GPU-Based Rigid Body Dynamics
A Thesis by Severin Strobl

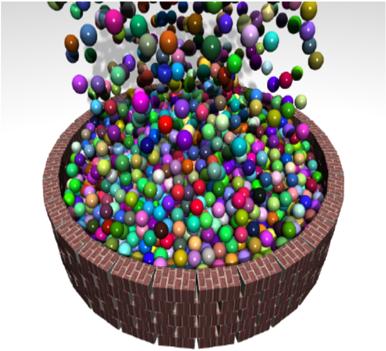
Jeremy Betz

RPI Computer Science

March 1, 2011

Introduction

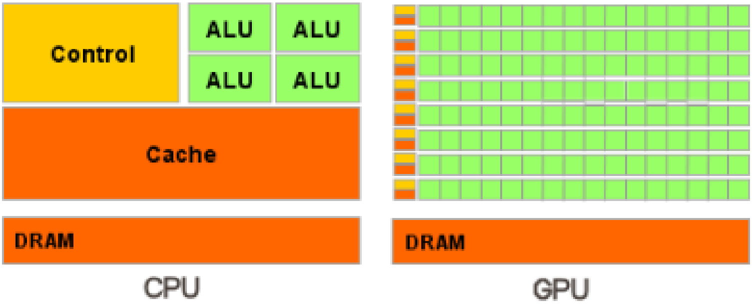
- GPU Architecture presents unique challenges
- Part of the simulator is CPU, part GPU
  - Presents memory sync issues



Why use the GPU?

- The CPU is a multipurpose device designed for minimal latency for a single thread.
  - Large cache on chip
- The GPU is a specialized device designed for maximum throughput
  - Multiple threads hides latency (Smaller caches)
- Reduced cache allows more transistor space for higher arithmetic density

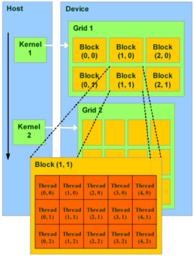
Architecture Comparison



Reduced cache allows more transistor space for higher arithmetic density

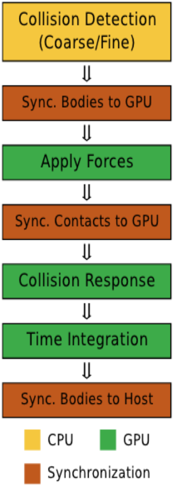
GPU Basics

- Threads are grouped into blocks that work together
- Blocks are grouped into kernels and all run the same program
- Kernels can run different programs
  - Usually only 1-2 kernels per card



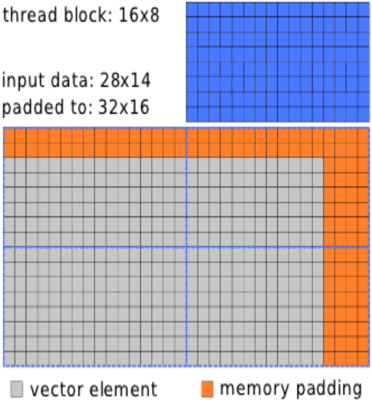
Integration of Solver

- Most components ported to GPU
- Some components still run on CPU
    - Memory synchronization between systems hurts performance



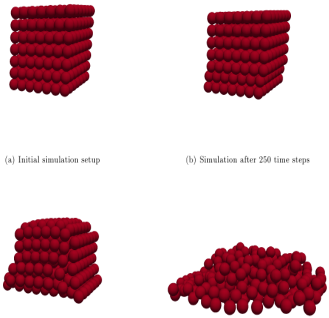
Data Structures

- Memory reads need to be aligned for maximum efficiency
  - 2D and 3D memory mapped to linear memory
  - Alignment to 16 words of 32bit
- Page locked memory used on host
- Data structures used on host and GPU



Collision Response

- Simple dynamics simulator *ballpark* ported to GPU
  - Only supports spheres
  - Uses LCP
  - Not very parallel



(a) Initial simulation setup

(b) Simulation after 250 time steps

(c) Simulation after 500 time steps

(d) Simulation after 1 000 time steps

LCP Solver Types

- Originally *ballpark* used Gauss-Sidel
  - Stable for convergence
  - Can't be run in parallel
- Jacobi method is similar to Gauss-Sidel
  - Less convergence
  - Computations for each element can be done in parallel

Algorithm

Choose an initial guess x^0 to the solution

while convergence not reached do

for i := 1 step until n do

$\sigma = 0$

for j := 1 step until n do

if j != i then

$$\sigma = \sigma + a_{ij} x_j^{(k-1)}$$

end if

end (j-loop)

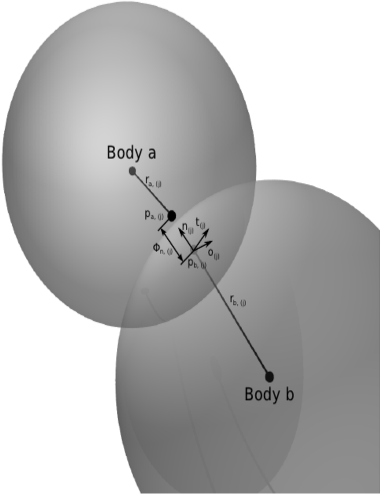
$$x_i^{(k)} = \frac{(b_i - \sigma)}{a_{ii}}$$

end (i-loop)

check if convergence is reached

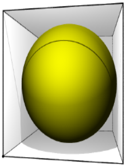
end (while convergence condition not reached loop)

- Linear and Angular movement seperated into two kernels
    - CUDA compiler has difficulty reusing registers
    - Operations are completely different
- Bounding boxes also updated during this step

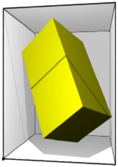


Bounding Boxes

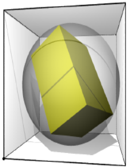
- Implemented for objects other than spheres
- Less optimal than CPU implementation
- Allows all threads to run same operation



(a) Axis aligned bounding box (AABB) of a sphere



(b) Optimal AABB of a box



(c) Wrapping bounding sphere of a box with the AABB of the sphere

Applying Forces

- Simple, easily parallel
- Each body's forces are independent, update step natural for GPU

Conclusion

- Time Integration ports well
- Collision Response poses many problems
- Doesn't give specific comparisons in paper