USING GPUS FOR COLLISION DETECTION

AMD
Takahiro Harada
GPU RAW PERFORMANCE

- High memory bandwidth
- High TFLOPS
- Radeon HD 7970
  - 32x4 SIMD engines
  - 64 wide SIMD
  - 3.79 TFLOPS
  - 264 GB/s
RAW PERFORMANCE IS HIGH, BUT

- GPU performs good only if used correctly
  - Divergence
  - ALU op / Memory op ratio
  - etc

- Some algorithm requires operation GPUs are not good at
EX) HASH GRID (LINKED LIST)

- Low ALU op / Mem op ratio
- Spatial query using hash grid

\[
do \\
\hspace{1cm} \{ \\
\hspace{2cm} \text{write( node );} \\
\hspace{2cm} \text{node = node->next;} \\
\hspace{1cm}\}\text{while( node )}
\]

- CPUs is better for this
EX) HASH GRID (LINKED LIST)

- Low ALU op / Mem op ratio
- Spatial query using hash grid
  - while()  
  - {
    - Fetch element
    - Copy
  - }
- CPUs is better for this
REASON OF PERFORMANCE JUMP

- Key architectural changes
  - No VLIW
  - Improved memory system
GPU COLLISION DETECTION
COLLISION DETECTION WAS HARD FOR GPUS

- Old GPUs were not as flexible as today’s GPUs
  - Fixed function
  - Programmable shader (Vertex shader, pixel shader)

- Grid based fluid simulation
- Cloth simulation

Crane et al., Real-Time Simulation and Rendering of 3D Fluids, GPU Gems3 (2007)
UNIFORM GRID

- Simplest but useful data structure
- Pre Compute Language
  - Vertex shader, pixel shader
  - Vertex texture fetch -> Random write
  - Depth test, blending etc -> Atomics

- Now it is very easy
  - Random write and atomics are supported

```c
struct Cell {
    u32 m_counter;
    u32 m_data[N ];
}
```
UNIFORM GRID

- Simplest but useful data structure
- Pre Compute Language
  - Vertex shader, pixel shader
  - Vertex texture fetch -> Random write
  - Depth test, blending etc -> Atomics

- Now it is very easy
  - Random write and atomics are supported
**WITH UNIFORM GRID**

- Often used for particle-based simulation
- Other steps are embarrassingly parallel

- Distinct Element Method (DEM)

- Particles can be extended to...
SMOOTHED PARTICLE HYDRODYNAMICS

- Fluid simulation
- Solving the Navier-Stokes equation on particles

Harada et al., Smoothed Particle Hydrodynamics on GPUs, CGI (2007)
RIGID BODY SIMULATION

- Represent a rigid body with a set of particles
- Rigid body collision = Particle collision

Harada et al., Real-time Rigid Body Simulation on GPUs, GPU Gems3 (2007)
COUPLING RIGID BODIES + FLUID
CLOTH COLLISION DETECTION

- Particle v.s. triangle mesh collision
- Dual uniform grid
  - 1\textsuperscript{st} grid: particles
  - 2\textsuperscript{nd} grid: triangles

Harada et al., Real-time Fluid Simulation Coupled with Cloth, TPCG (2007)
SEVERAL PHYSICS PROBLEMS WERE SOLVED
PROBLEM SOLVED??

- Not yet

- Uniform grid is not a perfect solution

- More complicated collision detection is necessary
  - E.g., rigid body simulation
  - Broad-phase collision detection
  - Narrow-phase collision detection
BROAD-PHASE COLLISION DETECTION

- Sweep & prune
  - Sort end points
  - Sweep sorted list

- Optimizations
  - Split sweep of a large object
  - Workspace subdivision
  - Cell subdivision

Liu et al., Real-time Collision Culling of a Million Bodies on Graphics Processing Units, TOG(2010)
NARROW-PHASE COLLISION DETECTION

- Variety of work
  - So many shape combinations -> divergence
- Unified shape representation
- Each collision computation is parallelizable
MULTIPLE GPU COLLISION DETECTION
MULTIPLE GPU PROGRAMMING
Parallelization using Multiple GPUs

- Two levels of parallelization
- 1GPU

- Multiple GPUs
Programming Model for Multiple GPUs

- Have to choose a programming model for particle methods
  1. Server-client model
  2. Peer-to-peer model

1. Server-client model
   - Data is managed by the CPU, working as a server
   - Overhead of parallelization can be big
   - The process of CPU is serialized
   - GPUs have to be idle while CPU is processing

2. Peer-to-peer model
   - No processor manages the entire data
   - Each GPU manages its own data
   - Small overhead of parallelization
   - No sequential process, completely parallel
PARTICLE SIMULATION ON MULTIPLE GPUS
Decomposition of Simulation

- Peer-to-peer model is employed
  - Each GPU manages its own data
  - No processor manages all the data
- Computation domain is decomposed into subdomains
- A processor is responsible for a subdomain
  - Compute particles in its subdomain
- Grid-based
  - Domain decomposition is natural choice because elements in a subdomain does not change
- Particle-based
  - Have to assign particles to GPUs dynamically because they move
  - Overhead of parallelization can be big
Decomposition of Simulation

- Computation of particle values requires neighboring values
  - Inside of subdomain: all the data is in the memory of its own
  - Boundary of subdomain: Some data may be in the memory of others
- Have to read data in other GPUs
- Communicating when required makes the granularity of transfer smaller and inefficient

- Introduced “Ghost Region” and “Ghost Particles”
  - Each GPU
    - Not update value of ghost particles
    - Just refer the data
Data Management

- Not all the data have to be sent
- Data required for the computation has to be sent

- Two kinds of particles have to be sent to adjacent GPU
  1. Escaped particles: Particles get out from adjacent subdomain (adjacent GPU will be responsible for these particles in the next time step)
  2. Ghost particles in the ghost region
Data Transfer

- After computation (advection etc.)
- Particles have to sent to adjacent GPU
  - Escaped particles
  - Ghost particles
- How to choose them?
  1. Scan particles and set flag to particles in the region
  - Need a lot of additional computations
- Overhead can be big
Data Transfer

- Where were these particles at time t?
- Using the Courant condition (particle does not move more than their diameter in a time step)
- Escaped particles were in the blue area
- Ghost particles were in the red area
- Particles have to be sent is the particles in the red area
- Grid constructed in the previous step can be used for selection
  - The Sliced Grid

Slided Data Structure for Particle-based Simulations on GPUs.
Harada et al., GRAPHITE, 55-62(2007)
ShaderX 7, To Appear
Data Transfer between GPUs

- No direct data transfer is provided on current hardwares
- Transfer via main memory
  - The buffers equal to the number of connectors are allocated
  - Each GPU writes data to the defined location at the same time
  - Each GPU read data of the defined location at the same time
Computation of 1 Time Step

Main Memory

GPU0 -> GPU1 -> GPU2 -> GPU3

Compt Force -> Update -> Prepare Send -> Send -> Synchronization -> Receive -> Post Receive

$PNQU$ PSDF $PNQU$

$PNQU$ PSDF $PNQU$

$PNQU$ PSDF $PNQU$
Environment

- 4GPU server (Simulation) + 1GPU (Rendering)
  - 1M particles

- 6GPU server boxes (Simulation) + 1GPU (Rendering) @ GDC2008
  - 3M particles
MOVIE

Harada et al., Massive Particles: Particle-based Simulations on Multiple GPUs, SIG Talk(2008)
Results

Number of Particles ($10^3$ particles)

- Total (1GPU)
- Sim (2GPU)
- Total (2GPU)
- Sim (4GPU)
- Total (4GPU)

Computation Time (ms)

Number of Particles (10^3 particles)
PROBLEM SOLVED??

- Not yet

- Most of the problems had identical object size
  - e.g., Particles

- The reason is because of GPU architecture
  - Not designed to solve non-uniform problem
HETEROGENEOUS COLLISION DETECTION
PARTICLE BASED SIMULATION ON THE GPU

- Large number of particles
- Particles with identical size
  - Work granularity is almost the same
  - Good for the wide SIMD architecture

Harada et al. 2007
MIXED PARTICLE SIMULATION

- Not only small particles
- Difficulty for GPUs
  - Large particles interact with small particles
  - Large-large collision
CHALLENGE

- Non uniform work granularity
  - Small-small (SS) collision
    - Uniform, GPU
  - Large-large (LL) collision
    - Non Uniform, CPU
  - Large-small (LS) collision
    - Non Uniform, CPU
FUSION ARCHITECTURE

- CPU and GPU are:
  - On the same die
  - Much closer
  - Efficient data sharing

- CPU and GPU are good at different works
  - CPU: serial computation, conditional branch
  - GPU: parallel computation

- Able to dispatch works to:
  - Serial work with varying granularity $\rightarrow$ CPU
  - Parallel work with the uniform granularity $\rightarrow$ GPU
METHOD
TWO SIMULATIONS

- Small particles
  - Position
  - Velocity
  - Force
  - Grid

  - Build Acc. Structure
  - SS Collision
  - S Integration

- Large particles
  - Position
  - Velocity
  - Force

  - Build Acc. Structure
  - LL Collision
  - L Integration
CLASSIFY BY WORK GRANULARITY

- Small particles
  - Uniform Work
    - Build Acc. Structure
    - SS Collision
    - S Integration
  - Non Uniform Work
    - Build Acc. Structure
    - LL Collision
    - LS Collision
    - L Integration

- Large particles
CLASSIFY BY WORK GRANULARITY, ASSIGN PROCESSOR

- Small particles
  - Build Acc. Structure
  - SS Collision
  - S Integration

- Large particles
  - Build Acc. Structure
  - LL Collision
  - LS Collision
  - L Integration
• Small particles

• Large particles

• Grid, small particle data has to be shared with the CPU for LS collision
  – Allocated as zero copy buffer
- Small particles

- Large particles

- Grid, small particle data has to be shared with the CPU for LS collision
  - Allocated as zero copy buffer
VISUALIZING WORKLOADS

- Small particles
  - Position
  - Velocity
  - Force
  - Grid

- Large particles
  - Position
  - Velocity
  - Force

- Grid construction can be moved at the end of the pipeline
  - Unbalanced workload
Small particles

- To get better load balancing
  - The sync is for passing the force buffer filled by the CPU to the GPU
  - Move the LL collision after the sync

Large particles
MULTI THREADING
(4 THREADS)
OPTIMIZATION2: IMPROVING SMALL-SMALL COLLISION

Harada, Heterogeneous Particle-Based Simulation, SIG ASIA Talk (2011)
QUESTIONS?