Dead Reckoning on the GPU: A Comparative Study vs. the CPU

Glenn A. Martin, Ronald M. Jewett, Christopher Hollander, Cory Hicks
University of Central Florida
Orlando, FL
martin@ist.ucf.edu, ronald.m.jewett@lmco.com, cholland@ist.ucf.edu, coryhicks@digilla.com

ABSTRACT

In many simulation systems, dead reckoning is used to minimize network bandwidth utilization. The Distributed Interactive Simulation (DIS) standard is one example protocol that uses dead reckoning. Many game engines also use the technique. Until a few years ago graphics hardware used a fixed pipeline. In recent years PC video cards have been built with a programmable architecture. Collectively, the programmable pipeline is referred to as the Graphics Processing Unit (GPU). As GPU programming has progressed, a growing research field into applying non-graphical algorithms onto the GPU has started. Image processing, numerical equations and illumination computation are some examples of what is called General Purpose GPU programming.

We performed a computational study of dead reckoning comparing the GPU with the Central Processing Unit (CPU). We tested various quantities of simulated entities using a variety of CPUs and GPUs. GPUs have the possibility of dead reckoning millions of entities in a single pass, but suffer the requirement of data readback from the video card, which is often slower than "outbound" data transfer. The study is presented and then analysis of the results discussed.

ABOUT THE AUTHORS

Glenn A. Martin is a Senior Research Scientist at the University of Central Florida's Institute for Simulation and Training where he leads the Interactive Realities Laboratory, pursuing research in multi-modal, physically realistic, networked virtual environments and applications of virtual reality technology. Since joining the institute in 1995, he has worked on numerous projects related to virtual reality including a testbed for evaluating VR for training uses, an investigation of network architectures, a library for VR applications and the study of human factors. In 2005 he was given a University of Central Florida Research Incentive Award (RIA) for his research in these areas. He received his B.S. in computer science in 1992, and his M.S. in computer science in 1995 both from the University of Central Florida. He is currently pursuing a Ph.D. in Modeling & Simulation at the University of Central Florida.

Ronald M. Jewett is a Software Engineer Senior at Lockheed Martin Simulation, Training and Support (LM-STS). At LM-STS, he has worked on several Virtual simulation programs including the Close Combat Tactical Trainer (CCTT), Synthetic Environment CORE (SE Core), and United Kingdom Combined Arms Tactical Trainer (UKCATT). He is currently pursuing an M.S. in Modeling & Simulation at the University of Central Florida.

Chris Hollander holds a B.A. in Mathematics from the University of South Florida. He is currently a Ph.D. student and research assistant at the University of Central Florida. His previous work includes web development and general programming. He is presently investigating problems in human-agent team interaction under funding from the Army Research Laboratory.

Cory Hicks currently holds a B.A. in Digital Media interactive systems from the University of Central Florida. He is an M.S. student in Modeling and Simulation intelligent system at the University of Central Florida. His previous work included web and design, programming, and game design. He is currently in a joint effort to create a digital interactive entertainment company.

Dead Reckoning on the GPU: A Comparative Study vs. the CPU

Glenn A. Martin, Ronald M. Jewett, Christopher Hollander, Cory Hicks
University of Central Florida
Orlando, FL
martin@ist.ucf.edu, ronald.m.jewett@lmco.com, cholland@ist.ucf.edu, coryhicks@digilla.com

INTRODUCTION

There has long been a desire to simulate large exercises (those with 1000's of entities or more). One major aspect of such quantity of entities is keeping each entity updated. Dead reckoning reduces network bandwidth by using an estimated state of the entity until an error metric becomes too large and then an update is broadcasted. However, performing the dead reckoning on large quantity of entities can be computationally expensive. Solutions that can address this computational load can provide the ability to simulate such large exercises.

BACKGROUND

Dead Reckoning

Dead reckoning is a term that is not new to either the computational era, or to virtual simulations. It is defined by the American Heritage Dictionary (2000) as "a method of estimating the position of an aircraft or a ship without astronomical observations, as by applying to a previously determined position the course and distance traveled since."

However, its usage in terms of virtual simulation is best exemplified by Calvin et al (1993), in that each object [in the simulation] extrapolates the new positions of remote objects from the states last reported by those objects. Sending objects are responsible for generating updates before discrepancies become too large. In effect, a "contract" exists between the objects. Each object agrees to maintain a dead reckoning model of itself that corresponds to the model used by remote objects to "see" that object. The updates contain externally "visible" information, including position, velocity, orientation, positions of components (such as a turret and gun barrel on a tank), dust clouds, smoke, muzzle flash, thermal signature, and other electromagnetic emissions. The dead reckoning technique is very efficient compared to sending data at fixed intervals.

The justification for using dead reckoning in virtual simulations is a simple one; it limits the number of messages communicated among the objects within the simulation. This results in reduced network bandwidth and a general decrease in latency as shown in Capin et al (1997). Despite the original domain of the military and its simulators, they are not the only users of dead reckoning. As simulation integrates into the civilian world, this technique is being adapted to almost everything which involves movement in a virtual world.

One of the more interesting examples is in the use of dead reckoning on components of virtual humans within a virtual simulation. Research conducted by Capin used this approach to reduce network overhead by nearly 50 percent. They accomplished this improvement by reducing the number of messages communicated among the participants in a virtual environment by applying a simple dead reckoning algorithm to each component of a humanoid skeletal system.

The Modern Graphics Processing Unit

The Graphics Processing Unit (GPU) has existed in various forms for decacdes. However, the PC-based Graphics Processing Unit with full 3-D capabilities (including hardware-accelerated transform and lighting) has been a technology innovation since the late 1990s, producing multiple generations of hardware development. The GeForce 256 was the first commercial GPU with such capability to the market for NVIDIA®.

As defined by NVIDIA® (2007), the GPU is "a single-chip processor with integrated transform, lighting, triangle setup/clipping, and rendering engines that is capable of processing a minimum of 10 million polygons per second." Today's GPUs from ATI® and NVIDIA® can be interfaced with either the Peripheral Component Interconnect (PCI) Express or Accelerated Graphics Port (AGP) busses and are easily replaceable for upgrades and fixes.

The GPU is specialized to handle the graphical processing that in years past the CPU would handle. Such offloading of graphical instructions to a GPU can

provide some amazing results such as providing improved processing of lighting and texturing. The progression of GPU programming is a benefit that will be discussed further in this study.

For this study, we are only concerned with working on a non-graphical problem being processed by the GPU and offloading some processing from the CPU. A brief background on the programming of GPUs is discussed in the next subsection.

GPU Programming

As GPUs have grown, the manufacturers have provided a programmability capability of the hardware, which allows programmers to develop specialized GPU programs for many special effects. Common shading languages today include the High Level Shading Language (HLSL), C for graphics (Cg), and OpenGL Shading Language (GLSL). For our work presented in this paper we chose to use GLSL (2007) as it allows for an open operating system and open hardware. The ability to adapt relatively easily between NVIDIA and ATI cards and the ability to go to multiple operating systems allows for better flexibility with our analysis. GLSL, along with other shading languages, is made of two important types of programs. These two programs are the vertex and fragment shader (the latest GPUs also support a geometry shader).

The vertex processor is a programmable unit that operates on incoming vertex values and their associated data. According to NVIDIA (2007), the vertex processor is intended to perform traditional graphics operations such as:

- Vertex transformation
- Normal transformation and normalization
- Texture coordinate generation
- Texture coordinate transformation
- Lighting
- Color material application

The fragment processor is a programmable unit that operates on fragment (pixel) values and their associated data. The fragment processor is intended to perform traditional graphics operations such as:

- Operations on interpolated values
- Texture access
- Texture application
- Fog
- Color

Our focus in this paper is in the fragment processor.

GPU and Dead Reckoning

Since GPUs are built for graphics processing, the question is why use the GPU for processing the dead reckoning algorithm? General Purpose Graphics Processing Unit (GPGPU) has provided some interesting research for this relatively new field in computer science. Papers on simulation that were of interest stem from our focus on the dead reckoning algorithm. For example, the line of sight, collision detection, and route planning algorithms research used by the One Semi-Automated Forces (OneSAF) Computer Generated Forces simulation to accelerate the runtime processing speed of the development program shows the promise of such approaches as shown in Verdesca et al (2006). The research focused on the GPU chip and how to design the existing algorithms to map with the GPU architecture. Samples of key gains include 150-200 times improvement in LOS query, ten times simulation speed improvement in the collision detection, and 10-30 times improvement in route computation.

In addition, GPUs provide for speedy floating point calculations (up to thirty-two bit floating point) and, as mentioned in the programming section, high-level programming that is easier to program compared to assembly. The load balancing between the CPU and GPU is also a benefit that can be realized if the dead reckoning algorithm is offloaded and replaced with more processing of other CPU designated instructions for simulation systems.

EXPERIMENTAL DESIGN

With the promise of general-purpose GPU, such capability could be used to support dead reckoning of entities, especially large quantities of entities. We investigated this possibility by performing a computational study of various quantities of entities on various CPUs and various GPUs. The interesting question is in comparing the parallelism nature of the GPU with its potential read-back disadvantage with the CPU that runs at very high clock rates but is minimally parallel in nature.

We looked at 2-D arrays of entities that are powers of two on each side (1x1, 2x2, 4x4, etc.). This was chosen as it is the preferred texture size for GPUs and was felt acceptable since zero data could always be added to pad any quantity of entities up to the next power of two. Each entity was dead reckoned 100 times for each sample and 100 samples were made at

each entity size to reduce external influences (e.g. operating system).

Within the CPU collection, we surveyed modern processors that are representative of those used in simulation today. We studied processors with various clock rates and various multi-core processors and included some laptop processors. Table 1 lists the CPUs used in this experiment.

Table 1. List of CPUs Used in Experiment

"Oldest"	Intel Pentium M	2.13 Ghz
	Intel Dual Xeon	1.8
		2.4
		3.2
	Intel Pentium D	3.73
	Intel Core Duo	2.0
"Newest"	Intel Core 2 Duo	2.66

On the GPU side, we surveyed various families of GPUs that were available to us and studied both AGP and PCI-Express cards due to their different read-back speeds. While these all fall within one manufacturer, we feel that they are representative of GPUs overall and our results should generalize to a GPU vs. CPU comparison. Table 2 lists the GPUs used within the experiment.

Table 2. List of GPUs Used in Experiment

"Slowest"	AGP	NVIDIA 5600
		NVIDIA 7600GS
	PCI-Express	Quadro FX 540
		Quadro Go 1400
		Quadro FX 1400
		Quadro 2500M
		NVIDIA 7800GTX
		NVIDIA 7900GS
"Fastest"		NVIDIA 7950GX2

When considering the software component of the experiment, many dead reckoning algorithms exist for both position and orientation and in various coordinate systems as defined by the IEEE (1993). For our computational study, we focused on only position using both linear velocity and linear acceleration. However, the technique will generalize to orientation. The equation (shown in Equation 1) for position used in this study is:

$$p = p + vt + \frac{1}{2}at^2 \quad (1)$$

where p is the position, v is the linear velocity, a is the linear acceleration and t is time.

Implementing dead reckoning using this equation is straight-forward on the CPU. On the GPU we implement the position, velocity and acceleration terms each within a texture. Instead of RGB components of a pixel, we store the XYZ components of the term (the past few generations of GPUs have supported floating-point textures). A shading program is written and run on the GPU to update the position texture with the velocity and acceleration textures based on the dead reckoning equation. The new positions are then read back from the GPU to main memory for use by the simulation. In this case only a fragment shading program is needed and it is shown in shown in Figure 1.

```
uniform sampler2D positionTexUnit;
uniform sampler2D linearVelocityTexUnit;
uniform sampler2D linearAccelerationTexUnit;
uniform float frameTime;

void main(void)"
{
   vec2 texCoord = gl_TexCoord[0].xy;
   vec4 p  = texture2D(positionTexUnit, texCoord);
   vec4 lv = texture2D(linearVelocityTexUnit,
                       texCoord);
  vec4 la = texture2D(linearAccelerationTexUnit,
                      texCoord);

  vec4 result;
  result.r = p.r + lv.r * frameTime +
             0.5 * la.r * frameTime * frameTime;
  result.g = p.g + lv.g * frameTime +
             0.5 * la.g * frameTime * frameTime;
  result.b = p.b + lv.b * frameTime +
             0.5 * la.b * frameTime * frameTime;
  result.a = 1.0;

  gl_FragColor = result;
}
```

Figure 1. GLSL Fragment Shader

EXPERIMENTAL RESULTS

Recall that each entity size was tested 100 times in each CPU and GPU configuration. These 100 samples were then averaged to compute one final time for each CPU and GPU at each entity quantity. The overall CPU results are shown in Figure 2.

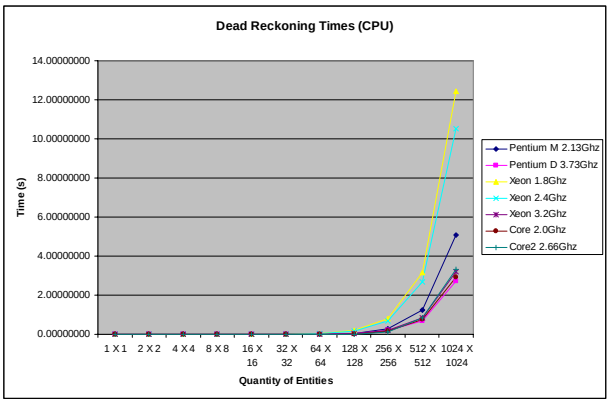


Figure 2. Plot of CPU Times

As expected, the CPU times increase as the quantity of entities increases. Interestingly, the older Xeon processors increase the most. This is likely caused by the cache sizes that those processors have relative to the others. More cache misses are occurring at the large quantities, which is causing a penalty for the processors.

The GPUs are less consistent. The overall GPU results are shown in Figure 3. Some show a longer time on small entities (1x1) that may result from overhead costs. In addition, some (but not all) increase at the largest entity quantities indicating they may be reaching their limit. Finally, some (but, again, not all) show some spikes in the middle entity quantities. This could be a cache issue or perhaps a driver implementation issue on how it sends data to the GPU.

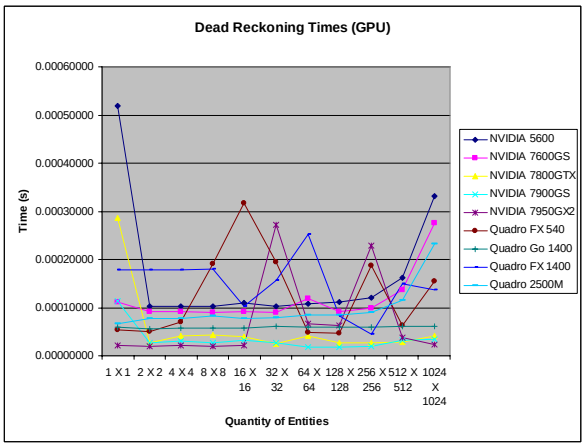
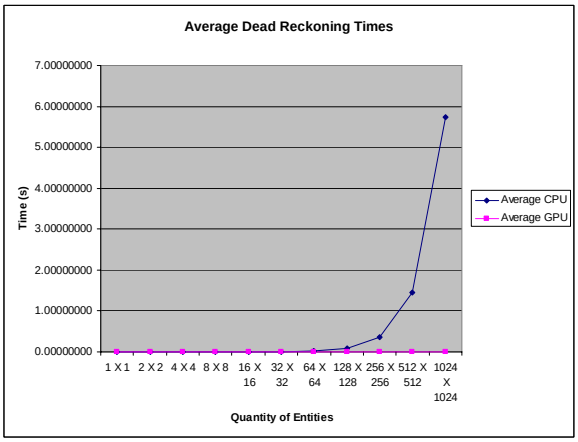
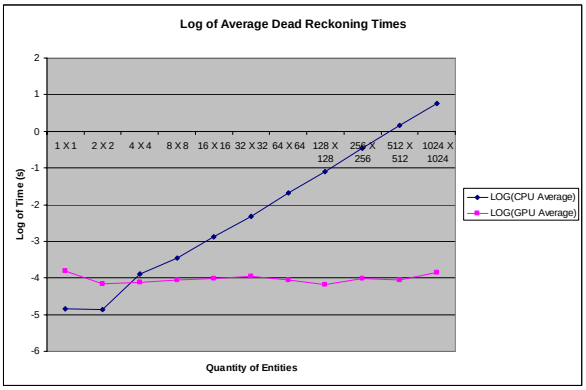


Figure 3. Plot of GPU Times

Considering the overall question of comparing the GPU with the CPU, we averaged across all the CPUs and all the GPUs and performed a comparison. Figure 4 (a) shows the increase of the CPU at larger quantities is clearly seen. However, the comparison at small quantities is unclear. Therefore, we also plot the logarithm of the data in order to see the detail more clearly (see Figure 4 (b))



(a)

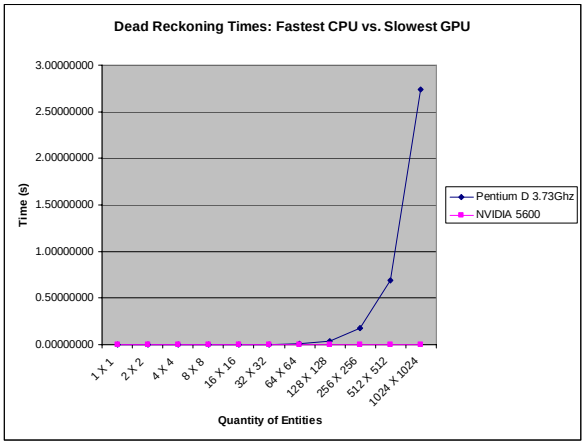


(b)

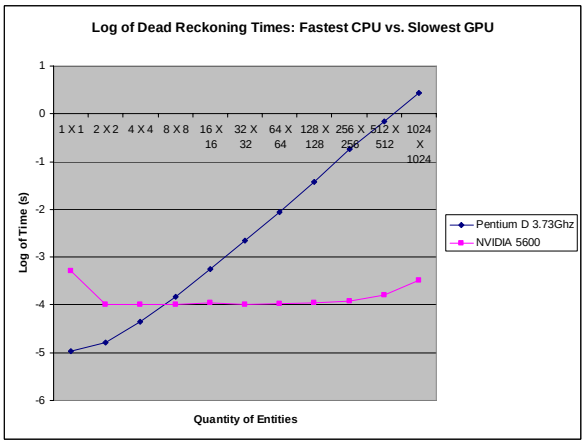
Figure 4. (a) Average Dead Reckoning Times, (b) Log of Average Dead Reckoning Times

The plot in Figure 4 (b) shows that the CPU is faster in the 1x1 and 2x2 cases but then loses to the GPU. This fulfills our initial idea that the GPU would take too much of a read-back penalty at lower quantities but that its parallelism nature would lead it to outperform the CPU. The surprising result is how quickly that happens (i.e. even dead reckoning 4x4 entities is faster on the GPU).

As an additional comparison and a full test of our initial question, we compared the fastest CPU with the slowest GPU. Figure 5 shows the results.



(a)



(b)

Figure 5. (a) Times for Fastest CPU vs. Slowest GPU, (b) Log of Times for Fastest CPU vs. Slowest GPU

Again, the initial plot shows the CPU increasing its time needed faster than the GPU as entity quantity increases. To see the lower end, the logarithm plot is again needed (Figure 5 (b)). The results were similar although slightly shifted. The fastest CPU performs better than the slowest GPU up to the 4x4 quantity (as opposed to up to just 2x2 in the overall average case from Figure 4).

CONCLUSIONS AND FUTURE WORK

Many exercises involve large quantities of entities and keeping them all updated can be quite challenging for the CPU. In addition, the CPU is required for other tasks such as user interfaces, reading and writing simulation data on the network, and environment issues such as collision detection and terrain following. Leveraging the GPU for some of the larger computational issues (especially those that parallelize well) can greatly aid overall support for large simulations.

The computational study presented in this paper showed that the GPU can indeed dead reckon large quantities of entities much more quickly than the CPU. Despite the potential read-back penalties from the video card, the parallelism capability of the GPU quickly outperforms even the fastest CPUs. Dead reckoning on the GPU can offload work from the CPU and allow the CPU to focus on user interfaces, network communication and other interactions (e.g. terrain following).

One interesting result of the study was the varying results of the GPUs even at varying quantities of entities. While we feel it is likely explained by cache or driver issues, this is one area of potential future work. If the variability can be further understood, dead reckoning on the GPU could be improved to be even faster. Additionally, even larger quantity of entities could be studied to see at what levels the GPUs may reach their limits. Finally, NVIDIA is generalizing general-purpose GPU use in their Compute Unified Device Architecture (CUDA) and leveraging this for dead reckoning would be further interesting future work.

ACKNOWLEDGEMENTS

The authors would like to thank Mr. Jason Daly of the University of Central Florida for his help on GPU design and shading program implementation. We also thank Dr. Charles Reilly of the University of Central Florida for helping to review this paper.

REFERENCES

American Heritage Dictionary, (2000). *Dead Reckoning.* Retrieved April 20, 2007 from http://www.bartleby.com/61/45/D0054500.html

Calvin, J., Dickens, A., Gaines B., Metzger P., Miller D. & Owen, D. (1993). The SIMNET Virtual World Architecture. *Virtual Reality Annual International Symposium*,. 450 – 455.

Capin, T.K. & Pandzic, I.S. (1997). A Dead-reckoning Algorithm for Virtual Human Figures. *Virtual Reality Annual International Symposium*, 161 – 169.

GLSL Tutorial, (2007). *Dead Reckoning.* Retrieved February 8, 2007 from

http://www.lighthouse3d.com/opengl/glsl/index.php?pipeline.

Institute of Electrical and Electronics Engineers (1993). "Standard for Distributed Interactive Simulation - Application protocols." IEEE Standard 1278.

NVIDIA GeForce 256. Retrieved March 1, 2007.from http://www.nvidia.com/page/geforce256.html

Verdesca, M., Munro, J., Hoffman, M., Bauer, M., & Manocha, D. (2006). Using Graphics Processor Units to Accelerate OneSAF: A Case Study in Technology Transition. *The Society for Modeling and Simulation International*, Volume 3, Issue 3, 177-187.