09: BOUNDING VOLUME HIERARCHIES

08/02/2016
RECAP: TWO-PHASE COLLISION DETECTION

Collision detection is often broken down further into broad phase (culling pairs) and narrow phase (e.g. primitive triangle intersection tests). In some cases an additional stage is employed such as progressive refinement.

Naive collision detection problems [Hubbard 1993]
Quote from havok source code comment

- Instead of checking whether the moving object is colliding with each of the triangles in the landscape in turn, which would be extremely time-consuming, the bounding box of the moving object is checked against the bounding volume tree - first, whether it is intersecting with the top-level bounding volume, then with any of its child bounding volumes, and so on until the check reaches the leaf nodes. A list of any potentially colliding triangles is then passed to the narrowphase collision detection. You can think of the bounding volume tree as a filter to the narrowphase collision detection system.

Source: hkpBvTreeShape.h © havok
BOUNDING VOLUME HIERARCHIES

© Denison Linus
BOUNDING VOLUME HIERARCHIES

© Machiel Van Hooren
BVH OVERVIEW

Overview of Main issues:

- BVH representation
- BVH generation / fitting
- BVH collision check + update and traversal
Model Hierarchy:

- each node is a simple volume that bounds a set of triangles
- BVH should be conservative over-approximations (for robustness)
- children contain volumes that each bound a different subset of the parent’s triangles

N.B. Children of each node must collectively cover all triangles covered by the parent
BVH principally designed for culling primitive level comparisons (pair-processing weakness)

- Most approaches have leaf node of BVH bounding individual primitives (usually triangles)

Above Example illustrates a binary sphere tree

- In some approaches, there may be multiple triangles in the smallest BVH node OR there may be more than one BV per triangle
BVH CONSTRUCTION

Top-down

```
A B C D
```

(a)

Bottom-up

```
A B C D
```

(b)

Insertion

```
A
```

(c)
TOP-DOWN CONSTRUCTION: SPLIT

Recursively split and bound geometric primitives

Start with top level Bounding Volume

Use split-plane

Sort using split-plane w.r.t. Triangle centroids

Find minimal boxes

And repeat...
BOTTOM-UP CONSTRUCTION: MERGE

- Wrap all primitives in leaf nodes
- Each level up (chosen based on various heuristics) needs to bound all primitives bound by children
- Parent bounds can be made to enclose child bounds for fitting speed

BOTTOM-UP CONSTRUCTION: MERGE

- Wrap all primitives in leaf nodes
- Each level up (chosen based on various heuristics) needs to bound all primitives bound by children
- Parent bounds can be made to enclose child bounds for fitting speed

BOTTOM-UP CONSTRUCTION: MERGE

- Wrap all primitives in leaf nodes
- Each level up (chosen based on various heuristics) needs to bound all primitives bound by children
- Parent bounds can be made to enclose child bounds for fitting speed

BOTTOM-UP CONSTRUCTION: MERGE

- Wrap all primitives in leaf nodes
- **Each level up (chosen based on various heuristics) needs to bound all primitives bound by children**

- Parent bounds can be made to enclose child bounds for fitting speed

BVH COLLISION DETECTION
BVH COLLISION DETECTION
BVH COLLISION DETECTION
BVH COLLISION DETECTION
BVH COLLISION DETECTION
BVH COLLISION DETECTION
BVH COLLISION DETECTION

Figure © M. Lin, UNC Chapel Hill
BVH COLLISION DETECTION

N.B. This example is Depth-first
PAIRWISE COLLISION DETECTION USING BVH

001: Check for collision between two parent nodes (starting from the roots of two given trees)

002: IF there is no interference between two parents,

003: THEN stop and report “no collision”

004: ELSE All children of one parent node are checked against all children of the other node

005: IF there is a collision between the children

006: THEN IF at leave nodes

007: THEN report “collision”

008: ELSE go to Step 4

009: ELSE stop and report “no collision”
MULTI-BODY BVH COLLISION DETECTION

Note that we cull not only collision checks but node updates

Update node only if parent is colliding

- Broadphase → potential colliding pairs
- For each pair
  - Update position/orientations of immediate children
  - Check collisions of nodes
  - Remove non-colliding nodes
  - Then traverse each colliding node (recursion)

Two traversal options: Depth first / Breadth first
Breadth first: resolve everything partially first before moving on

Depth first: pairs fully resolved before any others are processed
If we run out of time stop traversal of BVH (Interruptible)

- Any potential colliders will be treated as colliding
- Typically breadth first traversal preferred
- Not widely used with collision response due to stability issues but useful for pure intersection testing in highly complex scenes

Important question: which potential pairs do we process first
TIME CRITICAL CONTACT MODEL
TRADE-OFF IN CHOOSING BVH’S

- **Sphere**
- **AABB**
- **OBB**
- **6-dop**
- **Convex Hull**

Increasing complexity & tightness of fit

Decreasing cost of (overlap tests + BV update)

**FIGURE © M. LIN, UNC CHAPEL HILL**
COMPARING BVH APPROACHES

**Cost Function:**

\[ F = N_u C_u + N_v C_v + N_p C_p \]

where:

- \( F \): total cost function for interference detection
- \( N_u \): no. of bounding volumes updated
- \( C_u \): cost of updating a bounding volume,
- \( N_v \): no. of bounding volume pair overlap tests
- \( C_v \): cost of overlap test between 2 BVs
- \( N_p \): no. of primitive pairs tested for interference
- \( C_p \): cost of testing 2 primitives for interference

© M. LIN, UNC CHAPEL HILL
MEMORY REQUIREMENTS

Cost function: \[ M = N_i S_i + N_L S_L \]

where

\( M \) = total memory requirements in bytes
\( N_i \) = number of internal nodes
\( S_i \) = size of a single internal node in bytes
\( N_L \) = number of leaf nodes
\( S_L \) = size of a single leaf node in bytes

Optimised memory strategies:
- More than one primitive per leaf node
- Leaf node check not required
SOME BVH SCHEMES
1. SPHERE TREE

**Advantages**
- Nodes are rotationally Invariant
- Relatively low memory usage
- Some extension for deformable bodies

**Disadvantages**
- Bad fit for long/thin/flat objects

**Where used, some examples:**
- Reportedly Gran Turismo
- Unreal Tournament 2003 rag-doll: about 15 rigid bodies, spheres+capsules
1.1 UPDATE

Spheres are rotationally invariant so just update the positions of the child nodes based on the orientation of the object

Assuming c.o.m. is used as reference:

\[
\text{for } i=0 \text{ to number of children}
\]

\[
S_A.\text{child}[i].x(t) = R(t) \times S_A.\text{child}[i].x(0) + x(t);
\]

Where \(x(t)\) is position of object, \(R(t)\) is orientation matrix of object and \(S_A.\text{child}[i].x(0)\) is position of BVH node in Body space
bool CollisionCheck ( spheretree $S_A$, spheretree $S_B$)
{
    if ( distance($S_A.x(t)$, $S_B.x(t)$) < $S_B.radius + S_A.radius$ )
    {
        for i = 0 to num_children_of _S_A, denoted $S_A.child[i]$
        $S_A.child[i].x(t) = R(t) \ast S_A.child[i].x(0) + x(t)$;
        CollisionCheck ($S_B$, $S_A.child[i]$);
    }
    else return false;
}

In practice depth first recursion can be problematic (esp. For time
critical collision detection)

- Instead we might place $S_B, S_A \rightarrow child[i]$ on a potential colliders list for the next
iteration of processing
1.3 Generation: Octree-Based

**Simple technique:** Decompose object into Voxel Octree

- Relatively common data structure

**Each occupied node of the octree (an object-space ABB) is bounded by a sphere** (0.5*diagonal of octree cell)

**Quick to code BUT not very efficient fit**
1.3 GENERATION: MEDIAL AXIS-BASED

- Non-uniform cell-sizes usually provide much better fit
- Common example is to use a medial-axis (m-rep) base

Spheres are placed on the Voronoi vertices that are inside the object.

http://isg.cs.tcd.ie/spheretree/
COMPARISON

Medial-axis based sphere tree (right) tends to be about twice as efficient at bounding objects, compared to octree (left)

[Bradshaw] Sphere tree generation (code, binaries and papers): http://isg.cs.tcd.ie/spheretree
2. AABB TREES

Advantages

• Applies almost equally well to deformable objects
• Easy to compute structure
• Easy intersection tests
• Often tighter fit than sphere tree
• Low update – if object doesn’t rotate

Disadvantages

• Tree geometry varies with object rotation
• If object rotates, must recompute AABB at each frame

Generally very useful at toplevel - usefulness is debatable at finer levels (depends on application)
3. OBB TREES

**Advantages**

- Better fit than AABB and spheres
- Rotationally invariant in body space for rigid body

**Disadvantages**

- More expensive to update: must re-orient and reposition
- More expensive intersection test: **Two orders of magnitude** slower than checking two spheres for overlap
- More difficult to generate tree
For a set of polygons

- Consider their vertices
For a set of polygons
- Consider their vertices

And an arbitrary line
For a set of polygons
- Consider their vertices

And an arbitrary line
- Project the vertices onto the line
- Consider their variance (spread)
For a set of polygons

- Consider their vertices

And an arbitrary line

- Project the vertices onto the line
- Consider their variance (spread)
- Different line → different variance
For a set of polygons

- Consider their vertices

And an arbitrary line

- Project the vertices onto the line
- Consider their variance (spread)
- Different line $\rightarrow$ different variance

Given

- the maximum variance
**For a set of polygons**
- Consider their vertices

**And an arbitrary line**
- Project the vertices onto the line
- Consider their variance (spread)
- Different line → different variance

**Given**
- the maximum variance
- and the minimum variance
OBB CONSTRUCTION

For a set of polygons
- Consider their vertices

And an arbitrary line
- Project the vertices onto the line
- Consider their variance (spread)
- Different line → different variance

Given
- the maximum variance
- and the minimum variance

Find orientation of the best fitting box
based on eigenvectors of the covariance matrix of the vertices
The covariance matrix of point coordinates describes statistical spread of cloud.

The OBB is aligned with directions of greatest and least spread.
Let the vertices of the i'th triangle be the points \(a_i, b_i,\) and \(c_i,\) then the mean \(\mu\) and covariance matrix \(C\) can be expressed in vector notation as:

\[
\begin{align*}
\mu &= \frac{1}{3n} \sum_{i=0}^{n} (a_i^i + b_i^i + c_i^i) \\
c_{jk} &= \frac{1}{3n} \sum_{i=0}^{n} \left( \overline{a_j^i a_k^i} + \overline{b_j^i b_k^i} + \overline{c_j^i c_k^i} \right), \quad 1 \leq j, k \leq 3
\end{align*}
\]

where \(n\) is the number of triangles, and

\[
\begin{align*}
\overline{a_i} &= a_i - \mu, \quad \overline{b_i} = b_i - \mu, \quad \overline{c_i} = c_i - \mu
\end{align*}
\]
BUILDING AN OBB TREE

Good box
Add points: worse box
BUILDING AN OBB TREE

More points: terrible box
BUILDING AN OBB TREE

Compute with extremal points only
“Even” distribution: good box
“Uneven” distribution : bad box
BUILDING AN OBB TREE

Fix: compute facets of convex hull
BUILDING AN OBB TREE

Better: integrate over facets
BUILDING AN OBB TREE
For each primitive group, by Principal component analysis (covariance based)

- Find convex hull* of object (e.g. http://www.qhull.org/)
- Essentially a representation of the enclosing ellipsoid
- Find best BB to fit this

Building the tree: typically top-down (split operation)

Cost:
- $O(n \log n)$ fitting time for single BV
- $O(n \log^2 n)$ fitting time for entire tree
If overlapping bounding volumes A and B:

- Traverse into children of A
- Update (position and orientation) children
- Test children against B
- For each intersection repeat by recursion
**Separating axis theorem:** Two convex polytopes are disjoint iff there exists a separating axis orthogonal to a face of either polytope or orthogonal to an edge from each polytope.

**Separating Axis:** an axis on which the projections of two polytopes don’t overlap.
SEPARATING AXIS TEST (IN ONE SLIDE)

Determine Candidate Axes

Find Midpoint Separation

Find Half-length of projected intervals

Face normal directions

\[ u_0^A \quad u_1^A \quad u_2^A \]
\[ u_0^B \quad u_1^B \quad u_2^B \]

Cross product of edge directions (for boxes these are orthogonal to face normals)

\[ u_0^A \times u_0^B \quad u_0^A \times u_1^B \quad u_0^A \times u_2^B \]
\[ u_1^A \times u_0^B \quad u_1^A \times u_1^B \quad u_1^A \times u_2^B \]
\[ u_2^A \times u_0^B \quad u_2^A \times u_1^B \quad u_2^A \times u_2^B \]

Face normal directions

\[ \mathbf{C}_A \]

\[ \mathbf{C}_B \]

\[ \mathbf{L} \]

\[ S = \left| (\mathbf{C}_A - \mathbf{C}_B) \cdot \hat{l} \right| \]

\[ r_B = e_1^B |u_1^B \cdot \hat{l}| + e_2^B |u_2^B \cdot \hat{l}| + e_3^B |u_3^B \cdot \hat{l}| \]
COMPARISON

Volume-node intersection $C_v$ is one-order of magnitude slower than that for sphere trees or AABBs.

Primary advantage for OBB: Low $N_v$ and $N_p$

- In general, OBBs can bound geometry more tightly than AABB Trees and sphere trees.

Cost Function: $F = N_u x C_u + N_v x C_v + N_p x C_p$

- $F$: total cost function for interference detection
- $N_u$: no. of bounding volumes updated
- $C_u$: cost of updating a bounding volume,
- $N_v$: no. of bounding volume pair overlap tests
- $C_v$: cost of overlap test between 2 BVs
- $N_p$: no. of primitive pairs tested for interference
- $C_p$: cost of testing 2 primitives for interference
OBB VS AABB COMPARISON

OBB converges faster to the shape of the torus
K-DOP TREE: LEVEL 1

6-DOP

18-DOP

14-DOP

26-DOP

Images © Martin Held
K-DOP TREE: LEVEL 2

6-DOP

14-DOP

18-DOP

26-DOP

Images © Martin Held
K-DOP TREE: LEVEL 3

6-DOP

14-DOP

18-DOP

26-DOP
K-DOP TREE: LEVEL 4

6-DOP

18-DOP

14-DOP

26-DOP

Images © Martin Held