A Multi-core CPU Pipeline Architecture for Virtual Environments

Eric ACOSTA [a,1], Alan LIU [a], Jennifer SIECK [a], Gilbert MUNIZ [a], Mark BOWYER [a], and Rocco ARMONDA [b]

[a] *The National Capital Area Medical Simulation Center, Uniformed Services University*
[b] *National Capital Neurosurgery Consortium, National Naval Medical Center*

Abstract. Physically-based virtual environments (VEs) provide realistic interactions and behaviors for computer-based medical simulations. Limited CPU resources have traditionally forced VEs to be simplified for real-time performance. Multi-core processors greatly increase the computational capacity of computers and are quickly becoming standard. However, developing non-application specific methods to fully utilize all available CPU cores for processing VEs is difficult. The paper describes a pipeline VE architecture designed for multi-core CPU systems. The architecture enables development of VEs that leverage the computational resources of all CPU cores for VE simulation. A VE's workload is dynamically distributed across the available CPU cores. A VE can be developed once and scale efficiently with the number of cores. The described pipeline architecture makes it possible to develop complex physically-based VEs for medical simulations. Initial results for a craniotomy simulator being developed have shown super-linear and near-linear speedups when tested with up to four cores.

Keywords. Surgical simulation, Virtual Reality, Multi-core CPU, Parallel pipeline

Introduction

Computer-based medical simulations rely on physically-based virtual environments (VEs) for realistic object interactions and behaviors. Medical VEs quickly become very complex as deformable tissues and tool-tissue interactions are integrated. The computational limitations of computers has been a major bottleneck for medical VEs. Researchers must find innovative methods to simplify VEs and make approximations to required calculations for real-time performance. Multi-core CPUs are quickly becoming standard since they greatly increase the computational capacity of computers. To fully utilize all CPU cores, multiple execution threads can concurrently process tasks and expensive calculations. Multi-threading has been used extensively for VEs, but many approaches are application-specific. Additionally, many VE development tools are either single-threaded or only utilize multi-threading for specific tasks [1,2,3].

[1]Correspondence to: Eric Acosta, http://simcen.usuhs.edu; E-mail: Eric.Acosta.ctr@simcen.usuhs.edu.

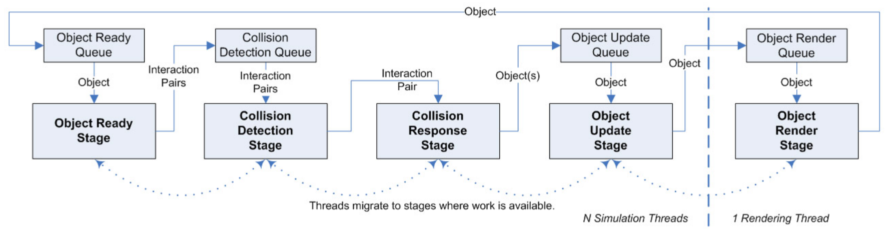

Figure 1. VE simulation and rendering pipeline overview. Multiple threads process the pipeline stages where work is available to dynamically distribute a VEs workload to all CPU cores.

1. Pipeline Virtual Environment Architecture

We have created a modular pipeline VE architecture designed for multi-core CPU systems. The architecture allows VEs to be modeled, and processes a VE's workload in parallel. A major difficulty with many parallel processing development tools is that developers must typically deal with the details on how a problem is broken down, processed in parallel, and assembled to obtain a final result. Our approach hides the underlying parallel processing from developers so they concentrate on developing a VE. A VE is developed by modeling objects and interactions between the objects. A modularized 3D object is provided to model the objects of a VE. Different objects are created by specifying geometrical representations and visual properties, as well as algorithms for graphics rendering and physical behavior. An "object interaction pair" is used to model pair-wise interactions between objects by specifying the collision detection and response algorithms for an interaction.

A multi-stage VE simulation and rendering pipeline processes a modeled VE, Figure 1. The *Collision detection* and *Collision response* stages trigger the algorithms specified in an interaction pair to simulate object interactions. *Object update* updates objects based on its modeled physical behavior and interaction data (e.g., forces). *Object render* generates images from the VE to display to the user. The graphics rendering and physical behavior algorithms specified for an object serve as callbacks for the object update and rendering stages. Queues serve as I/O between pipeline stages and allow work to be passed through the pipeline for processing.

To fully utilize the CPU resources, the architecture employs multiple threads to execute the pipeline. The number of threads used is based on the number of CPU cores. An application can also set the number of threads. All pipeline stages are designed to run in parallel; however, we found current GPUs do not fully support multi-threaded rendering. Thus, a token system is used so only one thread enters the rendering stage at a given time. It is difficult to create an efficient thread scheduler that balances the workload across the threads. Instead, the pipeline threads are designed to migrate to pipeline stages where work is available. This design automatically handles work-load distribution across the pipeline threads (and CPU cores). A VE is modeled once and the architecture automatically scales its performance based on the available CPU cores.

2. Results

A Virtual Reality-based training simulator is being developed (with the described architecture) to practice the skills required to perform a craniotomy. Figure 2 shows a haptic

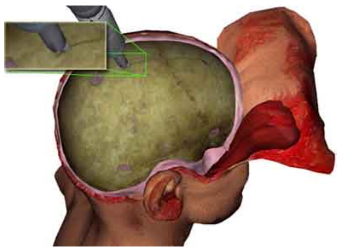

Table 1. Test results for craniotomy simulator.

Threads (p)	FPS	Speedup (FPS_p/FPS_1)	Efficiency $(Speedup/p)$
1	28	1.0	1.0
2	58	2.07	1.04
3	86	3.07	1.02
4	102	3.64	0.91

Figure 2. (Left) Craniotome cutting skull flap for a craniotomy simulator being developed and tested on the pipeline VE architecture. (Right) Test results for craniotomy simulator.

device-controlled craniotome cutting a skull flap from a virtual patient. Voxel-based algorithms are used to simulate tool-bone interactions and texture-based volume rendering displays the virtual bone material [4]. Fifteen anatomical structures are currently integrated into the VE (11 are real-time deformable mass-spring based objects). Collision detection is based on a GPU-based algorithm [5]. The test system is a 3.0 GHz Intel Core2 Extreme quad-core processor, 2 GB RAM, a NVidia GeForce 9800 GX2 (512 MB) graphics card in SLI mode and Windows XP. Table 1 provides the performance results. The VE frames/second (FPS) speedup is more that 1.75 times as the thread count doubles. Super-linear speedups (efficiency > 1.0) were seen for both 2 and 3 threads. Using 4 threads gave a near-linear speedup with a 91% thread efficiency.

3. Conclusion

The described pipeline VE architecture makes it possible to build and simulate complex physically-based VEs. We are in process of developing medical VEs for several simulators. The architecture is very modular and provides hooks for integrating and using different algorithms. This makes it possible to take independently developed algorithms and utilize them in a parallel environment. The ability to leverage all the CPU resources of multi-core CPUs will make it possible to push the technical boundaries and increase the realism of physically-based medical VEs.

References

[1] J. Allard, S. Cotin, F. Faure, P-J Bensoussan, F. Poyer, C. Duriez, H. Delingette, and L. Grisoni. SOFA - an open source framework for medical simulation. *Stud Health Technol Inform*, (2007).

[2] M. C. Cavusoglu, T. G. Goktekin, F. Tendick, and S. S. Sastry. GiPSi: An open source/open architecture software development framework for surgical simulation. *Stud Health Technol Inform*, 46-48, (2004).

[3] K. Montgomery and C. Bruyns, J. Brown, G. Thonier, A. Tellier, and JC Latombe. Spring: A general framework for collaborative, real-Time surgical simulation. *Stud Health Technol Inform*, (2002).

[4] E. Acosta and A. Liu. Real-time volumetric haptic and visual burrhole simulation. *IEEE Virtual Reality*, pages 59-66, (2006).

[5] N. K. govindaraju, M. C. Lin, and D. Manocha. Quick-CULLIDE: Fast Inter- and Intra-Object Collision Culling Using Graphics Hardware. *Virtual Reality*, 247-250, (2007).