Collision Detection Towards Natural Interaction in VEs

Prof. Gabriel Zachmann
Clausthal University, Germany
zach@in.tu-clausthal.de

Volkswagen Tagung, Sep 2006
**Hierarchical Collision Detection**

**Motivation**

- Observation: on most sides, child boxes almost touch parent box
- Combination of k-d tree and AABB

**Restricted Boxtrees**

- Observation: on most sides, child boxes almost touch parent box
- Combination of k-d tree and AABB

- Storage: 1 float, 1 axis ID, 1 pointer
Results

<table>
<thead>
<tr>
<th>BVH</th>
<th>Bytes / Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restricted Boxtree</td>
<td>9</td>
</tr>
<tr>
<td>Sphere Tree</td>
<td>20</td>
</tr>
<tr>
<td>AABB Tree</td>
<td>28</td>
</tr>
<tr>
<td>OBB Tree</td>
<td>52</td>
</tr>
<tr>
<td>24-DOP-Tree</td>
<td>100</td>
</tr>
</tbody>
</table>

Object-Space Coll. Detection on the GPU

- Background on stream architectures (and GPUs)
  - Stream Programming Model = "Streams of data passing through computation kernels."
  - Stream = ordered set of data of arbitrary datatype.
  - Kernel = program to be performed on each element of the input stream

Sample stream program:

```c
{   stream A, B, C;
    kernelfunc1( input: A, output: B);
    kernelfunc2( input: B, output: C);
    ...
}
```
Overview

- Simultaneous overlap testing of multiple BVs
- Implementation:
  - Stream = list of BVs = texture
  - Kernel = BV intersection test = fragment program

Dedicated Hardware for Coll.Det.

- General problem of "general purpose" computations on the GPU — competition among resources
- FPGA board (Xilinx Virtex II Pro) for prototyping
Results

- FPGA implementation has no cache yet(!)
- FPGA is much slower than ASIC (100 MHz, slow mem interface)
- With FPGA, the CPU is completely idle

Further Speedup

- Observation: absolute accuracy is often not necessary

→ New notion: approximative collision detection
- Goal: continuous and controlled balancing between running time and accuracy
- Benefit: time-critical computation


**General Approach**

- **Average-Case approach:**
  - Estimate probability of intersection for whole sets of polygons (at inner nodes of BVH)
  - BVH traversal guided by probability (P-Queue)
- **Modification of BVHs:** store *simple* description
- **Advantage** of our approach: can be applied to (almost) any kind of BVH / hierarchical collision detection
Estimating the Probability

1. Partition $A \cap B$ with grid of $s$ cells
2. Compute number of "well-filled" cells: $s_A$
3. Dito for $B$: $s_B$
4. Compute probability that $x$ cells are "well-filled" from both $A$ and $B$:

\[
Pr[c(A \cap B) \geq x] = 1 - \sum_{t=0}^{x-1} \frac{\binom{s_B}{t} \binom{s-s_B}{s-t}}{\binom{s_A}{s}}
\]

Results

<table>
<thead>
<tr>
<th>Distance</th>
<th>Time (msec)</th>
<th>Errors (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>pmin=0.99</td>
<td>pmin=0.90</td>
<td>pmin=0.80</td>
</tr>
<tr>
<td>1.2</td>
<td>0.1</td>
<td>5</td>
</tr>
<tr>
<td>1.3</td>
<td>0.2</td>
<td>6</td>
</tr>
<tr>
<td>1.4</td>
<td>0.3</td>
<td>7</td>
</tr>
<tr>
<td>1.5</td>
<td>0.4</td>
<td>8</td>
</tr>
<tr>
<td>1.6</td>
<td>0.5</td>
<td>9</td>
</tr>
<tr>
<td>1.7</td>
<td>0.6</td>
<td>10</td>
</tr>
<tr>
<td>1.8</td>
<td>0.7</td>
<td>11</td>
</tr>
<tr>
<td>1.9</td>
<td>0.8</td>
<td>12</td>
</tr>
<tr>
<td>2.0</td>
<td>0.9</td>
<td>13</td>
</tr>
</tbody>
</table>
Point Clouds

- Motivation: renaissance of points as object representation because of 3D scanners

- Goal:
  - Fast collision detection between 2 given point clouds
  - No polygonal reconstruction

Definition of the surface

- Idea: assume "distance function" $f$ from surface, then surface $S$ is

$$S = \{ x \in \mathbb{R}^3 \mid f(x) = 0 \}$$

- "Distance" function $f$ by Weighted Least Squares:

$$f(x) = n(x) \cdot (x - a(x))$$
Motivation

Visualization \( f(x) \) using Euclidean distance:

Problems:

Cause and solution:

Which neighborhood graph?
\( \rightarrow \) k-SIG (sphere-of-influence graph)
Benefits

- Much less artifacts
- Automatic, sampling-density independent detection of boundaries
- Automatic kernel bandwidth selection → handles different sampling densities automatically

CD using Point Cloud Hierarchies

Point Cloud Collision Detection

Jan Klein, Gabriel Zachmann
Eurographics 2004 – Grenoble, France
Coll. Det. of PCs using Stochastic Sampling

- Given two point clouds A and B (or subsets thereof), construct a sampling of
  \[ Z = \{ x \mid f_A(x) = f_B(x) = 0 \} \]

- Overall method:
  \[ (p_p, p_s) \in A \text{ on different sides of } B \rightarrow \text{Approx. intersection points} \rightarrow \text{Refined intersection point} \]
Results

- Theoretical complexity: $O(\log \log N)$

Kinetic Bounding Volume Hierarchies

- For collision detection of deformable objects ...
- ... but not just for collision detection!
  - Can be applied to ray-tracing, occlusion culling, etc.
- Pre-processed hierarchy becomes invalid when object deforms
  → BVH must be rebuilt or updated after deformations
Our Approach

- Observation:
  - Motion in the physical world is normally continuous
  - Changes in the combinatorial structure of the BVHs occur only at discrete time points
  
  ➔ We store only the combinatorial structure of the BVH and use an event-based approach for updating (kinetization)

### Kinetic Toy Example

Event Queue

(t1, Q, R, Max x)
Results

Shirt Scene (~ 100,000 triangles)

Total time incl. collision detection time

Total num triangles
Natural Interaction

- Direct manipulation is more intuitive and sometimes even more efficient
- Goal:
  - Model and simulate the real human hand
  - Interaction between virtual environment and virtual hand
  - Not necessarily physically correct but physically plausible
- Applications:
  - Virtual assembly Simulation
  - 3D Sketching
  - Medical surgery training

Projekt: Natürliche Objekt-Manipulation

- Plausible, realitätsnahe, Echtzeit-Simulation
- DFG und EU-Antrag (Integrated Project)
### Implementation

- 17k quad mesh hand model
- Skeletal representation
- OpenSG for visualization
- Data Collection with VRJuggler
- Physical simulation by OpenDE
- Spring model for virtual grasping
- Does not rely on heuristics to estimate user intend or grasp state

### Result (work in progress)
Real-Time Camera-Based 3D Hand Tracking

- **Goals**
  - Observe hand with cameras
  - Determine global hand position and orientation in 3d-space
  - Determine hand state, i.e. angles between fingers

Challenges

- Measurement noise
- Camera lens distortion
- Uncontrolled illumination
- Mutual occlusions of the hand
- Large working volume
- Fast hand motion
- High problem dimensionality (~ 27 DOFs)
**Approach**

- Capture hand with cameras from several viewpoints
- Preprocessing of captured images (correction of camera lens distortion, noise filtering)
- Edge detection
- Skin segmentation (bad illumination correction, skin region detection)
- Generate hand model (cylinders, triangle-mesh)
- Predict hand state
- Match hand model with image edges and skin regions in images

**Overview of the System**

1. Lens distortion correction
2. Noise filtering
3. Bad illumination correction

1. Edge detection
2. Skin color segmentation
Summary