Introduction

» Based on my research at the university of Tokyo
  Not at havok

» The details can be found in my publications
  Takahiro Harada, “Real-time Rigid Body Simulation on GPUs”, GPU Gems 3
  Takahiro Harada, Issei Masaie, Seiichi Koshizuka, Yoichiro Kawaguchi, Massive Particles: Particle-based Simulations on Multiple GPUs, SIGGRAPH 2008 Talk
etc...
  http://www.iii.u-tokyo.ac.jp/~takahiroharada/

Physics Simulation

» Physics simulation is highly parallel
» Grid-based fluid simulation is well mapped on the GPU
» How about rigid bodies?
  No general solution yet
  Simplified approach
  Takahiro Harada, “Real-time Rigid Body Simulation on GPUs”, GPU Gems 3

GPU

» GPU is designed for graphics
» GPU is good at
  Many similar computations
  Simple computations
  Not complicated computations
  All the thread taking the same path is ideal
» Ex. particle simulation without interaction

Particle-based Simulation

» Smoothed Particle Hydrodynamics
  Compressible fluids
SPH Simulation

» Overview
  » For each particle
    » Look for neighboring particles
  » For each particle
    » Calculate pressure from neighbors
  » For each particle
    » Force on a particle is calculated using values of neighbors
  » Integrate velocity and position

» Problem is neighbor search
  » Use uniform grid to accomplish this
  » Discuss later

Rigid Body Simulation using Particles

» Extension to particle based simulation
» Use particles to calculate collision
» Rigid body is represented by particles
  » Not accurate shape
  » Trade off between accuracy and computation
  » Simple, uniform computations -> Good for GPUs

Data Structure

» For each rigid body
  » Positions
  » Quaternion
  » Linear momentum
  » Angular momentum

» For each particle
  » Position
  » Velocity
  » Force
  » For neighbor search
    » 3D grid

Grid Construction

» Storing particle indices to 3D grid
» Can limit the number of particle in a cell if particles does not penetrate
» Each thread read particle position, write the index to the cell location
» But this fails when several particles are in the same cell
  » Divide this into several pass
  » 1 index is written in a pass
  » Repeat n times (max number of particles)

Demo
Extension

- If there are more than particles
  - Particles + Mesh (cloth)
- Can solve using several grids
  - A grid for particle
  - A grid for mesh
- Still not general

Broadphase Collision Detection

- Uniform grid is suited for the GPU
  - But not good for objects of not the same sizes
- Other approaches?
  - Sweep and prune
  - Tree
- Good for objects varying sizes
  - Much complicated than uniform grid
  - Can implement and accelerate on the GPU?

Tree traversal on the GPU

- Well studied in the field of ray tracing
  - Octree
  - Kd tree
- 2 problems when using for a real-time rigid body simulation
  - Dynamic construction of the tree
    - Several studies but few of them can beat the CPU
  - Traversal
    - Packet based for ray tracing -> cannot use this for collision detection
    - What is good for collision detection?

Dynamic Construction of Tree

- Tree construction is recursive subdivision of inputs -> not good for GPUs
- Convert the problem to a sorting problem
  - Calculate morton key of objects
  - Sort them
  - Add child-parent information to the sorted list
  - Lauterbach et al., Fast BVH Construction on GPUs, Eurographics 2009

Tree Traversal

- Using stack is most common
  - But the requirement of resources is too much -> kill the performance
- Stackless traversal with additional info
  - Dynamic update?
  - High overhead
- Restart
  - Cannot restart because we want the overlap of bounding boxes (maybe can truncate BB...)

Tree Traversal using History Flags

- Observation
  - Descending a tree does not need any information
  - Start from first element of children
  - Ascending a tree needs where to get back
- Instead of stacking node indices, stores the history of traversal
- Data can be small
**Tree Traversal using History Flags**

- For each level, store 4 bits
- Initialize 0000
- After visiting a node, flip the flag 1000
- Descending to the next level
  Just leave the flag and do the same to the next level
- Visiting the next element
  Find "0" in the history flag
- Ascending the tree
  When cannot find "0", ascend

**Example:**

```
0000
0000
1000
1000
```

**Ascending the tree**

When cannot find "0", ascend
Tree Traversal using History Flags

- For each level, store 4 bits
- Initialize 0000
- After visiting a node, flip the flag
- 1000
- Descending to the next level
  - Just leave the flag and do the same to the next level
- Visiting the next element
  - Find "0" in the history flag
- Ascending the tree
  - When cannot find "0", ascend

Discarding the flags of the level because they are used when descending to this level again

7 level octree traversal only requires 4bit x 7level = 28bit

Can use shared memory for the storage of history flag -> fast access
**Consideration**

- Can implement tree construction and traversal on the GPU
  
  *If compare this to best solution on the CPU?*

- Octree is not the best solution on the CPU

- Kd tree on the GPU is also studied

- But the CPU is better
  
  
  - Zhou et al., "Real-Time KD-Tree Construction on Graphics Hardware", SIGGRAPH Asia 2008

**Solving Constraint**

- Usually, constraints are solved for velocity

- Penalty based
  
  *No problem for parallel computation*
  
  *Input: position, output: force*

- Impulse based
  
  *Problem when parallelizing*
  
  *Input: velocity, output: velocity*

  *How to parallelize on the GPU?*

**Problem of Parallel Update**

- If a rigid body is colliding to another rigid body, no problem

- If a rigid body is colliding to several rigid bodies, cannot update in parallel

**Batching**

- Not update everything at the same time

- Divide them into several batches

- Update batches in sequential

  - Update collisions in a batch in parallel

- But how to divide into batches?? GPU??

**Batch Creation on GPU**

- CPU can do this easily

- Chen et al., High-Performance Physical Simulation on Next-Generation Architecture with Many Cores, Intel Technology Journal, volume 11 issue 04

- To implement on the GPU, the computation has to be parallel

- Do it by partially serialize the computation

  - Synchronization of several threads, which is available on CUDA, OpenCL

**Batch Creation**

- A thread is assigned for a constraint

<table>
<thead>
<tr>
<th>Thread ID</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint</td>
<td>a, j</td>
<td>a, b</td>
<td>a, c</td>
<td>b, d</td>
<td>e, f</td>
<td>g, h</td>
<td>i, l</td>
<td>k, f</td>
<td>g</td>
<td></td>
</tr>
</tbody>
</table>
**Batch Creation**

- A thread reads a constraint data
  - Thread 0 reads 0, 9
  - And write a flag to 0, 9, if they are not flagged
  - Can serialize operation in a block

**Thread Id**

<table>
<thead>
<tr>
<th>Thread Id</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint</td>
<td>a, j</td>
<td>a, b</td>
<td>a, c</td>
<td>b, c</td>
<td>b, c</td>
<td>t, i</td>
<td>b, c</td>
<td>b, c</td>
<td>b, c</td>
<td>b, c</td>
</tr>
</tbody>
</table>

**Synchronization**

- Need another mechanism to solve this situation
- Need global synchronization

**Solving Inconsistency**

- Thread 1 -> (0, 1)
- Thread 6 -> (1, 4)

- What we get is
  - Thread 1 succeed, Thread 6 failed
  - Thread 1 failed, Thread 1 succeed

- Instead of flagging, write constraint index to rigid bodies in the constraint
  - Thread 1 writes 1 to 0, 1
  - Thread 6 writes 1 to 1, 4

**Procedure**

- Batch 0
  - Clear the buffer
  - Write indices sequentially in a warp
  - Check if the write was succeed
- Batch 1
  - Clear the buffer
  - Write indices sequentially in a warp
  - Check if the write was succeed
- Batch 2
  - Clear the buffer
  - Write indices sequentially in a warp
  - Check if the write was succeed

**Inconsistency**

- But it does not solve the conflict among blocks
- Thread 1 and Thread 6 run at the same time
- Both try to flag 1

- Need another mechanism to solve this situation

**Demo**
Batch

Using Multiple GPUs
- Cannot run applications developed for a GPU
- Need two levels of parallelization
- 1GPU
- Multiple GPUs

How to Design?
- Each GPU manages its own data
- No sequential process, completely parallel

Particle Simulation on Multiple GPUs
- Grid-based
  - Domain decomposition is a natural choice, because elements in a subdomain does not change
- Particle-based
  - Have to assign particles to GPUs dynamically, because they move
  - How??
  - Overhead can be big without careful design

Decomposition of Computation
- Computation of particle values requires values of neighbors
  - Inside of subdomain: all the data is in the memory of its own
  - Boundary of subdomain: some data is in the memory of others
- Have to read data from other GPUs
- Communicating when required makes the granularity of transfer smaller and inefficient
- Transfer only “Ghost Region” and “Ghost Particles”
  - Ghosts are not updated
  - Just refer the data

Environment
- 4GPUs(Simulation) + 1GPU(Rendering)
  - S870 + 8800GTS
- 6GPU(Simulation) + 1GPU(Rendering)
  - @GDC2008
  - QuadroPlex x 2 + Tesla D870 + 8800GTS
Results

```
0 200,000 400,000 600,000 800,000 1,000,000 1,200,000
0 20 40 60 80 100
0 200,000 400,000 600,000 800,000 1,000,000 1,200,000
1GPU 2GPUx 4GPUx
```

Number of Particles

Computation Time (ms)

Thanks

- takahiro.harada@havok.com
- Demos:
  - http://www.iii.u-tokyo.ac.jp/~takahiroharada/