OpenCL Game Physics
Bullet: A Case Study in Optimizing Physics Middleware for the GPU

Erwin Coumans
Overview

• Introduction
• Particle Physics Pipeline from the NVIDIA SDK
  – Uniform grid, radix or bitonic sort, prefix scan, Jacobi
• Rigid Body Physics Pipeline
  – Parallel Neighbor search using dynamic BVH trees
  – Neighboring Pair Management
  – Convex Collision Detection: GJK in OpenCL on GPU
  – Concave Collision Detection using BVHs
  – Parallel Constraint Solving using PGS
• OpenCL cross-platform and debugging
Introduction

• Bullet is an open source Physics SDK used by game developers and movie studios
• PC, Mac, iPhone, Wii, Xbox 360, PlayStation 3
• Bullet 3.x will support OpenCL acceleration
  – Simplified rigid body pipeline fully running on GPU
  – Developer can mix stages between CPU and GPU
• Implementation is available, links at the end
Some games using Bullet Physics

- FreeRealms
- Grand Theft Auto IV
- Trials HD
Some movies using Bullet Physics

HANCOCK
WILL SMITH
NOT YOUR AVERAGE SUPERHERO

BOLT
JOHN TRAVOLTA
MILEY CYRUS
IN THEATERS NOVEMBER 21
DISNEY DIGITAL 3D

2012
Authoring tools

• Maya Dynamica Plugin
• Cinema 4D 11.5
• Blender
Rigid Body Scenarios

3000 falling boxes
1000 stacked boxes
136 ragdolls

1000 convex hulls
1000 convex against trimesh
ray casts against 1000 primitives and trimesh
Performance bottlenecks

Collision Data
- Collision Shapes
- Object AABBs
- Overlapping Pairs
- Contact Points

Dynamics Data
- Motionstate Transforms, Velocities
- Rigid Bodies Mass, Inertia
- Constraints contacts, joints

Forward Dynamics
- Apply Gravity
- Predict Transforms
- Compute AABBs
- Detect Pairs
- Compute Contacts

Collision Detection
- Setup Constraints
- Solve constraints
- Integrate Position
Leveraging the NVidia SDK

- Radix sort, bitonic sort
- Prefix scan, compaction
- Examples how to use fast shared memory
- Uniform Grid example in Particle Demo
Particle Physics
CUDA and OpenCL Demo
Uniform Grid

<table>
<thead>
<tr>
<th>Cell ID</th>
<th>Count</th>
<th>Particle ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>D,F</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>B,C,E</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
### Sorting Particles per Cell

#### Grid Representation

- **Cell Index**: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
- **Cell Start**:
  - 0
  - 1
  - 2
  - 3
  - 4
  - 5
  - 6
  - 7
  - 8
  - 9
  - 10
  - 11
  - 12
  - 13
  - 14
  - 15

#### Array Index

<table>
<thead>
<tr>
<th>Array Index</th>
<th>Unsorted Cell ID, Particle ID</th>
<th>Sorted Cell ID Particle ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>9, A</td>
<td>4, D</td>
</tr>
<tr>
<td>1</td>
<td>6, B</td>
<td>4, F</td>
</tr>
<tr>
<td>2</td>
<td>6, C</td>
<td>6, B</td>
</tr>
<tr>
<td>3</td>
<td>4, D</td>
<td>6, C</td>
</tr>
<tr>
<td>4</td>
<td>6, E</td>
<td>6, E</td>
</tr>
<tr>
<td>5</td>
<td>4, F</td>
<td>9, A</td>
</tr>
</tbody>
</table>

#### Particle Distribution

- **Cell 0**: B
- **Cell 1**: C
- **Cell 2**: D
- **Cell 3**: E
- **Cell 4**: A
- **Cell 5**: A
- **Cell 6**: B
- **Cell 7**: C
- **Cell 8**: D
- **Cell 9**: E
- **Cell 10**: F
- **Cell 11**: F
- **Cell 12**: A
- **Cell 13**: B
- **Cell 14**: C
- **Cell 15**: D
Neighbor search

- Calculate grid index of particle center
- Parallel Radix or Bitonic Sorted Hash Array
- Search 27 neighboring cells
  - Can be reduced to 14 because of symmetry
- Interaction happens during search
  - No need to store neighbor information
- Jacobi iteration: independent interactions
Interacting Particle Pairs

<table>
<thead>
<tr>
<th>Array Index</th>
<th>Sorted Cell ID</th>
<th>Particle ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>4,D</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>4,F</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>6,B</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>6,C</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>6,E</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>9,A</td>
</tr>
</tbody>
</table>

Interacting Particle Pairs

- D,F
- B,C
- B,E
- C,E
- A,D
- A,F
- A,B
- A,C
- A,E
Using the GPU Uniform Grid as part of the Bullet CPU pipeline

- Available through btCudaBroadphase
- Reduce bandwidth and avoid sending all pairs
- Bullet requires persistent contact pairs
  - to store cached solver information (warm-starting)
- Pre-allocate pairs for each object
**Persistent Pairs**

**Before**

Before image with particle pairs:
- D,F
- B,C
- B,E
- C,E
- A,D
- A,F
- A,B
- A,C
- A,E
- C,F
- C,D

**After**

After image with particle pairs:
- D,F
- B,C
- B,E
- C,E
- A,D
- A,F
- A,C
- A,E
- C,F
- C,D

**Differences**

- A,B removed
- B,C removed
- C,F added
- C,D added
Broadphase benchmark

• Includes btCudaBroadphase
• Bullet SDK: Bullet/Extras/CDTestFramework
# From Particles to Rigid Bodies

<table>
<thead>
<tr>
<th></th>
<th>Particles</th>
<th>Rigid Bodies</th>
</tr>
</thead>
<tbody>
<tr>
<td>World Transform</td>
<td>Position</td>
<td>Position and Orientation</td>
</tr>
<tr>
<td>Neighbor Search</td>
<td>Uniform Grid</td>
<td>Dynamic BVH tree</td>
</tr>
<tr>
<td>Compute Contacts</td>
<td>Sphere-Sphere</td>
<td>Generic Convex Closest Points, GJK</td>
</tr>
<tr>
<td>Static Geometry</td>
<td>Planes</td>
<td>Concave Triangle Mesh</td>
</tr>
<tr>
<td>Solving method</td>
<td>Jacobi</td>
<td>Projected Gauss Seidel</td>
</tr>
</tbody>
</table>
Dynamic BVH Trees

![Dynamic BVH Trees Diagram]

- Nodes are positioned within a grid, which helps in visualizing the bounding boxes and the hierarchy of the BVH tree.
- The tree structure shows how nodes are grouped and organized, allowing for efficient spatial queries.
- The grid layout aids in understanding the spatial distribution and the relationships between different elements.

The diagram illustrates the concept of BVH trees, which are used to optimize collision detection and spatial queries in 3D graphics applications.
Dynamic BVH tree acceleration structure

• Broadphase n-body neighbor search
• Ray and convex sweep test
• Concave triangle meshes
• Compound collision shapes
Dynamic BVH tree Broadphase

- Keep two dynamic trees, one for moving objects, other for objects (sleeping/static)
- Find neighbor pairs:
  - Overlap M versus M and Overlap M versus S

S: Non-moving DBVT

M: Moving DBVT
DBVT Broadphase Optimizations

- Objects can move from one tree to the other
- Incrementally update, re-balance tree
- Tree update hard to parallelize
- Tree traversal can be parallelized on GPU
  - Idea proposed by Takahiro Harada at GDC 2009
Parallel GPU Tree Traversal using History Flags

• Alternative to recursive or stackless traversal
Parallel GPU Tree Traversal using History Flags

• 2 bits at each level indicating visited children
Parallel GPU Tree Traversal using History Flags

• Set bit when descending into a child branch
Parallel GPU Tree Traversal using History Flags

- Reset bits when ascending up the tree
Parallel GPU Tree Traversal using History Flags

• Requires only twice the tree depth bits
Parallel GPU Tree Traversal using History Flags

- When both bits are set, ascend to parent
Parallel GPU Tree Traversal using History Flags

• When both bits are set, ascend to parent
History tree traversal

do{
    if (Intersect(n->volume, volume)) {
        if (n->isinternal()) {
            if (!historyFlags[curDepth].m_visitedLeftChild) {
                historyFlags[curDepth].m_visitedLeftChild = 1;
                n = n->childs[0];
                curDepth++;
                continue;
            }
            if (!historyFlags[curDepth].m_visitedRightChild) {
                historyFlags[curDepth].m_visitedRightChild = 1;
                n = n->childs[1];
                curDepth++;
                continue;
            }
        } else {
            policy.Process(n);
        }
    }
    n = n->parent;
    historyFlags[curDepth].m_visitedLeftChild = 0;
    historyFlags[curDepth].m_visitedRightChild = 0;
    curDepth--;
} while (curDepth);
Find contact points

- Closest points, normal and distance
- Convention: positive distance -> separation
- Contact normal points from B to A
Voxelizing objects
OpenCL Rigid Particle Bunnies
Broadphase

- The bunny demo broadphase has entries for each particle to avoid \( n^2 \) tests
- Many sphere-sphere contact pairs between two rigid bunnies
- Uniform Grid is not sufficient
Voxelizing objects
General convex collision detection on GPU

- Bullet uses hybrid GJK algorithm with EPA
- GJK convex collision detection fits current GPU
- EPA penetration depth harder to port to GPU
  - Larger code size, dynamic data structures
- Instead of EPA, sample penetration depth
  - Using support mapping
- Support map can be sampled using GPU hardware
Parallelizing Constraint Solver

- Projected Gauss Seidel iterations are not embarrassingly parallel
Reordering constraint batches

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Diagram showing the reordering of constraint batches from A, B, C, D to A, B, C, D.
Creating Parallel Batches

DEVICE

- Integrate OpenCL
- Compacted Pairs
- Objects Used
- Setup Batches
- Batch ID Array

- Apply Gravity
- Update Spheres
- Sort Cell ID (Hash)
- Find Cell Start
- Find Pairs in Cell
- Scan Pairs
- Compact Pairs

- Compute Contacts
- Solve Constraints
OpenCL kernel Setup Batches

```c
__kernel void kSetupBatches(...) {
    int index = get_global_id(0);
    int currPair = index;
    int objIdA = pPairIds[currPair * 2].x;
    int objIdB = pPairIds[currPair * 2].y;
    int batchId = pPairIds[currPair * 2 + 1].x;
    int localWorkSz = get_local_size(0);
    int localIdx = get_local_id(0);
    for(int i = 0; i < localWorkSz; i++) {
        if((i==localIdx) && (batchId < 0) && (pObjUsed[objIdA]<0) && (pObjUsed[objIdB]<0)) {
            if(pObjUsed[objIdA] == -1)
                pObjUsed[objIdA] = index;
            if(pObjUsed[objIdB] == -1)
                pObjUsed[objIdB] = index;
        }
    }
    barrier(CLK_GLOBAL_MEM_FENCE);
}
```
Colored Batches
CPU 3Ghz single thread, 2D, 185ms
Geforce 260 CUDA, 2D, 21ms
CPU 3Ghz single thread, 3D, 12ms
Geforce 260 CUDA, 3D, 4.9ms
OpenCL Implementation

• Available in SVN branches/OpenCL
  – http://bullet.googlecode.com

• Tested various OpenCL implementations
  – NVidia GPU on Windows PC
  – Apple Snow Leopard on GeForce GPU and CPU
  – Intel, AMD CPU, ATI GPU (available soon)
  – Generic CPU through MiniCL
    • OpenCL kernels compiled and linked as regular C
    • Multi-threaded or sequential for easier debugging
Thanks!

• Questions?
• Visit the Physics Simulation Forum at
  – http://bulletphysics.com
• Email: erwin_coumans@playstation.sony.com