HARDWARE/SOFTWARE CO-DESIGN FOR REAL-TIME PHYSICAL MODELING

B. Bishop, T. P. Kelliher, M. J. Irwin

Department of Computer Science and Engineering
The Pennsylvania State University
University Park, PA 16802 USA

ABSTRACT

Physical modeling of a mass-spring system allows for realistic object motion and deformation in a virtual environment. Previous work in this type of physical modeling relies on general-purpose hardware, and cannot offer the performance necessary for real-time human-machine interaction. In this paper, we consider the co-design of software and hardware in order to achieve real-time physical modeling.

1. Previous Work

1.1 Software/Algorithms

There are multiple approaches to physical modeling, each having its own merits for different types of simulations. Rigid body techniques are computationally efficient, but can only model stiff objects. Mass-spring simulation is useful for modeling many different types of objects, but has been too computationally intensive for real-time use. Additional techniques exist for modeling liquids and gasses.

In rigid body simulation[1][2], the computation can be simplified because the objects are not allowed to deform. Inertial information can be precomputed and then used to determine object motion upon collision. Collision detection still represents a significant computational challenge, however there are techniques to ease this burden (see below).

In a mass-spring simulation[3][4], the objects are represented as a set of mass points connected by springs. This representation is useful for modeling many different materials. For example, a marshmallow can be modeled by using highly damped springs. Other materials could include Jell-O, cloth, metal or stone.

Stiff materials require stiff springs. In order to ensure stability, advanced numerical methods can be applied, or alternately large numbers of simulation steps can be performed using simpler techniques.

Collision detection can be very computationally intensive. However, it is possible to make use of the fact that collisions are fairly rare. A great deal of computation can be saved by ruling out pairs of objects that cannot possibly collide. This “pruning” can be accomplished through bounding box checking.

One approach for bounding box checks is to check every object bounding box against every other object bounding box. For **N** objects this gives $\mathbf{N}^2$ checks. However, there are techniques[5] that reduce this number.

Once a possible collision has been detected between two objects using bounding boxes additional work is needed. One popular approach is to check if any points from one object are contained in the other object. If these objects are convex, this test can be performed very efficiently, since the test must determine only if the point is “under” each plane defining the object. This technique does not work for mass-spring simulations, as it is difficult to guarantee that convex objects remain convex as deformations occur.

1.2 Hardware/Architecture

This section briefly sketches current and emerging hardware/architectural support for physical modeling.

Both AMD and Intel have recognized the value of high-performance floating point for interactive 3D applications. This is in the form of 3Dnow![6] and KNI[7] respectively.

These techniques are general-purpose processor extensions, which increase FLOPS through the use of specialized high-bandwidth floating-point units.

Some 3D accelerators[9] are starting to include hardware support for transform and lighting. This hardware acceleration for object transformation could be useful in rigid body simulations, however a significant penalty exists for transferring object data back to the CPU for collision detection.

2. Hardware/Software Co-design

We achieved a high degree of synergy during the system’s software and hardware design. The design effort began with writing a demonstration computer graphics application. From this we synthesized an instruction set architecture and designed a pipeline supporting the floating point and collision detection computations. The complexity of the resulting pipeline caused us to reorganize both hardware and software for the proof-of-concept implementation.

We discuss further design considerations in the remainder of this section.

2.1 Software Considerations

We have chosen to implement a real-time mass-spring simulation. Each step of this simulation can be broken down into the following sub-steps:

- Spring force computation
- Collision detection pruning
- Collision detection

In spring force computation, velocity increments are generated for each mass point as a result of the connected springs acting on it. These springs respond to any displacement from their ideal length according to Hookes law [11].

In collision detection pruning we try to reduce the number of object pairs that must be considered for collision detection.

Since we are working with a mass-spring system, the approach for collision detection described above cannot be used. Instead, we have developed a new approach. In our approach, when an object is defined, each point is associated with a line segment in the object (see fig. 1 for a graphical description). That line segment is then checked against the faces of the other object for intersections. If there is an intersection, a collision has occurred.

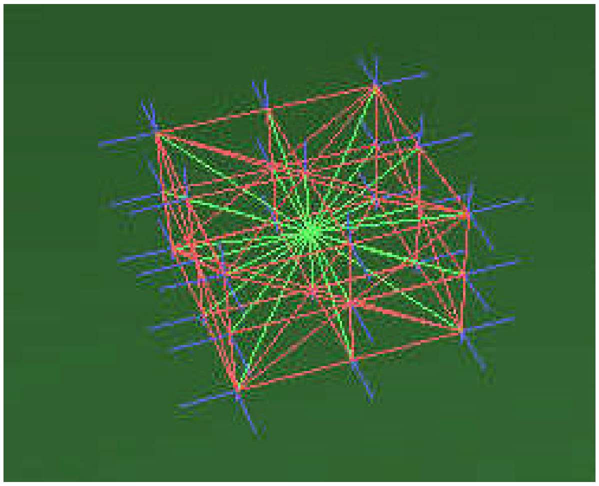

Fig. 1: Debug representation of cube object

2.2 Hardware Considerations

The unique features of this system are that we must process a large amount of data, and the computation is the same on each simulation step. Since the computation process does not change, we can consider a pipeline organization. This organization allows us to effectively address the other feature of this system – large amounts of data. In a pipeline organization, memory bandwidth is more efficiently used since intermediate values can be passed between pipeline stages without returning to memory. Additionally, the pipeline organization allows for very high utilization of processing capability.

2.3 “Coping” with the Hardware

This pipelined organization introduces some problems that must be addressed in software, however. The main problems with the static pipeline organization are data hazards[8] and lack of conditional execution.

Using the algorithm outlined in 2.1, each data set in the pipeline is very independent. However, with collision detection for example, only the maximum velocity increment for a given point among all data sets is considered. This results in data hazards when updating the maximum velocity increment. This hazard can be addressed by slight modification of the pipeline to allow for operand forwarding.

Another potential problem is the lack of conditional execution, which is necessary in the collision detection pruning phase of computation. This problem can be solved by pruning on the host CPU or through the use of a simple embedded processor since pruning is not generally as computationally intense as the other steps.

2.4 Dealing with Instability

One of the biggest problems in a mass-spring simulation is in achieving the necessary degree of stability. Since we perform collision detection as well as spring force computation on every iteration, it is not efficient to simply increase the number of iterations for the desired degree of stability.

One possible solution to this problem is to perform multiple spring force computations per collision detection computation. This is computationally efficient since spring force computation is relatively simple. However, collision detection fails since objects can slowly drift into one another during the spring force computation steps. This can be addressed by increasing the velocity increment for resolving collisions. However, increasing this increment leads to unrealistic vibration-like behavior.

One additional solution is through the use of techniques for breaking and bending of springs. When a velocity increment exceeds a certain threshold, the spring stiffness can be set to zero (breaking) or the ideal spring length can be modified (bending). The “plate demo” at [10] is an example of breakable springs.

3. Proof-of-concept – SPARTA

The SPARTA project (Simulation of Physics on a Real-Time Architecture) is an effort to develop a hardware/software experimental system for real-time physical modeling. Please see [10] for additional information about the project including source code, executables, and mpeg movies. The project has been split into the following stages of increasing complexity:

3.1 General-purpose CPU Implementation

This is an implementation of the algorithms outlined in 2.1 running under Linux and Windows. This implementation is meant to verify algorithm correctness. With heavy optimization, near real-time performance can be achieved only for very simple scenes. All of the images and movies from [10] were generated in this implementation. Figures 2-5 show one such animation sequence.

3.2 Compiled HDL Implementation

The goal of this implementation is to quickly verify the correctness of the pipeline organization. A hardware description language simulation is ideal, since changes can be made easily and verification can be accomplished through simulation. Since the pipeline design will be rather large and complex, a compiled HDL simulation would best ensure fast simulation.

3.3 FPGA Implementation

The goal of the Field Programmable Gate Array implementation is to develop a working prototype, which can at least partially accelerate the software beyond what is possible in 3.1.

The dominant concern in this implementation is density. The computational pipeline is very floating point intensive. Floating point units are difficult to implement in FPGAs due to routing overhead in barrel shifters. A number of approaches will be used to minimize this density problem, including reducing floating point precision and implementing only a partial pipeline. For example, the pipeline could perform only spring force computation with collision detection performed on the system CPU.

3.4 ASIC Implementation

The Application Specific Integrated Circuit implementation is the eventual goal of the SPARTA project. This implementation should be able to achieve tremendous speedups over 3.1. These speedups will be a result of efficient pipeline and memory organization in conjunction with the fast clock rates that are possible in ASICs.

4. Summary

We have described a novel real-time system for physical modeling, and have contrasted it with previous work. We have discussed the unique features of this experimental system from both a hardware and software perspective. Considering each of these perspectives, we present a codesign that is optimized for real-time physical modeling.

5. REFERENCES

[1] W. Armstrong, M. Green, ``The dynamics of articulated rigid bodies for purposes of animation'', *The visual computer*, Springer-Verlag, 1985.

[2] D. Baraff, ``Fast Contact Force Computation for Nonpenetrating Rigid Bodies'', *Computer Graphics Proceedings*, July 1994.

[3] D. Baraff, A. Witkin, ``Large Steps in Cloth Simulation'', *Computer Graphics Proceedings*, July 1998.

[4] D. Terzopoulos, J. Platt, A. Barr, K. Fleischer, ``Elastically Deformable Models'', *Computer Graphics*, Vol. 21, No. 4, July 1987.

[5] M.Lin, S. Gottschalk, ``Collision Detection between Geometric Models: A Survey'', *Proc. IMA Conference on Mathematics and Surfaces*, 1998.

[6] Inside 3DNow! Technology *http://www.amd.com/products/cpg/k623d/inside3d.html*

[7] Discover the New PentiumIII Processor *http://developer.intel.com/design/PentiumIII/prodbref/*

[8] J. Hennessy, D. Patterson, “Computer Architecture: A Quantitative Approach”, Morgan Kaufmann Publishers, San Mateo, CA.

[9] NVIDIA GeForce256 http://www.nvidia.com/Geforce256.nsf

[10] The SPARTA project http://www.cse.psu.edu/~mdl/sparta/

[11] D. Halliday, R. Resnick, J. Walker, *Fundamentals of Physics*, John Wiley and Sons, 1997.

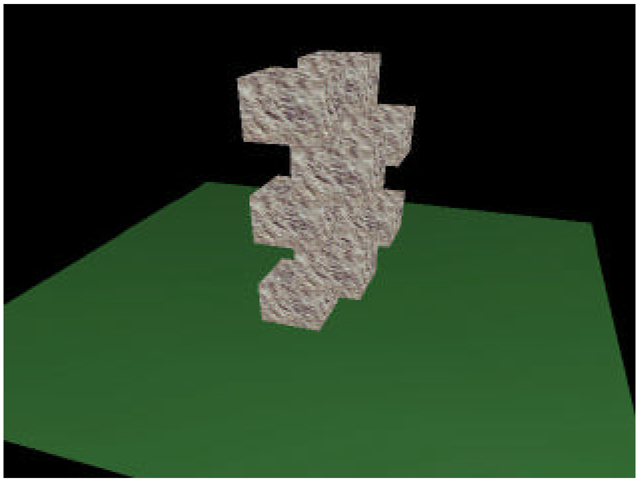

Fig. 2: Multiple Block Animation (frame 1)

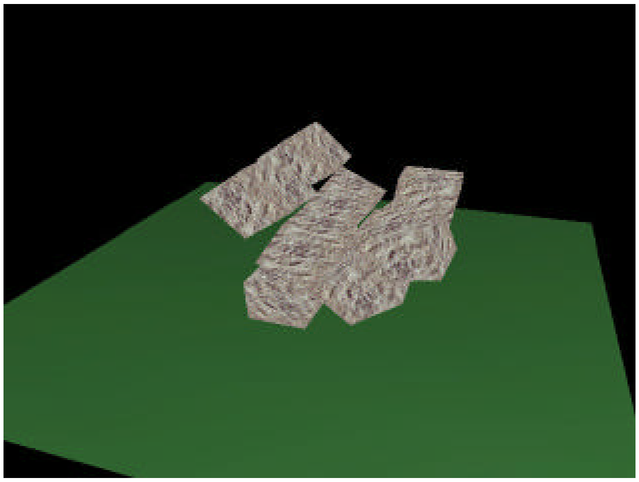

Fig. 4: Multiple Block Animation (frame 200)

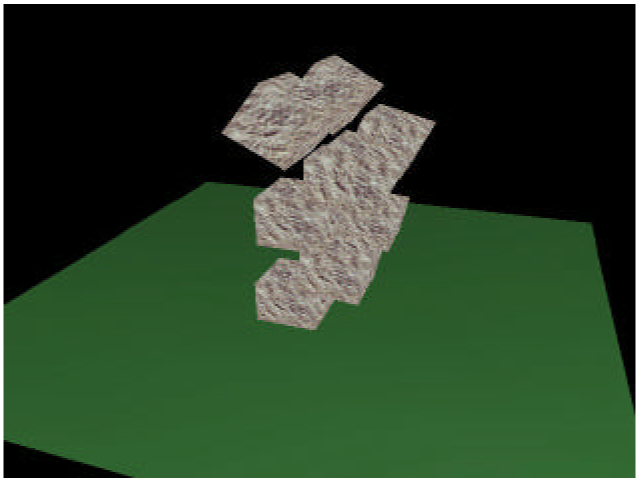

Fig. 3: Multiple Block Animation (frame 100)

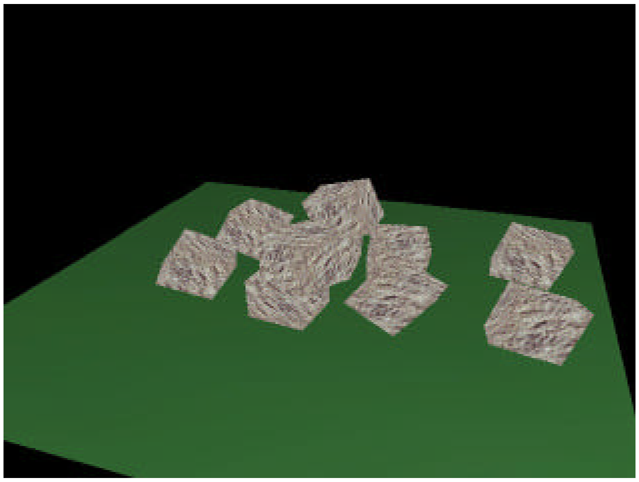

Fig. 5: Multiple Block Animation (frame 300)