FEASIBILITY OF HARDWARE-BASED COMPUTER GENERATED FORCES FOR EMBEDDED TRAINING

Stephen A. Schricker and Robert W. Franceschini
Institute for Simulation and Training
University of Central Florida
Orlando, Florida

Amar Mukherjee
School of Computer Science
University of Central Florida
Orlando, Florida

ABSTRACT

In one concept of embedded training, the goal is to outfit a military vehicle with computer hardware capable of battlefield simulation, and use this to train personnel in the field. The advantages are two-fold. First, personnel are trained on the same equipment they use in real-life exercises. Second, personnel can be trained locally; there is no need to transport them to dedicated simulation facilities that house expensive, single-use machinery. Such a training environment could benefit from the use of Computer Generated Forces (CGF) to provide automated opponents against which personnel may train. However, a critical problem with the use of CGF in embedded training is space: current CGF systems run on relatively large workstations that will not fit into operational equipment.

This paper discusses the feasibility of one technical approach for effective embedded training using CGF systems: hardware-based CGF. Rather than using an unwieldy and expensive high-performance workstation to provide the CGF functionality, it may be possible to construct a “CGF processor” using Very Large Scale Integration technology. A CGF processor would provide comparable performance at a size much smaller than the general-purpose workstations currently being used for CGF applications. This miniaturized “CGF-on-a-chip” would fit into preexisting onboard computer systems in operational equipment, eliminating the need for bulky workstations to provide CGF.

This paper identifies the set of operations performed by a CGF system, including those operations that are particularly relevant to embedded training, and are thus good candidates for implementation in hardware. The paper illustrates these ideas concretely by discussing a hardware algorithm for one CGF operation.

AUTHORS’ BIOGRAPHIES

Stephen A. Schricker is a Research Associate at the Institute for Simulation and Training and is currently a software engineer on the Eagle/ModSAF project. Mr. Schricker has earned a B.S. in Computer Science from the University of Central Florida and is currently pursuing an M.S. in Computer Science from UCF. His research interests include Computer Generated Forces, computer architecture, and networks.

Robert W. Franceschini is a Computer Scientist at the Institute for Simulation and Training, where he currently leads IST’s research on command entity technology. He has eight years of experience in distributed simulation and computer generated forces. Mr. Franceschini received a B.S. in computer science from the University of Central Florida and is pursuing a Ph.D. in computer science at UCF.

Amar Mukherjee is a Professor of Computer Science at the University of Central Florida. Professor Mukherjee received the D.Phil.(Sc.) degree from the University of Calcutta in 1962 at the Institute of Radiophysics and Electronics. Since then his research has been focused on VLSI design, algorithms, and architectures. He has held faculty positions at the University of Iowa, Montana State University, and Princeton University. Professor Mukherjee is a Fellow of the IEEE and served two terms as Editor of the journal *IEEE Transactions on Computers.*

FEASIBILITY OF HARDWARE-BASED COMPUTER GENERATED FORCES FOR EMBEDDED TRAINING

Steohen A. Schricker and Robert W. Franceschini
Institute for Simulation and Training
University of Central Florida
Orlando, Florida

Amar Mukherjee
School of Computer Science
University of Central Florida
Orlando, Florida

BACKGROUND

Computerized Simulation as a Training Tool

Computerized simulation has increasingly become the tool of choice for soldier training. Trainees can learn how to operate expensive equipment, but can do so in the virtual environment-avoiding costly wear-and-tear on operational equipment. As the use of simulation has become more widespread, its applications have expanded as well. Groups of trainees participate in networked exercises in which they learn cooperative behaviors. These virtual exercises allow vehicle crews to participate in realistic large-scale battles at a fraction of what it would cost to train in the field.

However, one problem with simulation is that providing opposing-force entities to interact with trainees is expensive. A force-on-force battle requires some crewed simulators to be allocated as opposing forces-preventing them from being used to train.

Computer Generated Forces

One solution to this problem involves the use of computers to provide additional entities required by an exercise. Computer Generated Forces (CGF) is a technique for supplying friendly and opposing-force entities to a simulated exercise. A CGF system typically consists of **a** collection of computers connected via a network; the computers generate and control multiple simulation entities using software and (possibly) a human operator.

Embedded Training

Computerized simulation as a training tool does have its drawbacks, however. Crewed simulators are often contrivances of the vehicles they are intended to represent. In addition, they are not readily portable.

To solve these problems, the concept of embedded training has emerged in which simulation technology is placed inside operational equipment. The benefit of embedded training is twofold: soldiers train using the equipment they would use in a real exercise; and their training platform is completely portable.

One problem with embedded training, however, is that the computer systems on which they reside are limited to the confines of the operational equipment in which they are installed. These spatial limitations put constraints on the capabilities of embedded training systems; specifically, they typically have only enough room to simulate a single entity. To participate in larger simulated exercises, embedded training systems must be connected to bulky CGF systems.

Techniques for Implementing Hardware Algorithms

The microelectronics/VLSI technology has made unprecedented progress during the last three decades. The demand for high performance computing and the need to process high volumes of data at a high speed or real-time is also increasing. The technology power must be utilized to solve these computation and data intensive problems by developing special-purpose architectures and hardware algorithms in VLSI (Mead 1980) (Mukherjee 1986). The idea is to off-load the bottleneck tasks on special-purpose chips.

The speed of computation can be improved by incorporating parallelism in hardware. There are two major techniques for hardware parallelism: pipelining and systolic hardware algorithms. The pipelining idea exploits simultaneous execution of the same operation on different sets of data which are streamlined across a bandwidth limited interface. For systolic computation, the basic idea is to develop a few basic cells or hardware modules that can be replicated in a regular geometry to reduce communication overhead and be easily realizable in VLSI. However, difficulty lies in distributing the necessary data to appropriate cells to keep the computation going; typically they need high data bandwidth. The ideal situation is to use pipelining and systolic computation simultaneously which produces not only high throughput but increased utilization of the hardware resources.

Potentially Applicable Hardware Technologies

A viable approach to embedded training using Computer Generated Forces (CGF) is to develop Application-Specific Integrated Circuits (ASICs) (Smith 1997) for operational equipment with simulation hardware algorithms built into a few chips. A challenge in the hardware approach would be to reconfigure the architecture to continual changes to ModSAF (Courtemanche 1995) releases on a yearly cycle. However, emerging results in the area of Field Programmable Gate Arrays suggest an alternative is possible (Villasenor 1997). Research in this area is focusing on ways to develop hardware implementations that have the speed and size advantages of "hard-coded" chips but are capable of being reprogrammed on the fly. This means that the entire chip might not need to be fabricated again to accommodate new software releases.

OPERATIONS PERFORMED BY A CGF SYSTEM

A CGF system is a complex piece of software that must perform a variety of complex tasks. The operations that a CGF system performs may be divided into seven categories (Petty 1995): core architecture, interoperability, operator interface, physical modeling, cognitive modeling, terrain representation and reasoning, and data storage. The following sections discuss operations that are typical of each of these categories.

Core Architecture

At the heart of any CGF system is its core architecture. Responsible for the fundamental operation of the system, the core architecture provides basic functionality in three key areas: task scheduling, time representation, and complex-data-type manipulation.

Perhaps the most important function of the core architecture is the scheduling of system tasks and events. These operations make sure all of the system's functional components are given sufficient time on the processor to perform the tasks required of them. The task scheduler may be implemented as a single all-encompassing task/event queue, or as a set of queues, each of which is used to signify tasks or events that occur at different intervals. In any case, examples of scheduling operations include:

- Instantiation of task/event queues.
- Manipulation of task/event queues.
- Invocation of scheduled tasks.

The core architecture also provides the CGF system with a representation of time to control the progression of the systems events and tasks. Many CGF systems have the capability to control the speed at which a scenario progresses; that is, a simulation can run faster or slower than real-time, typically for analytic purposes, Examples of timing operations include:

- Instantiation of the timer.
- Advancing the simulation clock.
- Converting between real- and simulation-time.

Finally, the core architecture of a CGF system provides basic functionality for the construction and manipulation of complex data types, such as lists, trees, and ordered arrays. CGF systems are responsible for storing and manipulating vast amounts of data; the system can benefit by providing the means of storing these data in ways that optimize their time and space requirements. In other words, the performance of a CGF system can be enhanced by storing particular sets of data in ways that make their retrieval as fast as possible. Since data storage and retrieval are such an integral part of a CGF system, it is imperative that these operations be optimized for size and speed. Typical data-type operations include the creation of various data types, and the insertion and deletion of items.

Interoperability

Most CGF systems support some form of interoperability-the use of a network to allow a group of computers to communicate, distributing the processing workload of a simulation exercise. To be interoperable, a CGF system must provide three services: a network service, a protocol service, and a routing service. The network service allows the CGF system to send data to and receive data from the network. The protocol service provides the CGF system with the functionality to encode and decode the network messages that it sends and receives, respectively. The routing service routes incoming and outgoing messages to and from other CGF system components. Each of these services provides two functions:

- Network: send and receive.
- Protocol: encode and decode.
- Routing: transmit and route.

Operator Interface

The operator interface is responsible for providing the CGF operator with relevant data from a simulation exercise, and for allowing the operator to provide input to the CGF system. The operator interface typically displays a representation of the virtual environment-usually in the form of a map or plan-view display. Most operator interfaces are graphical, although many provide facilities for text-based input and output as well. Examples of operator interface operations include:

- Registration of input events (definition of actions to take upon input).
- Handling of input events (activation of appropriate input responses).
- Parsing of text-based commands.
- Plan-view, text-based display updates.

Physical Modeling

Physical modeling addresses the way in which a CGF system represents the physical characteristics of the entities it simulates, and the way they behave in the physical environment in which they exist. Since most CGF systems model the real world, simulated entities must obey the laws of physics that apply to their actions. It is a CGF system's physical modeling components that apply and enforce these laws. Examples of physical modeling operations include:

- Orientation about the environment (roll, pitch, yaw).
- Visual characteristics (dimensions, visible emissions).
- Velocity and acceleration constraints.
- Performance characteristics (equipment failure rates, munitions capabilities).
- Aircraft/munitions flight dynamics.
- Collision detection.
- Damage assessment.
- Electromagnetic emissions/sensor capabilities (radar/detection).

Cognitive Modeling

Just as physical modeling represents the physical characteristics of a simulated entity, cognitive modeling represents the "intelligence" controlling the entity's actions-that is, its decision-making capability. For example, while an entity's physical model determines what happens as it moves along a road (how fast it can go, how quickly it can stop), its cognitive model is what allows it to decide that it should take the road in the first place. Cognitive models may be implemented using several methods, such as algorithms, rules, finite state automata, or neural networks. Examples of cognitive modeling operations include:

- Unit/vehicle routing.
- Plan generation.
- Task organization, prioritization, management.
- Targeting and weapons selection.
- Threat assessment.
- Unit organization.
- Rules of engagement.

Terrain Representation and Reasoning

Since most CGF systems model the real world, they require a simulated terrain on which exercises may be conducted. A terrain database provides this component, and terrain operations provide to simulated entities the means to interact with the terrain. Terrain operations may be divided into two categories: representation and reasoning. Terrain representation is concerned with how terrain data is presented to the CGF system. For example, a terrain database may represent terrain as a matrix of elevation posts; associated with each post is a value representing the latitude and longitude coordinates of the post, along with its height above sea level. The terrain may then be visualized as a set of polygons constructed by drawing straight lines between the posts. Thus, the height of a location on the terrain is interpolated from the height values of the posts that compose the polygon in which the location resides.

Terrain reasoning provides tools for analyzing the terrain. It is used to establish line-of-sight between points on the terrain database to determine if two entities can see one another, and to determine optimal routes for vehicle movement.

Examples of terrain operations include:

- Computation of height.
- Lookup of features (soil, canopies, treelines, cultural features).
- Reading terrain patches.
- Coordinate conversions.
- Inter-visibility computations.

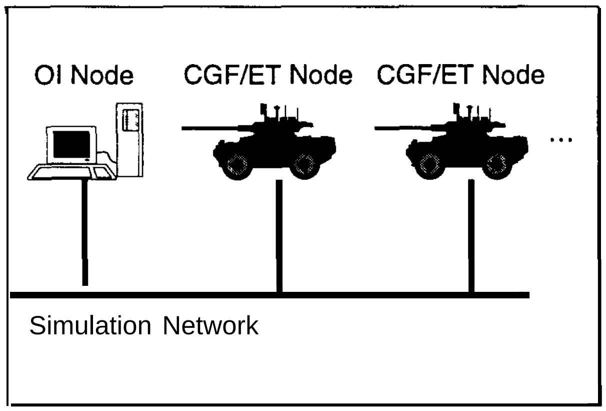

Figure 1. Conceptual CGF/ET Architecture

Entity/Unit Data Storage

All CGF systems maintain a set of data reflecting the state of the units and entities they simulate. Therefore, a CGF system requires a repository for this data. Entity/unit data storage operations involve either information retrieval (get) or information storage (set), and are typically defined for each field available in an entity/unit data storage record.

Embedded Training Criteria

An important limitation of CGF is that it requires a human operator; completely autonomous CGF systems have yet to be realized. This limitation has an important implication for embedded training: such a system requires a person to operate the CGF component. It is not feasible to have a member of a vehicle's crew double as the CGF operator-he or she must be allowed to concentrate on training. Therefore, an embedded training system that uses CGF requires an additional person to act solely as the CGF operator.

A Model for Embedded Training with CGF

An embedded training system must be small enough to fit within the confines of the vehicle in which it is to reside. Most vehicles, however, are extremely limited in the amount of space they have available for additional components. Therefore, it is unlikely that a vehicle would have sufficient space for an operator interface component and its accompanying CGF operator; these parts of the system would reside outside the vehicle.

Likewise, since CGF has been used to train cooperative behaviors among groups of vehicles (by using crewed simulators), it makes sense that a similar capability should exist in an embedded training environment that uses CGF. Therefore, the system must have the ability to interoperate with other similarly-equipped vehicles (and even with crewed simulators).

Figure 1 illustrates the concept for embedded training with CGF that IST has developed. It consists of an Operator Interface (01) node networked to one or more CGF/Embedded Training (CGF/ET) nodes. The 01 node resides outside of the vehicles; its primary function is to provide the operator interface for the CGF component. Each CGF/ET node provides the embedded training functions for its vehicle, along with a CGF capability for simulating entities.

Required Operations

The 01 node requires the following operations:

- Timing and scheduling core operations.
- All operator interface operations.
- All interoperability services.
- All terrain operations.

The CGFIET nodes require the following operations:

- All core operations.
- All interoperability operations.
- All physical modeling operations.
- All cognitive modeling operations.
- All terrain operations.
- All entity/unit data operations.

Candidate Operations for HW Implementation

The goal of this project is to define those operations of a CGF system that might benefit most from being implemented in hardware. This will result in a reduction in the space requirements of a complete CGF system, and the ability to install a complete CGF system within the confines of existing operational equipment. By carefully choosing the operations to implement in hardware (as opposed to software), it will be possible to construct an embedded training system that provides the most capability in the smallest space. Candidate operations will be those that are expensive to execute (that is, take a long time), are executed often, or operate on large amounts of data.

Research has shown that CGF systems spend a large portion of their time in three areas: terrain processing, interoperability, and data storage. Terrain representation requires large amounts of

Library Name	% Exec Time*	Associated Operation
Libvtab	7.89	Core
Libpduapi	5.25	Interoperability
Libctdb	4.72	Terrain
Libentity	4.34	Entity/unit data, physical modeling
Libroutemap	2.40	Cognitive
Libtask	2.35	Cognitive
Libmovemap	1.91	Cognitive
Libpbtab	1.57	Terrain
Libtaskpri	1.57	Cognitive
Libgcs	1.43	Terrain
*43.0% attributed to C librarv/svstem calls		
01 operations not shown-no GUI in experiment.		

Table 1. Most Frequently Used ModSAF Libraries

memory, while terrain reasoning operations typically take a long time to execute. Network operations, while inexpensive individually, become expensive out of sheer volume. Finally, since data storage is at the heart of any CGF system, it is imperative that entity and unit data be stored in the most efficient means possible for smallest size and greatest speed. Therefore, the CGF operations most likely to benefit from implementation in hardware would be interoperability, terrain representation and reasoning, or data storage operations.

Performance Analysis of ModSAF

IST conducted a performance analysis of ModSAF-a CGF system currently being used for both training and research-to determine the functions on which it spends the majority of its time. The system is divided into several hundred libraries. Each library is responsible for providing a different component of its overall functionality, and can be categorized into one of the seven CGF operations defined above (Core, Interoperability, Operator Interface, Physical Modeling, Cognitive Modeling, Terrain, and Entity/Unit Data Storage). IST gathered data that shows the amount of time ModSAF spends inside each function it calls.

Experimentation Details

ModSAF is equipped with a set of self-benchmarking routines (Vrablik 1994). These routines are designed to provide an indication of the capacity of the system at which the risk of overloading the system is minimized. These routines are executed while running a special exercise that guarantees to fully stress the system.

CGF Operation	% Exec Time*
Cognitive Modeling	15.67
Core Architecture	13.68
Terrain Rep/Reasoning	8.82
Entity/Unit Data Storage	7.15
Interoperability	6.09
Physical Modeling	5.59
*43.0% attributed to C library/system calls	
OI operations not shown–no GUI in experiment.	

Table 2. ModSAF Execution by CGF Operation

IST used this benchmarking scenario to collect its performance analysis data. The data was gathered using the CASEVision Workshop Performance View tool (cvperf). Cvperf gathered data showing the amount of time ModSAF spends inside each of its functions. IST then associated these times with each function's respective library. Table 1 displays the ten libraries inside which ModSAF spends the largest fraction of its time.

Table 2 provides a breakdown of ModSAF execution by CGF operation.

Table 1 shows that four out of the top 10 most costly operations in ModSAF are cognitive modeling operations. Table 2 shows that cognitive modeling operations account for almost 16% of ModSAF's total execution time. These results suggest that the implementation of cognitive modeling operations in hardware would provide the most benefit to an embedded training system.

ILLUSTRATION OF A CGF ALGORITHM IN HARDWARE

Inter-visibility Computation

A natural question to ask at this point is: "What would a CGF hardware implementation look like?" Here, we sketch the main ideas behind the development of a hardware algorithm for one CGF operation. It is important to point out that we are not claiming to have completely implemented a CGF system, nor are we claiming to have the best hardware algorithm for the illustrated problem. We simply wish to gain some understanding of the issues that arise in developing hardware algorithms for CGF.

For the purpose of illustration of a hardware-based CGF algorithm, we will present a hardware solution for the infer-visibility problem in battlefield simulation. The problem is to determine whether

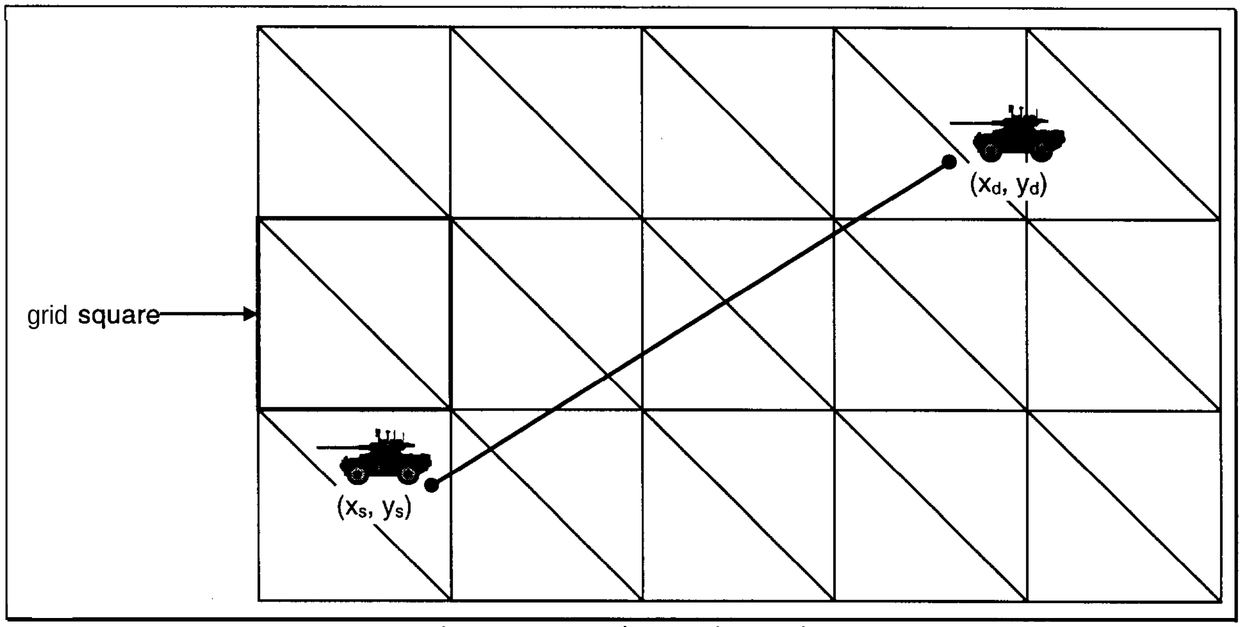

Figure 2. Example Terrain Database

two simulated entities can "see" each other, that is, whether their line of sight intersects any terrain polygon. Computing intervisibility is important in CGF systems and has been the subject of much research, for example (Petty 1992) and (Petty 1997).

Algorithm

The terrain database is assumed to be represented by the most commonly used CGF database format which is a matrix of elevation posts. In this format, the terrain elevation relative to vertical origin is given at points uniformly spaced in two dimensions relative to horizontal origin. Thus, in the XY-plane we assume a superimposed regular *grid,* with its origin at the point (0,0). We assume that the horizontal and vertical grid lines are uniformly spaced at an integral distance of k units (meters). The elevation posts are the Z-coordinate values of the terrain at the grid points whose XY-coordinate values are integral multiples of k. The polygonal (triangulated) representation of the terrain surface is constructed from the elevation posts by assuming a diagonal bisects each of the squares (from top left corner to bottom right corner) formed by the elevation posts in the XY-plane. The elevation of an intermediate point on the diagonal line is interpolated using the elevation values of the corner points. Thus each square surface of the terrain is described by two triangular surfaces in 3D as shown in Figure 2. A line of sight is specified by giving the coordinates in space of the source (xs,ys,zs) and the destination (xd,yd,zd). An intersection of the line of sight with a terrain polygon (triangle) blocks the line of sight. The object of the intervisibility algorithm is to determine whether such an intersection exists.

Since the terrain database is huge, it is impractical to try to perform the intersection computation with every triangle in the database. Not only will that be computationally expensive because such calculations will have to be performed over many different lines of sight, but it will also create a severe memory bandwidth bottleneck to download the entire terrain database to the processor. We would like to fetch from the database only those polygons that will potentially intersect with the given line of sight. A high level description of the algorithm is given below. The algorithm has two phases; in the first phase it computes the addresses of the relevant squares from the terrain database and in the second phase the actual intersection is performed. As we will see later, this structure fits well with the pipelined hardware algorithm given in the next section.

The first phase of the algorithm proceeds as follows. The addresses of the squares are expressed in terms of two indices both of which are normalized integers with respect to k, denoting the lower left hand corner of the square. The projection of the line of sight on the XY-plane is given by the equation y=mx+c, where both m and c (the slope and the y-intercept) can be computed

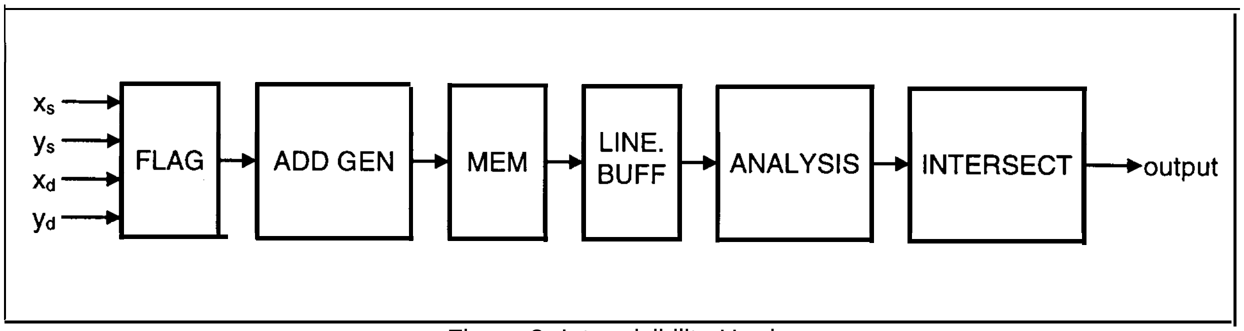

Figure 3. Inter-visibility Hardware

in terms of (x_s,y_s) and (x_d,y_d) as $m=\frac{y_d-y_s}{x_d-x_s}$ and $c=y_s-mx_s$. For simplicity of illustration, we assume that the slope is positive and the line of sight is in the positive XY-plane. The initial square containing the source has the address $(x=\frac{x_s}{k}, y=\frac{y_s}{k})$. Again for simplicity of illustration, we assume k is a power of 2, say 128 so that division can be performed by left shifts of integer-valued coordinates (allowing more general values for k requires more complicated hardware for performing the division computation). As we trace the line of sight sequentially from source to destination, we fetch the next grid square sequentially. At any given step of computation the algorithm generates two control bits in the first phase specifying whether the next square is to the right, above, or diagonally across from the current square (the corresponding control bits are 00, 01, and 11, respectively). A high-level description of the algorithm is given below:

```
$x \leftarrow x_s/k$; $y \leftarrow y_s/k$;
while $x \le x_d/k$ or $y \le y_d/k$ do
{
        if $m(x+1)+c < y+1$ then
        {
                $x \leftarrow x+1$;
                $flag \leftarrow 00$;
        }
        else if $(y+1-c)/m < x+1$ then
        {
                $y \leftarrow y+1$;
                $flag \leftarrow 01$;
        }
        else $flag \leftarrow 11$;

        output $flag$;
}
```

We note here that for our hardware implementation we have reformulated this algorithm in terms of multiplications and integer computations for simplicity. The algorithm can be reformulated in terms of multiplications as follows: Define $dx = x_d - x_s$ and $dy = y_d - y_s$. Our previous line equation $y=mx+c$ can be recast into the form $(dx)y=(dy)(mx)+(dx)c$ (note that $m=dy/dx$ from our previous equations). From here, we can rewrite the conditions tested in the algorithm above. For example, the first conditional test requires us to compute whether $m(x+1)+c < y+1$. We can compute this instead by computing whether $(dy)(x+1)+(dx)(c) < (dx)(y+1)$, which eliminates the requirement for division hardware in our algorithm. The other condition can be similarly expressed.

In the second phase, the algorithm generates the actual address of the grid square to be fetched. The code for this is the following:

```
Compute the initial square address $(x_{add}, y_{add})$;
if $flag = 00$ then $x_{add} \leftarrow x_{add}+1$
else if $flag = 01$ then $y_{add} \leftarrow y_{add}+1$
else { $x_{add} \leftarrow x_{add}+1$, $y_{add} \leftarrow y_{add}+1$}
```

Hardware Implementation

The proposed hardware of a simple prototype design to perform the intervisibility task is shown in Figure 3. It is designed as a pipelined architecture to produce high throughput. The input to the processor is a sequence of (source, destination) coordinate pairs giving the lines of sight. The output of the processor is a sequence of bits, one bit per line; a bit is set to 1 if the line of sight intersects any terrain polygon and is set to 0 if the line of sight does not intersect the terrain.

The first block, called "FLAG," in the pipeline computes the initial address of the square and then generates the addresses of the succeeding squares that need to be fetched to determine the intervisibility. This is conveyed in the form of a sequence of flag bits as described in the algorithm above.

The address generator block called "ADD GEN" delivers the addresses of the initial square and the addresses of the subsequent squares in pipelined fashion to the memory.

The memory may have to be multi-ported. Address decoding, memory access, and output to data buffer operations are pipelined to produce enough data to keep the pipeline full.

The output of the memory goes to "LINEBUFF" which stores the information about the polygons (and their elevation posts for the corner points) for the line under processing.

The information then goes into the "ANALYSIS" block which determines whether the Z-value of the line at the point where it could possibly intersect is above the interpolated Z-value of the point on the diagonal line. If yes, the intersection computation is skipped. If not, the actual intersection computation is performed.

The block "INTERSECT" performs the intersection computation. If the line is found to intersect any diagonal, further input from the LINE.BUFF is suspended, output is set to 1 and next set of inputs to LINE.BUFF is processed. It will be necessary to have a dual buffer to efficiently perform this operation.

One important technical point related to the pipelined architecture presented above deserves mention here. When accessing a grid square we need data about the four corner points. At worst, this requires us to perform four loads from memory. We can do this by having a very wide data path to and from the memory to allow the four loads to proceed concurrently. We can also have a small data path that allows only one memory access at a time. Here, we must recognize that we really only need to load some of the corner data, depending on whether we're loading the square directly above, directly to the right, or diagonally from the current square. In the first two cases, our current square already contains two of the corners, so we need only two loads from the memory. In the last case, our current square contains one corner, so we need to load three corners of the new square into memory. There is a difficulty here in that we cannot tell a priori whether we need to perform two or three memory accesses; this means that we must carefully coordinate the memory accesses with respect to the execution of the overall algorithm to maintain good performance.

We are implementing this algorithm in VHDL code and we intend to test it on a Field Programmable Gate Array in the VLSI Laboratory of the School of Computer Science at UCF.

Comparison of Software and Hardware Implementations

Even in our simplified algorithm presented above, we can see some of the benefits and difficulties involved in developing hardware algorithms. The primary benefit of the hardware approach is that we can exploit parallelism to perform computationally expensive tasks quickly. However, development of hardware algorithms can be a tedious process as compared to software algorithm development. For example, a software implementation of the above algorithm could easily deal with the division computations. As we saw, these divisions would require the addition of complicated hardware to our chip. However, we were able to avoid this by recasting the computations in terms of more easily realizable hardware algorithms.

CONCLUSIONS

The work reported here is a small first step toward investigating hardware-based Computer Generated Forces. We have presented a model for the use of CGF for embedded training. Based on this model and some experimental results, we have developed suggestions for which parts of a CGF system should be put into hardware for embedded training. Finally we have illustrated the idea of hardware-based CGF by presenting the main ideas of a hardware implementation of an important CGF algorithm: intervisibility computation.

Based on our results, we have identified the following areas for further work. Most importantly, the work of developing a hardware-based CGF system can continue. Along with that effort, work which defines more precisely the role of CGF within embedded training can be conducted. Finally, continued work on gathering performance data from ModSAF from a wide range of scenarios and expanding the implementation of our intervisibility algorithm can be performed.

ACKNOWLEDGEMENT

This research was sponsored by the Institute for Simulation and Training under the In House Grants program. That support is gratefully acknowledged.

REFERENCES

Courtemanche, A.J. and Ceranowicz, A. (1995) "ModSAF Development Status," Proceedings of *the Fifth Conference on Computer Generated Forces and Behavioral Representation,* Institute for Simulation and Training, Orlando FL, May 9-11 1995, pp. 3-13.

Mead, C. and Conway, L. (1980) *introduction to VLSI Systems.* Addison-Wesley.

Mukherjee, A. (1986) *introduction to nMOS and CMOS VLSI* Sysfems. Prentice-Hall.

Petty, M, Campbell, C, Franceschini, R, and Provost, M. (1992) "Efficient Line of Sight Determination in Polygonal Terrain," *Proceedings of the 7992 Image VI Conference,* Image Society, Inc., Scottsdale AZ, July 14-l 7 1992, pp.239-253.

Petty, M. (1995) "Computer Generated Forces in Distributed Interactive Simulation," *Distributed Interactive Simulation Systems for Simulation and Training in the Aerospace Environment,* SPIE Press, pp. 251-280.

Petty, M. (1997) *Computational Geometry Techniques for Terrain Reasoning and Data Distribution Problems in Distributed Battlefield Simulation.* Ph.D. Dissertation, University of Central Florida Department of Computer Science.

Smith, M.J.S. (1997) *Application-Specific Integrated Circuits.* Addison-Wesley.

Villasenor, J. and Mangione-Smith, W.H. (1997) "Configurable Computing," *Scientific American,* June, pp. 66-71.

Vrablik, R and Richardson, W. (1994) "Benchmarking and Optimization of ModSAF," *Proceedings of the Fourth Conference on Computer Generated Force and Behavioral Representation,* Institute for Simulation and Training, Orlando FL, May 4-6 1994.