportionately greater impact. Prior to combat, senior leaders must place greater emphasis on junior leader development [Ref. 4:p. 26].

One way to place greater emphasis on junior leader development is to develop a simulation to support the training of leaders at the company level and below. Use of a realistic simulation at those levels could augment a leader's experiences from training. Since developing such a simulation is a complex task, to reduce some complexity its development can start at a mid-level such as platoon level. Units that have the contingency mission to deploy anywhere in the world are the airborne, ranger and light infantry units. Thus, a simulation for a light infantry type unit seems most appropriate.

Two ingredients are paramount in a simulation for the light infantry platoon: a desire to realistically represent the information flow and to realistically portray the battlefield environment to the user of the simulation. Information flow is actually a subcomponent of a command and control system in terms of reports and orders.
1. Command and Control - The Unifying Thread

Organizations consist of people, procedures and equipment. The people use the equipment and procedures to accomplish a mission. The ingredient that integrates these into an organization and prevents chaos is command and control: the bonding that holds the organization together on the battlefield. A better understanding of this concept is obtained from the author's modified form of Lawson's Command and Control Process Model in Figure 1 [Ref. 5:p. 24].

Orr introduces and explains Lawson's model in Combat Operations C³I: Fundamentals and Interactions. The Sense, Process, Compare, Decide, and Act (S-P-C-D-A) functions are unchanged from Lawson's model. Two modifications have been made. First, the inclusion of higher and lower level forces and where they interface with the model is shown. Secondly, the dotted box around the S-P-C-D-A labeled "PERSON" is added. All external input to the person box occurs through the Sense function. All output of the "PERSON" box occurs at the Act function as reports, orders, or action on the environment. In this form the model can be used at any level to represent command and control as it relates to the individual. Depending on what level one examines, the only thing that changes is the definition of the lower levels, higher levels, and the environment. This model provides a framework for modeling information flow and the Command and Control process in the Light Infantry Platoon Combat Model.

2. Bounding the Problem

To get a better understanding of the Command and Control (C²) process and how it relates to the light infantry platoon, the "onion skin"
C2 System Bounding technique introduced by Sweet [Ref. 6:p. 11] is useful. Figure 2 shows the "onion skin" as applied to the platoon command and control system. Of particular interest are the four boundaries:

- Outside the platoon force boundary but within the platoon's environment boundary are the terrain, weather, adjacent and higher friendly units and enemy forces.
Figure 2. Bounding the Problem

- Outside the C2 system boundary but within the force boundary are the platoon's organizational forces and their equipment.
- The C2 system is the platoon command and control system.
- The squad and platoon headquarters command and control systems are subsystems of the platoon command and control system.

This "onion skin" and the Command and Control Process Model provide an understanding of a framework within which to develop the Light Infantry Platoon combat model.

As mentioned, the development of such a combat model is a complex task, much beyond the scope of this thesis. In order to develop
such a simulation, there is a need to determine the feasibility of developing an inexpensive method to display the battlefield environment to the potential user: the platoon leader. Specifically, there is a need for a realistic display of the terrain and environment of the battlefield. If this task can be accomplished, then the feasibility of developing a light infantry combat model that can be available to leaders several times a month, not just once or twice a year, can become a reality.

B. THE DESIGN CHOICES

Two design choices are paramount to the development of a display for a light infantry combat model due to the constraints they impose. One is the target hardware and the other is the software programming package. The target hardware is the microcomputer based on the Intel 8086 family of processors. The software package is Turbo Pascal 5.5 Professional which consists of Turbo Pascal, Turbo Assembler and Turbo Debugger. There are several reasons for these choices.

- The microcomputer is readily available to most potential users.
- Numerous references have been written with Pascal as the discussion language.
- There are software libraries for Turbo Pascal code.
- The software package is inexpensive. Its list price is only $250.00.
- The computer hardware is inexpensive, especially when compared to a graphics workstation. Graphics workstations can cost anywhere from $20,000.00 to $100,000. A personal computer only costs $1,000.00 to $8000.00 depending upon the configuration.

Before discussing the selection of the terrain representation, a description of the hardware and software is in order.
1. The Microcomputer

The microcomputer based on the Intel 8086 family of processors has been in use since the early 1980's when IBM introduced the first personal computer. The Intel 8086 family consists of the 8088, 8086, 80186, 80286, 80386, and 80486 processors. All are backward compatible to the 8088 [Ref. 7:p. ix].

The operating system most common for these computers is the Disk Operating System (DOS). A significant limitation of DOS is the ability to address only one megabyte of memory. Of this one megabyte, less than 640 kilobytes are available for program use. There are ways around that barrier, but that topic is beyond the scope of this thesis [Ref. 8:p. 7]. There are several graphics adapters available for IBM compatible microcomputers. The one of interest in this thesis is the Enhanced Graphics Adapter (EGA). With this adapter and a suitable color monitor, the microcomputer can display up to 640 horizontal by 350 vertical pixels in 16 different colors. With 128 kilobytes of memory installed, the EGA in graphics mode can utilize a two page capability. This is useful for drawing to one page while displaying the other. Once drawing is completed, the pages can be flipped to give an instantaneous change in display. This is a technique referred to as page flipping [Ref. 9:p. 105].

The majority of graphics cards for the microcomputer rely on the microprocessor to perform the necessary calculations for display graphics. This setup is quite a limitation when compared to graphics workstations which have built in hardware to take some of the load off the main processor. Since the EGA does not take any load off the main processor, algorithms and code organization are critical to performance.
Thus, the two primary concerns about the microcomputer are the constraints put on it by EGA graphics card and its operating system. The available program memory is limited to less than 640 kilobytes. The resolution of display is limited to 640 by 350 pixels in resolution and only 16 colors can be displayed.

2. Programming Language Software

The programming language chosen for this task was Object-Oriented Turbo Pascal 5.5. This version of Turbo Pascal provides the use of object-oriented programming and a fairly comprehensive graphics unit. The graphics unit greatly facilitates the development of a graphics intensive program. The use of the object-oriented programming methodology will greatly enhance later development of the full combat model as soldiers, squads and platoons are defined as objects.

Turbo Pascal has the capability to link with Turbo Assembler. This capability is well documented in references on Turbo Pascal and provides the flexibility to use assembly language routines where needed to enhance speed of execution. Speed of execution is especially critical in graphics operations since slow graphics operations mean a slow display.

Turbo Pascal has some disadvantages. Code written in Turbo Pascal does not transfer to a mainframe computer without having to rewrite the code due to incompatibilities of Turbo Pascal with standard Pascal. Additionally, Turbo Pascal does not provide a compiler that uses the 32 bit capability of the Intel 30386 processor. Even with these disadvantages, Turbo Pascal 5.5 provides more capabilities than liabilities.
C. REQUIREMENTS OF THE BATTLEFIELD ENVIRONMENT MODEL

The requirements for simulating the battlefield environment in a high resolution model, such as the light infantry combat model, are divided into three categories: what the terrain model should theoretically represent, requirements imposed by the purpose of the light infantry platoon combat model, and the requirements imposed by the constraints of the computer system. These categories represent the total requirements of the terrain model.

The three theoretical requirements for simulating the battlefield are listed below:

- The environment model must provide a terrain profile that allows for calculation of the existence or nonexistence of Line of Sight (LOS) between individual entities on the battlefield.
- The environment model must provide a representation of the terrain surface, vegetation, and man-made features so that concealment from observation, cover from direct fire weapons, and mobility can be determined.
- The environment model must provide a representation of the atmosphere over the battlefield in terms of light conditions, weather, and obscurants such as smoke and fog.

A model of the battlefield environment that satisfies these three theoretical requirements is needed for a high resolution model. [Ref. 10:p. 3-1]

The intended use of the combat model into which this environment model will be integrated identifies two additional requirements.

- The environment model must provide for rapid creation of different environments, thus providing the capability to simulate battlefields anywhere in the world. Light Infantry forces need to train for world-wide deployment to accomplish their mission.
- The display of the terrain representation for the environment model must provide a realistic display that does not confuse the user. In
particular, it should make use of three dimensional graphics and present a view as if the user is at that location on the ground.

These two requirements are important if the model is to enhance experience of platoon leaders when used for training. If the model is used as a tool for experimentation, the capability to display any situation anywhere in the world will provide the researcher with a flexible tool that does not impose undue constraints.

The target computer system imposes several other requirements on the environment model in addition to the five already mentioned.

- The memory requirements of the representation cannot exceed 200 kilobytes. This will allow approximately 320 kilobytes of memory for the combat model program.

- The complexity of the display must be minimized in order to keep the time to draw the terrain on a display in three dimensions to a minimum. A draw time over ten seconds is unacceptable.

These last two requirements become constraints on the design of the model.

The seven requirements presented provide for a realistic three dimensional display of any desired terrain. An implementation that satisfies these requirements will provide the capabilities needed for the purpose of the light infantry combat model. The difficult task is transforming these requirements into a usable product. The next chapter addresses the selection of a terrain representation and its capabilities that makes this transformation possible.
A. SELECTION OF A TERRAIN REPRESENTATION

The method of representing terrain has a significant impact on the capabilities of any combat model. It affects the ability to determine geometric line of sight between two entities on the battlefield. Also, since the computer will have to make line of sight calculations between all entities on the battlefield at specified intervals, the speed with which the computer can accomplish this calculation becomes critical. Finally, since the requirement is to present the terrain in three dimensional graphics, the method chosen will affect display time. On the microcomputer, longer display draw times imply more load on the microprocessor in order to accomplish display calculations instead of battle calculations. The end result is a slower running simulation.

Due to the requirement to display the terrain in three dimensional graphics, the choice of accepted methods of terrain representation is narrowed to what is known as surface terrain models. A surface model is one that represents the surface of the terrain in such a way that it approximates the true continuous appearance of the terrain. This representation is sometimes referred to as macro terrain. Macro terrain refers to capturing the major detail of the terrain, such as a hill, but not features such as forest, vegetation, and small boulders. A picture of an ideal surface model representation is at Figure 3. Note how this representation captures the attributes of the appearance of terrain.
There are three methods for representing terrain that approximate Figure 3. The three methods are utilized in the Dynamic Tactical Simulation (DYNTACS), the Individual Unit Action (IUA), and the Simulation of Tactical Alternative Responses (STAR) combat models [Ref. 10: pp. 3-8 - 3-9]. Each of these representations are possible candidates for the model.

1. The DYNTACS Terrain Model

The first candidate to represent the macro terrain is that used by DYNTACS. It takes as input the elevation of points that are uniformly spaced at a specified interval. These points are grouped to form squares. Each square is divided into two triangles with a diagonal going from the upper left corner to the lower right corner. This methodology is depicted in Figure 4.

By breaking the square into two triangles, it is possible to represent the square area with two triangular planes, each forming a continuous surface. For example, imagine a table with four legs of unequal
length. One plane (i.e., the floor) will not intersect the bottoms of all four legs simultaneously. Now imagine a table with three legs of unequal length. No matter what the length of those three legs, a plane will intersect the bottoms of all three legs simultaneously.

With the DYNMACS terrain model, these diagonal lines and all lines forming the square are common edges of several triangles. The result is a representation that has facets similar to a cut diamond. It is characterized by discontinuities at the edges. Theoretically, if one makes the triangles small enough, these changes may not be noticeable to the naked eye.

In this representation, the coordinates of the three vertices of the triangle are known. Since the triangle is actually a planar surface when viewed in three dimensions, the elevation of an object located
anywhere on that surface is easily determined using formulas of plane geometry. To determine line of sight between two entities, again geometry is used to determine if a line from the observer to the target intersects any of the triangular surfaces between them. The algorithms for this procedure are clearly documented in The Tank Weapon System. [Ref. 11:pp 57-86]

There are several advantages to the DYNTACS terrain model. First, the model provides the capability to utilize Digital Terrain Elevation Data (DTED) from the Defense Mapping Agency (DMA). DMA produces two levels of DTED, referred to as Level One and Level Two. Both express elevations in meters. A data file of DTED provides the elevations of a matrix of uniformly spaced points. Level One DTED has an approximate spacing of 100 meters. Level Two DTED has an approximate spacing of 30 meters. The DTED format conforms exactly to the requirements for elevation data in the DYNTACS terrain model and provides a rapid capability to generate different battlefields. [Ref. 12:p. 1]

A second advantage of the DYNTACS model is the requirements for memory storage are reduced. As long as the location of the lower left corner and the interval between elevation points are stored as constants, then only the elevation data need to be stored in a matrix. There is no requirement to store a three dimensional coordinate for each elevation point. For a large terrain database, this capability greatly reduces the storage requirements. Most models store the data this way, as does DMA on its DTED files. [Ref. 11:pp. 58-61]
A final advantage of the DYNTACS terrain model is that the three dimensional displaying of polygons is well documented. Any reference on three dimensional graphics addresses this subject. The ability to find such documentation is important when it comes to implementing the method.

The DYNTACS terrain model does have a major disadvantage. It does not take advantage of terrain that may be uniform over a large expanse. Consider a piece of terrain that is relatively flat for several kilometers. Such a piece could easily be represented by only two triangles if unequal spacing of points is allowed. Instead the DYNTACS terrain model will represent this piece of terrain with several hundred triangles.

2. The IUA Terrain Model

A second alternative to representing the macro terrain is the IUA terrain model. Similar to DYNTACS, the IUA terrain model represents terrain as triangular surfaces. Instead of uniform spacing, however, the IUA method utilizes nonuniform spacing. The modeler places the vertices wherever he desires to represent the shape of the terrain. Calculating LOS with this model is similar to DYNTACS with one exception. The calculations are more involved because a determination has to be made as to which triangle the entity occupies, since the spacing of points is not uniform. [Ref. 10:p. 3-9]

Thus, the IUA model offers the major advantage of making use of only those data points necessary to represent the terrain. In locations where the number of data points required to represent the terrain is small, the drawing of the display will be quick. A variant of this representation is what Lee Adams advocates in building a flight simulator for a
microcomputer [Ref. 13:pp. 243-280]. In that variant, any polygonal shape may be used. From this author's examination of several microcomputer games that have three dimensional terrain graphics, it appears to be the method used by them.

There are several disadvantages to this model. First, the ability to incorporate DMA DTED is limited. Without developing a program that can convert DTED to a format for this representation, DTED is of no use. That means that someone has to create the data points for a given piece of terrain. This disadvantage would significantly affect the ability to rapidly develop different scenarios. Another problem, already discussed, was the amount of computation required to determine what triangle the entity occupies. In a similar fashion, LOS calculations would become difficult as a search would be required to determine which triangles are between the observer and target. These disadvantages are significant.

3. The STAR Terrain Model

The third alternative for representing macro terrain is the representation used by the STAR combat model. The STAR terrain model is parametric in nature. Instead of using stored digital data for elevations, the STAR model uses a slightly altered bivariate normal probability density function to represent a hill mass. Several of these equations together can represent a battlefield. To represent a piece of real terrain it is necessary to fit these parametric equations to a contour map. [Ref. 14:p. 7]

The advantage of this representation is that it reduces the amount of storage required to represent terrain. For a ten kilometer by
Page 21 missing from master copy per Samuel H. Parry

Thesis Advisor
NPS
NW 3/27/92
DYNTACS terrain model is the model of choice for representing the macro terrain.

5. Simulating the Micro Terrain

The DTED does not provide high enough resolution to capture the small folds in the terrain. The macro terrain is represented by planes with smooth surfaces. In order to simulate the micro environment, an additional technique is needed. Placing a soldier in a deliberate prepared position is relatively easy; simply change his height. The real issue is a method of representing the somewhat random folds in terrain a soldier or vehicle on the move would be able to find when engaged by an enemy force.

Documentation on the STAR terrain model does not address this issue, but the DYNTACS terrain model does. In the DYNTACS model a random adjustment is made to an object's elevation based on a Monte Carlo process. This adjustment can be a positive or negative adjustment. In order to accomplish this procedure, a normal probability distribution is used. The variance for this distribution is determined from a table of predictions that are output from a separate model: the Environmental Model. A similar technique is appropriate for the terrain model being developed. [Ref. 11:pp. 73-76]

6. Representing Forest and Other Terrain Features

There are three methods of representing forest and other terrain features such as man-made objects:

- Account for all trees and man-made objects individually.
• Assign a code to each triangle that indicates the type of feature, its height, and effect on line of sight. This is similar to the technique used on a hex grid terrain model.

• Represent forest and built up areas by a geometric shape, such as an ellipse, that is fitted to terrain areas as appropriate. The DYNTACS and STAR terrain models use this technique.

Accounting for all features on the battlefield is not an option on a microcomputer unless the number of trees and man-made objects is small. The memory requirements for any substantial number of objects would be prohibitive. Since the model is to be used to represent various locations in the world, some locations will have numerous objects. Thus, this technique can be eliminated.

The remaining two choices are possible solutions for the microcomputer environment. The assignment of a code to each triangle would not only allow the addition of the feature height to the surface height when calculating LOS, but it would also allow for the ability to draw the features in that triangle when it is displayed. The other option, using geometric shapes, is feasible, but would be more difficult to implement in terms of drawing the features. This difficulty becomes more evident when one examines the mechanics of displaying the terrain. This topic is addressed in Chapter V.

Once a method is decided upon, there is still the question of the data source for the forests and man-made objects. One option is to use maps of the area of interest. Another option is to use DMA Digital Feature Analysis Data (DFAD). Similar to DTED, DMA produces two levels of DFAD data: Level One and Level Two. Level One DFAD approximates the density of 1:200,000 scale cartographic products. Level Two DFAD approximates the
density of a 1:50,000 scale cartographic product. Again, due to the microcomputer memory constraints, use of DFAD may be prohibitive. These are alternatives that need further examination beyond the scope of this thesis. [Ref. 15: pp. 1-2]

B. LINE OF SIGHT CALCULATIONS FOR THE DYNTACS TERRAIN MODEL

Since the DYNTACS terrain model is the model of choice for representing the macro terrain, it is appropriate to explain the calculation of LOS. In order to calculate LOS, the elevation of the observer and the target have to be determined. Once this information is determined, a check is made to see if the observer has geometric LOS to the target.

1. Determining Elevation at a Location on the Terrain Model

Calculation of the elevation for a point on the surface of the terrain is relatively easy if the data are stored in the correct format. This format involves arranging the elevation data into an array. In order to cut down on calculations, the elevation data should be divided by the interval between the elevation points before storing in the array. This scaling allows the indexes in the array and the data to be on the same scale.

To illustrate this scaling and the elevation routine, a step by step example is given. This example will be kept simple and will use only a two-by-two array shown in Table I. The coordinate system used in Table I is the left-handed coordinate system. To visualize this coordinate system, imagine you are facing North. If you are standing at the origin, the positive z axis is straight ahead to the North. To your right, or the East, is the positive x direction. Straight up is the positive y axis. It is
the z coordinate that adds the third dimension or depth. This is the world coordinate system that will be used throughout this thesis. Finally, notice that the x and z interval between data points in Table I is 100 meters.

**TABLE I. ELEVATION DATA FOR EXAMPLE**

<table>
<thead>
<tr>
<th>DATA POINT NO.</th>
<th>X COORDINATE (METERS)</th>
<th>Z COORDINATE (METERS)</th>
<th>Y ELEVATION (METERS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>130</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>100</td>
<td>140</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>0</td>
<td>135</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>100</td>
<td>120</td>
</tr>
</tbody>
</table>

If the data in Table I are organized into an array structure, some of the data stored can be eliminated since the points are uniformly spaced. An array that has the same information is at Table II. Notice that the only data stored in the array are the elevation data which have been scaled by dividing by 100. The i index corresponds to the x coordinate divided by 100. The j index corresponds to the z coordinate divided by 100. Table II provides the same information as Table I, but requires less memory for storage. The only data stored are the elevation data. The location in the array provides the other two coordinates. Using this technique, the memory storage requirements are reduced by two-thirds of the requirement for Table I.

The method of data presentation in Table II is the same as if the points were arranged on the ground and the reader was above the ground looking down at the points. The top of the paper is North.
DYNTACS methodology, the two by two array would only represent one square divided into two triangles. Assuming the bottom left corner of this square is referred to as \((i,j)\), the lower triangle would consist of \((i,j)\), \((i,j+1)\), and \((i+1,j)\). The upper triangle would be formed with the triple \((i+1,j+1)\), \((i,j+1)\), and \((i+1,j)\). Substituting \(i=0\) and \(j=0\) into the above triples gives the correct indices into the array for the appropriate triangles.

Continuing the example, assume that an observer is located at world coordinates \(x = 0.02\) and \(z = 0.02\) and a target is located at \(x = 0.8\) and \(z = 0.8\). Target and observer heights are both 0.018. These world coordinates and heights are in hundreds of meters, the same scaling as the array. Does a LOS exist? To answer this question, first, one must determine the elevations at the locations of the target and the observer.

The steps to determining the elevation are:

- Determine the triangle, either the upper or lower, in which the observer or target is located by using equation 1 below.
- If the entity is in the lower triangle use equation 2A below.
- If the entity is in the upper triangle use equation 2B below. [Ref. 11:63]

Equation 1 is a condition statement. It is
If \((x_{\text{obs}}+z_{\text{obs}}) > (i+j+1)\) then observer is on upper triangle 
else observer is on lower triangle.

The values for \(i\) and \(j\) are easily determined by truncating the fractional portion of the \(x\) and \(z\) location of the observer. The remaining integers are the indices. Equation 2A is

\[
y_{\text{obs}} = y_{i+1, j+1} + \{(i+1-x_{\text{obs}})(y_{i+1, j+1} - y_{i, j+1}) + (j+1-y_{\text{obs}})(y_{i+1, j} - y_{i, j})\}
\]

Equation 2B is

\[
y_{\text{obs}} = y_{i, j} + (x_{\text{obs}} - i)(y_{i+1, j} - y_{i, j}) + (y_{\text{obs}} - j)(y_{i, j+1} - y_{i, j})
\]

In these equations, the \(y\) values are the elevation for a location identified by the subscript. The subscripts \(i\) and \(j\) are indices into the array table. To determine the elevation of the target, wherever the formula uses observer information, use target information.

Continuing with the example, using equation 1 reveals that the observer is in the lower triangle \((.2+.2)\) is less than \((0+0+1)\). Since the observer is in the lower triangle, the ground elevation at his location is

\[
1.30 + (0.2 - 0)(1.35 - 1.3) + (0.2 - 0)(1.4 - 1.3) = 1.33
\]

Using the same procedure, the target is determined to be in the upper triangle and its ground elevation is 1.27.
The next step is to add the respective heights of the target and observer to their ground elevations. If micro terrain effects are to be included, then this positive or negative value must be added also. For this example, micro terrain effects will not be included. Therefore, the determined elevations of the top of the observer and target are:

\[
\begin{align*}
E_{\text{observer}} &= 1.33 + 0.18 - 1.51 \\
E_{\text{target}} &= 1.27 + 0.18 - 1.45
\end{align*}
\]

2. Line of Sight Routine

In the last section, determination of the elevations of a target and observer were illustrated. In order to determine if geometric LOS exist between an observer and a target, two additional procedures are required. First, the model needs a procedure to determine where a top down projection of the LOS onto the terrain model intersects the edges of the triangular planes.

Figure 5 presents two views of a situation for determining LOS between an observer and two targets. The side view shows that the LOS exists to target one but not to target two. From the top-down view this is not obvious. What is depicted in the top-down view are the intersections of the LOS projection onto the edges of the triangular planes. The DYNTACS model refers to these edges as "plane departure points." It is at the plane departure points that the maximum and minimum elevations occur. If the elevations of the entry and exit points of the plane are less than the elevation of the LOS line at those points (see side views), then any point between the entry and exit point is below the LOS line. In
other words, all the model needs to check are the plane departure points between the observer and target. If all of these are below the LOS line, geometric LOS exists. Therefore, the model needs a procedure to determine the plane departure points between the observer and the target. [Ref. 11:p. 78]

Figure 5. Line of Sight From Observer to Targets

Once the model determines the plane departure points, it requires a second procedure to determine if LOS exist. The procedure needs to check each plane departure point's elevation against the elevation of the LOS line at the same x and z location. Before doing this, if the triangle has been coded as having vegetation, then the vegetation height must be added to the elevation of the plane departure point. The results will inform the model only that LOS exists or does not exist. To determine if
only a portion of the target is visible, the model can do a second check where the height of the target is only half of its normal height. If LOS does not exist to the target midpoint, then the target is only partially visible. If LOS exist to the midpoint, the model assumes a completely exposed target. Due to the length of the procedures to determine the plane departure points and to check LOS, their algorithms are enclosed in Appendices A and B, respectively. [Ref. 11:p. 83]

C. MOVEMENT MODELING

Mobility over the terrain is a function of several variables; the three most important being slope of the terrain, soil conditions, and type of vegetation. To properly model movement requires the development of a functional equation that relates slope, soil conditions, and vegetation. This equation should result in a percentage of a maximum movement speed. The development of such an equation is beyond the scope of this thesis.

The determination of the slope on the terrain is provided in an equation developed as part of the DYNTACS model [Ref. 11:p. 66]. It solves for the angle of the slope using geometric relationships. The equation is as follows:

\[
\text{Slope} = \sin^{-1}\left(\frac{r_i - r_{i-1}}{d_i}\right); i=1, 2, \ldots, n+1;
\]

where

\[
(p_0, q_0) - (x_{max}, z_{max});
(p_{n+1}, q_{n+1}) - (x_{min}, z_{min});
(p_i, q_i) - \text{plane departure points;}
\]

\[
r_i = \text{elevation at } (p_i, q_i)
\]

\[
d_i = \sqrt{(p_i - p_{i-1})^2 + (q_i - q_{i-1})^2 + (r_i - r_{i-1})^2}
\]
With the above equation, the model can easily determine the slope of the terrain at any location on the battlefield. Determination of the values for soil conditions and vegetation depends on the representation method used in the model. The easiest method is coding the triangle with the values for vegetation and soil conditions as mentioned earlier. A more realistic representation is the use of geometric shapes to map the areas with similar vegetation or soil conditions, but the coded triangle method is faster for determining what codes apply to a given location.

D. MODELING TARGET ACQUISITION

Even though geometric LOS may exist between an observer and a target, its existence does not mean the observer detects the target. There are several reasons in the real world that detection might not occur. They are

- The observer is not looking in the direction of the target.
- The observer cannot distinguish the target from the background.
- Environmental factors may prevent him from detecting the target. For example, there may be fog or dust obscuring the target, or it could be dark.
- The observer is not alert.
- The observer is suppressed by enemy fire.

There are two methodologies for modeling the detection process that take the most important reasons for non-detection into consideration: the Night Vision and Electro-Optical Laboratories (NVEOL) detection model and the continuous looking detection model. According to Hartman [Ref. 10:p. 4-24] the NVEOL detection model is the better of the two methods.
The NVEOL model considers and evaluates the following events in order to determine if detection occurs [Ref. 10:p. 4-24 - 4-25]:

- the emitted or reflected target signature
- transmission of the target signature through the atmosphere
- the orientation of the observer's sensor
- the processing of an attenuated signal by the sensor to form an image
- the viewing of the display image and the response by the user.

By considering all of these events, the NVEOL model allows for an accurate representation of the process of detection and how it is affected by battlefield conditions such as smoke, fog, darkness, etc. In a model that has the luxury of adequate computational power, it is the method of choice.

The second method, the continuous looking model [Ref. 10:p. 4-12] represents the process of detection as the cumulative distribution function of the negative exponential. The parameter for this process is the detection rate which needs to be derived from detection experiments. The advantage to this equation is it keeps the process of detection determination simple. Everything is rolled into the one equation. Different parameters are assigned based on the conditions. Because of its simplicity, it is the method most promising for the personal computer environment. Implementing the continuous looking model in the program would not require a substantially amount of work. The real work will be in getting some valid parameters for the model based on already available data or new experiments.
IV. DISPLAYING THE DYNTACS REPRESENTATION

The last chapter discussed the methodology for developing a personal computer based simulation using the DYNTACS representation for terrain. The intent of this chapter is to illustrate the various considerations and decisions needed to implement a three-dimensional display of the DYNTACS terrain model on a microcomputer. Because of the limitations of the Enhanced Graphics Adapter (EGA), all of the procedures have to be implemented in the software.

To fully explore the feasibility of using the microcomputer, a program was built from scratch. The program created to implement the three-dimensional display of the DYNTACS terrain model has code that is divided into three categories:

- Unmodified code that was adapted directly from existing sources and programs.
- Modified code from existing sources and programs. In this category is code that needed some modifications or translation from another language.
- Code written to implement known algorithms. This category also includes code written as a derivative of known algorithms and created as innovative solutions to a problem.

Although there were some very useful procedures available in the first two categories, the majority of the code for the program is in the third category. Appendix C contains a listing of the interface portion of all units used by the program. The listing classifies the category for the code of each procedure according to the above list.
To insure the reader understands what the program does, it is appropriate to describe its capabilities. After a description of the capabilities, the topic shifts to the discussion of implementation decisions that affect the two most important issues about graphics - speed and realism.

A. PROGRAM OVERVIEW

The graphics program is best described by listing its capabilities and providing a few captured images; however, the black and white images do not do justice to the display. A true assessment of the program can only be obtained by seeing it in action at TRADOC Analysis Command (TRAC), Monterey. Its capabilities are as follows:

- Displays a three dimensional representation of the DYNTACS terrain model in color with moving soldiers.

- Uses a 20 square kilometer terrain database of processed DMA Level 1 DTED (approximately 100 meter spacing). It can move anywhere within this square and displays a view out to three kilometers.

- Has the ability to change viewing angles, viewing altitude, viewing magnification, and viewing direction. The default setting is from the viewpoint of a soldier standing on the ground looking to his front.

- Moves the soldier's viewpoint as the soldier moves, which simulates moving across the terrain.

- On a Dell 25 MHz 386 computer with math coprocessor, a VGA card, a VGA monitor, and cache memory displays one frame per 1.2 to 1.5 seconds. It uses EGA mode, so only a EGA card and monitor are required.

- Displays information regarding current location, heading, and view angle.

- Has the ability to change location and intensity of the light source, to change ambient light conditions, and thus change the shades of color in the scene.
• Using digital halftoning, provides 24 different shades and tones each of red, blue, and green.

• With minor processing of DTED Level 1, can display terrain anywhere in the world for which DTED Level 1 is available. DTED Level 1 does not include cultural features.

• With enhancements, it can be incorporated into a training simulator for small unit leaders; i.e., platoon leaders.

Three figures are provided to illustrate some of the capabilities of the three dimensional display. Figure 6 shows a view from behind a fire team (an element of 4 soldiers) at approximately six feet above the ground; Figure 7 show a side view of the fire team at six feet above ground; and Figure 8 shows a view from approximately 100 hundred feet above the fire team. The reason only a fire team is shown is because that is the largest size of force currently implemented in the display. This size was sufficient to test the display algorithms. The figures only demonstrate a few of the capabilities and potential of the program.

B. GRAPHICS IMPLEMENTATION ISSUES

The user of a graphics program judges its value using two criteria. The first criterion is how fast the program displays the scene. The second criterion is how realistically the program displays the scene. These two criteria, speed and realism, become the major concern in implementing a graphics program. Therefore, it is appropriate to address what the program does to provide realism and speed in displaying the DYNTACS terrain representation.
Figure 6. View from Behind Fire Team
Figure 7. Side View of Fire Team
Figure 8. View from 100 Feet Above Fire Team
1. Providing Realism

The program employs three techniques to support a realistic representation of the DYNTACS terrain model with soldiers. They are:

- The use of the perspective projection technique of displaying three dimensions.
- The use of color filled polygons with hidden surfaces removed instead of a transparent wire-frame representation.
- The use of a shading model that calculates the light intensities in order to determine the appropriate shade of color for the surfaces of the objects.

a. Perspective Projection

Prospective projection displays objects so those at greater distance appear smaller. Using this technique, parallel lines on objects tend to converge. This technique resembles the way people view objects in the real world. If the reader looks at any building, he will notice that parallel lines on the building appear to converge in the distance. This technique provides for a more realistic representation of the terrain model by representing all three dimensions and not just two dimensions. [Ref. 16:p. 184]

The enhanced realism of a three dimensional display comes at a cost. Perspective projection techniques requires substantial mathematical calculations in terms of translating, rotating, and scaling objects to be in the correct location, orientation, and size. It is critical that the program perform these repetitive calculations as efficiently as possible. Fortunately, there are documented techniques for doing these calculations.
efficiently through matrix concatenation of the translation, rotation, and scaling operations [Ref. 16:pp. 220-233].

b. Filled Polygons

There are two options for displaying the images created by perspective projection. One of these techniques is displaying the object as a transparent wire-frame model. The problem with transparent wire-frame objects is that the viewer is sometimes confused as to the definition of the object. This can be overcome, to some extent, by removal of any lines that should be hidden from view. The advantage of the transparent wire-frame model is that it allows quick display generation. [Ref. 17:p. 45]

The second technique of displaying objects is to fully shade the object or color the surfaces. This technique is often referred to as rendering. It provides for more realistic images but greatly increases the display generation time due to the need to color more pixels. Although the display time is increased, the program uses this technique for the effect of the more realistic display. Additionally, the terrain and objects are drawn with hidden surfaces removed.

c. Shades of Color Dependent on Light Conditions

The last technique to enhance realism is to use a shading model to determine the shade of color for filling a polygon. A shading model was adapted in a modified form from Computer Graphics [Ref. 16:276-289]. It determines which shade to use as a function of five variables. They are the light source location, the light source intensity, the ambient light intensity, the reflectivity of the surface, and the orientation of the surface. Descriptions of these five variables are as follows:
• The location of the light source is expressed as a three dimensional vector of unit magnitude pointing to the light source. This vector can be changed to indicate movement of the sun or moon.

• The intensity of the light source is expressed as a value less than or equal to one. This value can be changed to model a cloudy versus clear day.

• The orientation of the surface is expressed as a surface normal vector. The dot product of the surface normal and the light source vector equals the cosine of the angle between the two vectors. When the dot product is negative, the light source is behind the surface and thus, the surface is not illuminated.

• The reflectivity of the surface is expressed as a coefficient less than or equal to one.

• The intensity of ambient light is expressed as a value less than one. The sum of the intensity of ambient light and the intensity of the light source must be less than one.

These five variables are used in the following equation to determine the intensity of the reflected light from the surface:

\[ I = k_r(I_a + I_s(N\cdot L)) \]

where

- \( I \) - Intensity of Reflected Light
- \( k_r \) - Coefficient of Reflectivity for Surface
- \( I_a \) - Intensity of Ambient Light
- \( I_s \) - Intensity of Light Source
- \( N\cdot L \) - The Dot Product of Surface Normal and Light Source Vector

After determining the intensity of the reflected light, a procedure is called to fix which of twenty four shades to use for shading that surface. This shading model provides more realism to the display by shading objects similar to the way they appear to be shaded when viewed by the human eye.
2. Providing Speed

When sitting in front of a computer display waiting for information to be displayed, the difference between a couple of seconds and ten seconds can seem like eternity to a user. A program that requires user involvement must have fast displays or the user will become frustrated as he continually waits for the computer to do its job. Since this terrain model is being built for interactive use by a user, speed is essential.

Many techniques and algorithms are being incorporated in this program to enhance the speed with which the display is generated. Of these, there are three algorithms that account for the majority of the results to date. They are:

- A specialized triangle filling routine.
- The soldier sorting algorithm.
- The integrated display algorithm.

Although there are many other areas that affect the speed of the display, these are the areas that have received the most effort. All are innovative solutions or application of known algorithms.

a. The Specialized Triangle Drawing Procedure

(1) The Need. As mentioned earlier, drawing filled polygons requires more time than drawing a transparent wire-frame. For more realism, it was decided to not only use filled triangles, but to simulate more shades of a given color by using digital halftoning. Digital halftoning is a technique to generate more shades by creating patterns of pixels with
two slightly different tones of the same color. Using this technique, four
tones of the same color can easily generate 16 different shades of a color.

Turbo Pascal 5.5 provides a procedure in its graphics
unit called Fillpoly that fills polygons with a specified color or a pattern.
Unfortunately, the pattern is restricted to a specified color and the
background color of the display. To employ digital halftoning requires
drawing the polygon twice, once with one pattern and the first tone, then,
a second time with the complementing pattern and the second tone. Since
the procedure's source code is not available, altering this limitation is not
easily done.

Initial use of this "draw it twice" technique provided a
good display, but at a terrible cost. On the Dell 386 computer, the time
to generate one picture was timed at just under 30 seconds. These initial
results forced a re-evaluation of the situation. Should the digital
halftoning technique be dropped or should a new procedure to draw the
triangles be created? The decision was to search for or design a better
procedure.

Most computer graphics references describe two
algorithms for area filling of polygons that are suitable for the needs of
this program. They are the border fill algorithm and the scan line
algorithm. The border fill algorithm [Ref. 9:pp. 252-263] can be described
as follows:

- Trace the border and create an ordered list of the border pixels.
- Perform a scan of the interior, checking for holes in the region
defined by another polygon.
• Connect the left and right boundaries on each scan line by filling with a horizontal line.

The scan line algorithm [Ref. 16:pp. 83-90] can be described as follows:

• Trace the borders in a color that is different from other colors on the screen.

• Scan a rectangular region that contains the polygon in order to determine the left and right border pixel on each line.

• Connect the left and right border pixel by filling with a horizontal line.

Both of these algorithms are general purpose algorithms. They work with many types of polygons if implemented correctly. With their general purpose design, the user gets flexibility at the expense of additional computation time. If a procedure needs to be prepared for use on any different number of polygons, then these are the tools that should be utilized.

The DYNTACS terrain representation only needs the capability to draw triangles, but a large number of them. Only other objects, such as people, require the capability to draw various polygons. Since a four kilometer square with 100 meter spacing requires 3200 triangles, any speed gained by a special procedure for triangles is worth the effort of designing it. As a bonus, other polygons can be made by putting several triangles together.

(2) The Specialized Algorithm. The development of a specialized algorithm is straightforward. A triangle is defined by three points connected by lines. Additionally, recall that the general polygon
algorithms determined the border pixels, then connected them with a horizontal line. A specialized triangle routine has the following steps:

- Sort the three points of the triangle such that they are ordered top to bottom, and in case of the same $y$ values, order them left to right. Refer to the ordered points as point one, two, and three. Refer to the lines between them as Line 1-2, 1-3, and 2-3.

- Incrementally draw lines 1-2 and 1-3, one scan line at a time in order to determine the left and right border pixels of the triangle for that scan line. Fill that scan line between the left and right border pixels.

- Continue incrementally processing line 1-2 and 1-3 for each scan line until point two is reached.

- Now, incrementally process line 2-3 while continuing on line 1-3 where it stopped with the previous line. Continue until reaching point three, the end of the lines.

This algorithm capitalizes on the fact that as the computer draws the lines from one point to another, it visits the pixels that comprise the border of the triangle. Instead of returning to these pixels later, as the scan line algorithm does, it makes the determination as part of drawing the left and right lines incrementally.

(3) Implementing the Special Algorithm. As mentioned earlier, it is essential that the algorithm be implemented as efficiently as possible. Since speed is critical, the procedure needs to push the capabilities of the hardware to the maximum. Michael Abrash makes a statement in Zen of Assembly Language that is appropriate. He states:

Comment your code, design it carefully, and write non-time critical portions in a high level language, if you wish, but when you write portions that interact with the user or affect response time, performance must be your paramount objective. Assembler is the path to that goal. [Ref. 18:p. 9]
A high level language, such as Pascal or ADA, is dependent upon the compiler to optimize code. Compilers for these languages are like the general purpose filling algorithms, they get the job done rather effectively, but sometimes not using the most efficient code. This deficiency is the price the programmer pays for the ease of implementation using a high level language.

A special purpose algorithm or code written in assembly language, if written properly, will get it done more efficiently than if written in a high level language. Using assembly language, the programmer can access hardware directly without having to use the Disk Operating System (DOS) or the Basic Input Output System (BIOS) routines. Some of the DOS and BIOS routines are not as efficient as they could be. By accessing the hardware directly, the programmer bypasses the inefficient DOS and BIOS routines and obtains better performance. In time critical code, such as graphics producing code, this technique results in substantial performance gains at the cost of increased programming complexity and development time.

To implement the special purpose algorithm efficiently, two techniques need to be integrated: line drawing and area filling. The most efficient way to draw lines for a personal computer display is to use Bresenham's Algorithm [Ref. 9:p. 168]. To accomplish area filling, the use of horizontal lines is the most efficient method on the personal computer display [Ref. 9:pp. 168-169].

Integrating these concepts into one algorithm in assembly language is a complex, but necessary task if better performance is desired. Needless to say, they are implemented in assembly language for this
program. A listing of the FillTri routine with some additional explanatory comments is included in Appendix D.

(4) The Results. After implementing the special purpose algorithm in assembly language, the program was run again to determine what improvements were obtained. On the Dell 386 25 MHz computer, the result was a display rate of just under five seconds. This display time was a vast improvement from the initial time of 30 seconds with the FillPoly procedure.

b. The Soldier Sorting Algorithm

Having developed a model that displays terrain in a reasonable amount of time, the next step is to add soldiers. After spending much time and effort to obtain the performance results mentioned above, it is essential that this step be equally efficient. The adding of soldiers to the display has the potential to increase display time significantly if not implemented in an efficient manner.

(1) Nature of the Problem. To fully understand the problem, one area needs to be addressed: the manner in which terrain is drawn by the program. In particular, the method employed must be designed so hidden surfaces are not displayed.

The method the terrain program uses is known as the painter’s algorithm. This algorithm gets its name from the way a painter paints a picture. First the most distant objects are drawn. Additional objects are added to the picture working closer toward the viewpoint. In the process, some or all of a previous object is covered up by the closer objects. The terrain displaying program uses this algorithm. It
starts with the most distant row or column of the terrain matrix and works its way back to the closest row or column. Thus, objects that should be hidden from view are covered up on the screen by the closer objects.

Implementing this algorithm is relatively simple for the terrain by itself; simply draw the columns or rows in the correct order. Placing soldiers into this process makes it more difficult. Their location is dynamic and changes from display to display. Drawing them in the correct sequence requires two considerations. First, the program must determine which triangle they occupy. This is easy to determine because the elevation routine explained earlier developed an algorithm for determining what triangle a given point occupies. The other consideration is determining the drawing order for two or more soldiers when they are in the same triangle. This problem is one of sorting.

Sorting problems have the potential to become time consuming. A bad sort algorithm can cause an otherwise efficient program to become inefficient. With this in mind, a search of several references provides some elegant solutions to the problem.

(2) Binary Search Trees. The best technique for this particular problem is to insert the soldiers distance from the viewpoint (depth) and a pointer to any other information required for the display into a binary search tree [Ref. 19:pp. 198-210]. A slight modification to the Binary Search Tree (BST) is in order. Instead of sorting by smallest values, the program needs to sort by the larger values. In this way, the more distant soldiers, in terms of depth, will be retrieved from the tree first.
As an added bonus, using Turbo Pascal 5.5, it is possible to implement the BST so that the objects inserted into the tree can be different, as long as they are descendants of the same object [Ref. 20:pp. 265-281]. Normally, due to Pascal's strong typing of variables, it is not possible to mix different types of pointers in a Binary Search Tree. By using objects, it is possible to create a BST that sorts not only the soldiers, but any other object that is to be displayed, i.e., a tank. This method of implementation of the tree is used by the program, allowing for future expansion.

(3) The Results. In implementing the BST, the amount of code increases only slightly. The overall effect in program running time with the soldiers added to the display is minimal. The display time on the Dell computer only increases approximately one-tenth of a second over the time reported in the last section when displaying four soldiers in the same triangle.

c. The Integrated Display Algorithm

In the search for better performance, the code has been scrutinized for inefficiency. Some initial improvements have been made in the organization and structure. This reorganization resulted in a more integrated display algorithm that capitalized on some of the capabilities of the programming language. The program currently uses the following integrated display algorithm:

- Check to determine if the current terrain data in memory provides at least two kilometers of display depth. If not, load more terrain data.
- Set the video write page to the page hidden from view on the display.
• Move soldiers and display viewing location from current locations as appropriate.

• Set trigonometric variables used in the transformation, rotation, scaling and perspective projection formulas as global variables.

• Determine which one of the fourteen drawing sequences to use.

• Identify which squares of the 40 x 40 array of data points need to be processed for the display using the left and right boundaries of the field of view.

• Perform three dimensional to two dimensional transformation on the terrain data points.

• Sort soldiers by inserting into the appropriate Binary Search Tree. One empty tree exist for each of the triangles in the 40 x 40 array.

• Draw triangles in the correct order. Draw any soldiers that are in a triangle after drawing that triangle and before drawing the next triangle.

• Display information in the information part of the display.

• Flip the hidden page to the display page, thus refreshing the display with a new frame.

• Free any memory that was allocated for the BST's.

• Return to beginning of algorithm.

This algorithm continues to be refined as more tests are done to check performance on the program. The results using this algorithm, however, are a significant improvement over the original attempt with the FillPoly procedure. The display now refreshes at a rate of between 1.2 and 1.5 seconds. There are still some areas that can be improved, but the program is definitely pushing the edge of the capability of the personal computer.
V. ENHANCEMENTS

This chapter discusses enhancements to the current capabilities of the terrain model that are essential for its incorporation into a light infantry platoon combat model. These enhancements involve:

- Displaying cultural features such as forest, buildings, etc.
- Line of Sight calculations
- Modeling Target Acquisition
- Building a detection list.

These enhancements were not implemented as part of the terrain model because of the difficulty in implementing the three-dimensional display. The time required to implement the display took more time to develop than initially planned.

Without the last three enhancements, high resolution simulation of combat is impossible. Display of cultural features is not necessary, but without them, the battlefield display will be unrealistic. Terrain void of vegetation will significantly decrease the realism of the display. Each of the enhancements will be addressed individually.

A. DISPLAYING CULTURAL FEATURES

In Chapter III, cultural feature modeling was discussed briefly. Two techniques were presented as possible solutions; the use of codes assigned to each triangle and the alternative of using geometric shapes to "map" the
vegetation to the terrain. Of these two methods, the use of codes for each triangle is the simplest to implement in terms of writing the code.

Including cultural features would not be very difficult if it were not for the requirement to display them. Part of the problem is the method the program uses when drawing the terrain; it draws the triangles in an order that "paints" over areas that are hidden (the painter's algorithm). As long as the base of cultural features do not extend outside of a triangle, they can be drawn in the correct order by placing them in the Binary Search Tree with the soldiers. If the feature occupies more than one triangle, it must be subdivided into pieces that are assigned to the respective triangles they occupy. Otherwise, the painter's algorithm will not work (i.e., objects that should be hidden are no longer hidden).

Due to the requirement to draw a triangle and its associated occupants (i.e., people or features) one after the other, it would be very difficult to utilize the method of geometric shapes to display the features. If geometric shapes are utilized, the program would require a procedure to interpret the shapes and determine which features each triangle requires as it draws the triangles. When a triangle is drawn, determination of whether it has any terrain features must be made before drawing the next triangle. If people are in that triangle, this issue has to be resolved before removing them from the BST and displaying. The complexity of implementing such an algorithm makes it prohibitive for the personal computer environment. Because of the painter's algorithm, the program is forced to utilize coded triangles to display the features.

An innovative solution to the cultural feature display problem would be to utilize the code for each triangle to represent not only the type of
cultural feature, but also its location. For example, assume there are three trees in a given triangle. The programmer could have several codes that represent three trees; each with the trees in a different location of the triangle. The encoding of such information would be a time consuming process, but will save significantly on memory requirements.

The topic of displaying cultural features deserves significant research in order to implement it properly. Of the four enhancements, it is the most difficult to implement and is deserving of a separate thesis. Implementation of the enhancement would probably take three to six months of effort.

B. THE REMAINING ENHANCEMENTS

The remaining three enhancements are so closely related that they should be implemented as a group. The modeling of the acquisition process needs the LOS determination and the capability to store its list of detections.

1. Adding Line of Sight Calculations

The LOS calculations were discussed in Chapter II and are documented in Appendix B. To implement LOS determination in the program will not require a significant amount of effort. The geometric calculations are straightforward. Of the four enhancements, it is the easiest to implement. It would take one to four weeks to implement, depending upon the programmer.

2. Adding Detection Calculations

The detection calculations consist primarily of implementing the negative exponential in an algorithm that determines if detection occurs.
In a time step model, the negative exponential would give the probability of a detection given LOS in the amount of time of the time step. Then the computer could generate a random number between zero and one. If the random number is less than the probability value obtained from the negative exponential, then detection occurs. If the random number is greater than the probability value then detection does not occur. As mentioned in Chapter II, use of this detection model is rather straightforward and easy to implement. The difficulty lies in determining the parameters for the negative exponential. To implement the negative exponential would only take about one week. Researching the parameters to be used in the simulation could take several weeks to several months depending upon the accuracy desired.

3. Building a Detection List

Once a detection has occurred, the model will need to store this fact in a list. Because the computer processes detection determinations on targets sequentially (only one at a time), it needs to build a list of detections from a given detection cycle. Then, it needs to process this detection list to make determinations of courses of action. To implement the capability to store such a list is only a problem of using well documented techniques for a list data structure. Implementation of this enhancement should only require a week or two of work.
VI. CONCLUSIONS

The intent of this thesis was to develop a program to display a three-dimensional representation of a terrain model and soldiers on a personal computer. From this research, there are several conclusions:

- The EGA card of the personal computer provides limited support for graphics programming. Graphics routines have to be implemented in software and for enhanced speed, they have to be implemented in assembly language.

- The limitation of usable memory by DOS on the personal computer severely constrains the display program in terms of the size of terrain that can be loaded in memory at one time.

- The DYNTACS terrain representation provides a feasible methodology for implementing a realistic three-dimensional display of the terrain and provides the capability to use DMA DTED data.

- The representation of cultural features (i.e., trees) is relatively straightforward until one examines the requirement to display them. The display of cultural features is a complex problem and deserving of further research and development.

- Routines provided in programming packages (i.e., FillPoly) are good general purpose routines but may not meet the requirements of a program. A specialized routine tailored to the needs of the program can greatly improve the speed with which a program generates a display (five versus thirty seconds per frame).

- The use of a Binary Search Tree to sort the order of displaying the soldiers had only minimal effect on the display time. The use of objects allows the use of mixed objects in the BST as long as they are all descendants of the same object. This allows for future expansion as tanks, helicopters, and other objects are added to the display.

- The development of an integrated display algorithm further improved the display time (1.2 versus 5.0 seconds per frame).

The results of this thesis indicate that it is possible to develop a display for a light infantry combat model on a personal computer that
provides a realistic image in three dimensions. From the programming standpoint, the graphics programming is the most difficult part of a light infantry combat model. From the research standpoint, there is much work to be done in order to fulfill the goal of developing the light infantry platoon combat model.

The enhancements that still need to be added to the display program before full development of the combat model were discussed in Chapter V. Three of these enhancements are necessary in order to model the target acquisition process: Line of Sight calculations, detection, and creation of a detection list. These three are documented and are relatively straightforward to implement. The fourth enhancement, adding of cultural features, is the most difficult to implement and is worthy of additional research. As stated earlier, the use of the painter's algorithm for hidden surface removal and the memory constraints of the target computer make the integration of displaying cultural features a complex task.

Once these four enhancements are implemented, the modified Lawson's Command and Control model for the individual and the "onion skin" diagram provide a framework with which to continue the development of the program until it becomes a combat model. An alternative path of development is to utilize the display with enhancements to conduct experiments to measure the effects of human factors on leader decision making.

The development of a light infantry platoon combat model using the personal computer can greatly enhance the experience and training of platoon leaders. With such a model, units would be better prepared for deployment on a contingency mission such as Operation "Just Cause."
Command and Control of platoons could be greatly enhanced through better trained leaders. A platoon that has better trained leaders results in a better trained company, which in turn means a better trained battalion.
APPENDIX A. PLANE DEPARTURE POINTS

This appendix is referenced in Chapter III of the thesis in the section regarding Line of Sight determination. The intent of this appendix is to outline the algorithm for determining the plane departure points between two locations. Plane departure points are the entry and exit points of the triangular planes along a constant heading from the beginning point to the end point.

The routine that would be developed based on this algorithm would be utilized by a movement routine that needs the plane departure points for calculating slopes along a path between two points. This will be necessary for calculating speed of movement.

This algorithm is adopted from The Tank Weapon System which is referenced in the thesis. Except for changes in notation so as to insure consistency with the thesis notation, the algorithm is the same as described in the above reference on pages 64-65.
A. NOTATION

Notation used in the algorithm is defined below:

\[(x_d, z_d) = \text{starting point} \]
\[(x_a, z_a) = \text{ending point} \]
\[\{P_i, q_i\}, i=1, 2, \ldots, n = \text{the set of plane departure points} \]
\[[x_a] = \text{the greatest integer less than or equal to the real value of } x_a \]

B. THE ALGORITHM

The algorithm consists of three components. The first component calculates the plane departure points along the vertical terrain lines. The second component calculates the plane departure points along the diagonal terrain lines. The third component calculates the plane departure points along the horizontal terrain lines. Once these three components have been used to determine the plane departure points, all that remains is to sort them in the order they would be visited going from the start point to the end point.

**Vertical Terrain Lines**

1) If \( x_a > x_d \) then \( \alpha = 0; \beta = -1 \)
   else \( \alpha = +1; \beta = +1 \)
2) \( m = \frac{z_d - z_a}{x_d - x_a} \)
3) \( p = \lfloor x_a \rfloor + \alpha \)
4) If \( \beta p \geq \beta x_d \) then go Step 7 below
5) \( q = m(p - x_d) + z_a \)
6) Place \((p, q)\) on Plane departure list; \( \alpha = \alpha + \beta \); Go Step 3
Horizontal Terrain Lines

7) If \( z_d \geq z_a \) then \( \alpha = 0; \beta = -1 \)
   else \( \alpha = +1; \beta = +1 \)
8) \( q = [z_d] + \alpha \)
9) If \( \beta q \geq \beta z_d \) then Go Step 12
10) \( p = \frac{1}{m} (q - z_a) + x_a \)
11) Place \((p, q)\) on plane departure list; 
    \( \alpha = \alpha + \beta; \) Go Step 8

Diagonal Terrain Lines

12) If \( x_d + z_d < x_a + z_a \) then \( \alpha = 0; \beta = -1; \)
    else \( \alpha = +1; \beta = +1; \)
13) \( b = ([x_d] + [z_d] + \alpha) \)
14) \( p = \frac{x_m - z_a + b}{1 + m} \)
15) \( q = -p + b \)
16) If \( \beta b \geq \beta (x_d + z_d) \) then all departure points
    are identified; Go SORT
16) Place \((p, q)\) on plane departure point list; 
    \( \alpha = \alpha + \beta; \) Go Step 13
APPENDIX B. LINE OF SIGHT

This appendix is referenced in Chapter III of the thesis in the section regarding Line of Sight determination. The intent of this appendix is to outline the algorithm for determining whether or not geometric Line of Sight (LOS) exist between two entities. Plane departure points are the entry and exit points of the triangular planes along a constant heading from the observer location to the target location.

The routine that would be developed based on this algorithm would be utilized by the model to build a list of potential targets. A prerequisite for detection is that LOS exists. From the list of targets to which LOS exists, the detection model would determine if detection occurred.

This algorithm is adopted from The Tank Weapon System which is referenced in the thesis. Except for changes in notation so as to insure consistency with the thesis notation, the algorithm is the same as it is described in the above reference on pages 80-83.
A. NOTATION

Notation used in the algorithm is defined below:

\[(x_0, z_0) = \text{location of the observer}\]
\[(x_t, z_t) = \text{location of the target}\]
\[(p, q) = \text{coordinates of intersection between a terrain line and a plane parallel to the y axis}\]
\[\lfloor x \rfloor = \text{the greatest integer less than or equal to the real value of } x\]
\[y = \text{the macro terrain elevation at } (p, q)\]
\[y' = \text{the macro terrain elevation adjusted for vegetation height}\]
\[h_f = \text{tree height in a forested area}\]
\[h_v = \begin{cases} 
  h_f & \text{if } (p, q) \text{ is in forested area} \\
  0 & \text{if } (p, q) \text{ is not in forested area}
\end{cases}\]

B. THE ALGORITHM

This algorithm checks geometric LOS in three parts. First, it checks to determine if LOS exists over the vertical terrain lines. Next, it checks to determine if LOS exists over the horizontal terrain lines. Last it checks to determine if LOS exists over the diagonal terrain lines. If a LOS check fails during any one of the checks, LOS does not exist and the algorithm exits.
Vertical Terrain Lines

1) Determine $z_o$ and $z_r$ using elevation procedure
2) If $x_i > x_o$ then $\alpha = 0; \beta = -1$
   else $\alpha = +1; \beta = +1$
3) $i = ([x_r] + \alpha)$
4) If $\beta i > \beta x_i$ then go Step 14 below
5) $q = \frac{x_r - x_o}{x_t - x_o}$
6) $j = [q]$
7) If $(i, q)$ is in forest, set $h_v = h_o$
   Else $h_v = 0$
8) $y' = \frac{y_t - y_o}{x_r - x_o}$
9) If $y' > \max(y_{i,p}, y_{i,j-1})$ then $i = i + \beta$, Go Step 4;
   Else Go Step 10
10) If $y' < \min(y_{i,p}, y_{i,j-1})$ then LOS does not exist, SO EXIT
11) Calculate elevation $y$ at $(i, q)$ using elevation procedure
12) If $y > y'$ then LOS does not exist, SO EXIT
13) $i = i + \beta$; Go Step 4
Horizontal Terrain Lines

14) If \( z_i > z_o \) then \( \alpha = 0; \beta = -1 \)
   else \( \alpha = +1; \beta = +1 \)
15) \( j = ([z_j] + \alpha) \)
16) If \( \beta j > \beta z_j \) then go Step 26 below
17) \( p = \frac{x_i - x_s}{z_i - z_o} (j - z_o) + x_o \)
18) \( i = [p] \)
19) If \( (p, j) \) is \( \in \) forest, set \( h_v = h_i; \)
   else \( h_v = 0 \)
20) \( y' = \frac{y_i - y_s}{z_i - z_o} (j - z_o) + y_o - h_v \)
21) If \( y' > \text{Max}(y_{k, p}, y_{k, j+1}) \) then \( j = j + \beta, \) Go Step 16;
   else Go Step 22
22) If \( y' < \text{Min}(y_{k, p}, y_{k, j+1}) \) then \( \text{LOS does not exist SO EXIT} \)
23) Calculate elevation \( y \) at \( (p, j) \) using elevation procedure
24) If \( y > y' \) then \( \text{LOS does not exist, SO EXIT} \)
25) \( j = j + \beta; \) Go Step 16
Diagonal Terrain Lines

26) If \( x_i + z_i < x_o + z_o \) then \( \alpha = 0; \beta = -1; \)
else \( \alpha = +1; \beta = +1; \)
27) \( b = ([x_i] + [z_i] + \alpha) \)
28) If \( \beta (x_i + z_i) < \beta b \), the observer and target are intervisible SO EXIT

\[
b = z_o + x_o \left( \frac{z_i - z_o}{x_i - x_o} \right)
\]

29) \( p = \frac{z_i - z_o}{1 + \frac{z_i - z_o}{x_i - x_o}} \)

\( i = [p] \)
30) \( q = -p + b \)
\( j = [q] \)
31) If \((p, q)\) is \( \in \) Forest then \( h_v = h_f \) Else \( h_v = 0 \)
32) \( y' = \frac{y_i - y_o (q - z_o) + y_o - h_v}{z_i - z_o} \)
33) If \( z' > \text{Max}(y_{i+1}, y_{j+1}) \) then \( b = b + \beta \); Go Step 28
   Else Go Step 34
34) If \( z' < \text{Min}(y_{i+1}, y_{j+1}) \) then LOS does not exist SO EXIT
35) Calculate elevation \( y \) at \((p, q)\) using elevation procedure
36) If \( z > z' \) then LOS does not exist SO EXIT
37) \( b = b + \beta \); Go STEP 28
APPENDIX C. INTERFACE LISTINGS

This Appendix is referenced in Chapter IV of the thesis in the section regarding Displaying the DYNTACS representation. The intent of this appendix is to provide the reader a feel for the complexity of this program by providing a listing of the interface portions of all units used by the main program to display the terrain with soldiers in three dimensions. Each of the procedures in these listings is identified as belonging to one of three categories:

- Unmodified code that was adapted directly from existing sources and programs. Code in this category is labeled Unmodified.
- Modified code from existing sources and programs. In this category is code that needed some modifications or translation from another language. Code in this category is labeled Modified.
- Code written to implement known algorithms. This category also includes code written as a derivative of known algorithms and created as innovative solutions to a problem. Code in this category is labeled New Code.

The listings of the units and the main program follow on succeeding pages.
(***************The Shades Unit***************

unit shades;
interface
uses graph, CRT;
{
This unit is used to create the ability to simulate different colors using digital
Halftoning. It provides procedures to set the palette for digital halftoning with the
colors of red, green and blue and to select one of the shades based on the intensity value
of the reflected color from the surface of the plane being drawn. The entire Unit is NEW CODE.)

type
ToneAttr = record
  KeyMatte, EitherColor, EitherPattern: byte;
end;

ToneMatrix = array[1..24] of ToneAttr;

var
  BlueTones, GreenTones, RedTones: ToneMatrix;
  Int_Amb, Int_Point: Single;

procedure change_palette;
{This procedure changes the palette to allow use of 4 tones of red, green, and blue. The remaining
4 colors are black, white, yellow, and grey.}

procedure InitTones;
{This procedure sets up variables in memory that contain the two tones of a color (i.e. red) and the
pattern to use in drawing a surface using these two colors to create up to 24 shades of the color.}

function Drawing_Tone(Intensity: real): byte;
{This function returns the index into the array that contains the 24 shades of a color based on the
intensity value that is passed in as a parameter. Intensity values are between 0 and 1}

implementation
{
IMPLEMENTATION OMITTED IN THESIS APPENDIX}

end.

(***************The Ground Unit***************

unit Ground;
interface
uses Shades, CRT, GRAPH;
{
This unit provides the basic procedures and functions for the drawing of triangles and lines and
initialization of the program. It provides the global variables for the program}
const
Ground_Ref_Coeff:Single = 0.45; {reflection coefficient of ground}
Spacing:Single = 100.0; {interval between elevation points}
LOWR = 0; RANGE = 39; {the range of values for the elevation points array}

type
Vector = record
  x,y,z:Single; {three dimensional vector coordinates}
end;
TwoVector = record
  SE_Corner,NE_Corner:Vector; {one vector for each triangle in square, the SouthEast (SE) triangle and the NorthWest triangle (mistakenly labeled NE throughout program)}
end;
Surface_Color = record
  SW_Corner,NE_Corner:byte; {one color setting for each triangle in a square}
end;
Two_D_Array = array[LOWR..Range,LOWR..Range] of POITNTYPE; {POITNTYPE is defined in the Turbo Pascal graphics unit GRAPE as record of x, y of integer}
TRI_TYPE = array[1..3] of PointType; {array of three vertices of a triangle}
DATA_ARRAY = array[LOWR..Range,LOWR..Range] of Single; {elevation points for a square piece of terrain that is (Range - Lowr + 1) x Spacing large}
Surface_Color_Array = array[Lowr..Range,Lowr..Range] of Surface_Color; {array of surface colors for all triangles in the square piece of terrain being displayed}
Normal_Vector_Array = array[Lowr..Range,Lowr..Range] of TwoVector; {array of Normal vectors for all triangles in the square piece of terrain being displayed}
Normal_Vector_Ptr = "Normal_Vector_Array; Row_Of_Pts = array[Lowr..Range] of Vector; {needed to prevent overflow of integer values when drawing triangles that are close to the viewer}
points = array[1..7] of single;
PointTypeReal = record
  x,y:single;
end;
Close_Rows = array[0..2] of Row_Of_Pts; {used to draw triangles that are close to the viewer}

var
ch: char;
Tone_To_Draw: ToneAttr; {ToneAttr defined in Shades unit}
Center:PointType;
ScreenImage: Pointer;
Light_Source: Vector;
Surface_colors:"Surface_Color_Array;
Normal_Vectors: Normal_Vector_Ptr;
Close_Rows:"Close_Rows;
DIRECTION: (NORTH,SOUTH,EAST,WEST,NORTHWEST,NORSEAST,SOUTHWEST,SOUTHEAST,NORTH, NORTHEAST, SOUTHERN, SOUTHEAST, SOUTHWEST);
Two_D_Data:"Two_D_Array;
TRI: TRI_TYPE;
S: string;
STOPPOINT, WRITEPAGE, ERRORCODE, GRAPHSIZE, GRAPHDRIVER, PAGE: integer;
procedure INIT3D;
{This procedure initializes the program to use EGA graphics mode and sets the boundaries of the screen for this mode. Initializes the Roll, Pitch and Taw angles of the viewer to 0 for the program. MODIFIED CODE}

procedure Allocate_Mem;
{This procedure allocates memory from the heap for the Surface color array, the Normal Vectors array, the Two D Data array, and the Close Row array. NEW CODE}

procedure SetPixel(x,y:word;n:byte);
{This procedure is implemented in assembly language. It sets a given pixel x,y to the nth color of the palette. UNMODIFIED CODE}

procedure Myline(x1,y1,x2,y2:word;n:byte);
{This procedure is implemented in assembly language. It draws a line from (x1,y1) to (x2,y2) using the nth color of the palette. UNMODIFIED CODE}

procedure MylineC(x1,y1,x2,y2:integer;n:byte);
{This procedure is used to draw a line that has one or both end points off the screen (it clips the line to fit the screen). UNMODIFIED CODE}

procedure SetPattern(p:byte);
{This procedure is used to set the pattern that will be used by the triangle drawing procedures FillTri and FillTriC. NEW CODE}

procedure FillTri(x1,y1,x2,y2,x3,y3:word;n,o,MMNbits:byte);
{This procedure is a specialized procedure implemented in assembly language that draws triangles. All three vertices must be on the screen. The pattern must be set before calling this procedure using SetPattern. The triangle is drawn with the primary color of the pattern as the nth color of the palette and the secondary color of the pattern as the oth color of the palette. NEW CODE mixed with some MODIFIED CODE from Myline which uses Bresenham's algorithm for drawing lines.}

procedure FillTriC(x1,y1,x2,y2,x3,y3:integer;n,o,MMNbits:byte);
{This procedure is similar to FillTri except that the three points of the triangle do not have to be on the screen. It draws only that part of the triangle that is on the screen. NEW CODE mixed with some MODIFIED CODE from Myline which uses Bresenham's algorithm}

procedure Restore;
{This procedure restores the graphics card to its default condition at the end of the program. MODIFIED CODE}
procedure FillWindow(FillColor, RBMbi8ts: byte);
(This procedure fills a window with a color. It assumes the window is already defined by the variables TLX, TLY, BX, BY and is implemented in assembly language. MODIFIED CODE)

procedure To_Unit_Vector(var UnitN: Vector);
(This procedure converts a vector passed in as UnitN to a unit vector. MODIFIED CODE)

function Dot_Product(UnitN, UnitL: Vector): single;
(This function returns the value of the Dot Product of the two vectors UnitN and UnitL. MODIFIED CODE)

procedure Cross_Product(XU, YU, ZU, XV, YV, ZV: Single; var Normal: Vector);
(This procedure sets the variable Normal to the result of the cross product of the vector (XU, YU, ZU) and (XV, YV, ZV). MODIFIED CODE)

procedure Set_Light_Source(XL, YL, ZL, IP, IA: Single);
(This procedure sets the vector that indicates the location of the point light source (the sun) to (XL, YL, ZL). It sets the intensity of the point source to IP and the intensity of ambient light to IA. NEW CODE)

function Elevate(xloc, zloc: single): single;
(This function implements the DTWACS algorithm for determining the elevation of a point on the terrain surface. It accepts as input the location (xloc, zloc) and returns the y value (the elevation) for that point. The values xloc and zloc are the relative coordinates in reference to lower left corner of the square piece of terrain in the terrain array. MODIFIED CODE)

function Elevate_World(xlocw, zlocw: single): single;
(This function is similar to the elevate function except xlocw and zlocw are the world coordinates relative to the lower left corner of the 20 square kilometer terrain database in the file 32n131e.da3 MODIFIED CODE)

procedure READ3D_FILE(var DATA: DATA_ARRAY; LLX, LLY: Longint);
(This procedure opens the file 32n131e.da3 and initializes the 4 kilometer square chunk in to the display array. NEW CODE)

procedure READ.Normal_FILE(var NORMDATA: Normal_Vector_Ptr; LLX, LLY: Longint);
(This procedure reads in the surface normals for each of the triangles in the 4 kilometer square of the display data. NEW CODE)

procedure Calculate_Surface_Norms;
(This procedure calculates Surface Normals for for the 4 kilometer square of terrain data and stores the results in the surface normal array. NEW CODE)

procedure Calculate_Surface_Colors;
(This procedure calculates the appropriate surface colors of each of the triangles in the 4 kilometer display square based on the light intensity values and stores them in the surface color array. NEW CODE)

procedure Line_Clip(var x10, y10, x20, y20: single);
(This procedure clips a line to draw only the portion that is on the screen/window. It accepts the line coordinates as real values. UNMODIFIED CODE)
procedure Polygon_Clip_Draw(col:byte;m:integer;x,y:points);
(This procedure draws triangles that are in the rows that are close to the viewer. To prevent overflow it uses real values. It was adapted directly from Computer Graphics pp. 137-138 with only slight modifications. MODIFIED CODE)

procedure Draw_Close(Pt1,Pt2,Pt3:Vector;Tri_Col:byte);
(This procedure is used to draw triangles that are in the two rows closest to the viewer. It uses real values to prevent integer overflow. It clips the triangles as necessary even if the triangle goes behind the viewer. It is an implementation of the theory of clipping in two and three dimensions MODIFIED CODE)

implementation
{
implementation omitted in thesis
}
end.

***************The Ground2 Unit***************

unit Ground2;

interface

uses people,List,BSTree,pieces,Ground,shades,crt,graph;

(This unit is a continuation of the ground unit but required the use of several other units before it could be implemented. Limitations on the size of units that could be edited and debugged forced the breaking of the units in this fashion.)

type
Moving_Obj = arrary[lowr..Range,Lowr..Range] of LinkObj;

var
Array_Of_Movers:Moving_Obj;
Proj_X,Proj_Z:Single;

procedure Set_Trig_Val;
(This procedure sets the global trigonometric values used by the ThreeD_To2D procedure. It sets CT (cosine of Yaw), CR (cosine of Roll), CP (cosine of Pitch), ST (sine of Yaw), SI (sine of Roll), SP (Sine of Pitch), and variables used in the translation, rotation, and scaling matrix (am,bm,cm,dm,em,fn,gn,hm,im). Using this procedure the values are set once before performing calculations on all of the terrain data points. NEW CODE)

procedure ThreeD_To2D_List(index1,index2:Integer);
(This procedure creates a dynamic list as necessary for each 100m square that has one or more moveable objects in it (i.e. soldiers) and then performs the calculations necessary to create the display data for each of those objects. NEW CODE)

procedure THREED_TO_2D;
(This procedure converts the three dimensional coordinates of the terrain into two dimensional coordinates that are suitable for display on the screen. It selectively handles only the data of the 4 km square that falls in the field of view of the viewer. This procedure is application of theory. NEW CODE)
implementation

{ IMPLEMENTATION OMITTED IN THESIS APPENDIX }
end.

unit Ground3;
interface
uses shades, ground, ground2, list, bstree, graph, info;

{This unit contains more procedures and functions that are related to the ground unit but use other additional units that the ground unit does not use.}

procedure DRAW(This_Color: byte);
{This procedure checks to determine if the triangle is completely on the screen or not. If it is completely on the screen it draws the triangle using FillTri and then outlines it with Myline. If it needs clipping it draws the triangle with FillTriC then outlines it with MyLineC. NEW CODE}

procedure Check_Display_Remain;
{This procedure checks to see if at least 2 kilometers of terrain display data are available to the front of the viewer and that at least 1.5 km are to the left and right of the viewer. If these conditions are not satisfied, the procedure loads a new square of data from the 20 km terrain database file into the display data array that provides 3.5 km to the front of the viewer. NEW CODE}

procedure VIEW;
{This procedure implements the painter's algorithm and draws the triangles for the terrain and the soldiers in the correct sequence so that hidden surfaces are hidden. In order to do this, it uses one of 14 drawing sequences dependent upon the view direction. Each of the 14 drawing sequences draws only the triangles and objects that are in the field of view of the viewer. This procedure uses the FillWindow, the Set_Trig_Val, and the ThreeD_To_2d procedures. NEW CODE}

implementation

{ IMPLEMENTATION OMITTED IN THESIS APPENDIX }
end.

unit List;
interface
uses Pieces;

{This unit implements a dynamic linked list using objects instead of records. This unit was easily adapted form the text Data Structures by Rick Decker pp. 73-77. The only modifications were to convert it to an object oriented list. The entire unit is MODIFIED CODE.}

type
  NodePtr = ^NodeRec;
  LinkPtr = ^LinkObj;

72
NodeRec = record  
  Next: NodePtr;  
  Item: ThreeLocPtr;  
  end;

LinkObj = object  
  First, Last: NodePtr;  
  procedure Init;  
  procedure Done;  
  procedure Add(Item: ThreeLocPtr);  
  function EmptyList:Boolean;  
    [Checks to see if List pointed to by L is empty and]  
    [returns Boolean answer]  
  function FirstList:NodePtr;  
    [Returns pointer to first Node in List]  
  function LastList:NodePtr;  
    [Returns pointer to last Node in List]  
end;

{Moving_Obj = array[Lowr..Range,Lowr..Range] of LinkObj;}

implementation

{ IMPLEMENTATION OMITTED IN THESIS APPENDIX }

end.

{******************************************The BSTree Unit******************************************}

Unit BSTree;
interface
uses people,ground;

{This unit is an implementation of a Binary Search Tree modified to work with this terrain program  
It is only slightly modified from the BST presented by Decker in Data Structures pp. 198-202. Some  
additional procedures were added to suit the main programs needs.)

type  
  Tree_Link = 'Node;
  Binary_Search_Tree = Tree_Link;
  Node = record
    Left, Right: Tree_Link;
    Tree_data: Data2dPtr;
  end;
  Two_Ptr = record
    W, S: Binary_Search_Tree;
  end;
  LandMark2d_Array = array[Lowr..Range,Lowr..Range] of Two_Ptr;
  LandMarks2d = "LandMark2d_Array;

73
var
  LandMarksData: LandMarks2d;

procedure Init_LandMark2d_Array;
  [This procedure initializes the LandMark2d_Array by first marking the top of the heap, then allocating memory from the heap, and last setting all pointers to nil. NEW CODE]

procedure Erase_LandMark_Data;
  [This procedure erases the LandMark2d_Array by freeing the memory that has been allocated since the top of the heap was marked in the initialization of the Landmark2d_array. Erasing in this manner prevents the program from having to go back and de-reference all pointers to the BSTs created. All of the memory allocated since marking of the heap top is freed at once. NEW CODE]

procedure Create(var B: Binary_Search_Tree);
  [initializes B to point to a new empty binary tree. UNMODIFIED CODE]

procedure Insert(a: Data2d_Ptr; var B: Binary_Search_Tree);
  [inserts atom a into tree is such a manner that the resulting tree is still a BST. If there is a node with the same value as the key already then the atom is inserted as a right child. MODIFIED CODE]

procedure Clear_Tree(var P: Tree_Link);
  [deallocates all pointers in tree so that no garbage is left in heap. MODIFIED CODE]

procedure Display_LandMarks(P: Binary_Search_Tree);
  [displays objects in binary search tree by doing an inorder traversal of tree. NEW CODE]

implementation
  [ IMPLEMENTATION OMITTED IN THESIS APPENDIX ]
end.

{********************The GText Unit***********************}

unit GText;
{ an extended set of text routines for graphics mode adapted directly from the reference Power Graphics Using Turbo Pascal by Keith Weiskamp et al pp. 74-79 with no modification necessary The entire unit is UNMODIFIED CODE}

interface
cost
  CR = $13;
  ESC = $27;
  BS = $08;

  [ These routines are available to any programs that "use" this unit ]
function IntToStr(Num: longint): string;
  [This function returns an input integer value as string value (text).]
function RealToStr(n: real; width, decimals: integer): string;
{This function returns an input real value as a string (text).}

procedure GWrite(S: string);
{This procedure writes a string to the screen in graphics mode at the location where the cursor is already pointing.}

procedure GWriteXY(x, y: integer; S: string);
{This procedure writes a string to the screen at a specific location (x,y)}

procedure GWriteCh(ch: char);
{Writes a single character to the screen}

function GReadReal(var Num: real): boolean;
{Gets a real number as input from the screen followed by a carriage return.}

function GReadStr(var S: string): boolean;
{Echoes input from the keyboard to the screen in graphics mode}

implementation
uses
  Graph, Crt;

{IMPLEMENTATION OMITTED IN THESIS APPENDIX}
end.

{******************************************************************The GPopPac Unit******************************************************************}

unit GPopPac;
{ This is a set of utilities that provides popup windows in graphics mode.
  The routines use Turbo Pascal's BGI tools to simplify the code. Most of
  the graphics settings are saved before a new window is put up and they
  are restored when the window is closed. This window data is saved on a
  stack. The stack is implemented as an array in order to simplify things. These
  Utilities were adopted directly with no modification from Power Graphics Using Turbo
  Pascal by Neiskamp et al pp.219-222 with no modifications. The entire unit is
  UNMODIFIED CODE}

interface
uses
  Graph;

const
  NumGWindows = 10;  { Allow for 10 pop-up windows }

type
GraphicsWindow = record
  Left, Top, Right, Bottom: integer; { Parent window boundaries }
  cpz, cpy: integer; { Current position in parent window }
  SaveArea: pointer; { Pointer to the saved region }
  DrawColor: word; { Current drawing color }
end;

var
  { Graphics window stack }
  WindowStack: array [1..NumWindows] of GraphicsWindow;
  { Index to the next available location on the stack to use }
  GWindowPtr: integer;

{ The externally visible routines from this package }
function GPopup(Left, Top, Right, Bottom, BorderType,
  BorderColor, BackFill, FillColor: integer): boolean;
procedure GUunpop;
procedure UnpopAllWindows;

implementation
  { IMPLEMENTATION OMITTED IN THESIS APPENDIX }
end.

{******************************The Frago Unit******************************}

Unit Frago;

{ This unit installs a keyboard interrupt service routine that intercepts certain keystrokes before reaching the main program. These interrupts are set up upon initialisation of the program and is hidden from main program. Only the variables below are usable by the program directly. This unit was adapted from the units explained in Turbo Pascal Advanced Techniques by Chris Ohlsen and Gary Stroker pp. 197-230. The entire unit is MODIFIED CODE. }

interface

var
  ViewLeft, ViewRight, ViewFront, ViewRear,
  PitchUp, PitchDown, HeightUp, HeightDown, ZoomIn, ZoomOut, Escape: Boolean;
  P: byte;

implementation

uses DOS, CRT, Gtext, gpoppac, graph, ground;

  { IMPLEMENTATION OMITTED IN THESIS }
end.
{********************The Info Unit********************}

unit Info;
{This unit handles the information display on the screen. It can easily be changed to display any information that is desired. The entire unit is NEW CODE}

interface
uses Graph,ground;

procedure Display_Information;
{This procedure displays information in a window at the top of the screen. Information displayed includes the view azimuth, the pitch angle, the yaw angle, and the viewer location}

implementation
uses Graph;

{ IMPLEMENTATION OMITTED IN THESIS APPENDIX }

end.

{********************The People Unit********************}

unit people;

{This unit provides functions and procedures to initialize the data for displaying the soldiers and other objects. The objects provide a procedure to display themselves on the screen (View_Obj_T). This unit is completely NEW CODE except for function Atan.}

interface
uses Graph,ground;

const
    People_Ref_Coeff: Single = 0.45;

type
    Three_Indices = array[1..3] of integer;
    Four_Indices = array[1..4] of integer;
    People_Vertices = array[1..32] of vector;
    Tree_Vertices = array[1..14] of vector;
    People_Vertices2d = array[1..32] of PointTypeReal;
    Tree_Vertices2d = array[1..14] of PointTypeReal;
    People_Norm = array[1..11] of vector;
    Tree_Norm = array[1..12] of vector;
    People_Col = array[1..11] of byte;
    Tree_Col = array[1..12] of byte;
    Color_Indices = array[1..11] of Three_Indices;
    Tree_Color_Indices = array[1..12] of Three_Indices;
    Seq_1 = array[1..32] of Four_Indices;
    Tree_Seq = array[1..7] of Four_Indices;
Seq_and_No = record
  no_Tri:integer;
  Sequence:Seq_i;
end;

Tree_Seq_and_No = record
  no_Tri:integer;
  Sequence:Tree_Seq;
end;

Seq_Ptr = 'Seq_and_No;
Tree_Seq_Ptr = 'Tree_Seq_and_No;
PeopleVert2dPtr = 'People_Vertices2d;
TreeVert2dPtr = 'Tree_Vertices2d;
ObjColPtr = 'byte;

Draw_Data = object
  procedure Init;
end;

TreeColPtr = 'Tree_Col;

Tree_Draw_Data = object(Draw_Data)
  Vertices_2d:Tree_Vertices2d;
  Draw_Seq:Tree_Seq_Ptr;
  Draw_Colors:TreeColPtr;
  procedure Init;
  procedure Set_Tree_Vertices2d(obj_head,base_x,base_y,
                               base_x:tree_scale:single);
  procedure Set_Tree_Draw_Seq(obj_head,base_x,base_y,
                               base_x:tree_scale:single);
  procedure Set_Tree_Draw_Colors(obj_head:tree_color);
  procedure Set_all_Tree_Draw_Data(obj_head,base_x,base_y,base_x,
                                   tree_scale:single);
  procedure View_Tree;
end; {Tree_Draw_Data_Object}

Tree_Draw_Data_Ptr = 'Tree_Draw_Data;

Forrest_Array = array[1..20] of Tree_Draw_Data_Ptr;

Forrest_Of_Trees = object
  Number_of_Trees,Tree_To_View:Integer;
  The_Trees:Forrest_Array;
  procedure Init;
  procedure Set_Tree_To_View(The_Index:integer);
  function Get_Tree_To_View:integer;
  procedure Set_No_of_Trees(Num:integer);
  function Get_No_of_Trees:integer;
  procedure View_Tree(index:integer);
end;
Forrest_Ptr = 'Forrest_of_Trees';

Data_2d_Ptr = object
  Depth:Single;
  constructor Init(value:Single);
  destructor Done;virtual;
  procedure Set_Depth(value:Single);
  function Get_Depth:Single;
  procedure Set_Draw_Colors(ptr:ObjColPtr);
  procedure View_Obj_Y;virtual;
end;

PeopleCol_Ptr = 'People_Col';
People_2d_Ptr = object(Data_2d_Ptr)
  Data:PeopleVert2dPtr;
  DrawSeq:Seq_Ptr;
  Draw_Colors:PeopleColPtr;
  constructor Init(Value:Single);
  destructor Done;virtual;
  procedure Set_Data(Vert2d:PeopleVert2dPtr);
  function Get_Data:PeopleVert2dPtr;
  procedure Set_Draw_Seq(Ptr:Seq_Ptr);
  function Get_Draw_Seq:Seq_Ptr;
  procedure Set_Draw_Colors(Ptr:PeopleColPtr);
  function Get_Draw_Colors:PeopleColPtr;
  procedure View_Obj_Y;virtual;
end;

Tree_2d_Ptr = object(Data_2d_Ptr)
  Tree_index:integer;
  constructor Init(Value:Single);
  destructor Done;virtual;
  procedure Set_Tree_index(num:integer);
  function Get_Tree_index:integer;
  procedure View_Obj_Y;virtual;
end;

Draw_Seq = record
  Seq_0_to_90, Seq_270_to_360,
  Seq_90_to_180, Seq_180_to_270: Seq_Ptr;
end;

TreeDraw_Seq = record
  Seq_0_to_45, Seq_45_to_90,
  Seq_90_to_135, Seq_135_to_180,
  Seq_180_to_225, Seq_225_to_270,
  Seq_270_to_315, Seq_315_to_360: Tree_Seq_Ptr;
end;

PeopleVert_Ptr = 'People_Vertices;
TreeVert_Ptr = 'Tree_Vertices;

79
PeopleNormPtr = *People_Norm;
TreeNormPtr = *Tree_Norm;
DrawSeqPtr = *Draw_Seq;
TreeDrawSeqPtr = *TreeDraw_Seq;
ColorIndPtr = *Color_Indices;
TreeColorIndPtr = *Tree_Color_Indices;
Data2dPtr = *Data_2d_Obj;
PeopleData2dPtr = *People_2d_Obj;
TreeData2dPtr = *Tree_2d_Obj;

var
Forrest: Forrest_Ptr;
People_Data: PeopleVertPtr;
Tree_Data: TreeVertPtr;
People_Norms: PeopleNormPtr;
Tree_Norms: TreeNormPtr;
People_Draw_Seq: DrawSeqPtr;
Tree_Draw_Seq: TreeDrawSeqPtr;
People_Color_Vector_IND: ColorIndPtr;
Tree_Color_Vector_IND: TreeColorIndPtr;
People_Colors: PeopleColPtr;
Tree_Colors: TreeColPtr;
People_Data2d: PeopleData2dPtr;
Tree_Data2d: TreeData2dPtr;

procedure Init_People_Graph_DB;
(This procedure initializes the soldier three dimensional display data that all soldiers use to display themselves. NEW CODE)

function Compute_People_Colors: PeopleColPtr;
(This function determines the color to draw triangles of the soldier data base. It assumes that the Set_Trig_Val_Obj has already been called. NEW CODE)

function ATan(X, Y: Single): Single;
(This function returns the value of the arc tangent of x and y values input. MODIFIED CODE)

procedure Set_Trig_Val_Obj(Obj_Taw, Obj_Roll, Obj_Pitch: Single);
(This procedure sets the trigonometric values used in the View_Obj_Y procedures of each display object. NEW CODE)

function Depth_Obj(X_Obj, Y_Obj, Z_Obj: Single): Single;
(This function returns the depth or distance of an object from the view location. this depth value is necessary to determine the order in which to draw the various objects. NEW CODE)

function ThreeD_To_2D_Obj(XLoc, YLoc, ZLoc, Head_Obj: Single): PeopleData2dPtr;
(This function returns a pointer to the two dimensional display coordinates of an object that has been translated, rotated and scaled as appropriate for the display. NEW CODE)
implementation
uses Shades;

[ IMPLEMENTATION OMITTED IN THESIS APPENDIX ]
end.

(**********************************************************************)
Unit Pieces;
(This unit creates objects for data structures for the platoon soldiers and equipment.
The entire Unit is NEW CODE)
uses People;

Type
WeaponList = (M16,M203,M60,M249,M1911,M47);
MoveList = (Marching,ForcedMarching,Running,Rushing,LowCrawling,
           HighCrawling,Standing,Kneeling,Laying);
ShootList = (HighVolAimed,LowVolAimed,HighVolArea,LowVolArea,Loading,
            Jammed,NotFiring);
CommList = (Talking,Listening,Listening,Radioing,Signing);
EquipList = (Prc77,Prc68,PVS5,PVS4,Bayonet);
AmmoList = (RifleBullets,MGbullets,SAMBullets,Laws,Grenades,
            Smokes,Flares,HEEP,M203,M203smk,Claymores);
EquipStatus = (Working,Broken);
AlertStatus = (Awake,Sleeping);
Protection = (Covered,Concealed,Exposed);
DefStatus = (Prepared,Ready,None);
LifeStatus = (Alive,Dead,Wounded);

EquipRec = record
  IsPresent:Boolean;
  Status:EquipStatus;
end;

EquipArray = Array[Prc77...Bayonet] of EquipRec;
  {stores whether the indiv has a piece of equip and its status}
AmmoArray = Array[RifleBullets...Claymores] of Integer;

ThreeDLocPtr = "ThreeDLocObj;
PersPtr = "PersObj;
TreePtr = "TreeObj;

ThreeDLocObj = Object
  x,y,z,heading,ObjYaw,ObjPitch,ObjRoll: single;
  constructor Init(Ptx,Pty,Ptz,Orien,Yaw,Pitch,Roll:single);
  procedure move(Ptx,Pty,Ptz:single);
  procedure Change_Heading(New_Head:Single);
  procedure Change_Yaw(New_Yaw:Single);
  procedure Change_Pitch(New_Pitch:Single);

81
procedure Change_Roll(New_Roll:single);
function GetX:single;
function GetF:single;
function GetT:single;
function GetHeading:single;
function GetTaw:single;
function GetRoll:single;
function GetPitch:single;
destructor Done; virtual;
end;

PersObj = Object(ThreeDLocObj) {
  ThreeDDataPtr:PeopleVertPtr;
  NormalsPtr:PeopleNormPtr;
  SequencesPtr:DrawSeqPtr;
  TypeWpn:WeaponList;
  TypeMort:MoveList;
  TypeShoot:ShootList;
  TypeComm:CommList;
  AmmoCarried:AmmoArray;
  EquipCarried:EquipArray;
  Brain:AlertStatus;
  Exposure:Protection;
  DefPosture:DefStatus;
  BodyStatus:LifeStatus;
}
constructor Init(Ptx, Py, Ptz, Orien, Taw, Pitch, Roll:single);
procedure SetThreeDDataPtr(ptr:PeopleVertPtr);
procedure SetNormalsPtr(ptr:PeopleNormPtr);
procedure SetSequencesPtr(ptr:DrawSeqPtr);
procedure SetWpn(Wpn:WeaponList);
procedure SetMort(Mvt:MoveList);
procedure SetShoot(Sht:ShootList);
procedure SetComm(Commode:CommList);
procedure SetAmmo(Amm:AmmoList; Amt:integer);
procedure UseAmmo(Amm:AmmoList; Amt:integer);
procedure IssueEquip(Equ:EquipList);
procedure BreakEquip(Equ:EquipList);
procedure FixEquip(Equ:EquipList);
procedure SetBrain(cat:AlertStatus);
procedure SetExposure(Vis:Protection);
procedure SetDefPosture(Post:DefStatus);
procedure SetBody(Cond:LifeStatus);
function GetThreeDDataPtr:PeopleVertPtr;
function GetNormalsPtr:PeopleNormPtr;
function GetSequencesPtr:DrawSeqPtr;
function GetWpn:WeaponList;
function GetMort:MoveList;
function GetShoot:ShootList;
function GetComm:CommList;
function GetAmmo(Amm:AmmoList):integer;
function CheckEquipStat(Equ:EquipList):EquipStatus;
function CheckEquipHere(Equip: EquipList): Boolean;
function GetBrain: AlertStatus;
function GetExposure: Protection;
function GetDefPosture: DefStatus;
function GetBody: LifeStatus;
destructor Done; Virtual;
end;

TreeObj = Object(ArborLocObj)
constructor Init(Xtx, Ptx, Xori, Taw, Pitch, Roll: single);
destructor Done; Virtual;
end;

implementation

{ IMPLEMENTATION OMITTED IN THESIS APPENDIX }
end.

{*************** The ARFOR Unit******************}

Unit ARFOR;
(This unit is a skeleton for setting up the friendly forces that are necessary for the combat model. It provides the structure for a light infantry platoon organization and can be expanded to provide the structure for a company size force. The entire Unit is NEW CODE)

interface
uses pieces, people, List;

const
Default_Interval: Single = 0.1;

type
TeamForms = (TeamWedge, TeamOnLine, TeamFile, Mod_Wedge, Diamond); TeamPositions = (TeamLdr, AutoRifle, Grenadier, Rifleman, Attachment); SquadForms = (SquadColumn, SquadLine, SquadFile); PltForms = (PltColumn, LineLine, LineCol, PltVee, PltWedge, PltFile); MovTech = (Traveling, Traveling_Overwatch, Bounding); SqdModeList = (Move, Assault, Support, Defend, Delay, Withdraw, Reserve); FireTeamPtr = 'FireTeamObj'; FireTeamObj = Object
TL, AR, GNDR, RM, ATT: PersPtr;
SoldInt: Single; {Interval between soldiers} TeamForm: TeamForms;
Detections: LinkPtr;
procedure Init(xTL, yTL, xTL: single; Dir: single; Form: TeamForms);
procedure Done;
procedure SetGNDR(PPtr: PersPtr);
procedure SetAR(PPtr: PersPtr);
procedure SetTL(PPtr:PersPtr);
procedure SetEN(PPtr:PersPtr);
procedure SetATT(PPtr:PersPtr);
procedure SetSoldierInterval(Sp:Single);
procedure SetFormation(Form:TeamForms;Interval:Single);
procedure SetDetections(Dptr:LinkPtr);
procedure AttachMan(PPtr:PersPtr);
procedure DetachMan(Posit:TeamPositions;var PPtr:PersPtr);
function GetDetections:LinkPtr;
function GetGND:PersPtr;
function GetNR:PersPtr;
function GetTL:PersPtr;
function GetHN:PersPtr;
function GetATT:PersPtr;
function GetSoldierInterval:Single;
function GetFormation:TeamForms;
procedure ResupplyTeam(Perc:Single);
procedure MoveTeam;
procedure ChangeTeamHeading(As:single);
end;

SqPtr = "Squad;  

Squad = Object  

  SqLdr:PersPtr;  
  Alpha,Bravo:FireTeamPtr;  
  SqForm:SquadForms;  
  SqHs:SqHsList;  
  TeamInt:Single;  
procedure Init(xTL,yTL,zTL:single;Dir:single;Form:SquadForms);  
procedure Done;  
procedure SetAFireTeam(ABPtr:FireTeamPtr);  
procedure SetBFireTeam(ABPtr:FireTeamPtr);  
procedure SetSqLdr(PPtr:PersPtr);  
procedure SetTeamInterval(Sp:Single);  
procedure SetSqForm(Form:SquadForms;Interval:Single);  
procedure AttachMan(PPtr:PersPtr);  
procedure DetachMan(Posit:TeamPositions;var PPtr:PersPtr);  
procedure ResupplySq(Sq:CsvsqForm);  
procedure MoveSq(Tech:MoveTech);  
procedure ChangeSqHeading(As:single);  
function GetAFireTeam:FireTeamPtr;  
function GetBFireTeam:FireTeamPtr;  
function GetSqLdr:PersPtr;  
function GetTeamInterval:Single;  
function GetSqForm:SquadForms;  
procedure GetSqLoc(var xs,ys,zs:single);  
function GetSqHeading:single;  
end;

PItPtr = "Platoon;
Platoon = Object
FistSqd, SecSqd, ThdSqd: SqdPtr;
 procedure Init(xTL, yTL, sTL: single; Dir: single; Form: PtForm);
 procedure Done;
end;

var
 Offset, Set_Pitch, Alternate_alt: single;

implementation

uses ground;

{ IMPLEMENTATION OMITTED IN THESIS APPENDIX }
end.

{******************************The Main Program******************************}

program Main;
{This is the main program that uses all of the units listed below. The interface portion of these units
is presented above. The main program is NEW CODE}

uses Prago, GText, Arfor, people, List, BSTree, Pieces, Ground3, Ground2, Ground,
 shades, crt, graph;
var
 BlueTeam: FireTeamPtr;
 cti: integer;
 HalfPaw: Single;
 Start_Heading: Single;

procedure Initialize_Disposables;
var
 i: integer;
begin
 Init_LandMark2d_Arrary; {also Marks HeatTop while creating array of BST's}
 New(Array_of_Movers); {creates array of Lists for each square}
 FillChar(Array_of_Movers, SizeOf(Array_of_Movers), 0); {set all Pointers to nil}
 Array_of_Movers[Trunc(BlueTeam'.TL'.GetX-Map_BLC_X),
 Trunc(BlueTeam'.TL'.GetX-Map_BLC_X)].Add(BlueTeam'.TL);
 Array_of_Movers[Trunc(BlueTeam'.AR'.GetX-Map_BLC_X),
 Trunc(BlueTeam'.AR'.GetX-Map_BLC_X)].Add(BlueTeam'.AR);
 Array_of_Movers[Trunc(BlueTeam'.GNDR'.GetX-Map_BLC_X),
 Trunc(BlueTeam'.GNDR'.GetX-Map_BLC_X)].Add(BlueTeam'.GNDR);
 Array_of_Movers[Trunc(BlueTeam'.RM'.GetX-Map_BLC_X),
 Trunc(BlueTeam'.RM'.GetX-Map_BLC_X)].Add(BlueTeam'.RM);
 if BlueTeam'.ATT <> nil then
 Array_of_Movers[Trunc(BlueTeam'.ATT'.GetX-Map_BLC_X),
 Trunc(BlueTeam'.ATT'.GetX-Map_BLC_X)].Add(BlueTeam'.ATT);
end;
procedure Initialize_Model;
begin
  Writeln('Enter the heading for movement at startup in degrees i.e. 180.0');
  Readln(Start_Heading);
  Start_Heading:=pi*Start_Heading/180;
  Writeln('Enter the x coordinate for lower left corner of of map start');
  Readln(Map_BLC_X);
  Writeln('Enter the y coordinate for lower left corner of of map start');
  Readln(Map_BLC_Y);
INIT3D;
Shades.Change_Palette;
Shades.InitTones;
FillWindow(11.0);
WRITEPAGE:=1-WRITEPAGE;
SETVISUALPAGE(1-WRITEPAGE);
SETACTIVEPAGE(WRITEPAGE);
FillWindow(11.0);
read3d_file(data,Map_BLC_X,Map_BLC_Y);
View_Ht:=0.02;
Offset:=0.5;
Yaw_Dif:=0.0;
Trans_x:=21.25; Trans_y:=10.0;
TRANS_Y:= View_Ht + elevate(Trans_x,Trans_y);
Set_Light_Source([0.0,0.0,0.0,0.35,0.025]; [0.7071068,-0.7071068,0.75,0.25]);
Set_Pitch:= pi*3/180; [No higher than 89 deg]
Pitch_Ang:=Set_Pitch;
Allocate_mem;
Calculate_Surface_Norms;
Calculate_Surface_Colors;
HalfFOV:=0.523598775;
tlt:=18; tlx:=25;
trye:=331; brx:=614;
scale:=(1+HRE-YLE)*Cos(HalfFOV)/((2 Sin(HalfFOV));
Init_People_graph_DB;
New(BlueTeam);
BlueTeam'.Init(Map_BLC_X+11.25,elevate(11.25,15.1),Map_BLC_Y+15.1,Start_Heading,TmWedge);
end;

procedure Set_View_Coord(PPtr:PersPtr;Off:Single);
var
  Off_Alt:Single;
begin
  if (ViewFront = True) then begin
    Yaw_Dif:=0.0;
    ViewFront:=False
  end
  else if (ViewLeft = True) then begin
    Yaw_Dif:=-1.570796;
    ViewLeft:=False
  end
  else if (ViewRight = True) then begin

Yaw_Dif:= 1.570796;
ViewHeight:=False
end
else if (ViewRear = True) then begin
Yaw_Dif:= 3.141593;
ViewRear:=False
end
else if (PitchUp = True) then begin
Pitch_Ang:=Pitch_Ang - Set_Pitch;
PitchUp:=False
end
else if (PitchDn = True) then begin
Pitch_Ang:=Pitch_Ang + Set_Pitch;
PitchDn:=False
end
else if (HeightUp = True) then begin
View_Ht:= View_Ht + 0.2;
HeightUp:=False
end
else if (HeightDn = True) then begin
View_Ht:= View_Ht -0.2;
HeightDn:=False
end
else if (ZoomIn = True) then begin
Scale:= 2*Scale;
ZoomIn:=False
end
else if (ZoomOut = True) then begin
Scale:=Scale*0.5;
ZoomOut:=False
end

Yaw_Ang:= PPtr^.GetHeading + Yaw_Dif;
Trans_x:= PPtr^.Getz-Map_BLC_X-off*Sin(Yaw_Ang);
Trans_y:= PPtr^.Getz-Map_BLC_Y-off*Cos(Yaw_Ang);
view:= elevate(Trans_x,Trans_y)+view_Ht;
end;

begin
Initialize_Model;
Set_View_Coord(BlueTeam^.TL,Offset);
Check_Display_Remain;
Initialize_Disposables;
view;
for ctr:=1 to 5 do begin
Release(NeedTop);
BlueTeam^.MoveTeam;
Set_View_Coord(BlueTeam^.TL,Offset);
Check_Display_Remain;
Initialize_Disposables;
view;
end;
Release(HeapTop);
BlueTeam".SetFormation(TmWedge,Default_Interval);
BlueTeam".MoveTeam;
Set_View_Coord(BlueTeam".TL,Offset);
Check_Display_Remain;
Initialize_Disposables;
view;
for ctr:=1 to 5 do begin
  Release(HeapTop);
  BlueTeam".MoveTeam;
  Set_View_Coord(BlueTeam".TL,Offset);
  Check_Display_Remain;
  Initialize_Disposables;
  view;
end;
Release(HeapTop);
BlueTeam".SetFormation(TmWedge,Defautl_Interval);
BlueTeam".MoveTeam;
Set_View_Coord(BlueTeam".TL,Offset);
Check_Display_Remain;
Initialize_Disposables;
view;
Release(HeapTop);
BlueTeam".SetFormation(TmWedge,Defautl_Interval);
BlueTeam".MoveTeam;
Set_View_Coord(BlueTeam".TL,Offset);
Check_Display_Remain;
Initialize_Disposables;
view;
for ctr:=1 to 5 do begin
  Release(HeapTop);
  BlueTeam".MoveTeam;
  Set_View_Coord(BlueTeam".TL,Offset);
  Check_Display_Remain;
  Initialize_Disposables;
  view;
end;
Release(HeapTop);
BlueTeam".SetFormation(TmWedge,Defautl_Interval);
BlueTeam".MoveTeam;
Set_View_Coord(BlueTeam".TL,Offset);
Check_Display_Remain;
Initialize_Disposables;
view;
for ctr:=1 to 5 do begin
  Release(HeapTop);
  BlueTeam".MoveTeam;
  Set_View_Coord(BlueTeam".TL,Offset);
  Check_Display_Remain;
  Initialize_Disposables;
  view;
end;
Release(HeapTop);
BlueTeam".SetFormation(TmWedge,Defautl_Interval);
BlueTeam".MoveTeam;
Set_View_Coord(BlueTeam".TL,Offset);
Check_Display_Remain;
Initialize_Disposables;
view;
for ctr:=1 to 5 do begin
  Release(HeapTop);
  BlueTeam".MoveTeam;
  Set_View_Coord(BlueTeam".TL,Offset);
  Check_Display_Remain;
  Initialize_Disposables;
  view;
end;
Release(HeapTop);
BlueTeam".SetFormation(TmWedge,Defautl_Interval);
BlueTeam".MoveTeam;
Set_View_Coord(BlueTeam".TL,Offset);
Check_Display_Remain;
Initialize_Disposables;
view;
for ctr:=1 to 5 do begin
  Release(HeapTop);
  BlueTeam".MoveTeam;
  Set_View_Coord(BlueTeam".TL,Offset);
  Check_Display_Remain;
  Initialize_Disposables;
view;
end;

Release(HeapTop);
BlueTeam'.SetFormation(Diamond, Default_Interval);
BlueTeam'.MoveTeam;
Set_View_Coord(BlueTeam'.TL, Offset);
Check_Display_Remain;
Initialize_Disposables;
view;

for ctr:=1 to 5 do begin
  Release(HeapTop);
  BlueTeam'.MoveTeam;
  Set_View_Coord(BlueTeam'.TL, Offset);
  Check_Display_Remain;
  Initialize_Disposables;
  view;
end;

Release(HeapTop);
BlueTeam'.SetFormation(Triangle, Default_Interval);
BlueTeam'.MoveTeam;
Set_View_Coord(BlueTeam'.TL, Offset);
Check_Display_Remain;
Initialize_Disposables;
view;

for ctr:=1 to 10 do begin
  Release(HeapTop);
  BlueTeam'.MoveTeam;
  Set_View_Coord(BlueTeam'.TL, Offset);
  Check_Display_Remain;
  Initialize_Disposables;
  view;
end;

Release(HeapTop);
BlueTeam'.MoveTeam;
Set_View_Coord(BlueTeam'.TL, Offset);
Check_Display_Remain;
Initialize_Disposables;
view;

BlueTeam'.ChangeTeamHeading(0.78539);
for ctr:=1 to 20 do begin
  Release(HeapTop);
  BlueTeam'.MoveTeam;
  Set_View_Coord(BlueTeam'.TL, Offset);
  Check_Display_Remain;
  Initialize_Disposables;
  view;
end;
view
end;

BlueTeam^.ChangeTeamHeading(1.570796);
for ctr:=1 to 20 do begin
  Release(HeapTop);
  BlueTeam^.MoveTeam;
  Set_View_Coord(BlueTeam^.TL, Offset);
  Check_Display_Remain;
  Initialize_Disposables;
  view
end;

BlueTeam^.ChangeTeamHeading(2.3561945);
for ctr:=1 to 20 do begin
  Release(HeapTop);
  BlueTeam^.MoveTeam;
  Set_View_Coord(BlueTeam^.TL, Offset);
  Check_Display_Remain;
  Initialize_Disposables;
  view
end;

BlueTeam^.ChangeTeamHeading(pi*190/180);
for ctr:=1 to 45 do begin
  Release(HeapTop);
  BlueTeam^.MoveTeam;
  Set_View_Coord(BlueTeam^.TL, Offset);
  Check_Display_Remain;
  Initialize_Disposables;
  view
end;

Release(HeapTop);
Trans_x:=trans_x+3.535553539;
Trans_y:=trans_y-1.4644661;
Trans_y:=elevate(trans_x, trans_y) + View_Ht;
Yaw_ang:=Yaw_ang+0.785398163;
Check_Display_Remain;
Initialize_Disposables;
view;

Release(HeapTop);
Trans_x:=trans_x+1.4644661;
Trans_y:=trans_y-3.5355539;
Trans_y:=elevate(trans_x, trans_y) + View_Ht;
Yaw_ang:=Yaw_ang+0.785398163;
Check_Display_Remain;
Initialize_Disposables;
view;

Release(HeapTop);
Trans_x:=trans_x-3.5355539;
Trans_y:=trans_y-1.4644661;
Trans_y:=elevate(trans_x, trans_y) + View_Ht;

90
Yaw_ang:=Yaw_ang+0.785398163;
Check_Display_Remain;
Initialize_Disposables;
view;
Release(HeapTop);
Trans_x:=trans_x+1.4644661;
Trans_y:=trans_y-3.5355339;
Trans_y:=elevate(trans_x,trans_y) + View_Ht;
Yaw_ang:=Yaw_ang+0.785398163;
Initialize_Disposables;
view;
ch:=readkey;
Release(HeapTop);

Restore;
CLOSEGRAPE;
end.
APPENDIX D. ASSEMBLY CODE ROUTINES

This Appendix is referenced in Chapter IV of the thesis in the section regarding Graphics Implementation Issues. The intent of this chapter is to provide a complete listing of the FillTri and FillTriC routines and the routines they need to operate properly. A complete listing of the source code file is provided on the following pages.

```
.MODEL TPASCAL

BytesPerLine EQU 80 ; Number of Bytes in video buffer per line
OriginOffset1 EQU 0 ; Byte offset of (0,0) ON FIRST PAGE
OriginOffset2 EQU 8000h ; Byte offset of (0,0) ON SECOND PAGE
VideoBufferSeg EQU 0A000h ; Video memory location Page 1
ByteOffsetShift EQU 3 ; used to convert pixels to byte offset

Fill50a EQU 0Ah ; 10101010b
Fill50b EQU 55h ; 01010101b
Fill50c EQU 0Ah
Fill50d EQU 55h
Fill150a EQU 44h ; 0100100b
Fill150b EQU 55h
Fill150c EQU 44h
Fill150d EQU 11h
Fill125a EQU 0Ah ; 1010001b
Fill125b EQU 11h
Fill125c EQU 0Ah
Fill125d EQU 11h
Fill112a EQU 20h ; 0010000b
Fill112b EQU 02h ; 00000010b
Fill112c EQU 80h ; 1000000b
Fill112d EQU 08h ; 0000100b

.DATA
VarFill1a DB ? ; var for keeping current fill for 1st row
VarFill1b DB ? ; " " " " " " " 2d row
VarFill1c DB ? ; 3d row
VarFill1d DB ? ; 4th row
CurrFill DB ? ; Byte code to keep track of which VarFill
                  ; to use next
PatCode DB ? ; var to store fill pattern for this row
COLOR DB ? ; var to store current fill color
COLOR1 DB ?
COLOR2 DB ?
X1TEMP DW ? ; var to store temporarily ordered tri
X2TEMP DW ? ; values
X3TEMP DW ? ; var to store temporarily ordered tri
Y1TEMP DW ? ; values
```
TILTEMP DW ? ; var to store temporarily ordered tri
T3TEMP DW ? ; values
RTLT DW ? ; Right Limit of horizontal line
LFLT DW ? ; Left Limit of Horizontal line
RTLTI2 DW ? ; Right Limit of horizontal line
LFLT1 DW ? ; Left Limit of Horizontal line
RTLT2 DW ? ; Used as Right Limit if 1-2 and 1-3
; are low slope
LFLT2 DW ? ; Used as Left Limit if
FIRST13 DB ? ; Used to indicate if 1-3 uses first or last
; value in low slope routine
FIRST12 DB ?
LAST23 DB ?
RTLTA DT DW ?
TCURR DW ? ; Current Y value for Horizontal line
DI13 DW ?
DI12 DW ?
DI23 DW ?
VAR1INC13 DW ?
VAR2INC13 DW ?
VAR1INC12 DW ?
VAR2INC12 DW ?
VAR1INC23 DW ?
VAR2INC23 DW ?
HORIZ1N13 DW ?
HORIZ1N12 DW ?
HORIZINC23 DW ?
LOANS1 DW ? ; Used when multiplication is required
HIANS1 DW ?
SLOPE13 DB ?
SLOPE23 DB ?
SLOPE12 DB ?
ROUTINE13 DW ?
ROUTINE12 DW ?
ROUTINE23 DW ?
COUNTER1 DW ?
COUNTER2 DB ?
SPECASE DB ?

EXTRN TLY:WORD ; Top left y coordinate of view window
EXTRN TLX:WORD ; Top left x coordinate of view window
EXTRN BRY:WORD ; Bottom right y coordinate of view window
EXTRN BRX:WORD ; Bottom right x coordinate of view window
EXTRN WRITEPAGE:WORD ; Video Page to write on
DATA ENDS

; Changing 16 bits to 18h XOR, 08h = and, 10h = or, 00 = Replace

.CODE
EXTRN MYLINE: NEAR

93
PixelAddr PROC NEAR
PUBLIC PixelAddr

; Function: Determine buffer address of pixel in native EGA and VGA:
; 320x200 16 Color
; 640x200 16 Color
; 640x350 16 Color
; 640x480 2 Color
; 640x350 monochrome
; 640x480 16 Color

; Caller: AX = y - coordinate
; BX = x - coordinate
; Returns AX = Bit mask
; BX = byte offset in Buffer
; CL = number bits to shift left
; ES = video buffer segment

MOV CL, BL ; CL := low order byte of x
PUSH DX ; preserve DX
MOV DX, BytesPerLine ; AX := y*BytesPerLine
MUL DX
POP BX
SHR BX, 1
SHR BX, 1
SHR BX, 1 ; BX := x/8
ADD BX, AX ; BX := y*BytesPerLine + x/8
MOV AX, WRITEPAGE
CMP AX, 0
JNE OTHERPAGE
ADD BX, OriginOffset1
JMP GTG

OTHERPAGE: ADD BX, OriginOffset2 ; BX := byte offset in Video Buffer

GTG: MOV AX, VideoBufferSeg
MOV ES, AX ; ES:BX := byte address of pixel
AND CL, 7 ; CL := x & 7
XOR CL, 7 ; CL := number of bits to shift left
MOV AN, 1 ; AN := unshifted bit mask
ret

PixelAddr ENDP

; configure pattern variables for fill

SetPattern PROC pat_no:byte
PUBLIC SetPattern

94
This routine sets the pattern to fill a triangle.

0 = 50% fill
1 = 25% fill
2 = 12.5% fill

; determine which pattern is desired
MOV AL,pat_no ; AL := pat_no
CMP AL,0 ; if pat_no = 0 then go to P01
JE P01

CMP AL,1 ; if pat_no = 1 then go to P02
JE P02 ; else pattern = 2 (12.5% fill)

MOV AH,Fill12a ; (12.5% fill)
MOV [VarFilla],AH
MOV AH,Fill12b
MOV [VarFillb],AH
MOV AH,Fill12c
MOV [VarFillc],AH
MOV AH,Fill12d
MOV [VarFilld],AH
JMP PEEXIT

P01:

MOV AH,Fill150a ; 50% fill
MOV [VarFilla],AH
MOV AH,Fill150b
MOV [VarFillb],AH
MOV AH,Fill150c
MOV [VarFillc],AH
MOV AH,Fill150d
MOV [VarFilld],AH
JMP PEEXIT

P02:

MOV AH,Fill125a ; 25% fill
MOV [VarFilla],AH
MOV AH,Fill125b
MOV [VarFillb],AH
MOV AH,Fill125c
MOV [VarFillc],AH
MOV AH,Fill125d
MOV [VarFilld],AH

PEEXIT: MOV AL,80h ; Set pattern code for 1st row
MOV [PCODE],AL
RET

SetUpPattern ENDP
ConfigGraph PROC NEAR

; configure graphics controller

    MOV  DX,3Ceh        ; DX := Graphics Controller port addr
    MOV  AL,[color]    ; AL := pixel color
    XOR  AL,AL          ; AL := set/reset register number
    OUT  DX,AX

    MOV  AX,0P01h       ; AX := 1111b (bit plane mask for
                        ; Enable Set/Reset Register)
    OUT  DX,AX          ; AL := Enable Set/Reset Register

    MOV  AH,RNwbits     ; bits 3 and 4 of AH := function
    MOV  AL,3           ; AL := Data Rotate/Func Select Reg
    OUT  DX,AX

RET

ConfigGraph ENDP

Horline PROC near

; This routine draws a horizontal line using a fill pattern. It is only used by FillTri and FillTriD.
; The variables YCURR, PATCODE, lfttl, and rtttl must be set by FillTri before calling this procedure.

; Set fill for this line using pattern code

    MOV  AL,80h
    MOV  DX,CX
    MOV  CX,[YCURR]
    AND  CL,03h
    SHL  CL,1
    XOR  AL,CL
    MOV  [PATCODE],AL
    MOV  CX,DX

    CMP  AL,80h          ; check code to determine which row to use
    JE   Q01

    CMP  AL,20h
    JE   Q02

    CMP  AL,08h
    JE   Q03

    MOV  AH,[VARFILLD]
    MOV  [CURRFILL],AH
    JMP  QEXIT

Horline ENDP

96
Q01: MOV AX,[VARFILLA]
    MOV [CURRFILL],AH
    JMP QEXIT

Q02: MOV AX,[VARFILLB]
    MOV [CURRFILL],AH
    JMP QEXIT

Q03: MOV AX,[VARFILLC]
    MOV [CURRFILL],AH

; preserve SI & DI

QEXIT: PUSH SI
    PUSH DI

; routine for horizontal lines (slope = 0)

    MOV AX,[TCURR]
    cmp bx,cx
    jb nochange
    xchg bx,cx

    nochange: mov [lflt1],bx
                mov [rcfl],cx
    CALL PIXELADDR ; AH := Bit Mask
                ; ES:BX -> video buffer
                ; CL := # bits to shift left
    MOV DI,BX ; ES:DI -> video buffer
    MOV DE,AH ; DH := Bit mask for first byte
    NOT DH ; DH := reverse bit mask for first byte
    SHL DE,CL
    NOT DH ; DH := bit mask for first byte

    MOV CX,[RTL1]
    AND CL,7
    XOR CL,7 ; CL := number of bits to shift left
    MOV DL,0FFh ; DL := unshifted bit mask for rightmost byte
    SHL DL,CL ; DL := bit mask for last byte

; determine byte offset of first and last pixel in the line
    MOV AX,[RTL1]
    MOV BX,[LFLT1]

    MOV CL,ByteOffsetShift

    SHR AX,CL ; AX := byte offset of x2
SHR BX, CL            ; BX := byte offset of AL
MOVB CX, AX
SUB CX, BX            ; CX := (#bytes in line) - 1

; get graphics controller port address into DI

MOVB BX, DI           ; BH := bit mask for first byte
                      ; BL := bit mask for last byte

; tentative begin of loop save BX, CX, DI, SI
PUSH BX
PUSH CX
PUSH DI
PUSH SI

HorizLine:
    and BL, [currfill]   ; get pattern correct for first
                      ; and last byte
    and BH, [currfill]

MOVB DX, 3CEh          ; DX := Graphics Controller Port
MOVB AL, 8             ; AL := Bit Mask Register

; make video buffer addressable through DS: SI
PUSH DS                ; preserve DS
PUSH ES
POP DS
MOVB SI, DI           ; DS: SI -> video buffer

; set pixels in leftmost byte of the line
OR BH, BH
JS L43               ; jump if byte aligned (xl is leftmost
                      ; pixel in byte
OR CX, CX
JNZ L42              ; jump if more than one byte in the line
AND BL, BH            ; BL := bit mask for 1st byte
JMP SHORT L44

L42:
    MOV AH, BH         ; update graphics controller
    OUT DX, AX         ; AH := bit mask for 1st byte

MOVSB                ; update bit planes
DEC CX

; use a fast 8086 machine instruction to draw the remainder of the line

L43:
    POP DS            ; make dat segment addressable
MOV  AR,[currfill]
PUSH DS ; PRESERVE DS
PUSH ES ; MAKE VIDEO BUFFER ADDRESSABLE THROUGH DS:SI
POP DS
OUT DX,AX
REPE MOVSB ; Draw line a byte at a time

; set pixels in the rightmost byte of the line
L44: MOV  AH,BL ; AH := bit mask for last byte
OUT  DX,AX ; update graphics controller
MOVSBB ; update bit planes
POP  DS ; restore DS
MOV  AL,[CURRFILL] ; AL := current fill pattern
NOT  AL ; AL := reverse fill pattern
MOV  [CURRFILL],AL ; new fill pattern for same horizontal line. Set pattern to get at pixels not changed on first pass

MOV  AH,[COLOR2] ; set color to background fill color
MOV  [COLOR],AH ; for second pass
MOV  DX,3CEh ; DX := Graphics Controller port addr
XOR  AL,AL ; AL := set/reset register number
OUT  DX,AX

; POP REGISTERS THAT WERE SAVED
POP  SI
POP  DI
POP  CX
POP  DX

and  bl,[currfill] ; get pattern correct for first
; and last byte
and  bh,[currfill]

MOV  DX,3CEh ; DX := Graphics Controller Port
MOV  AL,8 ; AL := Bit Mask Register

; make video buffer addressable through DS:SI
PUSH DS ; preserve DS
PUSH ES
POP DS
MOV  SI,DI ; DS:SI -> video buffer
; set pixels in leftmost byte of the line

OR BH,BH                 ; jump if byte aligned (xl is leftmost
JS L43B                  ; pixel in byte
OR CX,CX
JMP L42B
AND BL,BH
JMP SHORT L44B

L42B:

MOV AH,BH                ; update graphics controller
OUT DX,AX
MOVSB
DEC CX

; use a fast 8086 machine instruction to draw the remainder of the line

L43B:

POP DS                   ; MAKE DAT SEGMENT ADDRESSABLE
MOV AH,[currfill]
PUSH DS                  ; PRESERVE DS
PUSH ES                  ; MAKE VIDEO BUFFER ADDRESSABLE THROUGH DS:SI
POP DS
OUT DX,AX
REP MOVSB               ; Draw line a byte at a time

; set pixels in the rightmost byte of the line

L44B:

MOV AH,BL                ; AH := bit mask for last byte
OUT DX,AX                ; update graphics controller
MOVSB
POP DS                   ; restore DS
MOV AL,[CURRFILL]        ; AL := current fill pattern
NOT AL                   ; AL := reverse fill pattern
MOV [CURRFILL],AL        ; new fill pattern for same horizontal
                        ; line. Set pattern to get at pixels
                        ; not changed on first pass

L55:

MOV AH,[COLOR1]          ; restore primary color after second
MOV [COLOR],AH
MOV DX,3CEh              ; DX := Graphics Controller port addr
XOR AL,AL
OUT DX,AX

SKIPJUMP:
POP DI
POP SI
RET

NorLine ENDP

HISLOPE PROC NEAR
; This routine is used by the HiSlope routines (i.e. HiSlope12) and returns the
; increment to move horizontally to find the border pixel in register DI. Th AX register
; returns the new DI variable
OR AX,AX
JNS NONNEGDI
ADD AX,BX
XOR BX,BX
JMP HIBYE

NONNEGDI: ADD AX,CX
MOV BX,DX

HIBYE: RET
HISLOPE ENDP

LOSLOPE PROC NEAR
; This routine is used by the LowSlope routines (i.e. LowSlope12) and returns the
; increment to move horizontally to find the border pixel in register BX. Th AX register
; returns the new DI variable
PUSH SI
XOR SI,SI ; zero SI

LOSLO: ADD SI,DX ; add horizontal increment
OR AX,AX ; check DI for to see if positive
JNS PODI
ADD AX,DX
JMP LOSLO

PODI: MOV BX,SI
ADD AX,CX
POP SI

RET
LOSLOPE ENDP

HiSlope13 PROC NEAR
; This routine implements Bresenham's algorithm for the High Slope case
MOV AX,[DI13]
MOV BX,[VAR1INC13]
MOV CX,[VAR2INC13]
MOV DX,[HORIZIN13]
CALL HISLOPE
MOV [DI13],AX
MOV AX,[LFLT]
ADD AX,BX
MOV [LFLT],AX
RET

HiSlopel3 ENDP

LowSlopel3 PROC NEAR
MOV AX,[DI13]
MOV BX,[VAR1INC13]
MOV CX,[VAR2INC13]
MOV DX,[HORIZIN13]
CALL LOSLOPE
MOV [DI13],AX
MOV AX,[LFLT]
ADD AX,BX
MOV [LFLT],AX
RET

LowSlopel3 ENDP

Vertical13 PROC NEAR
MOV BX,[XITEMP]
MOV [LFLT],BX
RET

Vertical13 ENDP

HiSlopel2 PROC NEAR
MOV AX,[DI12]
MOV BX,[VAR1INC12]
MOV CX,[VAR2INC12]
MOV DX,[HORIZIN12]
CALL HISLOPE
MOV [DI12],AX
MOV AX,[RTLTL]
ADD AX,BX
MOV [RTLTL],AX
RET

HiSlopel2 ENDP
LowSlope23 PROC NEAR
    MOV AX,[DI23]
    MOV BX,[VARINC23]
    MOV CX,[VAR2INC23]
    MOV DX,[HORIZINC23]
    CALL LOSLOPE
    MOV [DI23],AX
    MOV AX,[RTLTA2]
    ADD AX,BX
    MOV [RTLTA],AX
    RET

LowSlope23 ENDP

Vertical23 PROC NEAR
    MOV BX,[X2TEMP]
    MOV [RTLTA],BX
    MOV [DI23],0
    RET

Vertical23 ENDP

HiSlope23 PROC NEAR
    MOV AX,[DI23]
    MOV BX,[VARINC23]
    MOV CX,[VAR2INC23]
    MOV DX,[HORIZINC23]
    CALL HISLOPE
    MOV [DI23],AX
    MOV AX,[RTLTA2]
    ADD AX,BX
    MOV [RTLTA],AX
    RET

HiSlope23 ENDP

LowSlope23 PROC NEAR
    MOV AX,[DI23]
    MOV BX,[VARINC23]
    MOV CX,[VAR2INC23]
    MOV DX,[HORIZINC23]
    CALL LOSLOPE
    MOV [DI23],AX
    MOV AX,[RTLTA2]
    ADD AX,BX
    MOV [RTLTA],AX
    RET

LowSlope23 ENDP
RE?

LowSlope23 ENDP

Vertical23 PROC NEAR
    MOV BX,[E2TEMP]
    MOV [RTYALT],BX
    MOV [DI23],0
    RET
Vertical23 ENDP

Horizontal23 PROC near
    MOV BX,[E2TEMP]
    MOV [RTY],BX
    RET
Horizontal23 ENDP

PUBLIC FillTri

; This routine fills a triangle identified by its three vertices with a pattern of two colors, n and o. The variable RMbits set the graphics controller to write using AND, OR, or XOR Logic.
; The pattern must be set before calling this routine with routine SetPattern. This routine uses the following routines -- ConfigGraph, PixelAddr, Horline, all HiSlope (i.e. HiSlope2), all LowSlope (i.e. LowSlope2), all Vertical and Horizontal Routines (i.e. Vertical23)
; Order values so that T1 is Y min and T3 is Y max
    PUSH SI
    PUSH DI

; Set fill color for this line using color code

    MOV AL,n           ; AL := fill color************
    MOV [COLOR],AL ; Color = fill color************
    MOV [COLOR1],AL
    MOV AL,0
    MOV [COLOR2],AL
    CALL CONFIGGRAPH
    XOR AL,AL
    MOV [SLOPE13],AL  ; Set all slopes equal to zero
    MOV [SLOPE12],AL
    MOV [SLOPE23],AL
    MOV AX,Y1           ; Move all X & Y values to
    MOV BX,Y2           ; to processor registers before
    MOV CX,Y3           ; beginning sort.
    MOV DX,X1           ; Values are sorted from lowest to
    MOV DI,X2           ; highest Y values. If Y values are
    MOV SI,X3           ; same then sort by lowest X value.
    CMP AX,BX           ; BEGIN SORT
    JE FO1              ; NEED TO CHECK TO ORDER BY X VALUES

104
`NEXT:`

`JA`  `F02`  `; NEED TO REORDER`

`CMP`  `BX,CX`

`JE`  `F03`  `; NEED TO CHECK TO ORDER BY X VALUES`

`JA`  `F04`  `; NEED TO REORDER`

`JMP`  `ORDERED`

`F01:`

`CMP`  `DX,DI`  `; ORDER IS OKAY IF X1 - X2 <= 0 ELSE GO`

`JBE`  `NEXT`  `; TO F02`

`F02:`

`XCHG`  `AX,BX`  `; Exchange X and Y values`

`XCHG`  `DX,DI`  `; Go back and start on point Two.`

`JMP`  `NEXT`

`F03:`

`CMP`  `DI,SI`  `; ORDER IS OKAY IF X2 - X3 <= 0 ELSE GO`

`JBE`  `ORDERED`  `; TO F04`

`F04:`

`XCHG`  `BX,CX`  `; Exchange X and Y values`

`XCHG`  `DI,SI`

`CMP`  `AX,BX`

`JE`  `ALTCHECK`

`JA`  `REORDER`

`JMP`  `ORDERED`

`ALTCHECK:`

`CMP`  `DX,DI`  `; ORDER IS OKAY IF X2 - X3 <= 0 ELSE GO`

`JBE`  `ORDERED`

`REORDER:`

`XCHG`  `AX,BX`  `; Exchange X and Y values`

`XCHG`  `DX,DI`

`ORDERED:`

`MOV`  `[YTEMP] AX`  `; Save the Correctly ordered`

`MOV`  `[Y2TEMP] BX`  `; X and Y Values`

`MOV`  `[Y3TEMP] CX`

`MOV`  `[X1TEMP] DX`

`MOV`  `[X2TEMP] DI`

`MOV`  `[X3TEMP] SI`

`MOV`  `[LFT] DX`  `; Initialize left limit, right limit`

`MOV`  `[RTLT] DX`  `; right limit alternate, and Y current`

`MOV`  `[RTLTALT] DI`

`MOV`  `[YCURR] AX`

`MOV`  `AX,SI`  `; AX := X3`

`MOV`  `BX,1`

`SUB`  `AX,DX`  `; AX := X3 - X1 = DX13`

`JE`  `VERT13`  `; JUMP IF LINE FROM 1 TO 3 IS VERTICAL`

`JNS`  `F05`  `; JUMP IF POSITIVE`

`NEG`  `AX`

`NEG`  `BX`  `; MAKE HORIZ INCR FOR LINE 1-3 NEGATIVE`
P05:

MOV [HORIINC13],BL
MOV BX,[YTEMPL]
MOV CX,[XTEMPL]
SUB CX,BX ; CX = DX13
JE MORI13
CMP CX,BX
JL P06 ; DT < DX LOW SLOPE
XCHG AX,CX ; EXCHANGE DT AND DX
MOV BL,01
MOV [SLOPE13],BL ; Set code for high slope
MOV BX,OFFSET HISHOPE13
MOV ROUTINE13,BX ; Set Routine13 to HISHOPE13
MOV [FIRST13],0 ; Set First13 to False
JMP P06ALT

P06:

MOV BL,0
MOV [SLOPE13],BL ; Set code for LOW slope
MOV BX,OFFSET LOWSLOPE13
MOV ROUTINE13,BX

P06ALT:

SHL CX,1 ; CX := 2 * DT
MOV [VAR1INC13],CX ; INCR 1 FOR 1 - 3 = 2 * DT
SUB CX,AX
MOV [D113],CX ; D113 := (2 * DT) - DX
SUB CX,AX ; CX := 2*(DY - DX)
MOV [VAR2INC13],CX ; VAR2INC13 := 2*(DY-DX)
JMP START12

VERT13:

XOR BX,BX ; LINE FROM 1 TO 3 IS VERTICAL
MOV [HORIINC13],BX ; HORIZ INCR = 0
MOV BX,OFFSET VERTICAL13
MOV ROUTINE13,BX
MOV [SLOPE13],3 ; Set Slope Code = 3
MOV [FIRST13],0 ; Set First13 to False
JMP START12

MORI13: ; ALL THREE POINTS ARE HORIZONTAL DRAW LINE FROM 1 - 3
MOV AX,[YTEMPL]
MOV [YCURR],AX ; YCURR = YTEMPL
MOV BX,[XTEMPL]
MOV [LFLTL],BX ; LFLTL = XTEMPL
MOV CX,[X3TEMPL]
MOV [RTLTL],CX ; RTLTL = X3TEMPL
MOV BX,[LFLTL]
MOV CX,[RTLTL]
CALL MORLINE
JMP FTREXIT

START12:
MOV  AX, DI   ; AX := X2
MOV  BX, L   ; BX = Horizontal Increment
MOV  DI, [X1TEMP]
SUB  AX, DX   ; AX := X2 - X1 = DX12
JX  VERT12   ; JUMP IF LINE FROM 1 TO 2 IS VERTICAL
JNS  P07     ; JUMP IF POSITIVE
NEG  AX
NEG  BX       ; MAKE HORIX INCR FOR LINE 1-2 NEGATIVE

P07:      MOV  [HORIXIN12], BX
MOV  AX, [Y1TEMP]
MOV  CL, [Y2TEMP]
SUB  CL, BX   ; CL = D112
MOV  [COUNTER1], CX
JNE  P08
JMP  MOR1112
NOW:      CMP  AX, AX
JL   P08     ; DT < DX LOW SLOPE
XCHG  AX, CX   ; EXCHANGE DT AND DX
MOV  BL, 01
MOV  [SLOPE12], BL   ; Set code for high slope
MOV  BX, OFFSET HISELOPE12
MOV  ROUTINE12, BX
MOV  [FIRST12], 0
JMP  P08ALT

P08:      MOV  BL, 0
MOV  [SLOPE12], BL   ; Set code for LOW slope
MOV  BX, OFFSET LOWSLOPE12
MOV  ROUTINE12, BX

P08ALT:   SHL  CL, 1   ; CL := 2 * DT
MOV  [VAR1INC12], CX   ; INCR 1 FOR 1 - 2 = 2 * DT
SUB  CX, AX
MOV  DI12, CX   ; DI12 := (2 * DT) - DX
SUB  CX, AX   ; CX := 2*(DT - DX)
MOV  [VAR2INC12], CX   ; VAR2INC12 := 2*(DT-DX)
JMP  START23

VERT12:  XOR  BX, BX   ; LINE FROM 1 TO 2 IS VERTICAL
MOV  [HORIXIN12], BX   ; HORIX INCR = 0
MOV  AX, [HORIXIN13]
CMP  AX, BX
JX   BOTHVERT   ; 1 - 3 AND 1 - 2 ARE VERTICAL
MOV  BX, OFFSET VERTcal2
MOV  ROUTINE12, BX
MOV  BX, [Y1TEMP]
MOV  CX, [Y2TEMP]
SUB  CX, BX   ; CX = D112
MOV  [COUNTER1], CX   ; Counter1 = Y2-Y1
MOV  [SLOPE12], 03   ; Set Slope Code = 3

107
NOV [FIRST12],0 ; Set First12 to False
JMP START23

NOTEVERT:
MOV AX,[YTEMP] ; YTEMP = YTEM
MOV [YCurr].AX ; YTEMP = YTEM
MOV CX,[XTEMP] ; ESTABLISH COUNTER
SUB CX,AX ; SAVE Y1
PUSH AX ; SAVE Y1
MOV AX,[XTEMP] ; LFLT and RLT = XTEMP
MOV [LFLT].AX
MOV [RTLT].AX
PUSH CX
MOV BX,[LFLT]
MOV CX,[RTLT]
CALL NORLINE ; Set one pixel on Y1 line

SNLOOP:
POP AX ; Loop to set one pixel on
POP DX ; each line from y1+1 to y3
CMP AX,0 ; thus drawing a vertical line
JA WHY
JMP FIREXIT

WHY:
DEC AX
INC DX
MOV [YCurr].DX
PUSH DX
PUSH AX
MOV BX,[LFLT]
MOV CX,[RTLT]
CALL NORLINE
JMP SNLOOP

HORIZ12: ; THE LINE 1 - 3 IS THE LEFT BORDER OF THE TRI RIGHT BORDER ; IS FORMED BY LINE 2 - 3
MOV AX,OFFSET VERTICAL12 ; This routine will return
MOV ROUTINE12,AX ; X2 as right limit
MOV AX,2
MOV [HORIZ12].AX
MOV [SLOPE12].2 ; Set Slope Code to horizontal
MOV [FIRST12].0 ; Set Code for First12 to False

START23:
MOV AX,S1 ; AX := X3
MOV NX,1
SUB AX,DX ; AX := X3 - X2 = DX3
JE VERT23 ; JUMP IF LINE FROM 2 TO 3 IS VERTICAL
JBS F09 ; JUMP IF POSITIVE
NEG AX
NEG BX ; MAKE HORIZ INCR FOR LINE 2 - 3 NEGATIVE

F09:
MOV [HORIZ123].BX
MOV BX,[YTEMP]
MOV CX,[I3TEMP]
SUB CX,BX ; CX = DY23
MOV [COUNTER2].CX
JZ HORIZ23 ; 2 - 3 IS HORIZONTAL
CMP CX,AX
JL P10 ; DT < DX LOW SLOPE
XCHG AX,CX ; EXCHANGE DT AND DX

MOV BL,01
MOV [SLOPE23].BL ; Set code for high slope
MOV BX,OFFSET H2SLOP23
MOV ROUTINE23,BX ; Set ROUTINE23 to H2SLOP23
JMP P10ALT

P10:
MOV BL,0
MOV [SLOPE23].BL ; Set code for LOW slope
MOV BX,OFFSET L2SLOP23
MOV ROUTINE23,BX

P10ALT:
SRL CX,1 ; CX := 2 * DT
MOV [VAR1NC23].CX ; INCR 1 FOR 2 - 3 = 2 * DT
SUB CX,AX
MOV [DI23].CX ; DI13 := (2 * DT) - DX
SUB CX,AX ; CX := 2*(DY - DX)
MOV [VAR2INC23].CX ; VAR2INC23 := 2*(DY-DX)
JMP DONE23

VERT23:
XOR BX,BX ; LINE FROM 2 TO 3 IS VERTICAL
MOV [HORIZIN23].BX ; HORII INCR = 0
MOV DX,OFFSET VERTICAL23
MOV ROUTINE23,DX
MOV [SLOPE23].3 ; Set Slope Code for Vertical
MOV BX,[I2TEMP]
MOV CX,[I3TEMP]
SUB CX,BX ; CX = DT23
MOV [COUNTER2].CX
JMP DONE23

HORIZ23:
MOV DX,OFFSET HORIZONTAL23
MOV ROUTINE23,DX
MOV [SLOPE23].2 ; Set Slope Code For Horizontal

DONE23:
MOV AL,[SLOPE12]
MOV BL,[SLOPE13]
CMP BL,0
JZ POSSPECCASE ; JUMP to this label to check for a possible special case
MOV [SPECCASE].0 ; Otherwise set special case to False
CMP AL,0 ; Check to see if Line 1 - 2 is
POSSPECASE: MOV [SPECCASE],1
CMP AL,0 ; If both lines are lowslope jump
JZ CONTCK ; to continue checks
CMP AL,2 ; check to see if 1 - 2 is horizontal
JNZ SETLAST13 ; jump if not horizontal
MOV [FIRST13],1 ; set first13 to true
JMP CHECK23

CONTCK: MOV AL,[HORIZ113] ; compare horizontal increments
MOV AX,[HORIZ112] ; jump if both lines are going the
CMP AX,AX ; the same direction
JZ WHICHOMELOW

CALL LOWSLOPE12 ; Steps to set routine for 1 - 2
MOV BX,[HORIZ112] ; to last
SUB AX,AX
MOV [RLLT].AX
MOV [SPECCASE],0
MOV [FIRST12],0

SETLAST13: CALL LOWSLOPE13 ; Steps to set routine for 1 - 3
MOV BX,[HORIZ113] ; to last
SUB AX,AX
MOV [LFLT].AX
MOV [FIRST13],0
JMP CHECK23

SETLAST12: CALL LOWSLOPE12 ; Steps to set r
MOV BX,[HORIZ112] ;
SUB AX,AX
MOV [RLLT].AX
MOV [FIRST12],0
JMP CHECK23

WHICHOMELOW:

MOV CX,[VAR2INC13]
MOV AX,[VAR1INC13]
MOV BX,AX
SUB AX,CI
MOV CX,[VAR1INC12]
MUL CX
MOV [LANSH1],AX
MOV [M1ANS1].BX
MOV AX,CI
MOV CX,[VAR2INC12]
SUB AX,CI

110
MUL  BX

MOV  BX, [BIANS1]
CMP  BX, BX
JS   NOANSWER
JG   LOWERIS12
JL   LOWERIS13
MOV  [FIRST13], 0
MOV  [FIRST12], 1
JMP  SETLAST13

NOANSWER: MOV  BX, [LOANS1]
OR   BX, BX
JS   POSNEGCASE
OR   AX, AX
JS   LOWERIS13
CMP  BX, AX
JL   LOWERIS13
JE   DIAGLINE

LOWERIS12: MOV  [FIRST13], 0
MOV  [FIRST12], 1
JMP  SETLAST13

LOWERIS13: MOV  [FIRST13], 1
JMP  SETLAST12

DIAGLINE: CALL  LOWSLOPE13
MOV  BX, [HORIZ13]
SUB  AX, AX
MOV  [LYLT], AX
MOV  [FIRST13], 0
MOV  [FIRST12], 1
MOV  [LAST23], 0
JMP  STARTDRAW

POSNEGCASE: OR   AX, AX
JNS  LOWERIS12
CMP  AX, BX
JG   LOWERIS13
JE   DIAGLINE
MOV  [FIRST13], 0
MOV  [FIRST12], 1
JMP  SETLAST13

CHECK23: MOV  [LAST23], 0
MOV  BL, [SLOPE23]
CMP  BL, 0
JNE  STARTDRAW
MOV  BL, [SPECCASE]
CMP  BL, 1
JNE  STARTDRAW
; DRAW TRIANGLE

STARTDRAW:
  MOV    CX, [COUNTER1]
  CMP    CX, 0
  JZ     SECONDLARGE
  PUSH   CX
  MOV    BX, [LPLT]
  MOV    CX, [RTLTL]
  CALL   HORLINE
  POP    CX
  DEC    CX
  CMP    CX, 0
  JZ     SECONDLARGE

LOOP29:
  MOV    AX, [YCURL]
  INC    AX
  MOV    [YCURL], AX
  idea:
    PUSH   CX
    CALL   ROUTINE13
    CALL   ROUTINE12
    MOV    BX, [LPLT]
    MOV    CX, [RTLTL]
    CALL   HORLINE
    POP    CX
    loop  loop29

SECONDLARGE:
  MOV    CX, [COUNTER2]
  CMP    CX, 0
  JE     LASTLINESP
  MOV    AX, [YCURL]
  INC    AX
  MOV    [YCURL], AX
  CALL   ROUTINE13
  MOV    BL, [FIRST12]
CUP

BL, 0
JX SECONDHALF
CALL ROUTINE12
MOV CX,[RLT]
JMP SECSKIP

SECONDHALF:

MOV CX,[RLTALT]
SECSKIP:

MOV BX,[LFT]
CALL NOLINE
MOV CX,[COUTER]
DEC CX
JX LASTLINE
JNS LOOP30
JMP PTREXIT

LOOP30:

PUSH CX
MOV AX,[TCURR]
INC AX
MOV [TCURR], AX
CALL ROUTINE13
CALL ROUTINE23
MOV BX,[LFT]
MOV CX,[RLTALT]
CALL NOLINE
POP CX
LOOP LOOP30

LASTLINE:

MOV AX,[TCURR]
INC AX
MOV [TCURR], AX
MOV BL,[FIRST13]
CMP BL, 1
JNZ THISCASE
CALL ROUTINE13
JMP OTHERLINE

LASTLINESP:

MOV AX,[TCURR]
INC AX
MOV [TCURR], AX
MOV BL,[FIRST13]
CMP BL, 1
JNZ THISCASESP
CALL ROUTINE13
JMP OTHERLINESP

THISCASESP:

MOV AX,[X3TEMP]
MOV [LFT], AX
MOV BL,[SLOPE12]
This routine fills triangles that need to be clipped to fit inside the window.

It is very similar to FillTri except it checks to make sure the pixel is inside the window before it sets its color.

Order values so that Y1 is Y_min and Y3 is Y_max.

```
PUBLIC FillTri

FillTri PROC
  x1:WORD, y1:WORD, x2:WORD, y2:WORD, x3:WORD, y3:WORD, o:BYTE, RMBits:BYTE

ENDP
```

114
; Set fill color for this line using color code

MOV AL,W ; AL := fill color
MOV [COLOR],AL ; Color = fill color
MOV [COLOR1],AL
MOV AL,0
MOV [COLOR2],AL
CALL CONFIGGRAPH
XOR AL,AL
MOV [SLOPE13],AL ; Set all slopes equal to zero
MOV [SLOPE12],AL
MOV [SLOPE23],AL
MOV AX,Y1 ; Move all X & Y values to
MOV BX,Y2 ; to processor registers before
MOV CX,Y3 ; beginning sort.
MOV DX,Y1 ; Values are sorted from lowest to
MOV DI,Y2 ; highest Y values. If Y values are
MOV SI,Y3 ; same then sort by lowest X value.
CMP AX,BX ; BEGIN SORT
JE FO1C ; NEED TO CHECK TO ORDER BY X VALUES
JNS FO2C ; NEED TO REORDER
NEXTC:
CMP BX,CX
JE FO3C ; NEED TO CHECK TO ORDER BY X VALUES
JNS FO4C ; NEED TO REORDER
JMP ORDEREDC

FO1C:
CMP DI,DX ; ORDER IS OKAY IF X1 - X2 <= 0 ELSE GO
JNS NEXTC ; TO FO2

FO2C:
XCHG AX,BX ; Exchange X and Y values
XCHG DX,DI
JMP NEXTC ; Go back and start on point Two.

FO3C:
CMP SI,DI ; ORDER IS OKAY IF X2 - X3 <= 0 ELSE GO
JNS ORDEREDC ; TO FO4

FO4C:
XCHG BX,CX ; Exchange X and Y values
XCHG SI,SI
CMP AX,BX
JE ALTCHECKC
JNS REORDERC
JMP ORDEREDC

ALTCHECKC:
CMP DI,DX
JNS ORDEREDC

REORDERC:
XCHG AX,BX ; Exchange X and Y values
XCHG BX,DI

ORDEREDC:
MOV [Y1TEMP].AX ; Save the correctly ordered
MOV [Y2TEMP].AX ; X and Y values
MOV [Y3TEMP].CX
MOV [XTEMP].DX
MOV [X2TEMP].DI
MOV [X3TEMP].SI
MOV [LEFT].DX ; Initialize left limit, right limit
MOV [RIGHT].DX ; right limit alternate, and Y current
MOV [RIGHTALT].DI
MOV [TCURR].AX

MOV AX, SI ; AX := X3
MOV BX, 1
SUB AX, DX ; AX := X3 - X1 = DX13
JE VERT13C ; JUMP IF LINE FROM 1 TO 3 IS VERTICAL
JNS F05C ; JUMP IF POSITIVE
NEG AX
NEG BX ; MAKE HORIZ INCR FOR LINE 1-3 NEGATIVE

F05C:
    MOV [HORIZ13], BX
    MOV BX, [Y1TEMP]
    MOV CX, [Y3TEMP]
    SUB CX, BX ; CX = DY13
    JL F05C ; DY < DX LOW SLOPE
    XCHG AX, CX ; EXCHANGE DY AND DX
    MOV BL, 01
    MOV [SLOPE13], BL ; Set code for high slope
    MOV BX, OFFSET NISLOPE13
    MOV ROUTINE13, BX ; Set Routine13 to NISLOPE13
    MOV [FIRST13], 0 ; Set First13 to False
    JMP F06ALTC

F06C:
    MOV BL, 0
    MOV [SLOPE13], BL ; Set code for LOW slope
    MOV BX, OFFSET LOWSLOPE13
    MOV ROUTINE13, BX

F06ALTC:
    SEL CX, 1 ; CX := 2 * DY
    MOV [VAR1INC13], CX ; INCR 1 FOR 1 - 3 = 2 * DY
    SUB CX, AX
    MOV [DI13], CX ; DI13 := (2 * DY) - DX
    SUB CX, AX ; CX := 2*(DY - DX)
    MOV [VAR2INC13], CX ; VAR2INC13 := 2*(DY-DX)
    JMP START12C

VERT13C: XOR BX, BX ; LINE FROM 1 TO 3 IS VERTICAL

116
MOV [HORI13],DX ; Horiz Incr = 0
MOV BX,OFFSET VERTICAL13
MOV ROUTINE13,DX
MOV [SLOPE13],3 ; Set Slope Code = 3
MOV [FIRST13],0 ; Set First13 to False
JMP START12C

HORI13C: ; All three points are horizontal draw line from 1 - 3
MOV AX,[Y1TEMP]
MOV DX,TLY
CMP AX,DX
JS HOGO
MOV DX,AX
CMP AX,DX
JNS HOGO
MOV [YCURR],AX ; YCURR = Y1TEMP
MOV BX,[X1TEMP]
MOV CX,[X2TEMP]
CMP BX,CX
JS NOCHANGE35
XCHG BX,CX

NOCHANGE35: MOV DX,TLY
CMP DX,AX
JNS ALLRIGHT29
MOV BX,AX

ALLRIGHT29: MOV AX,DX
CMP AX,DX
JNS HOPROBU
JMP FTREXITC

HOPROBU: CMP AX,CX
JNS ALLRIGHT30
MOV CX,AX

ALLRIGHT30: CMP CX,DX
JNS GOAHEADU
JMP FTREXITC

GOAHEADU: CALL HORLDINE

HOGO: JMP FTREXITC

START12C: MOV AX,DI ; AX := X2
MOV BX,L ; BX = Horizontal Increment
MOV DX,[X1TEMP]
SUB AX,DX ; AX := X2 - X1 = DX12
JS VERT12C ; Jump if line from 1 to 2 is vertical
JNS F07C ; Jump if positive

117
NEG AX
NEG BX ; MAKE HORIZ INCR FOR LINE 1-2 NEGATIVE

PO7C:
MOV [HORIZIN12],BX
MOV BX,[T1TEMP]
MOV CX,[T2TEMP]
SUB CX,BX ; CX = D112
MOV [COUNTER1],CX
JNZ WOMC
JMP HORIZ12C

WOMC:
CMP CX,AX
JL PO8C ; DT < DX LOW SLOPE
XCHG AX,CX ; EXCHANGE DX AND DX
MOV BL,01
MOV [SLOPE12],BL ; Set code for high slope
MOV BX,OFFSET HISHAPE12
MOV ROUTINE12,BX
MOV [FIRST12],0
JMP PO8ALTC

PO8C:
MOV BL,0
MOV [SLOPE12],BL ; Set code for LOW slope
MOV BX,OFFSET LOWSLOPE12
MOV ROUTINE12,BX

PO8ALTC:
SLL CX,1 ; CX := 2 * DT
MOV [VAR1INC12],CX ; INCR 1 FOR 1 - 2 = 2 * DT
SUB CX,AX
MOV [D112],CX ; D112 := (2 * DT) - DX
SUB CX,AX ; CX := 2*(DY - DX)
MOV [VAR2INC12],CX ; VAR2INC12 := 2*(DY-DX)
JMP START23C

VERT12C:
XOR BX,BX ; LINE FROM 1 TO 2 IS VERTICAL
MOV [HORIZIN12],BX ; HORIZ INCR = 0
MOV AX,[HORIZIN13]
CMP AX,BX
JE BOTHVERTC ; 1 - 3 AND 1 - 2 ARE VERTICAL
MOV BX,OFFSET VERT12C
MOV ROUTINE12,BX
MOV BX,[T1TEMP]
MOV CX,[T2TEMP]
SUB CX,BX ; CX = D112
MOV [COUNTER1],CX ; Counter1 = Y2-T1
MOV [SLOPE12],03 ; Set Slope Code = 3
MOV [FIRST12],0 ; Set First12 to False
JMP START23C

BOTHVERTC:
MOV AX,[T1TEMP]
MOV CX,[T3TEMP]

118
MOV DX,TY
CMP AX,DX
JNS NOPROBBV
MOV AX,DX

NOPROBBV: MOV DX,BY
CMP DX,CX
JNS NOPROBBV2
MOV CX,DX

NOPROBBV2: MOV [YCURL],AX
SUB CX,AX
PUSH AX
MOV AX,[YITMP]
MOV DX,TLX
CMP AX,DX
JNS NOPROBBV3
POP AX
JMP PTREEITC

NOPROBBV3: MOV DX,BX
CMP DX,AX
JNS NOPROBBV4
POP AX
JMP PTREEITC

NOPROBBV4: MOV [LFLT],AX
MOV [RTLTY].AX
PUSH CX
MOV BX,[LFLT]
MOV CX,[RTLTY]
CALL HORLINE

SNOOPC: POP AX
POP DX
CMP AX,0
JNZ WHTC
JMP PTREEITC

WHTC: DEC AX
INC DX
MOV [YCURL],DX
PUSH DX
PUSH AX
MOV CX,[RTLTY]
CALL HORLINE
JMP SNOOPC

HORIZITC: ; THE LINE 1 -3 IS THE LEFT BORDER OF THE TRI  RIGHT BORDER
; IS FORMED BY LINE 2 - 3
MOV AX,OFFSET VERTICAL12 ; This routine will return
MOV ROUTINE12,AX ; X2 as right limit
MOV AX,2
MOV [HORISIN12].AX
MOV [SLOPE12].I ; Set Slope Code to horizontal
MOV [FIRST12].0 ; Set Code for First12 to False

START23C:
MOV AX,SI ; AX := X3
MOV BX,1
SUB AX,DI ; AX := X3 - X2 = D123
JE VERT23C ; JUMP IF LINK FROM 2 TO 3 IS VERTICAL
JNS PS9C ; JUMP IF POSITIVE
NEG AX
NEG BX ; MAKE HORIS INCR FOR LINE 2 - 3 NEGATIVE
PS9C:
MOV [HORISIN23].BX
MOV BX,[Y2TEMP]
MOV CX,[Y3TEMP]
SUB CX,BX ; CX = DI23
MOV [COUNTER2].CX
JE HORIS23C ; 2 - 3 IS HORIZONTAL
CMP CX,AX
JL F10C ; DY < DX LOW SLOPE
Xchg AX,CX ; EXCHANGE DX AND DX

MOV BL,01
MOV [SLOPE23].BL ; Set code for high slope
MOV BX,OFFSET HISLOPE23
MOV ROUTINE23,AX ; Set ROUTINE23 to HISLOPE23
JMP F10ALTC

F10C:
MOV BL,0
MOV [SLOPE23].BL ; Set code for LOW slope
MOV BX,OFFSET LOLSLOPE23
MOV ROUTINE23,AX

F10ALTC:
SRL CX,1 ; CX := 2 * DY
MOV [VAR1INC23].CX ; INCR 1 FOR 2 - 3 = 2 * DY
SUB CX,AX
MOV [DI23].CX ; DI13 := (2 * DT) - DX
SUB CX,AX ; CX := 2*(DT - DX)
MOV [VAR2INC23].CX ; VAR2INC23 := 2*(DT-DX)
JMP DONE23C

VERT23C:
XOR BX,BX ; LINK FROM 2 TO 3 IS VERTICAL
MOV [HORISIN23].BX ; HORIZ INC = 0
MOV BX,OFFSET VERTICAL23
MOV ROUTINE23,DX
MOV [SLOPE23].3 ; Set Slope Code for Vertical

120
```assembly
MOV   BX,[HORI$12]
SUB   AX,BX
MOV   [FLT$],AX
MOV   [FIRST$],0
JMP   CHECK23C

WHICHWHEL: 
  MOV   CX,[VAR$1NC13]
  MOV   AX,[VAR$1NC13]
  MOV   BX,AX
  SUB   AX,CX
  MOV   CX,[VAR$1NC12]
  MUL   CX
  MOV   [LOANS$1],AX
  MOV   [HIAN$1],DX
  MOV   AX,CX
  MOV   CX,[VAR$2INC12]
  SUB   AX,CX
  MUL   BX
  CMP   BX,DX
  JLE   NOANSWERC
  JG    LOWERIS12C
  JL    LOWERIS13C
  MOV   [FIRST$],0
  MOV   [FIRST$],1
  JMP   SETLAST13C

NOANSWERC: 
  MOV   BX,[LOANS$1]
  OR    BX,BX
  JS    POSNEGCASEC
  OR    AX,AX
  JS    LOWERIS13C
  CMP   BX,AX
  JL    LOWERIS13C
  JE    DIAGLINEC

LOWERIS12C: 
  MOV   [FIRST$],0
  MOV   [FIRST$],1
  JMP   SETLAST13C

LOWERIS13C: 
  MOV   [FIRST$],1
  JMP   SETLAST12C

DIAGLINEC: 
  CALL  LOWSLOPE13
  MOV   BX,[HORI$13]
  SUB   AX,BX
  MOV   [FLT$],AX
  MOV   [FIRST$],0
```

122
MOV [FIRST12], 1
MOV [LAST23], 0
JMP STARTDRAWC

POSGECASEC: OR
JNS AX, AX
CMP AX, RX
JG LOWERIS12C
JE DIAGLINEC
MOV [FIRST13], 0
MOV [FIRST12], 1
JMP SETLAST13C

CHECK23C:
MOV [LAST23], 0
MOV BL, [SLOPE23]
CMP BL, 0
JNE STARTDRAWC
MOV BL, [SPECCASE]
CMP BL, 1
JNE STARTDRAWC
MOV AX, [HORIXIN13]
MOV BX, [HORIXIN23]
CMP AX, RX
JNE STARTDRAWC
MOV AL, [FIRST13]
CMP AL, 1
JNE STARTDRAWC
CALL LOWSLOPE23
MOV BX, [HORIXIN23]
SUB AX, RX
MOV [RTLALT], AX
MOV [LAST23], 1

; DRAW TRIANGLE

STARTDRAWC:
MOV CX, [COUNTER1]
CMP CX, 0
JX GONSECONDHALFC
PUSH CX
MOV AX, [TCURR]
MOV DX, TLY
CMP AX, DX
JS ABOVETOP
MOV AX, [LVLT]
MOV CX, [RTLTY]
CMP CX, RX
JNS OKAY9
XCHG RX, CX
OKAY9: MOV AX, [YLI]
MOV DX, [BX]
CMP CX, [AX]
JNS OKAY10 ; JUMP IF LEFT LIMIT IS RIGHT OF LEFT SCREEN
CMP CX, AX ; LIMIT IS ON SCREEN
JS ABOVETOP ; JUMP IF RIGHT LIMIT IS OFF LEFT SIDE
OF SCREEN
MOV DX, AX ; SET LFLT = LEFT BORDER
CMP CX, AX ; JUMP IF RIGHT LIMIT IS ON SCREEN
MOV CX, DX ; RIGHT LIMIT IS OFF RIGHT SO MAKE LIMIT
JMP DRAW11 ; EQUAL TO RIGHT OF SCREEN THEN JUMP

GOSECONDHALFC:
JMP SECONDHALFC

OKAY10: CMP BX, DX ; CHECK TO SEE IF LFLT IS ON SCREEN
JNS ABOVETOP ; JUMP IF LEFT LIMIT IS OFF RIGHT SIDE
CMP DX, CX ; CHECK RT SIDE
JNS DRAW11 ; IF RTLT IS ON SCREEN JUMP
MOV CX, DX ; CHANGE RTLT TO RIGHT BORDER

DRAW11: CALL HORLINE

ABOVETOP: POP CX
DEC CX
CMP CX, 0
JS SECONDHALF1NCC

LOOP29C:
PUSH CX
CALL ROUTINE13
CALL ROUTINE12
MOV AX, [YCURR]
INC AX
MOV [YCURR], AX
MOV DX, TLY
CMP AX, DX
JS ABOVETOP2
MOV DX, BRT
CMP DX, AX
JS BELOWBOT
MOV BX, [LFLT]
MOV CX, [RTLT]
CMP CX, AX
JNS OKAY19
XCHG BX, CX

OKAY19: MOV AX, TLY
MOV DX, BRX
```
CMP     BX,AX       ; JUMP IF LEFT LIMIT IS RIGHT OF LEFT SCREEN
JNS     OKAY20      ; JUMP IF LEFT LIMIT IS RIGHT OF LEFT SCREEN
CMP     CX,AX       ; LVLT IS OFF SCREEN SO CHECK TO SEE IF RIGHT
                   ; LIMIT IS ON SCREEN
JS      ABOVETOP2   ; JUMP IF RIGHT LIMIT IS OFF LEFT SIDE
                   ; OF SCREEN
MOV     BX,AX       ; SET LVLT = LEFT BORDER
CMP     BX,CX       
JNS     DRAWIT2     ; JUMP IF RIGHT LIMIT IS ON SCREEN
MOV     CX,DX       ; RIGHT LIMIT IS OFF RIGHT SO MAKE LIMIT
JMP     DRAWIT2     ; EQUAL TO RIGHT OF SCREEN THEN JUMP
OKAY20:  CMP     BX,BX       ; CHECK TO SEE IF LVLT IS ON SCREEN
JNS     ABOVETOP2   ; JUMP IF LEFT LIMIT IS OFF RIGHT SIDE
CMP     BX,CX       ; CHECK AT SIDE
JNS     DRAWIT2     ; JUMP IF RIGHT LIMIT IS ON SCREEN
MOV     CX,DX       ; CHANGE RTL TO RIGHT BORDER
DRAWIT2:  CALL    HR NL INE
ABOVE TOP2:  POP    CX       
loop    loop29C
SECONDHALFINC:  MOV     CX,[COUNTER2]  
CMP     CX,0        
JE      GOLASTLINESPC
CALL    ROUTINE13
MOV     AX,[TCURR]  
INC     AX
MOV     [TCURR],AX  
MOV     BL,[FIRST12]  
CMP     BL,0        
JE      SECONDHALFPC
CALL    ROUTINE12
MOV     CX,[RTLTY]
JMP     SECSKIPC
GOLASTLINESPC:   JMP     LASTLINESPC
BELOWROT:  POP    CX       
BELOWBOTPOP:   JMP     PTRX1ITC
SECONDHALFPC:   MOV     CX,[RTLTYALT]
SECSKIPC:      MOV     BX,[LTY]
                MOV     AX,[TCURR]  
                MOV     BX,LTY
                CMP     AX,BX
```
OKAY39:
  MOV AX,TLX
  MOV BX,BX
  CMP BX,AH
  JNS OKAY39 ; JUMP IF LEFT LIMIT IS RIGHT OF LEFT SCREEN
  CMP CX,AH
  ; LFLT IS OFF SCREEN SO CHECK TO SEE IF RIGHT LIMIT IS ON SCREEN
  JS ABOVETO4 ; JUMP IF RIGHT LIMIT IS OFF LEFT SIDE
  ; OF SCREEN
  MOV BX,AH
  ; SET LFLT = LEFT BORDER
  CMP DX,CX
  JNS DRAWIT4 ; JUMP IF RIGHT LIMIT IS ON SCREEN
  MOV CX,DX
  ; RIGHT LIMIT IS OFF RIGHT SO MAKE LIMIT
  JMP DRAWIT4 ; EQUAL TO RIGHT OF SCREEN THEN JUMP

BELOWBOT21:
  POP CX
  JMP PTRXITC

OKAY40:
  CMP BX,DX
  ; CHECK TO SEE IF LFLT IS ON SCREEN
  JNS ABOVETO4 ; JUMP IF LEFT LIMIT IS OFF RIGHT SIDE
  CMP DX,CX
  ; CHECK RT SIDE
  JNS DRAWIT4 ; IF RTLTY IS ON SCREEN JUMP
  MOV CX,DX
  ; CHANGE RTLTY TO RIGHT BORDER

DRAWIT4:
  CALL HORLINE

ABOVETO4:
  POP CX
  LOOP LOOP30C

LASTLINEC:
  MOV AX,[TCURR]
  INC AX
  MOV [TCURR],AX
  MOV DX,80H
  CMP DX,AX
  JS BELOWBOT20
  MOV BL,[FIRST13]
  CMP BL,1
  JS FIXIT10C
  JMP THISCASEC

FIXIT10C:
  CALL ROUTINE13
  JMP OTHERLINEC

BELOWBOT20:
  JMP PTRXITC

LASTLINESC:
  MOV AX,[TCURR]
  INC AX
  MOV [TCURR],AX
  MOV DX,80H

127
CMP DX, AX
JS BELOWBOT2
MOV BL,[FIRST13]
CMP BL, 1
JNZ THISCASESPC
CALL ROUTINE13
JMP OTHERLINESPC

THISCASESPC: MOV AX,[X3TEMP]
MOV [LFLT], AX
MOV BL,[SLOP12]
CMP BL, 0
JNL DRAWLASTC
MOV BL,[FIRST12]
CMP BL, 1
JNL DRAWLASTC
CALL ROUTINE12
JMP DRAWLASTSPC

OTHERLINESPC: MOV BX,[X2TEMP]
CMP [RTLT], BX

DRAWLASTSPC: MOV BX,[LFLT]
MOV CX,[RTLT]
CMP CX, BX
JNS OKAY49
XCHG BX, CX

OKAY49: MOV AX, TLX
MOV DX, BRX
CMP BX, AX
JNS OKAY50 ; JUMP IF LEFT LIMIT IS RIGHT OF LEFT SCREEN
CMP CX, AX
JS BELOWBOT2 ; JUMP IF RIGHT LIMIT IS OFF LEFT SIDE
JS BELOWBOT2 ; JUMP IF RIGHT LIMIT IS ON SCREEN
MOV BX, AX
CMP CX, DX
JNS DRAWIT5 ; JUMP IF RIGHT LIMIT IS OFF RIGHT SIDE
MOV CX, DX
CMP DX, CX
JNS DRAWIT5 ; RIGHT LIMIT IS OFF RIGHT SO MAKE LIMIT
JMP DRAWIT5 ; EQUAL TO RIGHT OF SCREEN THEN JUMP

OKAY50: CMP BX, DX ; CHECK TO SEE IF LFLT IS ON SCREEN
JNS BELOWBOT2 ; JUMP IF LEFT LIMIT IS OFF RIGHT SIDE
CMP DX, CX
JS BELOWBOT2 ; JUMP IF RTLT IS ON SCREEN
MOV CX, DX
JNS DRAWIT5 ; CHANGE RTLT TO RIGHT BORDER

DRAWIT5: CALL HORIZLINE

128
BELLOWBOT2: JMP FTEXITC

THISCASEC: MOV AX,[X3TEMP]
           MOV [LFLT],AX

OTHERLINEC: MOV BL,[LAST23]
             CMP BL,1
             JNE OTHERCASEC
             MOV BX,[X3TEMP]
             MOV [RTLTLALT],BX
             JMP DRAULASTC

OTHERCASEC: CALL ROUTINE23

DRAULASTC:
           MOV BX,[LFLT]
           MOV CX,[RTLTLALT]
           CMP CX,BX
           JNS OKAY59
           XCHG BX,CX

OKAY59:
           MOV AX,TLX
           MOV DX,BRX
           CMP BX,AX
           JNS OKAY60 ; JUMP IF LEFT LIMIT IS RIGHT OF LEFT SCREEN
           CMP CX,AX ; LFLT IS OFF SCREEN SO CHECK TO SEE IF RIGHT LIMIT IS ON SCREEN
           JS FTEXITC ; JUMP IF RIGHT LIMIT IS OFF LEFT SIDE
           JS DRAULASTC ; OF SCREEN
           MOV BX,AX ; SET LFLT = LEFT BORDER
           CMP DX,CX
           JNS DRAUIT6 ; JUMP IF RIGHT LIMIT IS ON SCREEN
           MOV CX,DX ; RIGHT LIMIT IS OFF RIGHT SO MAKE LIMIT
           JMP DRAUIT6 ; EQUAL TO RIGHT OF SCREEN THEN JUMP

OKAY60:
           CMP BX,DX ; CHECK TO SEE IF LFLT IS ON SCREEN
           JNS FTEXITC ; JUMP IF LEFT LIMIT IS OFF RIGHT SIDE
           CMP DX,CX ; CHECK RT SIDE
           JNS DRAUIT6 ; IF RTLTL ALT IS ON SCREEN JUMP
           MOV CX,DX ; CHANGE RTLTL ALT TO RIGHT BORDER

DRAUIT6:
           CALL HORLINE

FTEXITC: POP DI
          POP SI
          RET

FillTriC ENDP

Restore PROC

         PUBLIC Restore
; Restores default graphics controller state and returns to caller

XOR AX, AX ; AX := 0, AL := 0
OUT DX, AX ; restore Set/Reset Register

INC AX ; AX := 0, AL := 1
OUT DX, AX ; restore Enable/Reset Register

MOV AL, 3 ; AX := 0, AL := 3
OUT DX, AX ; AL := DataRotate/Func Select reg 

MOV AX, OFFFH ; AX := llllllll, AL := 8
OUT DX, AX ; restore Bit Mask register

RET

Restore ENDP

FillWindow PROC FillColor:byte, RHNbits:Byte
PUBLIC FillWindow

; This routine fills a window determined by TLX, TLX, BLY, BRY with
; the color set by the variable Fillcolor. The RHNbits variable sets
; sets the controller to OR, XOR, or AND Logic the filling to the pixel
; color that is already set.

; preserve SI & DI

PUSH SI
PUSH DI

; routine for Horizontal lines (slope = 0)

MOV AL, Fillcolor
MOV [COLOR], AL
CALL CONFIGGRAPH
MOV AX, TLX
MOV BX, BLY
SUB BX, AX ;Establish counter
ADD BX, 1
MOV [COUNTER1], BX ;SAVE COUNTER
MOV BX, TLX
MOV cx, BRY
mov [1fil1], bx
mov [rtlti], cx
CALL PIXELADDR ; AX := Bit Mask
; ES:BX -> video buffer
; CL := # bits to shift left

MOV DI, BX ; ES:DI -> video buffer

130
MOV  DH,AX    ; DH := bit mask for first byte
NOT  DH     ; DH := reverse bit mask for first byte
SEL  DH,CL  ; DH := bit mask for first byte

MOV  CX,[RTLT1]
AND  CL,7
XOR  CL,7    ; CL := number of bits to shift left
MOV  DL,OFFh ; DL := unshifted bit mask for rightmost byte

SEL  DL,CL   ; DL := bit mask for last byte

; determine byte offset of first and last pixel in the line

MOV  AX,[RTLT1]
MOV  BX,[LHLT1]

MOV  CL,ByteOffsetShift

SHR  AX,CL   ; AX := byte offset of x2
SHR  BX,CL   ; BX := byte offset of x1
MOV  CX,AX
SUB  CX,BX   ; CX := (#bytes in line) - 1

; get graphics controller port address into DX

MOV  BX,DX   ; BH := bit mask for first byte
            ; BL := bit mask for last byte

; tentative begin of loop save bx, cx, di, si

HorizLine2: PUSH  BX
            PUSH  CX
            PUSH  DI
            PUSH  SI

MOV  DX,3CEh  ; DX := Graphics Controller Port
MOV  AL,0     ; AL := Bit Mask Register

; make video buffer addressable through DS:SI

PUSH  DS      ; preserve DS
PUSH  ES
POP  DS
MOV  SI,DI    ; DS:SI -> video buffer

; set pixels in leftmost byte of the line

131
OR BH,BH  ; jump if byte aligned (xl is leftmost pixel in byte
JS L43P
OR CX,CX  ; jump if more than one byte in the line
JNZ L42P
AND BL,BH  ; BL := bit mask for 1st byte
JMP SHORT L44P

L42F: MOV AH,BH  ; update graphics controller
OUT DX,AX  ; AH := bit mask for 1st byte
MOVSB  ; update bit planes
DEC CX

; use a fast 8086 machine instruction to draw the remainder of the line
L43F: POP DS  ; MAKE DAT SEGMENT ADDRESSABLE
MOV AL,11111111b
PUSH DS  ; PRESERVE DS
PUSH ES  ; MAKE VIDEO BUFFER ADDRESSABLE THROUGH DS:SI
POP DS
OUT DX,AX
REP MOVSB  ; Draw line a byte at a time

; set pixels in the rightmost byte of the line
L44F: MOV AH,BL  ; AH := bit mask for last byte
OUT DX,AX  ; update graphics controller
MOVSB  ; update bit planes
POP DS  ; restore DS
MOV CX,[COUNTER1]
DEC CX
JZ L555

; Move loop counter so it is preserved
MOV [COUNTER1],CX

; POP REGISTERS THAT WERE SAVED
POP SI
POP DI
POP CX
POP BX

; CHANGE START ADDRESS TO NEXT LINE
ADD DI, BYTESPERLINE
JMP Horizline2 ; make another pass on line to fill
; background color

LSSS:

POP SI
POP DI
POP CX
POP BX

POP DI
POP SI

RET

FillWindow ENDP

CODE ENDS
END
LIST OF REFERENCES


INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Cameron Station
   Alexandria, VA 22304-6145

2. Library, Code 0142
   Naval Postgraduate School
   Monterey, CA 93943-5002

3. Director for Command, Control and Communications Systems, Joint Staff
   Washington, D.C. 20318-6000

4. C3 Academic Group, Code CC
   Naval Postgraduate School
   Monterey, CA 93943-5000

5. Deputy Undersecretary of the Army
   for Operations Research
   Room 2E261, The Pentagon
   Washington, D.C. 20310

6. Dr. Samuel H. Parry, Code OR/Py
   Department of Operations Research
   Naval Postgraduate School
   Monterey, CA 93943-5002

7. CPT Thomas G. Dodd
   P.O. Box 164
   Lovingston, VA 22949

8. Commander
   U.S. Army Training and Doctrine Command
   ATTN: ATCD
   Fort Monroe, VA 23651-5000

9. Commander
   U.S. Army TRADOC Analysis Command
   ATTN: ATRC
   Fort Leavenworth, KS 66027-5200
10. Director
U.S. Army TRADOC Analysis Command-Ft. Leavenworth
ATTN: ATRC-F
Fort Leavenworth, KS 66027-5200

11. Director
U.S. Army TRADOC Analysis Command-Ft. Leavenworth
ATTN: ATRC-FOQ (Technical Information Center)
Fort Leavenworth, KS 66027-5200

12. Director
U.S. Army TRADOC Analysis Command-WSMR
ATTN: ATRC-W
White Sands Missile Range, NM 88002-5502

13. Director
U.S. Army TRADOC Analysis Command-WSMR
ATTN: ATRC-WSL (Technical Library)
White Sands Missile Range, NM 88002-5502

14. Director
U.S. Army TRADOC Analysis Command-Research Directorate
ATTN: ATRC-RD
White Sands Missile Range, NM 88002-5502

15. Director
U.S. Army TRADOC Analysis Command-Monterey
ATTN: ATRC-RDM
P.O. Box 8692
Monterey, CA 93943-0692

16. Director
U.S. Army TRADOC Analysis Command-Ft. Benjamin Harrison
ATTN: ATRC-FB
Fort Benjamin Harrison, IN 46216-5000

17. Director
U.S. Army Concepts Analysis Activity
8120 Woodmont Ave.
Bethesda, MD 20814-2797

18. U.S. Army Combined Arms Research Library (CARL)
ATTN: ATZL-SWS-L
Fort Leavenworth, KS 66027