A Voxel-Based Parallel Collision Detection Algorithm

Orion Sky Lawlor
Univ. of Illinois at Urbana-Champaign

ICS 2002
Introduction

Simulated objects can occupy the same space at the same time. Physical objects can’t.
Collision Detection (Contact)

- Given a set of objects, which objects overlap?
- Equivalently, which pairs of objects overlap?
- Variations (not covered here):
  - When’s the next overlap?
  - How much do the objects overlap?
  - Which objects should be moved?
  - How much should the objects be moved?
A well-known, widely quoted, and totally irrelevant result

- For two convex polyhedra of $n$ vertices, can determine overlap in $O(lg n)$ time
Given $n$ independent objects, it takes $\Omega(n)$ time to determine if any collisions exist.

Proof via adversary argument:
- If you look at less than $n$ objects, I can change one you didn’t look at to make a collision.

Does not apply if you’ve seen the objects before—collision scheduling (bounded velocities).
Naive Collision Detection

- Test every pair of objects for intersection
- Algorithm is always $O(n^2)$
- Common useless optimization: first test object bounding boxes

Advantages:
- Easy to write
- Trivial to parallelize, if you ignore data replication
Voxel Algorithm: Motivation

- We don’t intersect if:
  - None of me touches none of you
  - None of my bounding box touches none of yours
  - I’m completely inside any box you’re completely outside

- Idea: Avoid all-pairs problem by finding a nice set of boxes
Voxel Algorithm: Idea

- Overlay regular grid (of voxels) over problem domain
- Add objects to each voxel they touch
- Collide objects in each voxel
- Advantages:
  - Immediately divide up far-away objects
  - Naturally parallel, even with data movement
Voxel Algorithm: Picture
Voxel Algorithm: Implementation

- Voxels form a sparse 3D array
- Voxels accept objects they touch from across the machine
- Voxels do serial collision detection work, so should be load balanced
- A voxel should be near its objects, for efficiency
- Perfect match for Charm++ Parallel Object Array framework
Charm++ Parallel Object Arrays

- Dynamic parallel objects scattered across the machine
- "Indexable" via sparse 3D index
- Migratable for automatic load balance
- Collective operations:
  - Broadcast (start colliding)
  - Reduction (collect collisions)
  - Intelligent creation (createhere)
Charm++ Arrays: Smart Creation

- New voxels created whenever objects are created or moved
- Default semantics: create new voxel on the same processor as its object
- Charm++ needs communication to resolve creation race
- Result: after creation, local voxels are communication-free
Serial Scaling

![Graph showing the relationship between the number of triangles and time per step, with a linear trend line.]
Parallel Scaled Problem
Conclusions

- Voxel algorithm [Turk 1989] an efficient collision detection method
  - Serial version quite fast
  - Easy to describe and implement
  - Naturally parallel
- Charm++ good foundation for sparse, dynamic parallel apps
- Collaborating on real application
Future Work

- Rather silly to use regular grid
  - Easy to compute, but poor match to objects
- Might use multiresolution grids
  - Large objects in large cells
  - Small objects in small cells
  - Must handle overlapping cells
Floating-Point Hack

- Float to int conversion is slow
- Can replace with “normalization add” followed by a cast
- [Chris Hecker 1996], others…
Floating-Point Hack in Decimal

- Start with number like 12.3
  - (Stored as $1,230,000 \times 10^{-5}$)
- Add 1,000,000.0
- Get 1,000,012.3
  - (Stored as $1,000,012 \times 10^0$)
- Note:
  - Integer part shifted to right
  - Rounding truncates result
### Floating-Point Hack Performance

<table>
<thead>
<tr>
<th>Machine</th>
<th>Normal</th>
<th>Fast</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5 GHz AMD Athlon XP</td>
<td>54 ns</td>
<td>5 ns</td>
</tr>
<tr>
<td>500 MHz Pentium 3 Xenon</td>
<td>236 ns</td>
<td>14 ns</td>
</tr>
<tr>
<td>195 MHz MIPS R10000</td>
<td>109 ns</td>
<td>21 ns</td>
</tr>
<tr>
<td>300 MHz SPARC Ultra 10</td>
<td>245 ns</td>
<td>30 ns</td>
</tr>
<tr>
<td>332 MHz PowerPC 604e</td>
<td>281 ns</td>
<td>127 ns</td>
</tr>
<tr>
<td>240 MHz PA-RISC 8200</td>
<td>648 ns</td>
<td>15 ns</td>
</tr>
</tbody>
</table>
$O(1)$ Collisions

- Sometimes we try to maintain a constant number of collisions
- Still have to look at all the objects
$O(n^2)$ Collisions

- It’s possible for every object to intersect every other object
- Pretty unlikely, though!
$O(n)$ Collisions

- Physical simulations often have linear number of collisions (bounded number per object)