Game Architecture

4/7/15: Collision Detection
Collision Detection

- ‘Collision detection’ is really a geometric intersection detection problem
- Pair reduction (reducing $N^2$ object pair testing)
- Intersection testing (triangles, spheres, capsules, etc.)
Intersection Testing

- General goals: given two objects with current and previous transforms specified, determine if, where, and when the two objects intersect
  
- Alternative: given two objects with only current transforms, determine if they intersect
  
- Sometimes, we need to find all intersections.
  
- Other times, we just want the first one.
  
- Sometimes, we just need to know if the two objects intersect and don’t need the actual intersection data
Collision Traverser is a task that runs every frame.
Collision Traverser is a task that runs every frame which goes through a list of "from objects" which it knows about via addCollider()
Collision Traverser is a task that runs every frame which goes through a list of "from objects" which it knows about via `addCollider()` and compares them against all "into objects" in the scene graph.
Collision Traverser which goes through a list of "from objects" and compares them against all "into objects" in the scene graph

is a task that runs every frame which it knows about via addCollider()

Collision Bitmasks to determine if these are two objects we care about colliding (by default, we care about all "froms" into all "intos")
Collision Traverser is a task that runs every frame. It goes through a list of "from objects" which it knows about via `addCollider()` and compares them against all "into objects" in the scene graph.

First, it compares the objects' Collision Bitmasks to determine if these are two objects we care about colliding (by default, we care about all "froms" into all "intos").

If the bitmasks have at least one bit in common, test for collision via the objects'.

Collision Solids which can be added manually or to the model via a MEL script.

what do we do if the objects are colliding?
Collision Traverser is a task that runs every frame.

Collision Bitmasks which goes through a list of "from objects" and compares them against all objects in the scene graph.

First, it compares the objects' Collision Bitmasks to determine if these are two objects we care about colliding (by default, we care about all "froms" into all "intos").

If the bitmasks have at least one bit in common, test for collision via the objects' Collision Solids which can be added manually or to the model via a MEL script.

Collision Handler which the traverser knows about via addCollider().

What do we do if the objects are colliding?
Collision Traverser

which goes through a list of
"from objects"
and compares them against all
"into objects"
in the scene graph

Collision Bitmasks

First, it compares the objects'
to determine if these are two objects
we care about colliding (by default, we care about all "froms" into all "intos")

Collision Solids

If the bitmasks have at least one bit in common, test for collision via the objects'
which can be added manually or to the model via a MEL script

Collision Handler

the most common of which is to generate a
which the traverser knows about via addCollider()

Collision Event

which are "accepted" just like input
Collision Traverser is a task that runs every frame, which goes through a list of "from objects" and compares them against all "into objects" in the scene graph.

First, it compares the objects' Collision Bitmasks to determine if these are two objects we care about colliding (by default, we care about all "froms" into all "intos"). If the bitmasks have at least one bit in common, test for collision via the objects' Collision Solids, which can be added manually or to the model via a MEL script. What do we do if the objects are colliding?

Collision Handler, which the traverser knows about via addCollider(), generates a Collision Event, which carries a Collision Entry which is passed to the function specified by the accept()
Bounding Objects

- Detecting intersections with complex objects is expensive
- Provide simple object that surrounds them to cheaply cull out obvious cases
- Use for collision, picking
Bounding Sphere

• Tightest sphere that surrounds model
• For each vertex, compute distance from center, save max for radius
Bounding Box

- Tightest box that surrounds model
- Compare points to min/max vertices
- If element less/greater, set element in min/max
Axis-Aligned Bounding Box

• Box edges aligned to world axes
• Recalculate when object changes orientation
• Collision checks are cheaper
Axis-Aligned Box-Box Collision

- Compare x values in min, max vertices
- If $\text{min}_2 > \text{max}_1$ or $\text{min}_1 > \text{max}_2$, no collision (separating plane)
- Otherwise check y and z directions

![Diagram showing two axis-aligned boxes with vertices labeled min and max]
Object-Oriented Bounding Box

- Box edges aligned with local object coordinate system
- Much tighter, but collision calculations more costly
OBB Collision

- Idea: determine if separating plane between boxes exists
- Project box extent onto plane vector, test against projection between centers
Separating Axis Theorem

http://www.metanetsoftware.com/technique/tutorialA.html
Capsule

• Cylinder with hemispheres on ends
• One way to compute:
  • Calc bounding box
  • Use long axis for length
  • Next largest width for radius
Capsule

- Compact
- Only store radius, endpoints of line segment
- Oriented shape w/faster test than OBB
Capsule-Capsule Collision

- Key: swept sphere axis is line segment with surrounding radius
- Compute distance between line segments
- If less than $r_1 + r_2$, collide
Line-Line Distance

\[ \frac{|(x_1 - x_0) \cdot (n_1 \times n_0)|}{|n_1 \times n_0|} \]

- where \( x_0 \) and \( x_1 \) are points on respective lines, and \( n_0 \) and \( n_1 \) are unit direction vectors along those lines
Segment-Segment Collision

- Determine closest point between lines
- If lies on both segments, done
- Otherwise clamp against nearest endpoint and recompute
First, compute signed distances of $a$ and $c$ to plane

\[ d_a = (a - b_0) \cdot n \]
\[ d_c = (c - b_0) \cdot n \]

Reject if both are above or both are below triangle

Otherwise, find intersection point $b$

\[ b = \frac{d_a c - d_c a}{d_a - d_c} \]
Test all 3 edges

\[(b - b_0) \cdot ((b_1 - b_0) \times n) > 0\]
Linear Interpolation

\[ L = \text{LERP}(A, B, \beta) = (1 - \beta)A + \beta B \]

\[ = [(1 - \beta)A_x + \beta B_x, (1 - \beta)A_y + \beta B_y, (1 - \beta)A_z + \beta B_z] \]
\[ \mathbf{b} = \alpha \mathbf{b}_0 + \beta \mathbf{b}_1 + \gamma \mathbf{b}_2 \]

\[ \alpha + \beta + \gamma = 1 \]
Barycentric Coordinates
Barycentric Coordinates

\[ x = \alpha x_1 + \beta x_2 + \gamma x_3 \]
\[ y = \alpha y_1 + \beta y_2 + \gamma y_3 \]
\[ \gamma = 1 - \alpha - \beta \]

\[ x = \alpha x_1 + \beta x_2 + (1 - \alpha - \beta)x_3 \]
\[ y = \alpha y_1 + \beta y_2 + (1 - \alpha - \beta)y_3 \]

\[ \alpha(x_1 - x_3) + \beta(x_2 - x_3) + x_3 - x = 0 \]
\[ \alpha(y_1 - y_3) + \beta(y_2 - y_3) + y_3 - y = 0 \]

\[
\begin{vmatrix}
  x_1 - x_3 & x_2 - x_3 \\
  y_1 - y_3 & y_2 - y_3 \\
\end{vmatrix}
\begin{vmatrix}
  \alpha \\
  \beta \\
\end{vmatrix} = \mathbf{b} - \mathbf{b}_3
\]

\[
\begin{vmatrix}
  \alpha \\
  \beta \\
\end{vmatrix} = \begin{vmatrix}
  x_1 - x_3 & x_2 - x_3 \\
  y_1 - y_3 & y_2 - y_3 \\
\end{vmatrix}^{-1} \mathbf{b} - \mathbf{b}_3
\]
Barycentric Coordinates

\[ \begin{vmatrix} \alpha \\ \beta \end{vmatrix} = \begin{vmatrix} x_1 - x_3 & x_2 - x_3 \\ y_1 - y_3 & y_2 - y_3 \end{vmatrix}^{-1} (b - b_3) \]

\[ \alpha = \frac{(y_2 - y_3)(x-x_3) + (x_3-x_2)(y-y_3)}{\det(T)} = \frac{(y_2 - y_3)(x-x_3) + (x_3-x_2)(y-y_3)}{(y_2 - y_3)(x_1 - x_3) + (x_3-x_2)(y_1 - y_3)} \]

\[ \beta = \frac{(y_3 - y_1)(x-x_3) + (x_1-x_3)(y-y_3)}{\det(T)} = \frac{(y_3 - y_1)(x-x_3) + (x_1-x_3)(y-y_3)}{(y_2 - y_3)(x_1 - x_3) + (x_3-x_2)(y_1 - y_3)} \]

\[ \gamma = 1 - \alpha - \beta \]

Reject if \( \alpha < 0 \), \( \beta < 0 \) or \( \alpha + \beta > 1 \)
Segment / Mesh Collision

• To test a line segment against a mesh of triangles, simply test the segment against each triangle

• Sometimes, we are interested in only the ‘first’ hit along the segment from \( \mathbf{a} \) to \( \mathbf{b} \). Other times, we want all intersections. Still other times, we just need any intersection.

• Testing against lots of triangles in a large mesh is time consuming. Data structures can be used to optimize
Bounding interval hierarchy

From Wikipedia, the free encyclopedia

A bounding interval hierarchy (BIH) is a partitioning data structure similar to that of bounding volume hierarchies or kd-trees. Bounding interval hierarchies can be used in high performance (or real-time) ray tracing and may be especially useful for dynamic scenes.

The BIH was first presented under the name of SKD-Trees, presented by Ooi et al., and BoxTrees, independently invented by Zachmann.

Contents [hide]
1 Overview
2 Operations
   2.1 Construction
   2.2 Ray traversal
3 Properties
   3.1 Numerical stability
4 Extensions
5 See also
6 References
   6.1 Papers
7 External links

Overview [edit]

Bounding interval hierarchies (BIH) exhibit many of the properties of both bounding volume hierarchies (BVH) and kd-trees. Whereas the construction and storage of BIH is comparable to that of BVH, the traversal of BIH resemble that of kd-trees. Furthermore, BIH are also
Entries are listed from oldest to newest, so often the last entry is the best. This table covers objects not moving; see the next section for dynamic objects.

<table>
<thead>
<tr>
<th>ray</th>
<th>plane</th>
<th>sphere</th>
<th>cylinder</th>
<th>cone</th>
<th>triangle</th>
<th>AABB</th>
<th>OBB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gems p.304;</td>
<td>IRT p.50,88;</td>
<td>IRT p.39,91; Gems p.388;</td>
<td>IRT p.91; Gems IV p.356;</td>
<td>IRT p.91; Gems V p.227;</td>
<td>Möller-Trumbore jgt 2(1);</td>
<td>IRT p.65,104; Gems p.395;</td>
<td>(IRT p.10,25) Gomez GTCG p.634;</td>
</tr>
<tr>
<td>SG; RTCD p.198;</td>
<td>SG; GTCG p.482;</td>
<td>GTCG p.501; RTCD p.127,177;</td>
<td>GTCG p.507; TGS; RTCD p.194;</td>
<td>GTweb; GTCG p.512;</td>
<td>Möller-Trumbore jgt 2(1);</td>
<td>3DG p.20; Terdiman (optimized Woo); Schroeder GTCG p.626; RTCD p.179; Mahovsky jgt 9(1); Williams jgt 10(3) Lagae jgt 10(4); SoftSurfer; RTR p.578; RTR p.746; Havel TVCG June 2009</td>
<td></td>
</tr>
<tr>
<td>TGS; RTR2 p.618;</td>
<td>TGS; RTCD p.175;</td>
<td>RTR2 p.568; RTR3 p.738</td>
<td>RTR p.127,177; RTCD p.194;</td>
<td>RTR p.127,177; RTCD p.194;</td>
<td>GTCG p.485; TGS; RTCD p.513;</td>
<td>Mahovsky jgt 9(1); Williams jgt 10(3) Lagae jgt 10(4); SoftSurfer; RTR p.578; RTR p.746; Havel TVCG June 2009</td>
<td></td>
</tr>
<tr>
<td>RTR3 p.781</td>
<td>SoftSurfer (more)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>plane</th>
<th>plane</th>
<th>plane</th>
<th>plane</th>
<th>plane</th>
<th>plane</th>
<th>plane</th>
<th>plane</th>
</tr>
</thead>
</table>
The 3D Quickhull Algorithm

Dirk Gregorius – Valve Software
Implementing GJK

- http://mollyrocket.com/849
The Gilbert-Johnson-Keerthi (GJK) Algorithm

Christer Ericson
Sony Computer Entertainment America

christer_ericson@playstation.sony.com
Talk outline

• What is the GJK algorithm
• Terminology
• “Simplified” version of the algorithm
  – One object is a point at the origin
  – Example illustrating algorithm
• The distance subalgorithm
• GJK for two objects
  – One no longer necessarily a point at the origin
• GJK for moving objects
GJK solves proximity queries

• Given two convex polyhedra
  – Computes distance $d$
  – Can also return closest pair of points $P_A, P_B$
GJK solves proximity queries

- Generalized for arbitrary convex objects
  - As long as they can be described in terms of a support mapping function
Supporting (or extreme) point \( P \) for direction \( d \) returned by support mapping function \( S_C(d) \).
Terminology 2(3)

0-simplex  1-simplex  2-simplex  3-simplex

simplex
Terminology 3(3)

Point set $C$

Convex hull, $\text{CH}(C)$
The GJK algorithm

1. Initialize the simplex set \( Q \) with up to \( d+1 \) points from \( C \) (in \( d \) dimensions)
2. Compute point \( P \) of minimum norm in \( \text{CH}(Q) \)
3. If \( P \) is the origin, exit; return 0
4. Reduce \( Q \) to the smallest subset \( Q' \) of \( Q \), such that \( P \) in \( \text{CH}(Q') \)
5. Let \( V = S_C(-P) \) be a supporting point in direction \(-P\)
6. If \( V \) no more extreme in direction \(-P\) than \( P \) itself, exit; return \(||P||\)
7. Add \( V \) to \( Q \). Go to step 2
INPUT: Convex polyhedron $C$ given as the convex hull of a set of points
1. Initialize the simplex set $Q$ with up to $d+1$ points from $C$ (in $d$ dimensions)

$Q = \{Q_0, Q_1, Q_2\}$
2. Compute point $P$ of minimum norm in $CH(Q)$

$Q = \{Q_0, Q_1, Q_2\}$
3. If $P$ is the origin, exit; return 0
4. Reduce $Q$ to the smallest subset $Q'$ of $Q$, such that $P$ in CH($Q'$)
5. Let $V = S_c(-P)$ be a supporting point in direction $-P$

$$Q = \{Q_1, Q_2\}$$
6. If \( V \) no more extreme in direction \(-P\) than \( P \) itself, exit; return \( ||P|| \)

7. Add \( V \) to \( Q \). Go to step 2

\[ Q = \{ Q_1, Q_2, V \} \]
2. Compute point $P$ of minimum norm in $\text{CH}(Q)$

$Q = \{Q_1, Q_2, V\}$
3. If $P$ is the origin, exit; return 0
4. Reduce $Q$ to the smallest subset $Q'$ of $Q$, such that $P$ in $\text{CH}(Q')$
5. Let $V = S_c(-P)$ be a supporting point in direction $-P$.

$Q = \{Q_2, V\}$
GJK example 10(10)

6. If $V$ no more extreme in direction $-P$ than $P$ itself, exit; return $||P||$

$$Q = \{Q_2, V\}$$

$V$ + $P$ + $Q_2$
Distance subalgorithm 1(2)

- Approach #1: Solve algebraically
  - Used in original GJK paper
  - Johnson’s distance subalgorithm
    - Searches all simplex subsets
    - Solves system of linear equations for each subset
    - Recursive formulation
    - From era when math operations were expensive
    - Robustness problems
  - See e.g. Gino van den Bergen’s book
Distance subalgorithm 2(2)

- Approach #2: Solve geometrically
  - Mathematically equivalent
    - But more intuitive
    - Therefore easier to make robust
  - Use straightforward primitives:
    - ClosestPointOnEdgeToPoint()
    - ClosestPointOnTriangleToPoint()
    - ClosestPointOnTetrahedronToPoint()
  - Second function outlined here
    - The approach generalizes
Closest point on triangle

- ClosestPointOnTriangleToPoint()
  - Finds point on triangle closest to a given point
Closest point on triangle

- Separate cases based on which feature Voronoi region point lies in
Closest point on triangle

\[ AX \cdot AB \leq 0 \]
\[ AX \cdot AC \leq 0 \]
Closest point on triangle

\[(BC \times BA) \times BA \cdot BX \geq 0\]

\[AX \cdot AB \geq 0\]

\[BX \cdot BA \geq 0\]
GJK for two objects

• What about two polyhedra, $A$ and $B$?
• Reduce problem into the one solved
  – No change to the algorithm!
  – Relies on the properties of the Minkowski difference of $A$ and $B$

• Not enough time to go into full detail
  – Just a brief description
Minkowski sum & difference

- Minkowski sum
  - The sweeping of one convex object with another

Defined as:

\[ A + B = \left\{ a + b : a \in A, b \in B \right\} \]
Minkowski sum & difference

• Minkowski difference, defined as:
  
  \[ A - B = \{a - b : a \in A, b \in B\} \]
  
  \[ = A + (-B) \]

• Can write distance between two objects as:
  
  \[ \text{distance}(A, B) = \min \left\{ \|a - b\| : a \in A, b \in B \right\} \]

  \[ = \min \left\{ \|c\| : c \in A - B \right\} \]

• A and B intersecting iff \( A - B \) contains the origin!
  
  • Distance between A and B given by point of minimum norm in \( A - B \)!
The generalization

• A and B intersecting iff $A - B$ contains the origin!
  – Distance between A and B given by point of minimum norm in $A - B$!
• So use previous procedure on $A - B$!
• Only change needed: computing

$$S_C(d) = S_{A - B}(d)$$
• Support mapping separable, so can form it by computing support mapping for A and B separately!

$$S_C(d) = S_{A - B}(d) = S_A(d) - S_B(-d)$$
GJK for moving objects
Transform the problem...
...into moving vs stationary
Alt #1: Point duplication

Let object $A$ additionally include the points $P_i + \mathbf{v}$

…effectively forming the convex hull of the swept volume of $A$
Alt #2: Support mapping

Modify support mapping to consider only points $P_i$ when $d \cdot v \leq 0$.
Alt #2: Support mapping

...and to consider only points $P_i + \mathbf{v}$ when $\mathbf{d} \times \mathbf{v} > 0$
GJK for moving objects

• Presented solution
  – Gives only Boolean interference detection result

• Interval halving over $\mathbf{v}$ gives time of collision
  – Using simplices from previous iteration to start next iteration speeds up processing drastically

• Overall, always starting with the simplices from the previous iteration makes GJK…
  – Incremental
  – Very fast
References


- Ruspini, Diego. *gilbert.c*, a C version of the original Fortran implementation of the GJK algorithm. ftp://labrea.stanford.edu/cs/robotics/sean/distance/gilbert.c
Line-Line Distance

• Vector $\mathbf{w}$ perpendicular to $\mathbf{u}$ and $\mathbf{v}$:

$$\mathbf{w} = P_1 - Q_1$$

$$\mathbf{u} \cdot \mathbf{w} = 0$$

$$\mathbf{v} \cdot \mathbf{w} = 0$$

• Two equations

• Two unknowns
Line-Line Distance

\[ P_1 = P_0 + \frac{be - cd}{ac - b^2} \times u \]

\[ Q_1 = Q_0 + \frac{ae - bd}{ac - b^2} \times v \]

\[ a = u \cdot u \]

\[ b = u \cdot v \]

\[ c = v \cdot v \]

\[ d = u \cdot (P_0 - Q_0) \]

\[ e = v \cdot (P_0 - Q_0) \]