Fast Collision and Proximity Computations

Dinesh Manocha
University of North Carolina at Chapel Hill
dm@cs.unc.edu
http://gamma.cs.unc.edu

Collaborators
- Sean Curtis (UNC)
- Christian Lauterbach (UNC/Google)
- Young Kim (Ewha)
- Ming Lin (UNC)
- Qi Mo (UNC)
- Rasmus Tamstorf (Disney)
- Min Tang (Zhejiang Univ.)
- Sungeui Yoon (KAIST)
- Liangjun Zhang (UNC/Stanford)

Proximity Queries

Geometric reasoning of spatial relationships among objects (in a dynamic environment)

Collision Detection
Contact Points & Normals
Closest Points & Separation Distance
Penetration Depth

Motivation

Proximity queries
Intersection
Separation & Penetration distance
Self-collision
2+ objects
Object
Problem Domain Specifications

Model Representations
- polyhedra (convex vs. non-convex vs. soups)
- CSG, implicits, parametrics, point-clouds

Type of Queries
- discrete vs. continuous query
- distance vs. penetration computation
- estimated time to collision

Simulation Environments
- pairwise vs. n-body
- static vs. dynamic
- rigid vs. deformable

Applications
- Robot motion planning
- Simulation of (dis-)assembly tasks
- Tolerance verification
- Simulation-based design
- Ergonomics analysis
- Haptic rendering
- Physics-based modeling and simulation

Prior work on Proximity Computations
- Fast algorithms for convex polytopes (1991 onwards)
- Bounding volume hierarchies for general polygonal models (1995 onwards)
- Deformable models & self-collisions (2000 onwards)
- Multiple software systems
  - I-Collide, RAPID, PQP, DEEP, SWIFT, SWIFT++, PIVOT
  - DeformCD, Self-CD,.....

Multiple software systems
- I-Collide, RAPID, PQP, DEEP, SWIFT, SWIFT++, DeformCD, PIVOT, Self-CD,.....
- More than 100,000 downloads from 1995 onwards
- Issued more than 50 commercial licenses (Kawasaki, MSC Software, Ford, Sensable, Siemens, BMW, Phillips, Intel, Boeing, etc.)
Do we need better or faster algorithms?

Reliable continuous, self-collisions for cloth simulation
(Model Courtesy: Disney Animation)

Penetration computation has high combinatorial complexity: Needed for dynamic response and path planning

Finite-Element Simulation for Crash Analysis: Collisions can take 50-90% of simulation time (Model Courtesy: BMW & LS-DYNA)
Our Recent Work

- Faster algorithms for continuous collision detection among deformable models
- Volumetric continuous collision methods
- Penetration depth computation
- Parallel algorithms for multi-core and many-core processors

Continuous Collision Detection

Compute the first time of contact between discrete time intervals

- Incremental hierarchy based methods
- Improved culling based on normal bounds
- Eliminate redundant elementary tests
- Simple filters to remove false positives

More than 10-20X improvement in performance

Continuous Collision Detection

Fast Collision Detection for Deformable Models using Representative-Triangles

Sean Curtis*
Rasmus Tamstorf*
Dinesh Manocha*

* University of North Carolina - Chapel Hill
* Walt Disney Animation Studios

Volumetric CCD

- New volumetric methods for FEM simulations
- Collision checking between internal nodes and elements
- Eliminate redundant elementary tests
- Simple filters to remove false positives

Up to 20X improvement in performance
[Tang et al. 2011]
Volumetric CCD

VolCCD: Fast Continuous Collision Culling between Deforming Volume Meshes

Submission ID: 0191

Penetration Depth Computation

- Generalized penetration depth formulation based on rotational motion
- Local and global penetration depth computation
- Retraction based planners for rigid and articulated models

[Zhang et al. 2006; Zhang et al. 2007; Zhang et al. 2008; Pan et al. 2010]

Retraction-based Planner using Penetration Depth Computations

Collision or proximity checking takes more than 90% of time in sample-based planners

A Parallel Revolution: 2005 Onwards

Power Wall = Brick Wall

End of way built microprocessors for last 40 years

New Moore’s Law is 2X processors (“cores”) per chip every technology generation, but ≈ same clock rate

“This shift toward increasing parallelism is not a triumphant stride forward based on breakthroughs ...; instead, this ... is actually a retreat from even greater challenges that thwart efficient silicon implementation of traditional solutions.”

The Parallel Computing Landscape: A Berkeley View, Dec 2006

- Sea change for HW & SW industries since changing the model of programming and debugging
Parallel Revolution has started!

- While evolution and global warming are "controversial" in scientific circles, belief in need to switch to parallel computing is unanimous in the hardware community (Dave Patterson, Berkeley)

- AMD, Intel, IBM, Sun, ... now sell more multiprocessor ("multicore") chips than uniprocessor chips
  - Plan on little improvement in clock rate (8% / year?)
  - Expect more cores every 2 years, ready or not
  - Note – they are already designing the chips that will appear over the next 5 years, and they’re parallel

Multi-Core and Many-Core Processors

- Multi-core CPUs (Intel, AMD, IBM)
  - Take the best serial core and fit as many cores on a single chip, as possible
  - Each serial core has large caches
  - Support limited SIMD and instruction-level parallelism

Many-Core Processors (GPUs)

2010: Fermi has 512 *scalar* fragment processors or cores
2009: GT285 240 *scalar* fragment processors or cores
2006: G80 (8800 GTX) has 128 fragment processors or cores
2005: G71 (7900) has 48 *vec4* pixel cores
2004: NV40 (6800) has 16 vec4 cores
2003: NV30 (5800) had 4 vec4 pixel shader pipes or cores

- Growth Rate of NVIDIA GPUs (2003 onwards)

Many-Core or High-Throughput Computing

- Notion of designing commodity processors with tens or hundreds of cores
- Combining fine-grain and coarse-grain parallelism
- High parallel code performance
- Improved memory throughput and power efficiency
GPU-based Algorithms

• Challenges in exploiting multiple cores
• Communication and synchronization between the cores is limited
• Limited cache hierarchy
• Use high number of threads to hide memory latency

High GPU Computing Throughput

• Provide a sufficient number of parallel tasks so that all the cores are utilized
• Provide several times that number of tasks just so that each core has enough work to perform while waiting for data from slow memory accesses

Dynamic GPU Work Distribution Methods [Lauterbach, Mo and Manocha 2009; Lauterbach & Manocha 2010]

Computing and Traversing Hierarchies

• Build or update hierarchies (Hard to parallelize)
• Traverse hierarchies recursively
  - Start with root nodes
  - Do nodes overlap?
    - Yes: Inner nodes: recurse on combinations of children
      Leaf nodes: put primitive pair in separate queue
  - Perform primitive overlap tests (Easy to parallelize)

Hierarchy-based proximity queries
**Primitive tests**

- **Discrete collision**: triangle-triangle test
  - Do triangles overlap?

- **Continuous collision**
  - Did moving triangles overlap at any time between $t_1$ and $t_2$?

**Related work**

- Use multi-core CPUs
  - [Kim et al. 08, Kim et al. 09, Tang et al. 09]

**Work organization on GPUs**

- Standard for recursive hierarchy operations
  - Global work queue, work stealing

- Problem
  - Shared access on GPU only via slow, non-consistent global memory

**Lightweight balancing**

- Our solution
  - Every thread/core has local queue (non-shared)
  - Keep track of other thread's state occasionally
    - One shared global idle counter
  - If above threshold, break and balance queues

- Avg. ~2-3x performance of work stealing
Parallel Hierarchy Operations

• Can also use vector units
  - Each vector lane handles one intersection pair
  - Potentially thousands of parallel tests
• Local work queue shared between lanes
  - Access synchronized by atomics or prefix sum
  - Does not change outside synchronization

Hierarchy Construction

BVH construction on GPUs
- Uses thread and data parallelism
- Fast linear BVH construction
- Interactive construction on current GPUs

Hierarchy Construction

Top-down methods
- E.g. recursively split primitives in half
Bottom-up methods
- Repeatedly combine primitives into groups
- Derive from scene graph

Bounding volumes

• We use oriented bounding boxes (OBBs) on GPUs
  - Operations: about 1-2 order of magnitude more instructions
• But:
  - Hierarchy construction only ~25% slower for OBBs
  - Better culling efficiency (fewer overall tests)
  - Overall performance win (especially for continuous collision and distance queries)
Bounding volumes

- Separation distance:
  Rectangular swept spheres (widely used in PQP)
  
  - Also has expensive construction
  - Similar advantages, easy extension of OBBs

Front tracking

- Exploit temporal coherence
  - Simulations typically have small timesteps

- Store last intersecting pair for each subtree
- Next frame: still intersecting?
  - Yes: test primitives
  - No: go up in tree until intersection found

Front tracking

- Advantage
  - Less steps in intersection
  - Not necessarily less work, but results in higher parallelism

- Overall
  - ~10-25% less overall time for our benchmarks

Results

- Implemented in CUDA on NVIDIA GTX 285
  - Hierarchies built and updated fully on GPU
  - Self-collision
    - Includes collision and update of BVH per frame
    - 10-20X speedup over CPU-based algorithms

49k tris, 34ms collision
40k tris, 29ms collision
92k tris, 38ms collision
146k tris, 74ms collision
Results & Application

- Ported to NVIDIA GeForce 480 desktop GPU
  - 2.5 – 3X improvement over NVIDIA GeForce 285
- Resulting package (gProximity) is available on the WWW
- Used for real-time high DOF motion planning (gPlanner)

Real-time High DOF Motion Planning

<table>
<thead>
<tr>
<th>PRM algorithm</th>
<th>GPU algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample generation</td>
<td>Parallel sampling</td>
</tr>
<tr>
<td>Milestone construction</td>
<td>Parallel kNN query</td>
</tr>
<tr>
<td>Proximity computation</td>
<td>BVH construction</td>
</tr>
<tr>
<td>Local planning</td>
<td>Parallel BVH collision</td>
</tr>
<tr>
<td>Query connection</td>
<td>Parallel graph search</td>
</tr>
</tbody>
</table>

PRM Motion Planning on GPUs

<table>
<thead>
<tr>
<th>Environment</th>
<th>C-PRM</th>
<th>C-RRT</th>
<th>G-PRM</th>
<th>GL-PRM</th>
</tr>
</thead>
<tbody>
<tr>
<td>piano</td>
<td>6.53s</td>
<td>19.44s</td>
<td>1.71s</td>
<td>111.23ms</td>
</tr>
<tr>
<td>helicopter</td>
<td>8.20s</td>
<td>20.94s</td>
<td>2.22s</td>
<td>129.33ms</td>
</tr>
<tr>
<td>maze3d1</td>
<td>1.88s</td>
<td>21.18s</td>
<td>14.78s</td>
<td>71.24ms</td>
</tr>
<tr>
<td>maze3d2</td>
<td>69.76s</td>
<td>17.45s</td>
<td>14.47s</td>
<td>408.6ms</td>
</tr>
<tr>
<td>maze3d3</td>
<td>8.45s</td>
<td>4.3s</td>
<td>1.40s</td>
<td>96.37ms</td>
</tr>
<tr>
<td>alpha1.5</td>
<td>65.73s</td>
<td>2.8s</td>
<td>12.86s</td>
<td>1.44s</td>
</tr>
</tbody>
</table>

OOPSMP on Intel 3.2GHz i7 (single core) CPU ($600)
gPlanner on NVIDIA GTX 285 GPU ($400)

Results on PR2 robot model
Conclusions

• Collision and proximity queries
  – Deformable models
  – FEM and volumetric meshes
  – Penetration depth computation
• Parallel GPU-based algorithms
• Application to real-time motion planning

Future Work

• Need faster algorithms
• Integration with dynamics and FEM simulation packages
• Real-time planning on physical robots
• Parallelism and scalability?

Request to the Community

• Please take the effort to make your source code available

Acknowledgments

• Funding agencies:
  – NSF
  – ARO
  – DARPA/RDECOM
  – NVIDIA
  – Intel
  – Willow Garage
• Models courtesy of Disney, Kineo CAM, BMW, LS-DYNA
Thanks!