Collision Detection

*In Simulation and Animation*

Kenny Erleben

Department of Computer Science
University of Copenhagen
Model Representations

- Polygon models
- Implicit surfaces \( f(x, y, z) = 0 \)
- Patch models (NURBS)
- Constructive Solid Geometry (CSG)
- Voxels
I like **polygons**

**Why?**

- Easy to obtain
- Easy to render
- Easy to understand
- Most people use them
- Most other model representations are easily converted into polygon models

**Why not?**

- They can be quite huge
- They are unphysical, discontinuously
- They are not scale invariant
Why is it such a difficult problem?

How hard can it be to test two triangles for overlap?

- One (very small) object = 1000 triangles
- Testing two pair of objects = 1000000 triangle tests
- Small configuration with 100 objects = \((100 \cdot 100 - 100)/2 = 4950\) object pair tests
- 4950 object pair tests = \(5 \cdot 10^6\) triangle tests
- Collision detection 100-1000 times per second = \(5 \cdot 10^9\) triangle tests per second

The problem is slightly worse because we have overlooked the time needed for rendering, state computation, AI and lots of other stuff.
Why is collision detection in simulation and animation different from collision detection in traditional computational geometry and algorithms?

<table>
<thead>
<tr>
<th>Yes - No query</th>
<th>Proximity Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Shot</td>
<td>Multiple Queries</td>
</tr>
</tbody>
</table>
Principles of Collision Detection

Three principles to make life easier

- The Approximation Principle
- The Locality Principle
- The Coherence Principle

The Fourth not so often mentioned principle

- Exploit knowledge of physics and trajectories etc.
A Typical Dynamic Simulator
A Typical Dynamic Simulator
A Typical Dynamic Simulator
A Typical Dynamic Simulator

[Diagram of a dynamic simulator showing components like collision detection, time control, motion solver, and constraint solver.]
A Typical Dynamic Simulator

[Diagram showing the process of collision detection and solving]
A Typical Dynamic Simulator
A Typical Dynamic Simulator
A Typical Dynamic Simulator
A Typical Dynamic Simulator
A Typical Dynamic Simulator
The commonly used approaches

- Exhaustive Search: Also known as “all-pair-test”, Only uses the approximation principle.
- Hierarchical Hashtables: Cells and Grids are similar algorithms, Hierarchical hashtable Uses all three principles.
- Coordinates Sorting: Also known as “Sweep N’ Prune”, Uses all three principles.

Bounding Volumes versus Sweeping Volumes

- The fourth principle
Narrow Phase Collision Detection

A Few Examples

<table>
<thead>
<tr>
<th>Spatial subdivision</th>
<th>Bounding Volume Hierarchies</th>
<th>Feature Based</th>
<th>Simplex Based</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSP-trees</td>
<td>Sphere-trees</td>
<td>VClip</td>
<td>GJK</td>
</tr>
<tr>
<td>OCTrees</td>
<td>OBB-trees</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Contact Determination (1)

From a pure geometrical perspective we know

- A Contact Area between polygon models
  - A point
  - A connected line of segments
  - A polygon

- The general case
  - Nonplanar contact areas
  - Multiple contact areas
Contact Determination (2)

Our way of representing the geometry

- Principal contacts (PCs)
  - (V-V), (V-E), (V-F), (E-E), (E-F) and (F-F)

- A Contact Formation (CF)
  - A set of PCs that represent the polygonal contact area
  - A CF is not unique
A simulator uses CF’s to compute physics, so we need to compute

- Contact Points
- Contact Normals

To be physically meaningful we typically also want

- Contact normals to be continuous

\[ \text{Discontinuity } \Rightarrow \vec{n} \cdot \vec{u} < 0 \Rightarrow \text{impact} \]

Exploiting the fourth principle to find the “minimum” CF’s, the support region.
A Simple Example

Approximating sphere trees a real hackeli-hack way of doing things.

- Classical Bounding Volume Hierarchy
- Using Crude Approximations
- Simplicity rather than performance and accuracy
Data Structure Overview

- Root encloses all the geometry of the object.
- A bounding Volume at depth $k$ encloses the geometry of its children at depth $k + 1$.
- Leaves only encloses geometry.
Traversal Algorithm (1)

Both Collision Detection and Contact Determination
Algorithm test(BV A, BV B)
    If A not overlaps B Then
        return
    End if

    If A and B are both leaves Then
        do contact determination
        return touching or penetration
    End if

    If descend to A Then
        For all A’s children C Do
            test(C, B)
        Next C
    End if

    If descend to B Then
        ....
    End if
End algorithm
The Contact Determination Problem

- The Contact Point

\[
\frac{|\vec{p} - \vec{c}_A|}{|\vec{p} - \vec{c}_B|} = \frac{r_A}{r_B}
\]

- The Contact Normal

\[
\vec{n} = \frac{\vec{c}_B - \vec{c}_A}{|\vec{c}_B - \vec{c}_A|}
\]

For more details see Dingliana and O’Sullivan
Descend or Tree Traversal Rule

- Descend to both BV’s
- Descend to BV that is most likely to prune
- Depth or Breadth first traversal?
Algorithm build(Geometry G)
    sphere = enclose(G)
    If G is big enough Then
        \((G_1, ..., G_m) = \text{split}(G)\)
        For i=1 to m Do
            Child = build\((G_i)\)
            sphere.addChild(child)
        Next i
    End if
    return sphere
End algorithm
The Enclose Problem

- Compute Minimum Volume, See Welzl
- Or just some volume

\[
\bar{c} = \frac{1}{N} \sum_{i=0}^{N-1} \bar{p}_i
\]

\[
r = \max (\bar{p}_0 - \bar{c}, \ldots, \bar{p}_{N-1} - \bar{c})
\]
The Split Problem

Find splitting plane or axe

- Covariance methods, Direction of maximum variance
- Predetermined Axes
- Others...

For more details see Gottschalk
Suggested Reading


- Thomas Möller: *A Fast Triangle-Triangle Intersection Test*,
