Title: SPARTA: Simulation of Physics on a Real-Time Architecture
Category: 1

Contact Author
Benjamin Bishop
202 Pond Lab
University Park, PA 16802
(814)863-7325 (voice)
(814)865-3176 (fax)
bishop@cse.psu.edu
Computer Science and Engineering - Pennsylvania State University

Coauthor
Mary Jane Irwin
202 Pond Lab
University Park, PA 16802
(814)865-1802 (voice)
(814)865-3176 (fax)
mji@cse.psu.edu
Computer Science and Engineering - Pennsylvania State University

Coauthor
Thomas P. Kelliher
Hoffberger 140
1021 Dulaney Valley Road
Baltimore, MD 21204
(410)337-6189 (voice)
(410)337-6408 (fax)
kelliher@bluebird.goucher.edu
Mathematics and Computer Science - Goucher College

SPARTA: Simulation of Physics on a Real-Time Architecture

Abstract - In this paper, we discuss hardware acceleration for real-time physical modeling that would allow for realistic virtual environments. Additionally, we propose algorithms and their architectural implementation (SPARTA), which is specifically tuned for real-time use. We expect performance orders of magnitude higher than general-purpose CPUs.

1 Introduction

The goal of the SPARTA project is to accelerate physical modeling of solid objects through the use of specialized hardware in order to achieve real-time performance. Physical modeling is necessary for creating realistic virtual environments. Using physical modeling, it is possible to simulate realistic object collision and deformation. Currently, much of the work in physical modeling is targeted at the film industry. Real-time physical modeling hardware would enable many additional applications such as Virtual Reality. Physical modeling hardware that could operate at much faster than real-time could also be useful in robotic motion planning. Such a system could plan movements by considering large numbers of possible motions looking for some desired outcome (similar to the way Deep Blue plays chess [1]).

2 Background

A wide range of techniques exist for physical modeling of solid objects [2, 3, 4, 5, 6]. Rigid body techniques [2, 5] tend to be computationally efficient on general purpose CPUs. Since the object is not allowed to deform, collision detection is the only significant computational challenge. The disadvantages of this approach include collision detection performance and inability to model object deformations.

More general techniques [4, 6] often represent the object as a collection of mass points connected by springs. This approach can yield very visually appealing results, but spring force computation can be problematic. The problem is that spring force computations can become unstable. Previous work [4] advocated the use of complex numerical methods [7] in order to reduce the number of steps required for spring force computation. The disadvantages of this approach include spring force computation performance and collision detection performance.

3 The SPARTA project

Currently, we are developing an FPGA-based proof-of-concept coprocessor accelerator board. This section gives details about the algorithms in use. We also discuss the advantages of SPARTA over general purpose CPUs and address the shortcomings of an FPGA-based approach.

3.1 Algorithms

We have selected the Mass-Spring model for use in SPARTA. Mass-Spring systems allow for modeling of many materials (jello, metal, cloth, stone, etc.). In addition, large numbers of simple spring force computations can be performed efficiently in hardware. Note that this "many simple steps" method is a different approach then previous work. We expect that this method will be easier to implement in hardware.

Overview
The type of computations performed in the Mass-Spring model can be broken down into two types: spring force computation and collision detection/resolution. These two types have very different computational requirements. Collision detection/resolution is very computationally intensive. In spite of large speedups due to early exiting, our initial simulations on an x86 show that collision detection/resolution is about 30 times slower than a single spring force computation. The following pseudo-code is a simple high-level outline of the algorithms used:

```
/* Spring step */
for every object
  for every spring
    find force due to displacement from resting length
    increment velocity of both points
    find force due to damping
    increment velocity of both points
/* Collision detection/resolution step*/
for every pair of objects
  if bounding boxes intersect
    for every face in object1 and line segment in object2
      if bounding boxes intersect
        find face-line intersection
        if intersection is on the line segment and intersection is on face
          fix velocities
```

Collision detection/resolution
One of the main problems in collision detection is the computational cost of checking every object against every other object for collisions (N^2 object checks). However, this problem can be solved efficiently [8] for larger Ns. After the initial

check, only the objects which have overlapping bounding boxes must be checked for collisions.

In the most simple form of low-level collision detection, points of one object are checked to see if they lie within another object. This method can be implemented very efficiently if all objects are assumed to be convex. Each point of the first object is checked against all planes formed by faces of the second object. If the point is "below" all planes, then a collision has occurred. This scheme is not feasible for SPARTA, since it is not possible to guarantee that deformable objects are convex.

For collision detection in SPARTA, we define line segments which are internal to each object. Our algorithm simply finds the point of intersection between line segments of one object and faces of the other object. It then does a bounding check to determine if the point of intersection is on the line segment and on the face.

Collision resolution is achieved by averaging velocity of the nearest line segment point and face points along the face normal. Friction can be modeled by modifying the velocity not along the face normal. Using the average velocity of the face to approximate the face velocity at the collision point can cause problems for sparse models.

Spring force computation

The goal of spring force computation is to determine the velocity increments (instantaneous forces) on points due to spring compression/ decompression. Spring force [9] can be computed as in the following equation:

$$F = -kd - bv$$

where F is the spring force, k is the spring constant (stiffness), d is the displacement, b is the damping constant, and v is the relative velocity along the spring. The bending of metal can be modeled by adjusting the ideal resting length of the spring. Breakable objects can be modeled by reseting stiffness and damping parameters if a spring exceeds a certain length.

The main problem in spring force computation is instability. Instability is due to the use of a discrete model to approximate a continuous system. Instability occurs when the velocity increment at each step overcompensates for spring compression/decompression at the previous step. This instability can be controlled by adjusting the step size and object stiffness.

3.2 CPU-based Implementation

We have developed a CPU-based physical modeling application based on the algorithms discussed above. Figures 3-6 show a physical modeling simulation of stiff and deformable blocks. Figures 7-10 show a demonstration of friction modeling. Figures 11-14 show modeling of a broken plate using breakable inter-object springs. In each simulation four snapshots were taken from 400 simulated frames. Source code and MPEG demo movies for the CPU-based implementation can be found at `http://www.cse.psu.edu/~mdl/sparta`.

3.3 Ideal SPARTA Implementation

The philosophy of the SPARTA architecture is to translate the high-level functions required for spring force computation and collision detection/resolution into hardware. Since the same complex operations are being performed many times, a pipelined organization is ideal. Figure 1 shows a block diagram of the Ideal SPARTA Implementation. This is a single chip ASIC implementation. Separate specialized pipelines are used for spring force computation and collision detection/resolution. An additional unit is required to perform object bounding box checking. A specialized, small, on-chip, high speed, high bandwidth SRAM memory is used to feed the pipelines. The control logic controls the pipeline operation and interfaces to the host CPU.

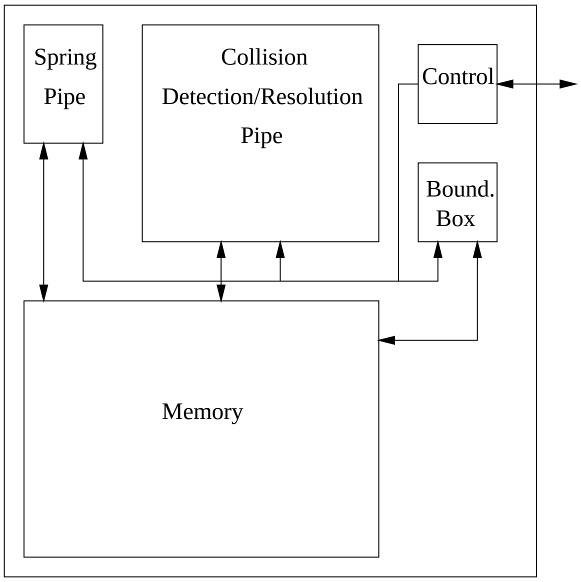

Figure 1: Ideal SPARTA Implementation

3.4 Why SPARTA over general purpose processors?

3dnow! and KNI [10, 11] are enhancements to general-purpose CPUs by AMD and Intel respectively. These enhancements offer more FLOPS through the use of SIMD floating point units. This subsection presents an argument as to why SPARTA would greatly exceed the performance offered by these enhancements.

1. Memory Organization: SPARTA only needs to store a small amount of information per object. This includes point positions, point velocities, spring information, and face information. This information could fit onto an on-chip SRAM, allowing for very high speed/high bandwidth access. We expect the SPARTA collision detection/resolution pipeline to require a large databus to on-chip memory. Additionally, intermediate values would be passed through the pipeline eliminating the need to store them in memory (as in general purpose processors).

2. Area Utilization: In a general-purpose processor, chip area is not well utilized during physical modeling computations. There is a great deal of area devoted to integer computation, instruction decoding, branching, etc.

In an ideal SPARTA implementation, the entire spring force computation and collision detection/resolution processes could be pipelined. The whole chip could be devoted to floating point units and memory.

3. Bus Bottleneck: In a general-purpose CPU, a bottleneck exists between the CPU and graphics hardware. SPARTA could avoid this bottleneck since it could be integrated directly with the graphics hardware.

4. Time Utilization: General-purpose CPUs cannot dedicate all of their resources to physical modeling. The CPU must also deal with operating system overhead, input processing, sound processing, etc.

5. Flexible Precision: The floating point hardware on a general-purpose CPU must usually be at least IEEE single precision. In SPARTA, it may be possible to reduce this precision in order to speed computation.

6. Implementation Risk/Time: General purpose CPUs are much more complex than the SPARTA design. The simplicity of the SPARTA design could lead to low-risk implementations with fast time-to-market.

3.5 Proof-of-concept Implementation

In order to show that physical modeling performance improvements are possible through the use of specialized hardware, a proof-of-concept implementation of SPARTA is being constructed. For this implementation, an Altera EPF10K250 FPGA [12] is being used in conjunction with a custom board and specialized memory. An FPGA implementation was selected in order to minimize design time. An FPGA-based implementation presents some problems, however. It is very difficult to implement barrel shifters (due to FPGA routing problems), which leads to high area and low performance for normalization steps in floating point operations. Additionally, it is impossible to implement the full collision detection/resolution and spring force calculation pipelines in hardware due to area constraints on the FPGA.

For the FPGA implementation, it is difficult to address the problem of inefficient floating point units, however a precision reduction may help to reduce performance/area losses. As for the problem of FPGA density, we plan to implement only the spring force computation pipeline (collision/detection resolution will be handled by the host CPU or an on-board coprocessor). The spring force computation pipeline was chosen since it is smaller. Figure 2 shows a block diagram of the planned FPGA organization.

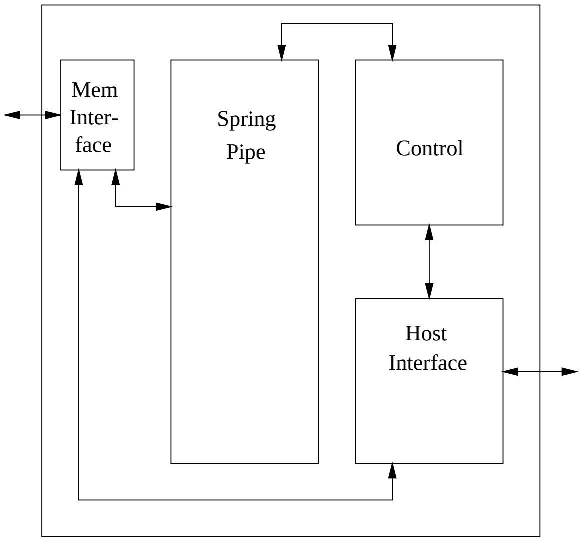

Figure 2: FPGA SPARTA Implementation

4 Future Work

The eventual goal of the SPARTA project is to develop a physical modeling ASIC, which is orders of magnitude faster than current processors. We would like to begin work on the ASIC implementation as soon as the FPGA implementation is realized.

Additionally, we are developing a public-domain physical modeling package which is based on the SPARTA algorithms.

References

[1] C. Tan, F. Hsu, M. Campbell, J. Hoane, G. Brody, "Deep Blue"
http://www.research.ibm.com/deepblue
http://www.research.ibm.com/resources/magazine/1996/issue_1/news196.html

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

[3] A. Witkin, W. Welch, "Fast Animation and Control of Nonrigid Structures", *Computer Graphics*, Vol. 24, No. 4, August 1990.

[4] D. Baraff, A. Witkin, "Large Steps in Cloth Simulation", *Computer Graphics Proceedings*, July 1998.

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

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

[7] W. Press, B. Flannery, S. Teukolsky, W. Vetterling, *Numerical Recipes*, Cambridge University Press, 1986.

[8] M.lin, S. Gottschalk, "Collision Detection between Geometric Models: A Survey", *Proc. IMA Conference on Mathematics and Surfaces*, 1998.

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

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

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

[12] "FLEX 10K Device Family" *http://www.altera.com/html/products/f10k.html*

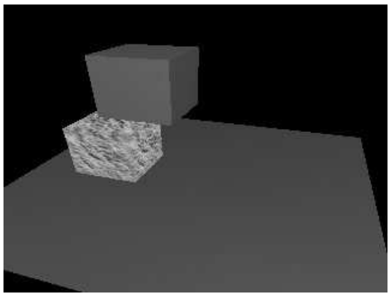

Figure 3: Jello - Frame 1

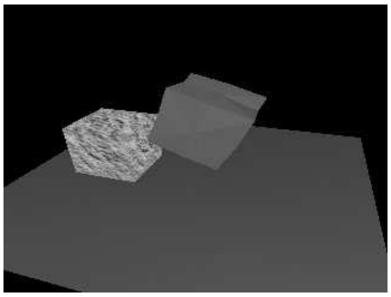

Figure 4: Jello - Frame 2

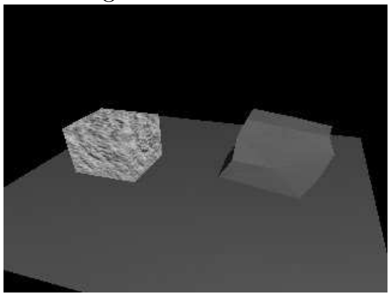

Figure 5: Jello - Frame 3

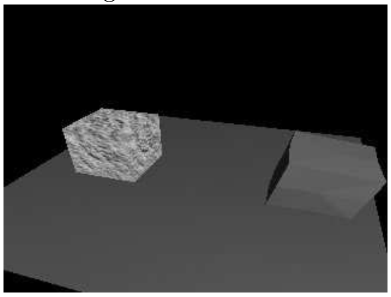

Figure 6: Jello - Frame 4

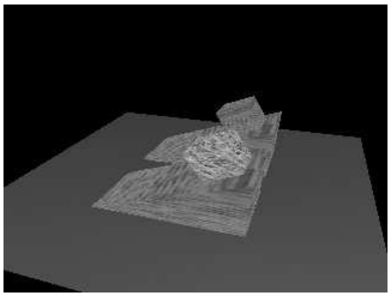

Figure 7: Friction - Frame 1

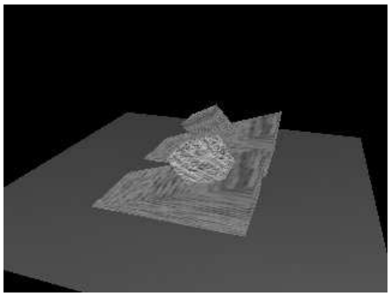

Figure 8: Friction - Frame 2

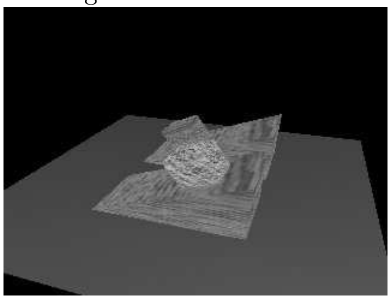

Figure 9: Friction - Frame 3

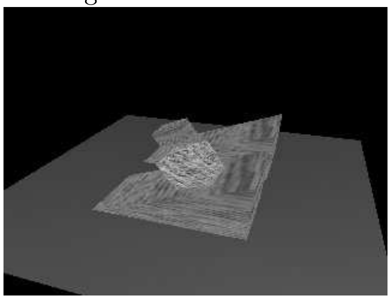

Figure 10: Friction - Frame 4

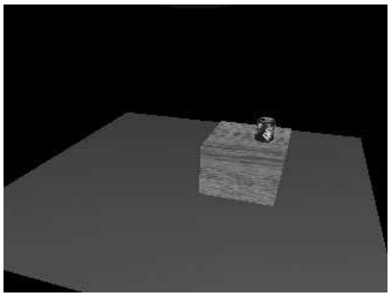

Figure 11: Plate - Frame 1

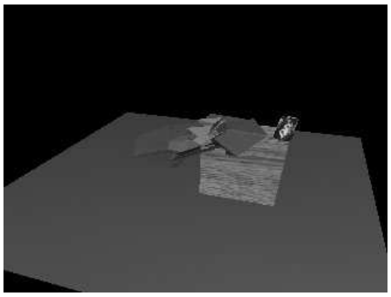

Figure 12: Plate - Frame 2

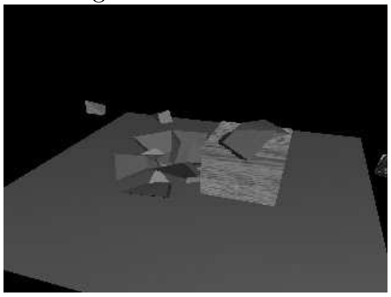

Figure 13: Plate - Frame 3

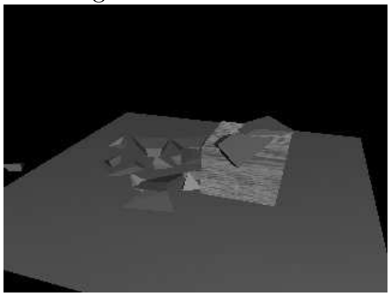

Figure 14: Plate - Frame 4