RL-TR-94-34
In-House Report
March 1994

PARALLEL IMPLEMENTATION OF THE TERRAIN MASKING ALGORITHM

Milissa M. Benincasa

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

This effort was funded totally by the Laboratory Director's fund.

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-94-34 has been reviewed and is approved for publication.

APPROVED: 

SAMUEL A. DINITTO, JR., Chief
Software Technology Division
Command, Control, & Communications Directorate

FOR THE COMMANDER: 

JOHN A. GRANIERO
Chief Scientist
Command, Control, & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL (C3CB) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1994	In-House May 91 - Dec 93

4. TITLE AND SUBTITLE
PARALLEL IMPLEMENTATION OF THE TERRAIN MASKING ALGORITHM

5. FUNDING NUMBERS
PE - 62702F
PR - LDFP
TA - 01
WU - H1

6. AUTHOR(S)
Milissa M. Benincasa

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Rome Laboratory (C3CB)
525 Brooks Road
Griffiss AFB NY 13441-4505

8. PERFORMING ORGANIZATION REPORT NUMBER
RL-TR-94-34

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Laboratory (CD)
26 Electronic Pky
Griffiss AFB NY 13441-4514

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Milissa M. Benincasa/C3CB (315) 330-7650

This effort was funded totally by the Laboratory Director's fund.

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

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The terrain masking algorithm is a key algorithm used in battle management and mission planning in choosing the optimal location site for mobile radar and jammer systems. This algorithm calculates the region of clear line-of-sight at a particular altitude for an emitting device. Currently, to calculate the site for one radar or jammer system it can take up to eight hours running on a uniprocessor system. This is unacceptable because it increases the time and cost involved in the planning of a successful mission strategy.

A solution to this problem is to utilize a parallel architecture and modify the existing sequential version of the terrain masking algorithm so that it can effectively execute in parallel. The approach presented uses the transputer architecture for executing the algorithm. This architecture was selected because it allows the simulation of a number of MIMD message passing architecture topologies, without having to drastically rewrite the parallel implementation of the algorithm. The approach for parallelizing the terrain masking algorithm involves reverse engineering the sequential version. The algorithm decomposition involved defining the data dependencies inherent in the sequential version so that the data can be properly partitioned for parallel execution.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Software Development, Parallel Processing, Software Engineering, Transputers, Terrain Masking	76
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	U/L

TABLE OF CONTENTS

CHAPTER 1: Terrain Masking Algorithm and Algorithm Analysis pg. 1

1.1 Introduction pg. 1

1.2 Electronic Mission Planning pg. 1

1.3 The Improved Many-On-Many Mission Planning System pg. 2

1.4 Analysis of the IMOM Terrain Masking Code pg. 6

1.5 The Multi-Source Integrated Viewing System (MIVS) pg. 9

1.6 Analysis of the MIVS Terrain Masking Code pg. 11

CHAPTER 2: Timing the Sequential Version of the Terrain Masking Algorithm pg. 14

2.1 Introduction pg. 14

2.2 Terrain Masking Test Cases pg. 15

CHAPTER 3: Parallelizing the Terrain Masking Algorithm pg. 20

3.1 Introduction pg. 20

3.2 Determining What Existing Code Needed to Be Modified pg. 20

3.3 Partitioning the Terrain Elevation Data pg. 22

3.4 Determining Communication between the Sun Host and Transputers pg. 24

3.5 Software Written to Run on the Transputer Architecture pg. 26

CHAPTER 4: Results pg. 28

4.1 Problems Encountered pg. 28

4.2 Possible Results If Parallel Execution was Successful pg. 32

CHAPTER 5: Conclusion pg. 34

BIBLIOGRAPHY pg. 35

APPENDIX A pg. 37

APPENDIX B pg. 38

APPENDIX C pg. 41

LIST OF FIGURES

Figure 1. Radial Approach to Terrain Masking pg. 3

Figure 2. Factors Affecting Terrain Masking Calculation pg. 2

Figure 3. Timing for Terrain Masking Varying Terrain Resolution pg. 5

Figure 4. Example of a Record in the Terrain Data Relation pg. 7

Figure 5. Example of a Record in the Terrain Subroutine Relation pg. 8

Figure 6. Matrix Grid Approach to Terrain Masking pg. 10

Figure 7. Executing Terrain Masking Varying the Range pg. 15

Figure 8. Executing Terrain Masking Varying Above Ground Level Height pg. 16

Figure 9. Executing Terrain Masking Varying the Field of View Angle pg. 17

Figure 10. Executing Terrain Masking Varying the Observer Position pg. 18

Figure 11. Data Distribution for Rows Above the Observer pg. 23

Figure 12. Data Distribution for Rows Below the Observer pg. 23

Figure 13. View of Sun and Transputer Configuration pg. 24

Figure 14. Sample of Meiko MRUN Command Using -M Option pg. 29

Figure 15. Sample of Meiko MRUN with sysHeapVec Size Reduced pg. 31

CHAPTER 1 Terrain Masking Algorithm and Algorithm Analysis

1.1 Introduction

The future for increased processing performance lies in the parallel architectures of today and the future. Although these architecture can provide increased processing power over current sequential Von Neuman architectures, the challenge is how to obtain this capability when developing software to run on these architectures.

The objective of this effort is attempt to parallelize a terrain masking algorithm which is part of the Multi-Source Integrated Viewing System (MIVS). This algorithm has an identified need for increased processing time. The challenge of this effort is to identify the inherent parallelism and select an appropriate method for parallelizing the algorithm and mapping it to the selected Meiko transputer architecture. Another objective of this effort is to identify the problems that exist when parallelizing existing sequential code for any given high performance architecture.

1.2 Electronic Mission Planning

Electronic Mission Planning is a process that is made up of a number of different functions which navigate an aircraft to and from a target [Benincasa92]. These functions include such things as logistics, threats, and resource availability. Terrain masking is one function that is utilized in a number of Air Force Electronic Mission Planning systems. For example, at Rome Laboratory, two systems the Improved Many-On-Many (IMOM) System [AFEWC89] and the Multi-Source Integrated Viewing System (MIVS) [Souza92] utilize terrain masking.

Terrain Masking is the process of determining where an electronic emitting device such as a radar or jammer has line-of-sight for a given range, field of view and above ground level height. Line-of-sight means that the emitting device is able to detect objects directly above the horizon at a given position [LaBatt90]. The results of the

terrain masking algorithm are displayed differently depending upon the system. For example, with the IMOM system the results of terrain masking produce an accurate display of rings around the radar or jammer with reference to the terrain. The MIVS system, on the other hand uses different colors to show where an electronic emitting device has line-of-sight with respect to the terrain.

1.3 The Improved Many-On-Many Mission Planning System

The Improved Many-On-Many system was developed by Air Force Electronic Warfare Center (AFEWC) in Texas to assist combat mission planners [LaBatt90]. The objective of this program was to develop a tool that would aid in planning Electronic Combat (EC) missions in the context of overall mission planning [AFEWC89]. Electronic Combat mission planning is a part of mission planning that deals with the electromagnetic spectrum of a battlefield. It was felt a tool was needed to help in the process of mission planning because of the growing complexity of current air defenses. IMOM consists of nine separate, one-on-one engineering models. These models include the capabilities of displaying terrain masking effects, Airborne Warning and Control System (AWACS) radar coverage, and infrared tracking range functions just to name a few [AFEWC89]. IMOM also provides the capability of developing requirements for further EC missions and provides the ability to derive new requirements for improving the system.

Terrain masking is one function in the overall IMOM system. Terrain masking provides a user with the capability to show how limitations of terrain can effect the coverage amount of a radar or weapon system. The IMOM terrain masking function calculates, where an electronic emitting device, both ground and air based, has visibility for a given altitude. The IMOM system provides radars and jammers as the two types of emitting devices. The output from terrain masking is a display of rings around the emitting device. These rings reflect the characteristics of the terrain that may impede

what the device can see if placed at the specified location. If the output results obtained do not meet the requirements for the optimal operation of the emitting device, new location parameters for the emitting device can be entered into the terrain masking function to see if they satisfy the requirements.

The terrain masking algorithm utilized in the IMOM system is called a radial or spoke wheel algorithm. It is called a radial or spoke wheel algorithm because the terrain masking calculation is performed by drawing radial lines from the specified radar system to a given target aircraft altitude at angular increments for a full 360 degrees around the radar system (refer to figure 1).

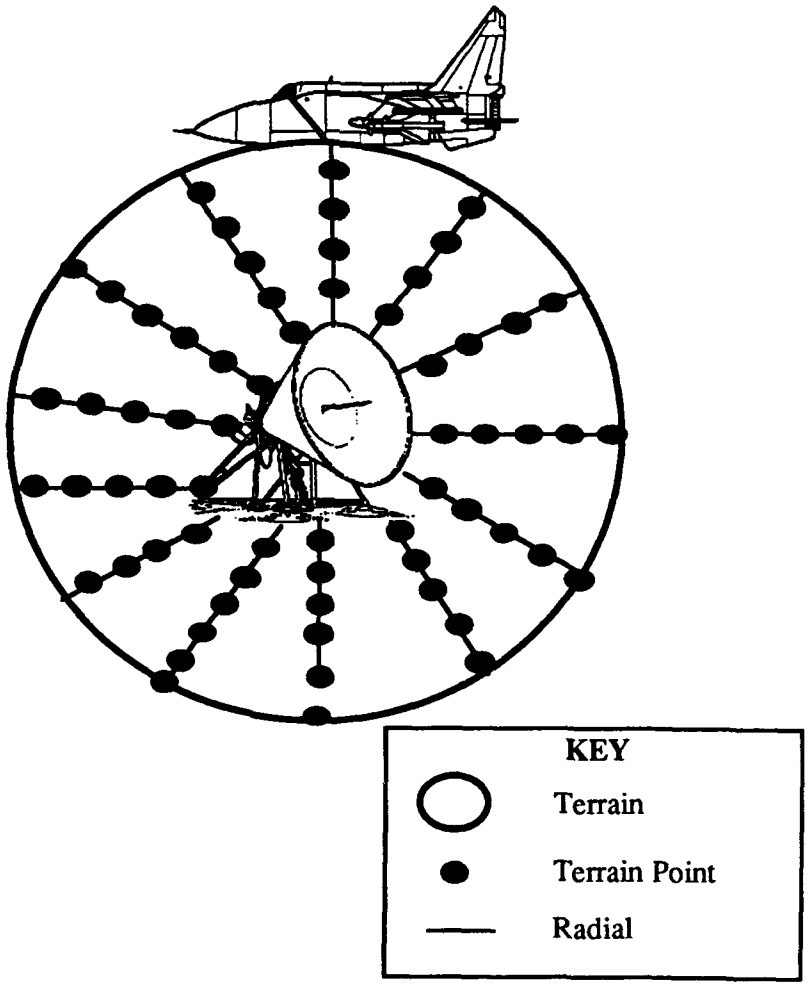


Figure 1. Radial Approach to Terrain Masking

The length the radial lines are drawn is based on how far the emitting device is from the given aircraft altitude. The aircraft altitude is a user supplied parameter. The distance between radial lines is defined as angular increment. Angular increment is measured in degrees and is also a user supplied parameter. The distance between the terrain points on each radial is defined as terrain resolution. The terrain resolution setting can vary between 1 and 9 and is also a user supplied parameter. A setting of 1 would specify a fine grain terrain resolution, while a setting of 9 will specify a coarse grain resolution. If fine grain terrain resolution (1) is selected, the distance between the points is set at .25 nautical miles (approximately 1500 feet) and the sampling along the radial is calculated more frequently. If coarse grain terrain resolution (9) is selected, the distance between the points is set at 2.25 nautical miles (approximately 13,500 feet) and the sampling along the radials is calculated less frequently. In general, the terrain resolution is calculated by multiplying the setting number by .25 nautical miles or by 1500 feet.

The IMOM terrain masking algorithm is computationally intensive due to four factors. These factors are: the number of radars specified to be masked, the altitude of the aircrafts, the terrain resolution, and the number of degrees between the radials. According the report by [Labatt90], the calculations of the rings around a emitting device are influenced by the above factors (refer to figure 2).

FACTOR TYPE	INCREASES TIME	DECREASES TIME
Number of Radars	more than one radar	less radars and small area
Terrain Resolution	fine grain resolution	coarse grain resolution
Aircraft Altitude	high altitude	low altitude
Angular Increments	small increments	large increments

Figure 2. Factors Affecting Terrain Masking Calculation

In order to decrease the amount of time to calculate terrain masking, the user can modify the number of radars, terrain resolution, aircraft altitude, and the angular increments of

the radials. For example, according to the report by [Labatt90], just by varying the terrain resolution, the time for terrain masking can be decreased (refer to figure 3).

TERRAIN RESOLUTION	TIME FOR MASKING IN SECONDS
1	50.2
2	33.8
3	29.0
4	26.4
5	25.8
6	24.4
7	24.7
8	23.5
9	23.2

Figure 3. Timing for Terrain Masking Varying Terrain Resolution

By changing the terrain resolution parameter from 1 to 9; 27 seconds is saved in calculating terrain masking. But this savings does not come without a price. Instead of checking the terrain approximately every 1500 feet, the terrain is now checked every 13,500 feet. The trend that can be discerned from this is that time is decreased, but the result is less accuracy in the terrain masking calculation. [Labatt90] showed a similar case for modifying the angular increment parameter. Using an angular increment of 90 degrees versus and angular increment of 1 degree saved 86.5 seconds. But the sacrifice again was the loss of accuracy in the terrain masking calculation.

As a result of the analysis conducted by [LaBatt90], terrain masking seemed like a viable candidate for parallelization. If the processing time could be reduced for the computationally intensive terrain masking case, then the user would not have to sacrifice accuracy for speed of execution.

1.4 Analysis of the IMOM Terrain Masking Code

In order to determine how to parallelize the terrain masking algorithm, the software for the IMOM system was analyzed. The IMOM system is composed of approximately 36,000 lines of Fortran77 code. Since terrain masking is only one function in the IMOM system, the first task was to determine what portion of the 36,000 lines of code constituted the terrain masking algorithm. Determining the number of lines of code was a manual process since the documentation available did not provide this information and a software tool for counting lines of code was not available. The process utilized consisted of constructing call trees of the IMOM software and manually counting the number of lines of code for the terrain masking algorithm (refer to appendix A for the call tree) [Benincasa92]. It was determined that terrain masking consisted of about 10,000 lines of code.

The next step was to examine data dependencies inherent in the terrain masking algorithm. This would help to identify any data parallelism present in the algorithm. Data parallelism is defined as a type of parallelism where the same operation or program instruction can be executed over a large array of data [Hillis]. This seemed to be a valid step because of the type of processing involved in the terrain masking algorithm. The terrain masking algorithm processes large volumes of data. At this point it would have been beneficial if there was a software tool available that would have helped in identifying inherent data dependency. But since software tool support was not available, the approach taken was to develop a relational database consisting of all global data used in the terrain masking algorithm. The database was developed on a Macintosh computer using Double Helix, a product from Odesta [Odesta91].

Double Helix provides the standard features of a relational database as well as a non-procedural visual dataflow language which uses a command string programming language. Double Helix defines a database as a collection. The notation of a collection is different than that of a file in a standard relational database. A file contains information

on a single type of data where as a collection can contain information on many types of data. One collection was defined for the terrain masking algorithm. This collection consists of two relations. The first relation contains the terrain masking data. The terrain masking data relation consists of the following information: data name, data description, data type, data range, and which subroutines use the data (for an example of a data relation record, refer to figure 4) [Benincasa92].

TERRAIN MASKING VARIABLE TYPES

NAME	AOI_SUB_LONG
DESCRIPTION	This variable defines a one dimensional array of real numbers of length two which holds the longitude angles in radians of the southwest and northeast corners respectively of the angle of intersection or the angle of intersection subset
DATATYPE	One-Dimensional Array of Reals
VALIDRANGE	-2Pi to 2Pi
USEDBY	CF_TERRAIN_MASKING MU_INTERSECTION_OF_TWO_RECTANGS

Figure 4. Example of a Record in the Terrain Data Relation

The second relation defines the terrain masking subroutines. This relation contains the following information: subroutine name, subroutine function, input parameters, output parameters, what subroutines call this subroutine, and subroutines called from this subroutine (for an example of a subroutine relation record, refer to figure 5) [Benincasa92].

TERRAIN MASKING SUBROUTINES

SUBROUTINE NAME	CF_TERRAIN_MASKING
SUBROUTINE FUNCTION	This subroutine is the interface between the Terrain Masking algorithm/utility code and the application layer. As such, it gathers feature attributes and data from the DCA online database, receives user and applications control inputs as parameters, and saves the ground shadow bitmap, corresponding ground shadow contours, and the above-ground shadow height matrix in the DCA online database as temporary features.
INPUT PARAMETERS	CF_RECORD, WKAREA_ADR,WKAREA _SIZE
OUTPUT PARAMETERS	CF_RECORD, CF_STATUS
CALLED FROM	N/A - IS THE MAIN PROGRAM
SUBROUTINES CALLED WITHIN	L_GET$FEATURE MS_MOVE_DATA L_PUT$MESSAGE MU_TRANSPOSE_MATRIX L_GET_MATRIXDEF L_CREATE_TEMP_MATRIX CA_SHADOW_SEEFAR L_GET_MATRIX MU_CALC_THREAT_BOUNDARY MU_BOUNDING_RECTANGLE MU_INITIALIZE_COORD_TRANS MU_INTERSECTION_OF_TWO_RECTANGS

Figure 5. Example of a Record in the Terrain Subroutine Relation

By utilizing the query feature in the Double Helix database system, analysis of the terrain masking algorithm could be conducted by using the two defined relations. For example, the query feature provided the capability of taking a given variable in the terrain masking algorithm and determining which subroutines used it. A query could also be set up to see which subroutines actually modified the variable. Another query provides information on the names of the subroutines that are called from a given subroutine. The development of the relational database helped in determining the data dependencies in the terrain masking algorithm. It also provided insight to partitioning the data.

The final step in the analysis process was to utilize a Rome Laboratory developed tool called Parallel Proto. Parallel Proto (PProto) is a rapid prototyping tool for building parallel programs [Acosta91]. PProto provides assistance in the high-level analysis of software. It was at this point in the research that the MIVS terrain masking algorithm was identified. The MIVS algorithm is based on some of the IMOM functionality but has a slightly different approach to calculating terrain masking. The biggest advantage of the MIVS terrain masking algorithm is that the execution time is less than the IMOM terrain masking algorithm. Since one of the goals of this research was to reduce the actual calculation time of the terrain masking algorithm, the MIVS terrain masking algorithm needed to be analyzed since it already reduces the calculation time. The final step in the analysis process of the IMOM terrain masking algorithm was postponed to review the MIVS system. This was necessary to determine if the MIVS terrain masking should be parallelized instead of the IMOM terrain masking algorithm or if parallelization was still an issue for this algorithm.

1.5 The Multi-Source Integrated Viewing System (MIVS)

The Multi-Source Integrated Viewing System was developed by Grumman Data Systems for the Rome Laboratory Directorate of Intelligence and Reconnaissance. The MIVS effort demonstrated the usefulness of cartography, spatial analysis, and image processing in the Sentinel Byte program Unit-level Intelligence environment [Souza92]. Sentinel Byte is a software system that provides intelligence data handling deployable on a unit-level workstation [Souza92]. This software system provides intelligence personnel with the following capabilities: order of battle maintenance, imagery dissemination, targeting updates, assessment of intelligence data, integrated threat pictures for Mission Support Systems, intelligence briefings and debriefings, and provides input to mission support functions.

MIVS is based on two previous Rome Laboratory systems. One system dealt with image intensification and annotation techniques and the other system was IMOM which deals with digital cartographic applications. The MIVS system combines the cartographic and spatial analysis features of IMOM with image processing capabilities. One of the major requirements of the MIVS system was to provide the capability to run it on multiple types of workstation platforms. IMOM can only run on a VAX mainframe system. MIVS runs on a DEC 3540 VAX station, SUN 4/370, and Dell 433TE 80486 PC. This capability provides MIVS with greatly flexibility for mission planning because the system is more portable.

The MIVS algorithm for terrain masking differs from the IMOM terrain masking algorithm. The MIVS system uses a matrix grid approach for calculating the terrain masking (refer to figure 6).

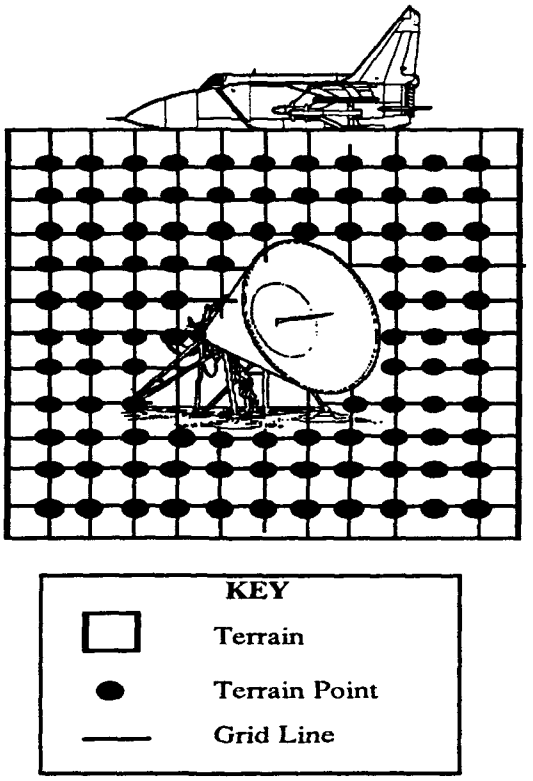


Figure 6. Matrix Grid Approach to Terrain Masking

In a matrix grid approach, a grid is drawn around the emitting device. This grid defines the field of view for the emitting device. Successive rows and columns are then

constructed in the grid to produce a NxN matrix. Then given the position of the emitting device and the height of the device, every point in the matrix is examined to determine visibility. Every point is visited only once by traversing columns successively farther from the emitting device. Columns are traveled from right to left. Within a column, points are traversed by rows starting at the emitting device and moving outward, first above the emitting device, then below. For each point, visibility is assessed by determining how tall an object would have to be to be seen at that point. The visibility algorithm calculates the visibility based on only one intersection with a previous column rather then comparing it to all previous columns. This is one reason why the algorithm is faster than the IMOM terrain masking algorithm. Another advantage to the matrix grid approach is that more terrain data can be processed to obtain a more accurate calculation of the given terrain.

1.6 Analysis of the MIVS Terrain Masking Code

The MIVS system consists of approximately 136,000 lines of C code. The terrain masking algorithm makes up 10,000 lines of the 136,000 lines. In analyzing MIVS, it was not necessary to construct call trees because the MIVS software design document provided a detailed description of the terrain masking software. [Grumman92].

Like IMOM, the MIVS terrain masking algorithm processes large amounts of data. In order to analyze and determine if data parallelism was present, the Parallel Proto (PProto) tool was utilized to do high level design analysis of the terrain masking algorithm. PProto provides a visual language and editor for specifying hierarchical dataflow graphs, a resource modeling tool for defining parallel and distributed architectures, mapping mechanisms for mapping the dataflow graph components to hardware components, simulation capabilities for simulating the defined prototype, and software reuse capabilities [Acosta91]. PProto was utilized to reengineer the existing code (refer to Appendix B for the PProto charts).

The first graph in Appendix B gives a top level view of the terrain masking algorithm. The circles represent process nodes. In PProto, a process node represents a unit of computation [ISSI91]. A process node contains behavior rules which can define a computation or an algorithm. It can communicate with other process nodes, it can contain local data, and it can access data stores. There are two types of process nodes in PProto. The plain process nodes represent elementary functions that cannot be further decomposed. The process nodes, with a picture in the circle, represent complex functions that have a decomposition. Opening a complex process node will produce a graph of the decomposition. For example, the process node Process_Matrix_Data (in the first graph in Appendix B) is a compound process node and contains a subgraph.

Process nodes communicate with other process nodes using ports and connections. If two process nodes are to communicate, they must first have ports associated with the node. A process node can accept and supply data to and from other process nodes via input and output ports. Ports are connected to other ports using connections. There are three different types of connections that can be used. The three types of connections are stream, synchronized, and sampled. In a stream connection, the sending process node sends the information to the receiving process node and then proceeds with execution. In a synchronized connection, the sending process node sends the information to the receiving process node and waits for acknowledgment of reception before proceeding with the execution. Finally, sampled connection provides the capability to monitor continuous data.

The square box on the first graph in appendix B represents a data store. A data store can contain any data objects that are supported in the PProto tool. Data stores can be both read from and written to. In this graph, Terrain_Masking_Data represents a global data store which is accessed by a number of process nodes. The dashed lines from the data store to the process node represent access to the data store. The arrow at the end of the dashed line specifies direction. For example, if the data store

Terrain_Masking_Data has a dashed line with an arrow pointing to it from a process node, this signifies that a process node is writing to the data store. If the arrow is pointing to the process node, the process node is reading from the data store. If the dashed line has bi-directional arrows the data store is being both read from and written to.

Based upon the analysis conducted with PProto, MIVS terrain masking has inherent data parallelism like the IMOM terrain masking. For example, referring to the subgraph of the Process_Matrix_Data node in appendix B, the data can be partitioned into 15 different pieces that can be processed in parallel. Since the MIVS terrain masking algorithm already out performs the terrain masking in IMOM, the MIVS terrain masking algorithm will be parallelized. This will involve parallelizing roughly 2500 lines of C code. This portion of code is where seventy percent of the processing time resides in calculating terrain masking.

CHAPTER 2: Timing the Sequential Version of the Terrain Masking Algorithm

2.1 Introduction

Timing the sequential version of the terrain masking algorithm is necessary in order to compare it against the parallel version of the algorithm for determining the delta in execution time. Also, various terrain masking scenario test cases are needed to determine under what conditions the execution time increases or decreases when calculating terrain masking. The parallel implementation will want to decrease the execution for cases which increase the execution time.

The first thing required determining where to insert time probes into the C source code. Based on the analysis performed using the PProto tool, the main terrain masking module is called ATint. ATint calls all other necessary modules for calculating terrain masking. The next step was to determine where in the ATint module the time probes should be inserted. The probes needed to be inserted so that they were as non intrusive as possible and so that the timing would reflect the portion of code that would be modified to include parallelism. When timing the code, portions that would remain sequential needed to be eliminated. Again, based on the PProto analysis, the start time would be sampled after the terrain masking variable initialization, since this initialization would not be parallelized. The end time is sampled after the completion of the terrain masking calculation before the image is written to the screen.

Once it was determined where to put the start and end time probes, how to store the time information was required. This raised the question whether the file storing the timing measurements should remain open the entire time or if it should be opened and closed just to sample the times. Both methods were tested on a couple test cases yielding the same timing estimates. Since it did not matter, it was decided to just open and close the file to sample the time. This method allows for a lower probability of file corruption which could result with an open file.

2.2 Terrain Masking Test Cases

After determining where to insert and store the timing information, it was necessary to define the test case scenarios. It was determined that there existed four possible scenarios that could increase or decrease the amount of time it takes to perform the terrain masking calculation. The first scenario would vary the range of the observer. This set of test cases involved keeping all other parameters constant and varying the range of the observer between 1000 meters and 20000 meters. The parameters that were kept the same included: elevation, interpolation, shadow high, shadow low, field of view angle, observer position, above ground level height, focus point, shade, type, fade percent, and color. Thirteen test cases were run with the result being that range increased the time for calculating terrain masking proportionally (refer to figure 7). For a complete list of all test cases refer to appendix C.

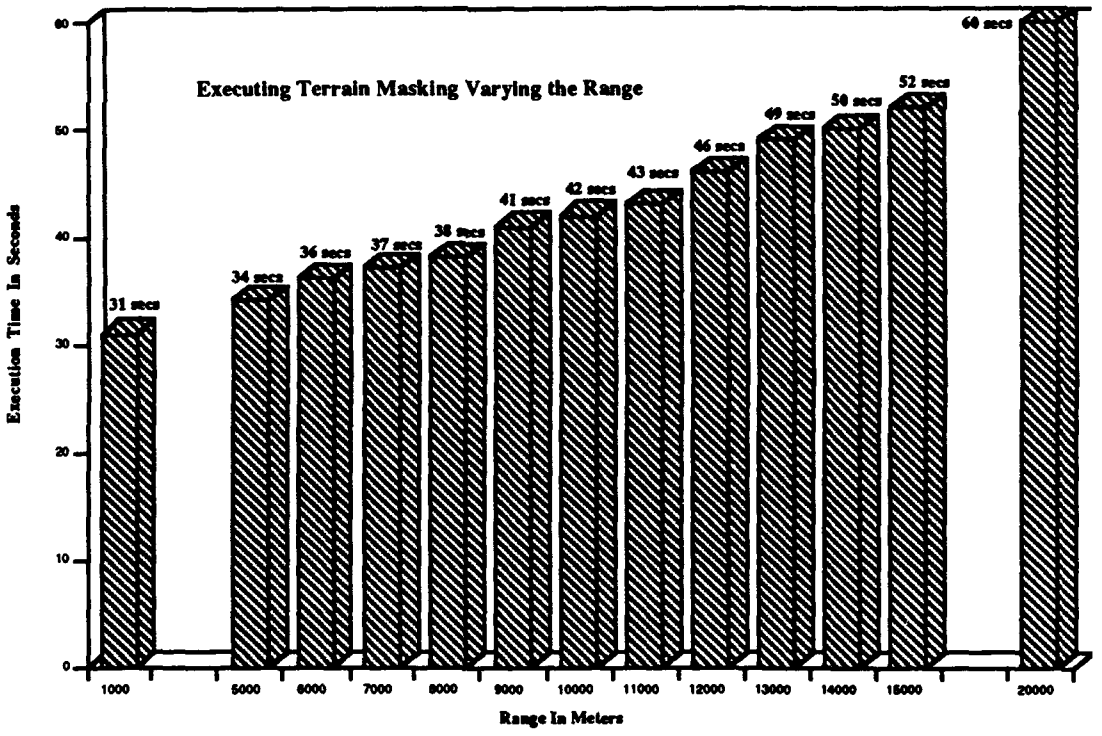


Figure 7. Executing Terrain Masking Varying the Range

Referring to figure 7, it can be observed that increasing the range from 1000 meters to 20000 meters almost doubles the amount of time for calculating the terrain masking. It can be concluded from this that range has a direct affect on the amount of time to calculate terrain masking and that the parallel implementation of the terrain masking algorithm will want to reduce the calculation time as range increases.

The second scenario varied the above ground level height of the observer. As in the first scenario all parameters were kept constant, but the above ground level height. The above ground level height of the observer was varied between 20 meters and 40 meters. For this scenario, 13 test cases were run. The result was that modifying the above ground level height of the observer had no effect on the terrain masking calculation time (refer to figure 8).

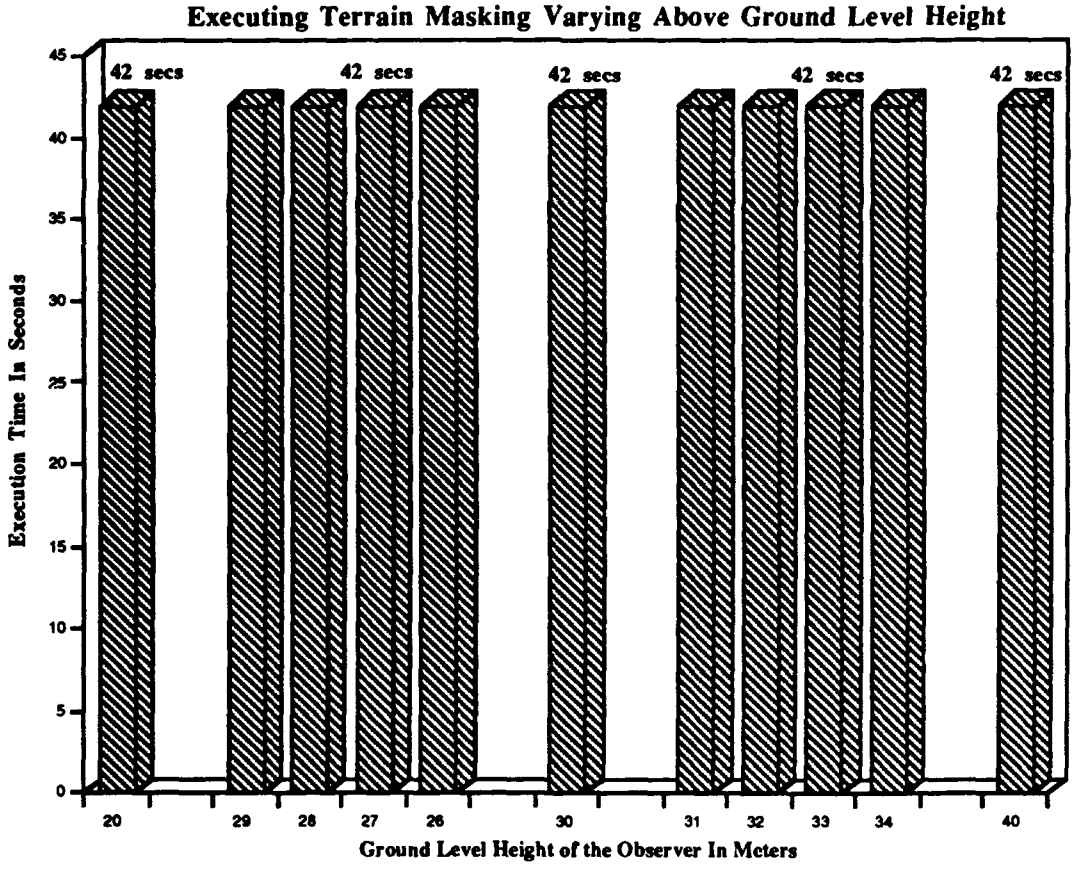


Figure 8. Executing Terrain Masking Varying Above Ground Level Height

It can be concluded that varying the above ground level height of the observer has no effect on the terrain masking calculation and that this will not be considered in the parallel version of the terrain masking code. The only thing that will be verified is that the parallel implementation for this set of test cases produces consistent execution times as the sequential version.

The third scenario varied the field of view angle of the emitting device. For example, with a radar system its field of view could range anywhere from 180 degrees to 360 degrees. The field of view angle directly affects visibility of the emitting device. For this set of test cases all parameters remained constant except the field of view angle and 19 test cases were run. The field of view was varied from 360 degrees to 30 degrees. The result was that once the field of view was narrowed to less then 360 degrees, the time for calculating terrain masking increased, and then remained constant at that execution time for all the test cases (refer to figure 9).

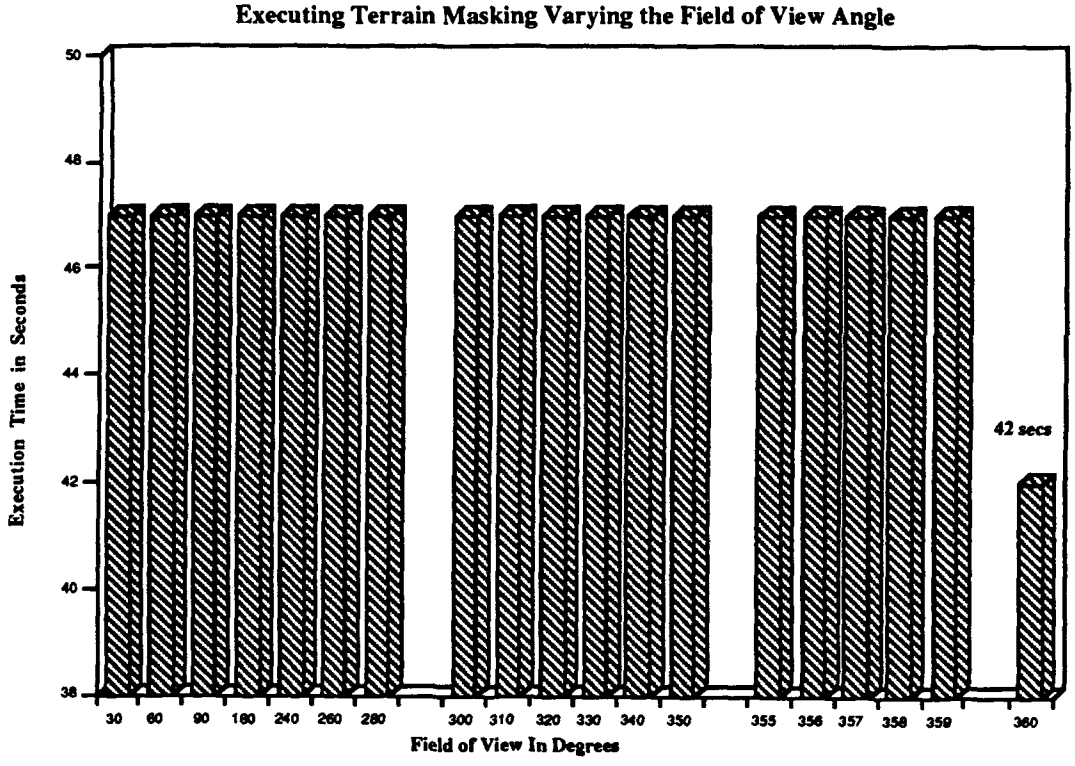


Figure 9. Executing Terrain Masking Varying the Field of View Angle

It was concluded from this set of test cases that the field view angle has a direct effect on the calculation time of the terrain masking algorithm. The parallel version of the terrain masking algorithm at a minimum will want to reduce the calculation time to at least the time to calculate for a field of view of 360 degrees.

The final scenario varied the position of the observer in both latitude and longitude. This was necessary to determine if moving the observer closer to or farther from the emitting device effected the calculation time of terrain masking. For this set of test cases, all parameters again remained constant except for the location of the observer. Twelve test cases were run with the result being the distance of the observer to the emitting directly effected the execution time of terrain masking. As the observer is moved closer to the emitting device, the execution time decreases and as the observer is moved farther from the emitting device, the execution time increased (refer to figure 10).

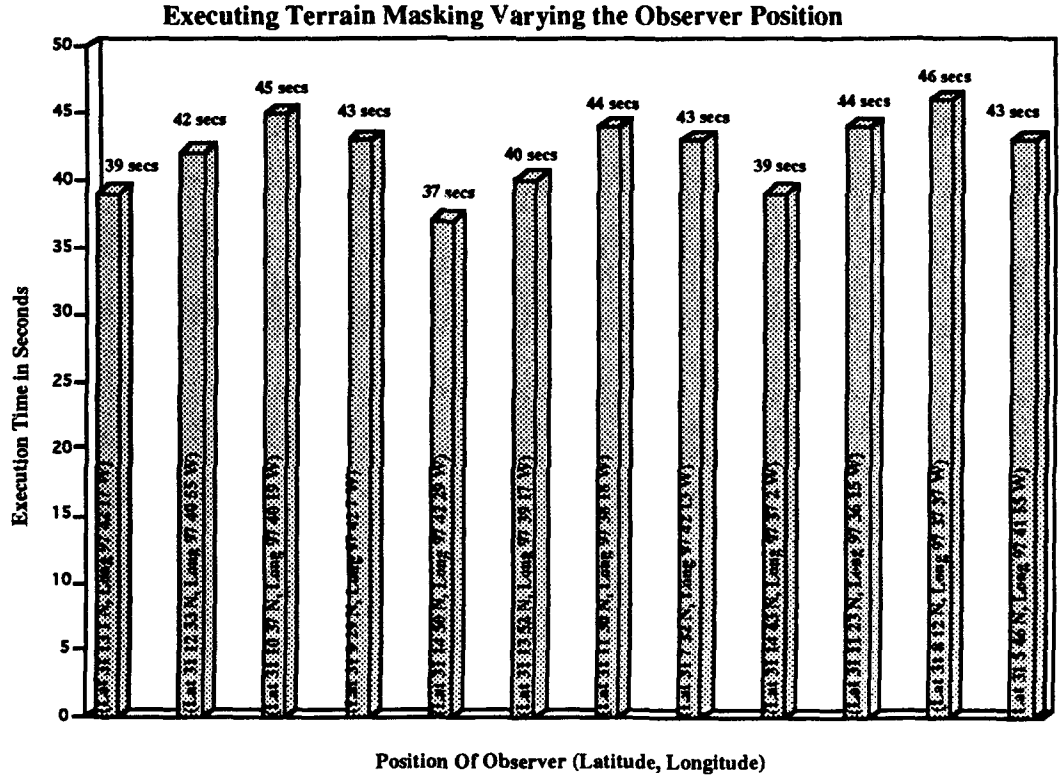


Figure 10. Executing Terrain Masking Varying the Observer Position

It can be concluded that the parallel version of the terrain masking algorithm at a minimum will want to decrease the execution time of the cases when the observer position is far away from the emitting device.

In general, the parallel version of the terrain masking algorithm will want to reduce the execution time for the baseline test case (refer to appendix C case 1) of the sequential version. This test case was used as the base test case for all four of the test scenarios. The parallel version of the terrain masking algorithm will need to execute in less than 42 seconds.

CHAPTER 3 Parallelizing the Terrain Masking Algorithm

3.1 Introduction

The process of parallelizing the IMOM terrain masking algorithm involved first determining which existing subroutines and functions needed to be modified. Once these subroutines and functions were identified, the next step was to determine the most efficient manner to partition the data for the transputer architecture. Once the data had been partitioned, the next step was to determine how the transputers and the host architecture would communicate. The final step was to determine what new software needed to be written to run on the transputers.

3.2 Determining What Existing Code Needed to Be Modified

When determining what code needed to be modified, the decision was made to try to modify as little existing code as possible. This decision was made so that terrain masking would still be able to run in the MIVS system environment. Based on analysis of the existing code, it was determined that the C_SEEFAR_SHADOW subroutine is where the majority of the terrain masking calculation is being performed. It is this algorithm that comsumes about seventy percent of the total terrain masking calculation time.

The C_SEEFAR_SHADOW subroutine given the position of the observer (latitude, longitude), the height of the observer, and the range of the given sensor, calculates the visibility of every terrain elevation point in the identified area. The C_SEEFAR_SHADOW subroutine receives the following inputs: the terrain region (cel_e), the observer latitude (obs_lat), the observer longitude (obs_lon), how far the observer can see (range), the observer height about above sea-level (obs_ht), the orientation of the terrain data (orientation), the number of rows of elevation data (nr_rows), the number of columns of elevation data (nr_cols), the longitude of the first

element in the elevation data (SW_Lon), the latitude of the first element in the elevation data (SW_lat), the number of radians between the columns of elevation data (x_inc), and the number of radians between the rows of the elevation data (y_inc). The C_SEEFAR_SHADOW subroutine produces the following outputs: the array of shadow data defining which points are visible and which points are not (bitmap), the array of shadow heights (shad_ht), and the current horizon elevations (horizon). The calculation to determine whether or not a given point is visible by comparing it to points on the horizon is accomplished by a function called C_VISIBILITY. This function is repeatedly called by the C_SEEFAR_SHADOW subroutine for column and row pairs of terrain data.

The terrain elevation data in the C_SEEFAR_SHADOW subroutine is processed by first going through all columns and rows to the right of the observer. For example, the processing begins by starting to the right of the observer and one row above the observer and processing all the rows above the observer. Then, still staying to the right of the observer and starting at one row below the observer, all the rows below the observer are processed. There are two more similar pairs of calculations of this type performed for the columns to the right of the observer. Once the visibility has been determined for columns to the right of the observer, the visibility is calculated for the columns to the left of the observer. The same type of calculations are performed as were done for the columns to the right. The total number of pairs of calculation that is conducted on the terrain elevation data is six. For each of these calculations, the elevation data is processed by the C_VISIBILITY function.

It was determined that no modifications needed to be made to the C_VISIBILITY function since this routine would work no matter how large or small the amount of terrain data passed to it. The only change required to the C_VISIBILITY function is that it would no longer execute on the Sun processor, rather it would execute on the transputer processors.

The C_SEEFAR_SHADOW subroutine would need to be modified in order to communicate (send and receive pertinent data) with the transputer platform and remove all references of calls to the C_VISIBILITY function since this would now be executed on the transputer platform.

3.3 Partitioning the Terrain Elevation Data

As stated in section 3.2, it was determined that there were six column and row pair calculations. In order to determine if the data in these column and row calculations was dependent or independent, a sample data set was traced through the program. The trace showed that the calculation for visibility of the rows and columns is independent. This meant that the calculation for visibility of data points in a row or column is not dependent on the calculation of data points in adjacent rows or columns. For example, when columns to the right and rows above the observer were being processed, columns to the right and rows below the observer could be processed at the same time. In the same manner, columns to the left and rows above the observer could be processed at the same time as columns to the left and rows below the observer.

In order to determine how to partition the terrain elevation data, a distribution of the sample trace data was plotted. Charts showing the data distribution for rows above and below the observer are given in figure 11 and 12. From these charts it can be observed that the data is linearly distributed. Because the distribution is linear, the data can be divided equally. The division of data is dependent upon the number of transputer processors available.

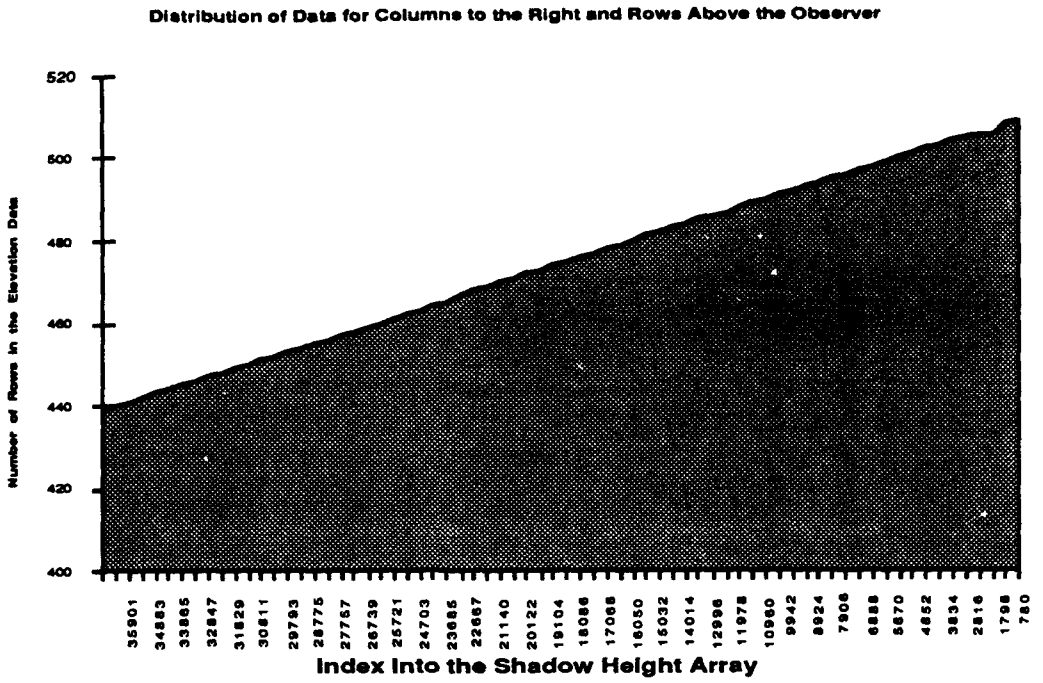


Figure 11. Data Distribution for Rows Above the Observer

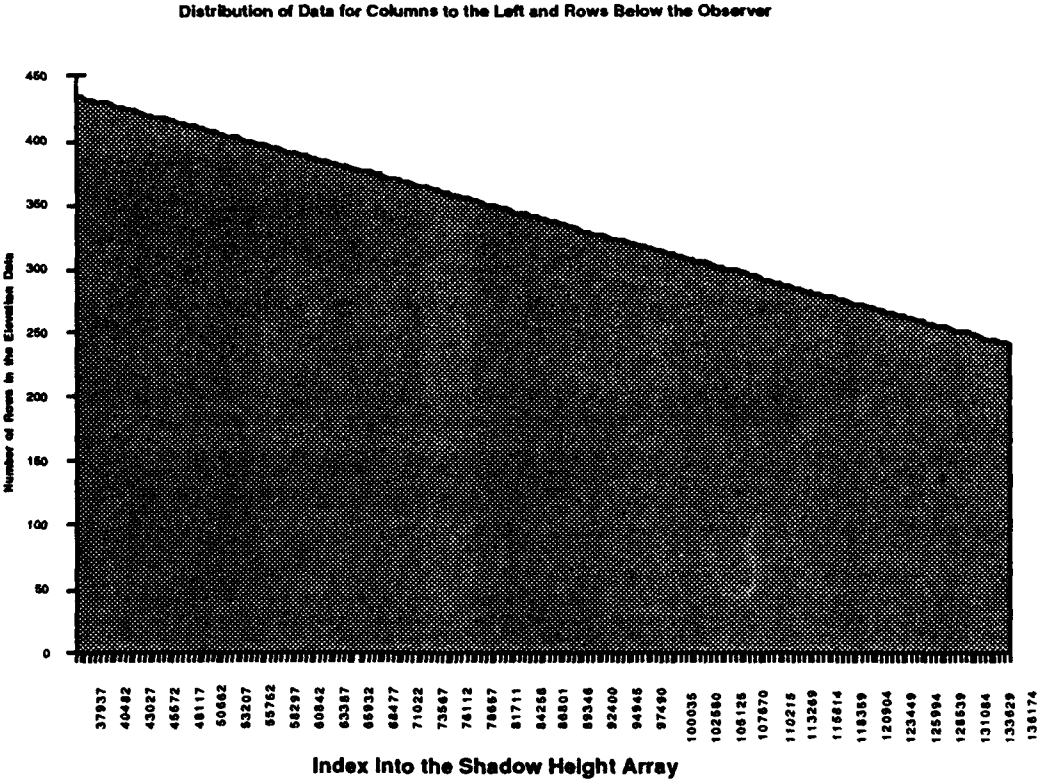


Figure 12. Data Distribution for Rows Below the Observer

3.4 Determining Communication between the Sun Host and Transputers

The transputer architecture that is being utilized is a Meiko MK202 16-processor transputer board with 2 mega-bytes of memory per processor. The Meiko board is housed in a Sun SparcStation 4/330 with 40 mega-bytes of memory and 3 giga-bytes of combined internal and external disk storage. Since there are 16 transputer processors available, it was decided that each pair of column/row calculations would be given 8 processors. Of the eight processors, one processor is assigned as the master process that interfaces with the Sun 4/330 processor. The transputer processors are configured as two binary trees with each master process node communicating with the Sun host processor (refer to figure 13).

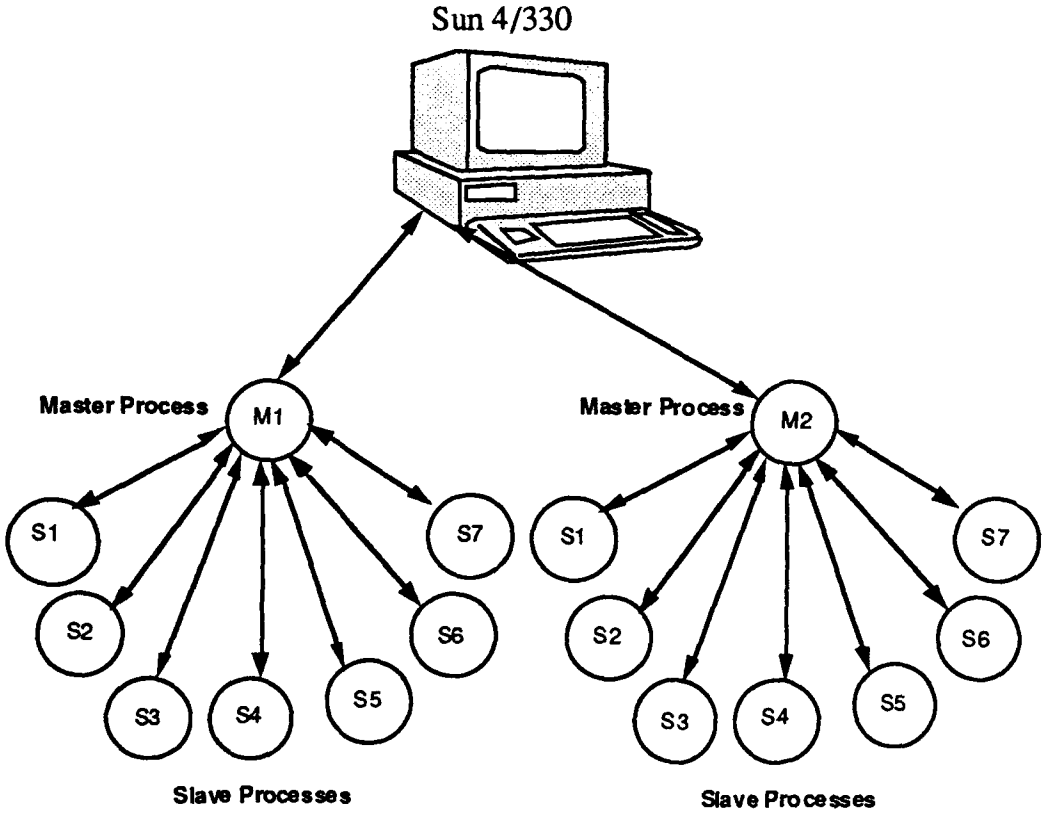


Figure 13. View of Sun and Transputer Configuration

The master process receives the terrain elevation data as well as other data required to calculate terrain visibility from the Sun processor. The master process divides the terrain

elevation data equally among the seven transputer processors. The master process also transmits any other relevant data required to perform the visibility calculation. The seven processors are referred to as slave processes, they only communicate with the master process. Also, the slave processes do not need to communicate amongst each other. Once the slave processes have finished their calculations on the terrain elevation data, they transmit the results back to the master process. The master process then sends the results to the Sun processor. This configuration was selected to reduce the overhead of communication and synchronization between the Sun host processor and the transputers. With a in-Sun Meiko transputer board there is only one port available for communicating between the host process and the transputer computing surface. If there was not a master process, all the transputers would have to communicate directly with the host processor. This would create a bottleneck for receiving and transmitting data thereby reducing the amount of useful processing time.

In order for the Sun host processor and master transputer process to share data, the External Data Representation Protocol (XDR) must be used. The XDR protocol is a set of routines that provides the ability to encode non-portable data types within a byte array so that it can be transmitted between processors and decoded [Meiko91]. This is required because the Meiko transputer data protocol is different from the Sun 4/330 data protocol.

For the terrain masking algorithm, the Sun host processor must transmit 21 data elements to the master process in order for the visibility calculation to be performed. To send the data to the master process, the data elements were encapsulated in a C structure so that only one transmit command sequence was required. In order to use the XDR protocol, a three step process is involved. The first step is to create the required amount of memory needed to transmit the data. The next step is to transmit the data. In order to transmit the data, a csn_tx command is used. The problem with the transmit command is that only one data element at a time can be transferred. Finally, the last step is to destroy the memory space allocated. If the data was not encapsulated 21 separate memory

allocations, transmits commands, and memory deallocations would have been required. This would have resulted in a tremendous amount of overhead in communication.

Finally, the master process needs to send the results of the visibility calculations performed by the slave processes back to the Sun host. The shadow height array and the bitmap array must be sent back to the C_SEEFAR_SHADOW routine running on the Sun processor.

3.5 Software Written to Run on the Transputer Architecture

Four software modules were written to run on the transputer architecture. Two subroutines called MASTER_ROWS_ABOVE and MASTER_ROWS_BELOW were developed to run on the two transputer nodes designated as the master nodes. MASTER_ROWS_ABOVE and MASTER_ROWS_BELOW are responsible for receiving the terrain elevation data from the C_SEEFAR_SHADOW routine running on the Sun processor. These two subroutines are basically the same except that MASTER_ROWS_ABOVE is only concerned with the rows above the observer and MASTER_ROWS_BELOW is only concerned with the rows below the observer. Each subroutine partitions the terrain elevation data evenly among the seven slave processors connected to them. Once the data has been transmitted to the slave processes, the master processes wait for the results. Upon receipt of all the results, the subroutines then transmit the shadow height array and the bitmap array data back to the C_SEEFAR_SHADOW subroutine running on the Sun processor.

The other two subroutines are called SLAVE_ROWS_ABOVE and SLAVE_ROWS_BELOW. These two subroutines run on the slave process nodes. Seven transputer nodes receive a copy of the SLAVE_ROWS_ABOVE subroutine and seven receive a copy of the SLAVE_ROWS_BELOW. These two subroutines are responsible for calculating whether a particular terrain elevation point has visibility at a given location. SLAVE_ROWS_ABOVE and SLAVE_ROWS_BELOW also contain

code that originally ran on the Sun processor. This includes the C_VISIBILITY function which determines the visibility of a terrain elevation point. The subroutines receive data from the master processes and return the results of the visibility calculation.

CHAPTER 4 Results

4.1 Problems Encountered

Memory limitations of the transputer processors coupled with the size of the terrain elevation data made it unfeasible to obtain a complete execution run of the parallel version of the terrain masking algorithm. The Meiko MK202 board provides a maximum of 2 mega-bytes of memory per processor which in the beginning of this effort seemed sufficient to support the terrain masking algorithm. Unfortunately, over a megabyte of the memory per processor is consumed by the Meiko Operating System and the heap stack. This left very little memory for storing the program and data required to calculate the terrain masking visibility. The data required to process the terrain masking algorithm consists of two 38,000 element integer arrays, one 38,000 element unsigned character array, and one 100 element floating point array and 17 miscellaneous data elements. This data needed to be transmitted from the Sun host processor to the master processes on the transputer architecture which then partitioned the data for the slave processes.

When terrain masking was executed a stack overflow was obtained on the transputer processors because there was not sufficient memory available to load the program and the required data onto the transputer processors. In order to determine that this was the problem, the -v (verbose) and -m (memory) options were used with Meiko MRUN command when executing the parallel version of terrain masking. The -m option shows how the memory on each transputer process is allocated and what is allocated to the memory (refer to figure 14).

```
carrie% mrun -v -m test.par
grab: proc1 maps to proc 0 type 8 mem 174K m255b255p255 (MK255)
grab: about to grab domain from svcs
MEMORY MAP : proc1
------------------
  Primary transputer memory : 80000000 - 801fffff
  ----------------------------------------------
    proc1`_localRteClient`_sysHeapVec     80023940-801fffff ( 1906K)
    proc1`_g_dataLinkInfo                 800238c0-8002393f
    proc1`_unixRteTron                    800238bc-800238bf
    proc1`_localRteTron                   800238b8-800238bb
    proc1`_csn`_bssVec                    80023494-800238b7 (   2K)
    proc1`_csn                            80022820-80023493 (   4K)
    proc1`_csnNetwork                     80022748-8002281f
    proc1`TEXT_unix_cxx.rt8               8001d178-80022747 (  22K)
    proc1`TEXT_locl_xxx.rt8               8001b290-8001d177 (   8K)
    proc1`TEXT_csnx_xxx.rt8               800170a0-8001b28f (  17K)
    proc1`master_rows_above`_bssVec       80014c64-8001709f (  10K)
    proc1`master_rows_above`_argsVec      80014c50-80014c63
    proc1`master_rows_above               8000fe20-80014c4f (  20K)
    proc1`TEXT_master_rows_above          8000a1b8-8000fe1f (  24K)
    proc1`_csnNetTable                    8000a1b4-8000a1b7
    proc1`_csnRouteTable                  8000a1ac-8000a1b3
    proc1`_csnHopTable                    8000a1a8-8000a1ab
    proc1`_csnTransportTable              800061a8-8000a1a7 (  16K)
    proc1`_unixRteClient`_bssVec          800038f8-800061a7 (  11K)
    proc1`_unixRteClient                  80002560-800038f7 (   5K)
    proc1`_localRteClient`_bssVec         80002010-8000255f (   2K)
    proc1`_localRteClient                 80001000-8000200f (   5K)
    proc1`_internalRam                    80000070-80000fff (   4K)
    ANONYMOUS                             80000000-8000006f      FIXED

boot: Booting  transputer proc1 (link 0) from cs_host (link 0)
load: Loading code segment ./master_rows_above
load: Loading code segment /usr/meiko/cstools/rte/unix_cxx.rt8
load: Loading code segment /usr/meiko/cstools/rte/locl_xxx.rt8
load: Loading code segment /usr/meiko/cstools/rte/csnx_xxx.rt8
init: send initializations for proc1
run : Run thread _localRteClient Wptr 80002008 Iptr 8001cf55 pri 0
run : Run thread _unixRteClient Wptr 800038f0 Iptr 80021a41 pri 0
run : Run thread master_rows_above Wptr 80014c48 Iptr 8000a1bc pri 1
run : Run thread _csn Wptr 8002348c Iptr 8001b079 pri 1
load: Domain up and running (1 children)
[proc1`master_rows_above] *** Stack overflow in main() in file master_rows_above.c
```

Figure 14. Sample of Meiko MRUN Command Using -M Option

The first step to solve this problem was to try and reduce the size of the program that needed to run on the transputer processors. The SLAVE_ROWS_ABOVE and SLAVE_ROWS_BELOW program could not be reduced at all, the size of the program remained at 34K. The MASTER_ROWS_ABOVE and MASTER_ROWS_BELOW were reduced slightly removing some unnecessary include files, but the size of the files remained at 24K.

The next step was to try and see if the size of sysHeapVec (refer to figure 14) could be reduced to allow more room for the program and data. There is an option available with the Meiko C compiler which allows the user to tell the linker to set the stack size to a certain number of bytes [MeikoOS]. The option is -tF(storage-bytes). In order to get the program to fit on the processing nodes the -tF option had to be set to 20480K. This solved the problem of getting the programs and data to fit on the transputers, but a SVCS services error was given that the processor wires did not exist (refer to figure 15).

```
carrie% mcc -o master_rows_above -tF20480k master_rows_above.c -lcs -lcsn
carrie% more test.par
par
  processor 0 (proc_type host) /appl/mivs/bin/Uc
  processor 1 master_rows_above
endpar
carrie% mrun -v -m test.par
grab: proc1 maps to proc 0 type 8 mem 20634K m255b255p255 (MK255)
grab: about to grab domain from svcs
svcs: requested processors or wires don't exist
MEMORY MAP : proc1
------------------
  Primary transputer memory : 80000000 - 81426b1f
  ------------------------------------------------
    proc1`_localRteClient`_sysHeapVec     8141eb20-81426b1f (  32K)
    proc1`_g_dataLinkInfo                 8141eaa0-8141eb1f
    proc1`_unixRteTron                    8141ea9c-8141ea9f
    proc1`_localRteTron                   8141ea98-8141ea9b
    proc1`_csn`_bssVec                    8141e674-8141ea97 (   2K)
    proc1`_csn                            8141da00-8141e673 (   4K)
    proc1`_csnNetwork                     8141d928-8141d9ff
    proc1`TEXT_unix_cxx.rt8               81418358-8141d927 (  22K)
    proc1`TEXT_locl_xxx.rt8               81416470-81418357 (   8K)
    proc1`TEXT_csnx_xxx.rt8               81412280-8141646f (  17K)
    proc1`master_rows_above`_bssVec       8140fe44-8141227f (  10K)
    proc1`master_rows_above`_argsVec      8140fe30-8140fe43
    proc1`master_rows_above               8000fe20-8140fe2f (20481K)
    proc1`TEXT_master_rows_above          8000a1b8-8000fe1f (  24K)
    proc1`_csnNetTable                    8000a1b4-8000a1b7
    proc1`_csnRouteTable                  8000a1ac-8000a1b3
    proc1`_csnHopTable                    8000a1a8-8000a1ab
    proc1`_csnTransportTable              800061a8-8000a1a7 (  16K)
    proc1`_unixRteClient`_bssVec          800038f8-800061a7 (  11K)
    proc1`_unixRteClient                  80002560-800038f7 (   5K)
    proc1`_localRteClient`_bssVec         80002010-8000255f (   2K)
    proc1`_localRteClient                 80001000-8000200f (   5K)
    proc1`_internalRam                    80000070-80000fff (   4K)
    ANONYMOUS                             80000000-8000006f      FIXED

Terminated with errors
```

Figure 15. Sample of Meiko MRUN with sysHeapVec Size Reduced

The SVCS is the scheduler daemon for the Meiko In-Sun Computing Surface[MeikoOS]. SVCS stands for Sun Virtual Computing Surface. The daemon boots the system processor on each board and provides a TCP service to handle user process requests. One of the services provided is the wiring of the processors. By reducing the size of sysHeapVec, the capability for SVCS was not available.

The next step was to reduce the size of the data. Reducing the size of the data would also help reduce the size of the programs because the size of the arrays that the programs had to handle would be smaller. The biggest problem existed with the arrays being transmitted from the Sun processor to the master processes running on the transputers. The following four arrays were targeted: shad_ht, cel_e, bitmap, and horizon. The shad_ht, cel_e, and bitmap arrays all contained 38,000 elements, the horizon array contained 100 elements. Reducing the size of the arrays allowed the programs to load, but in order to achieve this, each array could only have 10 elements. This would mean that in order to send all of the necessary data to the master processes 3800 separate XDR transmits would be required. The overhead produced by the communication would eliminate any possible chance of speedup. Attempts were made without modifying the existing design to get a complete terrain masking execution but success was not obtained.

4.2 Possible Results If Parallel Execution was Successful

If more memory per processor was available on the transputer nodes, the ability to execute the parallel version of the terrain masking algorithm would be possible. Assuming that the parallel implementation executed successfully, lets try to determine the possible speed-up attainable. The parallel approach taken reduces the number of row/column calculations from 12 sequential calculations to 2 parallel row/columns calculations that are executed six times. This could roughly provide a 2 to 1 speed-up over the current sequential implementation. Each of the calculation pairs partitions the data over 7 transputers nodes.

A number of factors must be considered. These factors are the cost for initializing the transputer nodes for execution, the cost for sending the data from the Sun host processor to the master process on the Meiko Computing Surface, the cost of sending the partitioned data to the slave processes, the cost of each slave process transmitting their

results back to the master process, and the cost of the master process transmitting the results back to the Sun host process. The cost to initialize the transputer network is actually neglible because the transputer network is initialized when the MIVS system is brought up and therefore has no bearing on the timing of the terrain masking calculation. The main penalty cost that must be considered is the transfer of data.

The amount of data being transmitted from the Sun host processor to each master process pair is roughly 195,400 bytes of data. The master processes must then communicate this data to the slave processes. The slave processors process the data and communicate their results back to the master processes. The master processes then transfer this data to the Sun processor. The total amount of result data produced by the slave processes is approximately 38,000 bytes. The data transfer rate between the Sun host processor and the Meiko transputer node is approximately 2.2 mega-bytes per second, and the data transfer rate between transputer nodes is approximately 1.4 mega-bytes per second. Based on this, roughly 1.74 secs would be required for data communication in the parallel version of the terrain masking algorithm.

An example will be used to show the possible speed-up attainable. Referring to test case 18 in Appendix C, it took 42 seconds to calculate terrain masking for the sequential version. Assuming half of the 42 seconds, 21 seconds is spent processing the data, 10.5 seconds would be required for the parallel version (this is because a 2 to 1 speed-up is possible). Add to this the data communication overhead of 1.74 and the total time for test case 18 is now 12.24 secs. Therefore, the total time to execute the parallel version of the terrain masking algorithm is approximately 12.24 seconds. To obtain the amount of speed-up attainable, 42 secs is divided 12.24 yielding a 3 to 1 speed-up. For test case 18, a speed-up of 3 to 1 is possible. For other test cases, the speed-up may be more or less depending upon the size of the data and the execution speed. If the size of the data is larger than the example above, and the execution time is less than 42 secs, the speed-up obtained will be less.

CHAPTER 5 CONCLUSION

The process of parallelizing existing sequential software is currently not an easy task. The software developer must first try to determine what is the most suitable architecture for executing their particular application. In making this selection, the software developer must be concerned with many issues such as the memory size available on the processors, the number of processors available, the speed of the memory, and the protocols available for communicating among the processors. This is not always easy to determine as can be seen from the research presented here. What may seem sufficient by the specifications provided by the hardware vendor may not be what they seem. One possible hope for making the process of selecting the appropriate parallel architecture easier is to provide software assessment tools. One such tool, called the Parallel Assessment Window System (PAWS), allows a user to run existing code on a number of characterized parallel architectures in order to determine the most appropriate architecture for a given application [PAWS91]

Once the architecture is selected the real work begins. The software developer must determine the type of parallelism that exists in the application and the best method for implementing it. Such issues as determining how to partition data, determining the number of identical or non-identical tasks, synchronizing messages between tasks, are among the problems encountered. Software tools for supporting design, code and testing of parallel software are required. Even after all the decisions are made and the algorithm is parallelized, there is no guarantee that what was projected will be obtained.

For the military community, these problems make the prospect of developing software targeted for execution on parallel architectures risky and expensive. This is because military systems are large and complex and consist of many different types of processing [Alexandridis86]. The future for parallel architectures lies in making software support available to aid the software developer.

BIBLIOGRAPHY

[Acosta91] R. D. Acosta, "Simulation Modeling of Parallel Programs in PPROTO.", Proceedings of the 1991 Simulation Conference, July 22-24, 1991.

[AFEWC89] Locus Inc., "User/Operations Manual For The Improved Many-On-Many Model Version 4.0.", Reference Manual April 1989.

[Alexandridis86] Alexandridis, N.A. "Adaptable Software and Hardware: Problems and Solutions", IEEE Computer, February 1986, pp 29-39.

[Benincasa92] M. Benincasa, and C. Burns, "Phase One Results of Parallelizing The Terrain Masking Algorithm.", In-House Technical Report RL-TR-92-72, Rome Laboratory, Griffiss AFB, NY, April 1992.

[Grumman92] Grumman Data Systems, "Software Programmer's Manual for the Multi-Source Integrated Viewing System (MIVS).", Rome Laboratory Contract Delivery, Contract No. F30602-89-C-0077, Rome Laboratory, Griffiss AFB, NY, March 1992.

[Hillis 86] W. Daniel Hillis, and G. L. Steele, Jr., " Data Parallel Algorithm", Communications of the ACM 29, December 1986, pp 1170-1183.

[ISSI91] International Software Systems, Inc., "PProto User's Manual", Rome Laboratory Contract Delivery, Contract No. F30602-89-C-0129, Rome Laboratory, Griffiss AFB, NY, June 1991.

[LaBatt90] E.C. LaBatt, "Analysis of Improved Many-On-Many", In-House Technical Report RADC-TR-90-115, Rome Air Development Center, Griffiss AFB, NY, April 1990.

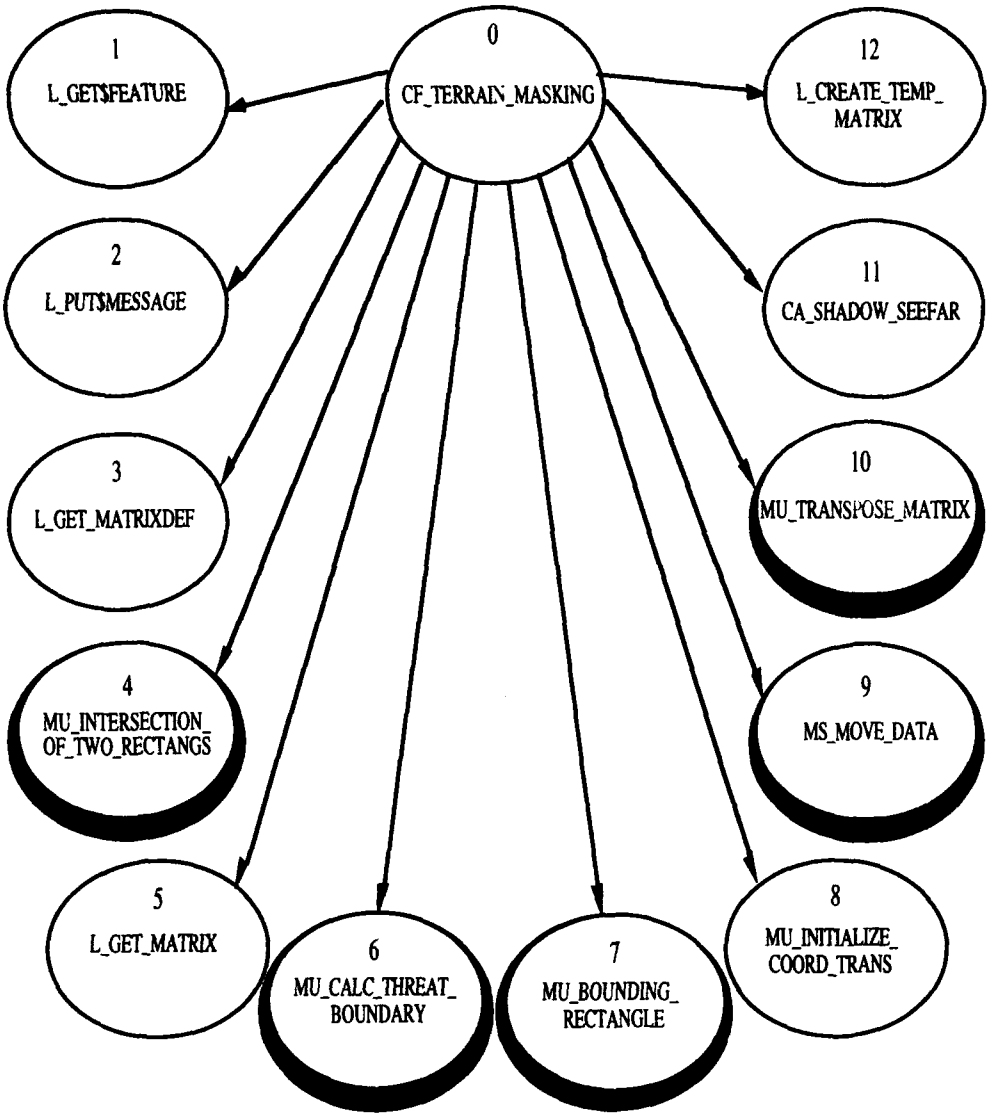
[Meiko91] Meiko Limited, "C for CS-Tools", Reference Manual, 1991.

[MeikoOS] Meiko Limited, "CSTools for SunOS", Reference Manual, 1991.

[PAWS91] D. Pease, A. Ghafoor, I. Ahmad, D. Andrews, K. Foudil-Bey, T. Karpinski, M. Mikki, and M. Zerrouki, "PAWS: A Performance Evaluation Tool for Parallel Computing Systems", IEEE Computer, January 1991, pp 18-29.

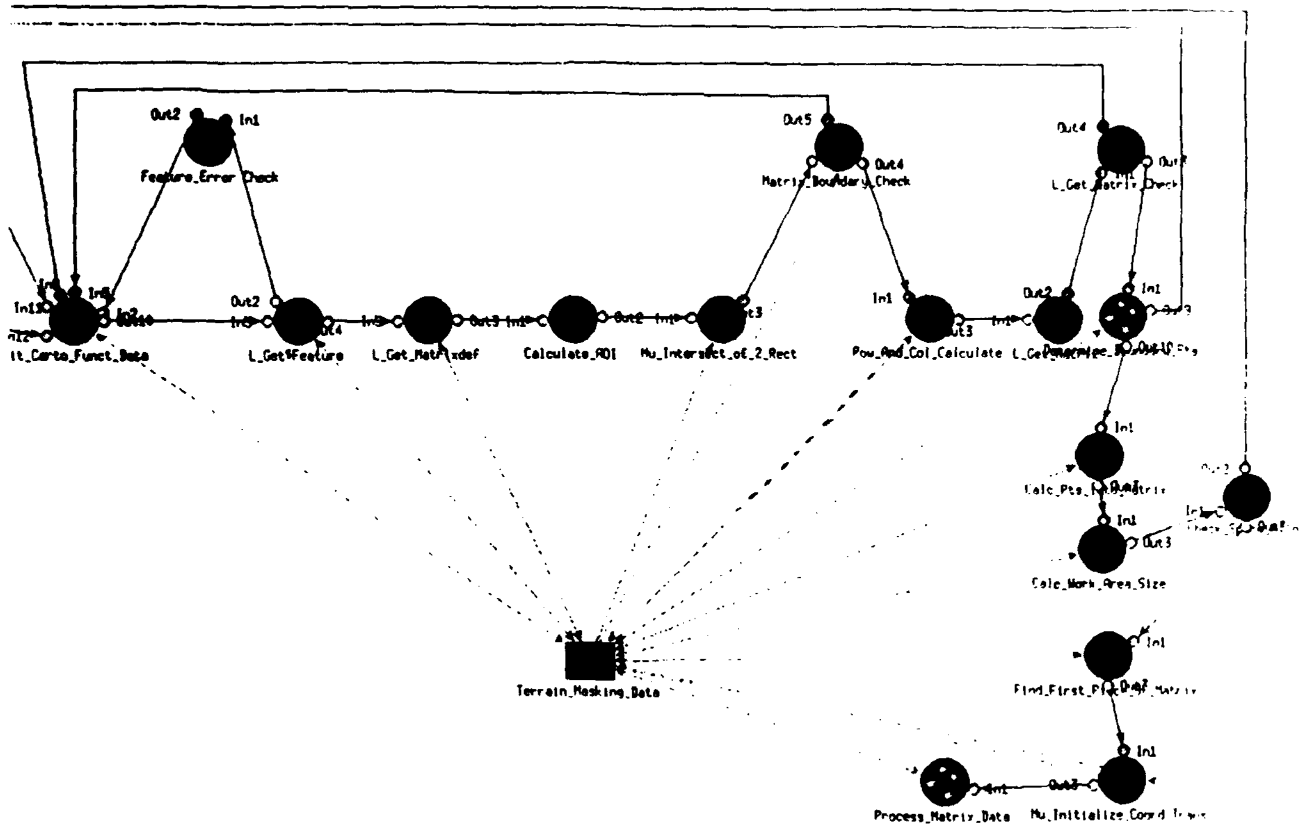
[Souza92] J. Souza, "MIVS Multi-Source Integrated Viewing System.", Final Technical Report, Rome Laboratory, Griffiss AFB, August 1992.

Appendix A - IMOM Terrain Masking Call Trees

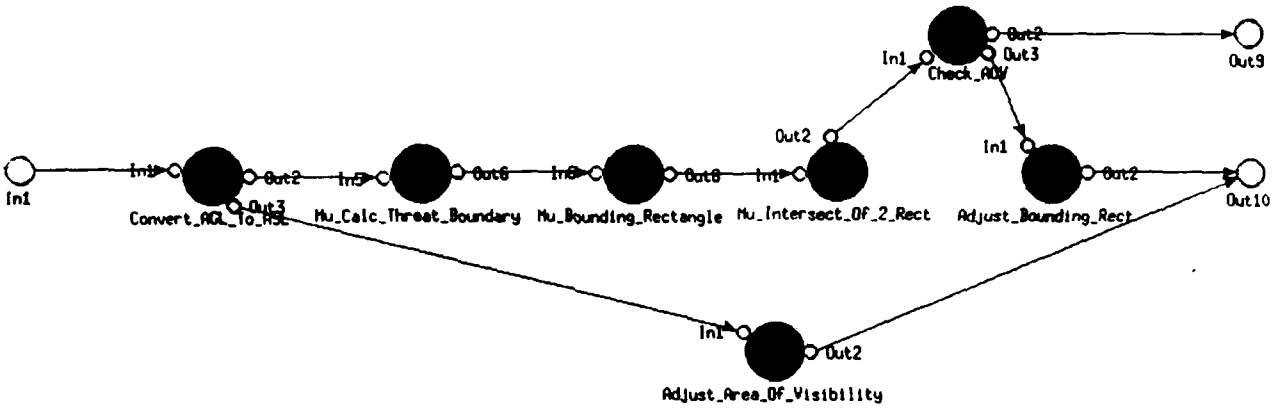


CF_TERRAIN_MASKING CALL TREE

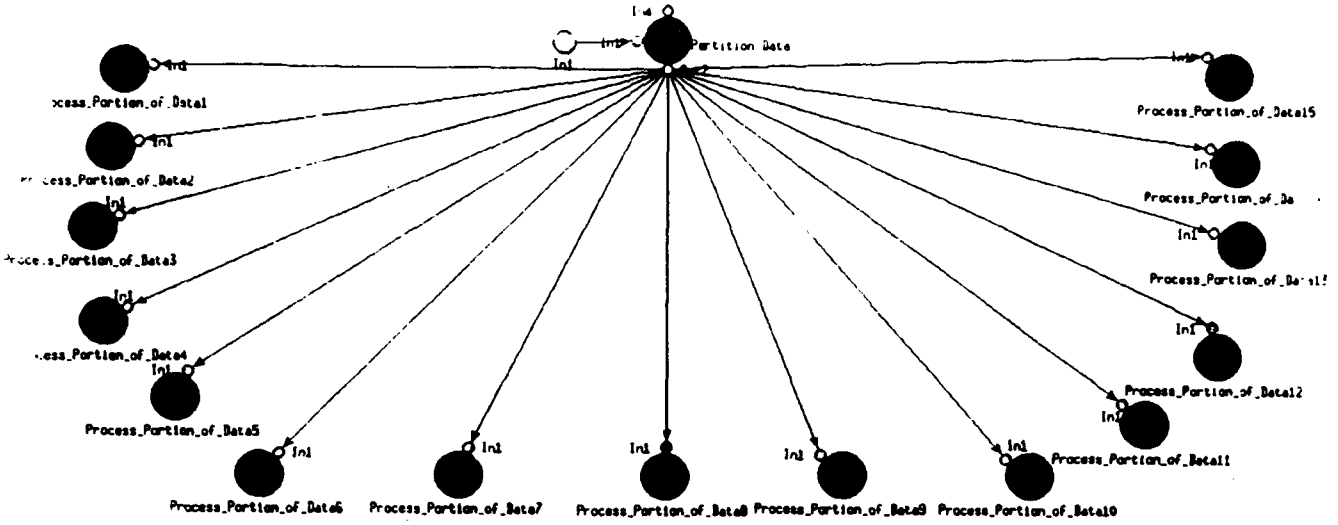
This is the top-level call tree for the IMOM terrain masking algorithm. For a detailed look at all the sub-call trees refer to [Benincasa92].



Top-Level View of the Terrain Masking Algorithm



Determine_Boundary_Pts Subgraph



Process_Matrix_Data Subgraph

Appendix C - Test Cases for Timing Sequential Version of Terrain Masking

Case 1: This is the base test case. The following information was entered into the terrain masking tinting function:

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	10000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738259288
end time measurement in secs	738259330
total execution time in secs	42

The following set of test cases varies the range.

Case 2: Everything is the same as in test case 1 except the range has been changed to 9000.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	9000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738174644
end time measurement in secs	738174685
total execution time in secs	41

Case 3: Everything is the same as in test case 1 except the range has been changed to 8000.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	8000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738174994
end time measurement in secs	738175032
total execution time in secs	38

Case 4: Everything is the same as in test case 1 except the range has been changed to 7000.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	7000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738175309
end time measurement in secs	738175346
total execution time in secs	37

Case 5: Everything is the same as in test case 1 except the range has been changed to 6000.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	6000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738175638
end time measurement in secs	738175674
total execution time in secs	36

Case 6: Everything is the same as in test case 1 except the range has been changed to 5000.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	5000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738175899
end time measurement in secs	738175933
total execution time in secs	34

Case 7: Everything is the same as in test case 1 except the range has been changed to 1000.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	1000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738176214
end time measurement in secs	738176245
total execution time in secs	31

Case 8: Everything is the same as in test case 1 except the range has been changed to 11000.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	11000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738176451
end time measurement in secs	738176494
total execution time in secs	43

Case 9: Everything is the same as in test case 1 except the range has been changed to 12000.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	12000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738176769
end time measurement in secs	738176815
total execution time in secs	46

Case 10: Everything is the same as in test case 1 except the range has been changed to 13000.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	13000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738177033
end time measurement in secs	738177082
total execution time in secs	49

Case 11: Everything is the same as in test case 1 except the range has been changed to 14000.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	14000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738177418
end time measurement in secs	738177468
total execution time in secs	50

Case 12: Everything is the same as in test case 1 except the range has been changed to 15000.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	15000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738177742
end time measurement in secs	738177794
total execution time in secs	52

Case 13: Everything is the same as in test case 1 except the range has been changed to 20000.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	20000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738178273
end time measurement in secs	738178333
total execution time in secs	60

The following set of test cases varies the agl.

Case 14: Everything is the same as in test case 1 except the agl has been changed to 29.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	10000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	29
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738249282
end time measurement in secs	738249324
total execution time in secs	42

Case 15: Everything is the same as in test case 1 except the agl has been changed to 28.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	10000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	28
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738249952
end time measurement in secs	738249994
total execution time in secs	42

Case 16: Everything is the same as in test case 1 except the agl has been changed to 27.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	10000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	27
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738250282
end time measurement in secs	738250324
total execution time in secs	42

Case 17: Everything is the same as in test case 1 except the agl has been changed to 26.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	10000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	26
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738258551
end time measurement in secs	738258593
total execution time in secs	42

Case 18: Everything is the same as in test case 1 except the agl has been changed to 20.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	10000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	20
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738258788
end time measurement in secs	738258830
total execution time in secs	42

Case 19: Everything is the same as in test case 1 except the agl has been changed to 31.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	10000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	31
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738259693
end time measurement in secs	738259735
total execution time in secs	42

Case 20: Everything is the same as in test case 1 except the agl has been changed to 32.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	10000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	32
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738259942
end time measurement in secs	738259984
total execution time in secs	42

Case 21: Everything is the same as in test case 1 except the agl has been changed to 33.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	10000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	33
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738260150
end time measurement in secs	738260192
total execution time in secs	42

Case 22: Everything is the same as in test case 1 except the agl has been changed to 34.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	10000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	34
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738263572
end time measurement in secs	738263614
total execution time in secs	42

Case 23: Everything is the same as in test case 1 except the agl has been changed to 40.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	10000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	40
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738266866
end time measurement in secs	738266908
total execution time in secs	42

The following set of test cases varied the FOV angle.

Case 24: Everything is the same as in test case 1 except the fov angle has been changed to 180.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	180
range	10000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738267144
end time measurement in secs	738267191
total execution time in secs	47

Case 25: Everything is the same as in test case 1 except the fov angle has been changed to 90.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	90
range	10000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738267494
end time measurement in secs	738267541
total execution time in secs	47

Case 26: Everything is the same as in test case 1 except the fov angle has been changed to 60.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	60
range	10000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738267722
end time measurement in secs	738267769
total execution time in secs	47

Case 27: Everything is the same as in test case 1 except the fov angle has been changed to 30.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	30
range	10000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738267961
end time measurement in secs	738268008
total execution time in secs	47

Case 28: Everything is the same as in test case 1 except the fov angle has been changed to 240.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	240
range	10000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738271609
end time measurement in secs	738271656
total execution time in secs	47

Case 29: Everything is the same as in test case 1 except the fov angle has been changed to 260.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	260
range	10000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738269193
end time measurement in secs	738269240
total execution time in secs	47

Case 30: Everything is the same as in test case 1 except the fov angle has been changed to 280.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	280
range	10000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	738269394
end time measurement in secs	738269441
total execution time in secs	47

Case 31: Everything is the same as in test case 1 except the fov angle has been changed to 300.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	300
range	10000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	739500850
end time measurement in secs	739500897
total execution time in secs	47

Case 32: Everything is the same as in test case 1 except the fov angle has been changed to 310.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	310
range	10000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	739501219
end time measurement in secs	739501266
total execution time in secs	47

Case 33: Everything is the same as in test case 1 except the fov angle has been changed to 320.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	320
range	10000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	739501586
end time measurement in secs	739501633
total execution time in secs	47

Case 34: Everything is the same as in test case 1 except the fov angle has been changed to 330.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	330
range	10000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	739501747
end time measurement in secs	739501794
total execution time in secs	47

Case 35: Everything is the same as in test case 1 except the fov angle has been changed to 340.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	340
range	10000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	739501905
end time measurement in secs	739501952
total execution time in secs	47

Case 36: Everything is the same as in test case 1 except the fov angle has been changed to 350.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	350
range	10000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	739502049
end time measurement in secs	739502096
total execution time in secs	47

Case 37 Everything is the same as in test case 1 except the fov angle has been changed to 355.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	355
range	10000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	739502208
end time measurement in secs	739502255
total execution time in secs	47

Case 38: Everything is the same as in test case 1 except the fov angle has been changed to 356.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	356
range	10000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	739502362
end time measurement in secs	739502409
total execution time in secs	47

Case 39: Everything is the same as in test case 1 except the fov angle has been changed to 357.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	357
range	10000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	739502509
end time measurement in secs	739502556
total execution time in secs	47

Case 40: Everything is the same as in test case 1 except the fov angle has been changed to 358.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	358
range	10000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	739502659
end time measurement in secs	739502706
total execution time in secs	47

Case 41: Everything is the same as in test case 1 except the fov angle has been changed to 359.

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	359
range	10000
observer_latitude	31 12 43 N
observer_longitude	97 36 52 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	739502825
end time measurement in secs	739502872
total execution time in secs	47

The following set of test cases varies the observer position.

Case 42: Everything is the same as in test case 1 except the observer position was changed to Latitude: 31 13 1 N, Longitude: 97 42 17 W

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	10000
observer_latitude	31 13 1 N
observer_longitude	97 42 17 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	739503180
end time measurement in secs	739503219
total execution time in secs	39

Case 43: Everything is the same as in test case 1 except the observer position was changed to Latitude: 31 12 33 N, Longitude: 97 40 55 W

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	10000
observer_latitude	31 12 33 N
observer_longitude	97 40 55 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	739503436
end time measurement in secs	739503478
total execution time in secs	42

Case 44: Everything is the same as in test case 1 except the observer position was changed to Latitude: 31 10 37 N, Longitude: 97 40 19 W

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	10000
observer_latitude	31 10 37 N
observer_longitude	97 40 19 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	739503634
end time measurement in secs	739503679
total execution time in secs	45

Case 45: Everything is the same as in test case 1 except the observer position was changed to Latitude: 31 9 29 N, Longitude: 97 42 7 W

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	10000
observer_latitude	31 9 29 N
observer_longitude	97 42 7 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	739503836
end time measurement in secs	739503879
total execution time in secs	43

Case 46: Everything is the same as in test case 1 except the observer position was changed to Latitude: 31 14 50 N, Longitude: 97 42 29 W

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	10000
observer_latitude	31 14 50 N
observer_longitude	97 42 29 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	739504018
end time measurement in secs	739504055
total execution time in secs	37

Case 47: Everything is the same as in test case 1 except the observer position was changed to Latitude: 31 13 52 N, Longitude: 97 39 17 W

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	10000
observer_latitude	31 13 52 N
observer_longitude	97 39 17 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	739504200
end time measurement in secs	739504240
total execution time in secs	40

Case 48: Everything is the same as in test case 1 except the observer position was changed to Latitude: 31 11 30 N, Longitude: 97 38 18 W

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	10000
observer_latitude	31 11 30 N
observer_longitude	97 38 18 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	739504392
end time measurement in secs	739504436
total execution time in secs	44

Case 49: Everything is the same as in test case 1 except the observer position was changed to Latitude: 31 7 32 N, Longitude: 97 42 13 W

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	10000
observer_latitude	31 7 32 N
observer_longitude	97 42 13 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	739504575
end time measurement in secs	739504618
total execution time in secs	43

Case 50: Everything is the same as in test case 1 except the observer position was changed to Latitude: 31 14 43 N, Longitude: 97 37 2 W

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	10000
observer_latitude	31 14 43 N
observer_longitude	97 37 2 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	739504749
end time measurement in secs	739504788
total execution time in secs	39

Case 51: Everything is the same as in test case 1 except the observer position was changed to Latitude: 31 11 23 N, Longitude: 97 36 15 W

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	10000
observer_latitude	31 11 23 N
observer_longitude	97 36 15 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	739504955
end time measurement in secs	739504999
total execution time in secs	44

Case 52: Everything is the same as in test case 1 except the observer position was changed to Latitude: 31 8 12 N, Longitude: 97 37 37 W

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	10000
observer_latitude	31 8 12 N
observer_longitude	97 37 37 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	739505150
end time measurement in secs	739505196
total execution time in secs	46

Case 53: Everything is the same as in test case 1 except the observer position was changed to Latitude: 31 5 46 N, Longitude: 97 41 55 W

elevation	level_1
interpolation	bilinear
shadow_low	0
shadow_high	0
fov_angle	360
range	10000
observer_latitude	31 5 46 N
observer_longitude	97 41 55 W
focus point latitude	31 5 57 N
focus point longitude	97 36 52 W
agl	30
shade	invisible to observer
type	fade
fade percent	50
color of tinting	yellow
start time measurement in secs	739505306
end time measurement in secs	739505349
total execution time in secs	43

MISSION

OF

ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Materiel Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.