Rigid Body Simulations
Physics on the GPU
What is a Rigid Body?

- A rigid body is a non-deformable object that is an idealized solid.
- Each rigid body is defined in space by its center of mass.
- To make things simpler we assume the mass has an equal distribution of mass.
- The rigid body also has a velocity.
- Unlike particles the rigid body has an associated rotation and angular velocity.
- Using these two distinctions, the velocity at any point on the body is equal and the rotational velocity at any point is equal.

GPU Gems 3: Fig. 29-2
Rigid Body Simulation

- 100s to 1000s of rigid bodies at once
  - All maybe interacting with one another or the user

- Complex shapes
  - The rigid bodies maybe different sizes and shapes
  - Bodies may also be concave, allowing them to get stuck in one another

- Physically realistic
  - Each object has visually realistic interactions with one another
  - Objects may have inter-dependencies

- Becomes more Difficult to parallelize
  - Arbitrary object shapes can create branching
  - Dependencies makes it hard to find calculations to do in parallel
Major Components of a Rigid Body Simulation

- **Integration**
  - Moving objects based on forces affecting them
  - Not discussed as much in papers of Rigid Body Simulation

- **Broad Phase Collision Detection**
  - Culling intersections to prevent performing $O(n^2)$ checks

- **Narrow Phase Collision Detection**
  - Performing intersections found in Broad Phase

- **Collision Response**
  - Bouncing, Sliding and Deflecting objects based on previous detections
  - Generally creates a force that is applied against colliding objects to change their velocity and thereby their position.
Motivations

- Entertainment
  - Video Games
  - Movies
  - Realistic Visual Effects

- Mechanical Design
  - Prototyping
  - Stress and Usability Testing
Motivations – Video Games

- Physics Effects adds realism to game
  - Interacting with objects, environment and other players
  - Creates an immersive world

CellFactor: Revolution, Developed by Artificial Studios
PhysX
Motivations – Video Games

Videos
  › CellFactor
    • http://www.youtube.com/watch?v=Q_ZF6hSiF9o
  › Half-Life
    • http://www.youtube.com/watch?v=HxaFDP0Ffw0&feature=related
Motivations – Movies

- Realistic looking Effects with great detail
- Much easier to animate than using keyframes
  - Litter and debris add detail
- Allows artists to play with physical properties to get intended result

WALL-E, Disney’s Pixar

2012, Sony Pictures
Motivations – Design

- Create a computer model of a mechanical design
- Allows for testing of mechanical motion, stress, pressure and physical properties
- Can allow designers to tune properties without building actual model
Other Works Not Covered

- **Physics Libraries**
  - NVIDIA’s PhysX
  - Havok FX
  - Open Dynamics Engine (ODE)

- **Other Papers**
  - ParallAX: An Architecture for Real-Time Physics
  - A New Physics Engine with Automatic Process Distribution between CPU-GPU
    - Mark Joselli, Esteban Clua, Anselmo Montenegro, Aura Conci, Paulo Pagliosa, ACM 2008
  - Evaluation of Real-Time Physics Simulation System
    - Adrian Boeing, Thomas Braunl, ACM 2007
Works Going to Cover

- GPU Gem 3. Chapter 29: Real Time Rigid Body Simulation on GPUs
  - Takahiro Harada

- Bullet: A Case Study in Optimizing Physics Middleware for GPU
  - Erwin Coumans
Works Going to Cover

- GPU Gem 3. Chapter 29: Real Time Rigid Body Simulation on GPUs
  - Takahiro Harada

- Bullet: A Case Study in Optimizing Physics Middleware for GPU
  - Erwin Coumans
Real Time Rigid Body on GPU
Overview

- Voxelization of Bodies
- Collision Detection using Grid and Particles
- Data Structures for Position, Rotation, etc
Voxelization of Rigid Bodies

- Rigid Bodies are broken down into sets of particles
  - Discretize space around bodies
  - Perform a raycast through the grid space to determine which particles are filled
  - Requires mesh to be closed, but allows for complex concave designs

- GPU Accelerations
  - Using Depth Peeling to perform ray cast.
  - Using fragment shader to hold information about voxelization.

GPU Gems 3: Fig. 29-3
Voxelization of Rigid Bodies

Generating Uniform Grid on GPU

- Stenciling Routing Technique
- Using vertex shader, one vertex for one particle
- Allocating a voxel for each pixel
- Allows for at maximum four particles per voxel
  - Using RGBA values for the values of each voxel
- Rendering Requires 4 Passes into buffer
  - Each pass rendering to RGBA respectively
  - Uses stencil buffer and depth buffer to determine channel to use
- Can be used for more than 4 particles per voxel, but requires multiple render targets.

GPU Gems 3: Fig. 29-8
Voxelization of Rigid Bodies

- Pseudocode of Passes

```c
01. //== 1 PASS
02. colorMask(GBA);
03. depthTest(less);
04. renderVertices();
05. //== 2 PASS
06. colorMask(RBA);
07. depthTest(greater);
08. stencilTest(greater, 1);
09. stencilFunction(increment);
10. clear(stencilBuffer);
11. renderVertices();
12. //== 3 PASS
13. colorMask(RGA);
14. clear(stencilBuffer);
15. renderVertices();
16. //== 4 PASS
17. colorMask(RGB);
18. clear(stencilBuffer);
19. renderVertices();
```

GPU Gems 3: Example 29 -1
Collision Detection Using Grid

- Broken Down Complex Shapes into Simple particles
- Now only Requires Particle-Particle Collision
  - Allows for control of speed versus Accuracy
  - Higher Resolution Grid = Higher Accuracy
  - Lower Resolution Grid = Higher Frame Rates
  - Multipurpose

- Uniform Grid Organization
  - Used to reduce check time between particles from $O(n^2)$
  - Domain is divided into voxels with index $g$ and each particle has an associated voxel index.
  - With in the grid particles are reduced to colliding with
    - Other particles within their grid
    - Particles in the surrounding voxels
  - Improves running time to $O(n)$
  - Most efficient when voxel side length is equal to the diameter of a particle, reducing checks to $3^3$ checks in 3 dimensions.
Collision Detection Using Grid

- Look up into Voxel grid to find particles within the 27 possible voxels around a given particle.
- No Broad or Narrow phase needed
  - Particles require only distance check
  - Only checking neighboring particles because we are using uniform grid
Collision Response Using Particles

- Consider inter-particle forces between pairs rather than groups of particles interacting.
- Discrete Element Method (DEM) for getting forces between granular materials:
  - Particles colliding are considered to have a repulsive force modeled by a spring, a damping force, and a shear force based on tangential velocity.
- We then sum all the forces affecting one rigid body and apply it to all the particles within the rigid body:
  - It is a similar process for the torques on each body.
  - Once the forces have been calculated, we perform an integration to find positions.
  - Pseudo-Code for Detection and Reaction

```plaintext
01. gridCoordinate = calculateGridCoordinate(iParticlePosition);
02. FOREACH of the 27 voxels
03.     jIndex = tex2D(gridTexture, gridCoordinate);
04.     FOREACH RGBA channel C of jIndex
05.         jParticlePosition = tex2D(positionTexture, index2Texture(C));
06.         force += collision(iParticlePosition, jParticlePosition);
07.     END FOR
08. END FOR
```

GPU Gems 3: Example 29-2
Data Structures of Rigid Bodies

- Rigid Bodies are defined like Particles
  - We need to hold their center of mass, velocity, angular momentum and orientation.
  - These can all be held in a texture

- Unlike Particles
  - Since each rigid body is turned into particles we need to increase the size of the texture to hold all information for each particle
  - Using the number of particles per rigid body we can calculate the rigid body’s position in the texture.
  - We use two textures, one for reading and one for writing
We also need to consider the uniform grid in which the particles are placed

- For this we create a 3D data structure on a 2D texture
- Called “Flat 3D Texture”
- At time of implementation, 3D textures are not as optimized as 2D textures in hardware
Results and Performance

- Tested on NVIDIA GeForce 8800 GTX
  - 16,384 Chess pieces, with 65,536 particles at 44 fps
  - 10,922 Tori, with 65,536 particles at 68 fps
  - 963 bunnies, 68 particles each at 112 iterations per second (ips)
  - 329 teapots, 199 particles each at 101 ips

- All grid sizes were uniform at $128^3$
  - Made note that uniform grids were not necessarily and not always best

- Projects could be easily adapted to granular materials or fluids

GPU Gems 3: Fig. 29-9

GPU Gems 3: Fig. 29-10
Results and Performance

GPU Gems 3: Fig. 29-1
Works Going to Cover

- GPU Gem 3. Chapter 29: Real Time Rigid Body Simulation on GPUs
  - Takahiro Harada

- Bullet: A Case Study in Optimizing Physics Middleware for GPU
  - Erwin Coumans
Bullet: Rigid Body Simulation Overview

- Portability
- Dynamic BVH Trees
- Convex Closest Points, GJK Collision detection
- Projected Gauss Seidel Solving
Bullet Portability

- Bullet is open source Physics SDK
  - Used by game developers and movie studios
- Bullet 3.0 and above supports OpenCL acceleration
  - Allows for developers to work on many platforms
    - PC, Mac, iPhone, Wii, Xbox360 and PlayStation 3
- Many different implementations given in library, allowing for mixing between CPU and GPU stages
  - Creating a much better library for varying games and movies
- Already has been implemented in tools commercially available
  - Maya Plugins
  - Cinema 4D 11.5
  - Blender
Dynamic Bounding Volume Hierarchy Trees (BVH Trees)

- **Trees**
  - Siblings are the bounding boxes closest to one another
  - Optimized so overlapping of boxes is minimal
  - Overall volume is minimized

- **Implemented without recursion**
  - Necessary on the GPU

- **Implemented for Broad phase collision**

- **Allows for concave triangle meshes and compound collision shapes**

- **Can also be used for ray tracing when drawing scene**
Keeping two dynamic trees
- One for moving objects and one for static objects.
- They consider objects moving relatively slowly are static.

Updating the Trees
- We do not need to update the non-moving BVHT since their velocities are small.
- Updating moving objects becomes hard to parallelize.
- Objects are allowed to move from one tree to the other.
- We must incrementally update and rebalance the tree.
History Flags

- This acts as an alternative to recursive or stackless traversal
- Each level in the tree has 2 bits of information to represent where we are in the recursion.
- We require twice the number of levels of bits to store the flags, which is relatively small compared to the entire tree
- The bit is 1 if you have visited or currently visiting the child
- The bit is then reset when you finish a node.
- This allows you to traverse the tree using the flag information without dealing with recursion.
BVH Trees Traversal
Setting bit when going down child branch
BVH Trees Traversal
Resetting bit when moving back up tree
BVH Trees Traversal
BVH Trees Traversal

Branch has been completed, so we back move up

Bullet: A Case Study in Optimizing Physics, Erwin Coumans
BVH Trees Traversal

Reset the flags at the previous branch

Bullet: A Case Study in Optimizing Physics, Erwin Coumans
Collision Detection with Non-Voxelized Objects

- Rather than voxelizing objects we can create a low polygon representation of the object
  - Voxelization may not be good for certain objects as you might need many spheres to get accurate collision and it no longer becomes faster
  - Get a “better” approximation with a convex hull of the object with faster speed up
  - Allows for usage of the Convex Closest Points, Gilbert-Johnson-Keerthi (GJK) Collision detection rather than sphere-sphere on a grid
  - Also allows easier, non-uniform grids that are not dependent on the voxelization
  - Acts as Narrow phase

The same bunny, voxelized versus Low polygon where the number of spheres is comparable to the number of faces

Bullet: A Case Study in Optimizing Physics, Erwin Coumans
GJK Algorithm for collision detection

- Bullet uses hybrid GJK (Gilbert-Johnson-Keerthi) algorithm with EPA
- Calculates distance between convex objects
  - The algorithm returns a close approximation of the shortest distance between two convex objects
  - The algorithm creates a set of vertices through iteration that contain the two closest points in each object
  - The algorithm will produce the actual distance of two points if the objects are polytopes.
  - To deal with hard cases such as spheres or cylinders, we augment the system with a mapping.
Projected Gauss Seidel for Force Solving

- Rather than trying to organize threads for each object being acted upon we create constraint batches
  - These batches represent the rigid interacting with one another
  - They can be sorted into dependency groups to be solved
  - Then we create the independent sets and solve them in parallel.
  - Finding the sets to do in parallel is hard to do in parallel

Bullet: A Case Study in Optimizing Physics, Erwin Coumans
Color Batches

Colored based on what interactions can be done in parallel

Bullet: A Case Study in Optimizing Physics, Erwin Coumans
Results and Performance

- **2D case**
  - 4000 Cubes
  - CPU 3Ghz single thread, 185 ms/f
  - Geforce 260 CUDA, 21 ms/f
  - 8.8 times speed up

- **3D Case**
  - 3000 Cubes
  - CPU 3Ghz single thread, 15 ms/f
  - Geforce 260 CUDA, 9 ms/f
  - 1.6 times speed up

- **Videos**
  - [http://www.youtube.com/watch?v=3lojinnYQ8](http://www.youtube.com/watch?v=3lojinnYQ8)
Summary

- The GPU offers a substantial way of speeding up rigid body simulations
- Use the right broad and narrow phase for your given situation
  - If you have many objects in the scene and you don’t care about accuracy, use voxelization
  - If you have larger and fewer objects you may consider using a lower polygon representation with DBVHs
- Find ways to speed up the number of force calculations you must consider.
  - By finding batches of force calculations that can be done in parallel, you decrease the amount of time to find the final force affecting any given body.
- Use the optimal data structure for your bodies to reduce space in the texture and to conserve bandwidth.
My Implementation

- **Final Project**
  - Create a rigid body simulator optimized for origami
  - Take in folding sequences and apply torque forces to make rigid bodies fold

- **Solving Torque Forces using sparse matrix**
  - Similar to batch idea
  - For Origami, adjacency between objects is always known and never changes

- **Collision methods that do not require closed meshes**
  - All methods I’ve found so far require a closed mesh or voxelize the mesh
  - I want to do neither, so to handle this case I perform AABB as a broad phase and simple triangle-triangle collision for the narrow phase.

- **Reduce bandwidth by sending minimum information to the GPU at each time step and not have to recalculate structures each step.**
Questions