Physics for Game Programmers: Spatial Data Structures

Gino van den Bergen
gino@dtecta.com
“I feel like a million tonight ... ...But one at a time.”

Mae West
Collision Queries

- Find pairs of objects that are colliding now, ...
- ... or will collide over the next frame if no counter action is taken.
- Compute data for response.
Continuous Collision Queries

- Fast moving objects, such as bullets and photons (for visibility queries), need to be swept in order to catch all potential hits.
- Querying a scene by casting rays (or shapes) is an important ability.
Collision Objects

- Static environment (buildings, terrain) is typically modeled using polygon meshes.
- Moving objects (player, NPCs, vehicles, projectiles) are typically convex shapes.
- We focus on convex-mesh collisions. Mesh-mesh is hardly ever necessary.
Convex Shapes

Convex

Concave
Three Phases

- **Broad phase**: Determine all pairs of independently moving objects that potentially collide.
- **Mid phase**: Determine potentially colliding primitives in complex objects.
- **Narrow phase**: Determine contact between convex primitives.
Mid Phase

- Complex objects such as triangle meshes may be composed of lots of primitives.
- Testing all primitives will take too long, especially if only a few may be colliding.
- How to speed things up? Or rather, achieve a more output-sensitive solution?
Spatial Coherence

- Expresses the degree in which a set of primitives can be ordered based on spatial location.
- Primitives occupy only a small portion of the total space.
- A location in space is associated with a limited number of primitives.
Which Is More Coherent?
Divide & Conquer

Capture spatial coherence by using some divide & conquer scheme:

- **Space Partitioning**: subdivide space into convex cells.
- **Model Partitioning**: subdivide a set of primitives into subsets and maintain a bounding volume per subset.
Uniform Grid

- Subdivide a volume into uniform rectangular cells (voxels).
- Cells need not keep coordinates of their position.
- Position \((x, y, z)\) goes into cell
  \[
  (i, j, k) = \left( \left\lfloor \frac{x}{e_x} \right\rfloor, \left\lfloor \frac{y}{e_y} \right\rfloor, \left\lfloor \frac{z}{e_z} \right\rfloor \right)
  \]
  where \(e_x, e_y, e_z\) are the cell dimensions.
Uniform Grid (cont’d)

Two alternative strategies:

- Add a primitive to all cells that overlap the prim’s bounding box. Overlapping boxes must occupy the same cell.

- Add a primitive to the cell that contains the center of its box. Neighboring cells need to be visited for overlapping boxes, but cells contain fewer objects.
Uniform Grid (cont’d)

- Grids work well for large number of primitives of roughly equal size and density (e.g. cloth, fluids).
- For these cases, grids have $O(1)$ memory and query overhead.
Spatial Hashing

- Same as uniform grid except that the space is unbounded.
- Cell ID \((i, j, k)\) is hashed to a bucket in a hash table.
- Neighboring cells can still be visited. Simply compute hashes for \((i\pm1, j\pm1, k\pm1)\).
Spatial Hashing (cont’d)

• As for grids, spatial hashing only works well for primitives of roughly equal size and density.

• Remote cells are hashed to the same bucket, so exploitation of spatial coherence may not be that great.
Binary Space Partitioning

- A BSP is a hierarchical subdivision of space into convex cells by (hyper)planes.
**k-D Trees**

- A *k*-D trees is a BSP with axis-aligned partitioning planes.
BSP Trees versus $k$-D Trees

- $k$-D trees have smaller memory footprints and faster traversal times per node.
- BSP Trees need fewer nodes to “carve out” a proper volume representation.
- The volume enclosed by a polygon mesh can be accurately represented by a BSP.
Point Queries on a BSP

while (!node->isLeaf) {
    node = node->plane.above(p) ? node->posChild : node->negChild;
}

return node->inside;
Volume Queries on a BSP

- Fat objects may overlap multiple cells, so multiple root paths need to be traversed.
- Shrink the query object to a point and dilate the environment by adding the negate of the query object [Quake2].
- The dilated environment is better known as Configuration Space Obstacle (CSO).
Exact CSO for a Disk
Approximate CSO for a Disk
Add Auxiliary Bevel Planes
Dynamic Plane-shift BSP Trees

- What if we have query objects with different sizes?
- Keep the original BSP and offset the planes while traversing [Melax, MDK2].
- Since obstacles occur both in the + and - half-space, we offset in both directions.
Bounding Volumes

- Should fit the model as tightly as possible.
- Overlap tests between volumes should be cheap.
- Should use a small amount of memory.
- Cost of computing the best-fit bounding volume should be low.
Bounding Volumes (cont’d)

Good bounding volume types are:

- Spheres
- Axis-aligned bounding boxes (AABBs)
- Discrete-orientation polytopes (k-DOPs)
- Oriented bounding boxes (OBBs)
# Bounding Volume Types

<table>
<thead>
<tr>
<th>Bounding Volume</th>
<th>Fit</th>
<th>Test (ops)</th>
<th>Memory (scalars)</th>
<th>Best-fit Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sphere</td>
<td>poor</td>
<td>11</td>
<td>4</td>
<td>high</td>
</tr>
<tr>
<td>AABB</td>
<td>fair</td>
<td>$\leq 6$</td>
<td>6</td>
<td>low</td>
</tr>
<tr>
<td>$k$-DOP</td>
<td>good</td>
<td>$\leq 2k$</td>
<td>$2k$</td>
<td>medium</td>
</tr>
<tr>
<td>OBB</td>
<td>excellent</td>
<td>$\leq 200$</td>
<td>15</td>
<td>high</td>
</tr>
</tbody>
</table>
Why AABBS?

- Offer a fair trade-off between storage space and fit.
- Overlap tests are fast.
- Allow for fast construction and update of bounding volumes.
- Storage requirements can be further reduced through compression.
Binary AABB Tree

- Each internal node has two children.
- Each leaf node contains one primitive.
- For N primitives we have N leaf nodes and N – 1 internal nodes (2N – 1 AABBs in total).
- Best trees need not be, and usually are not, height-balanced.
Volume Queries on an AABB Tree

- Compute the volume’s AABB in the AABB tree’s local coordinate system.
- Recursively visit all nodes whose AABBs overlap the volume’s AABB.
- Test each visited leaf’s primitive against the query volume.
Sequential Tree Traversal

```java
int i = 0;
while (i != nodes.size()) {
    if (overlap(queryBox, nodes[i].aabb)) {
        if (nodes[i].isLeaf)
            primTest(nodes[i]);
        ++i;
    } else i += nodes[i].skip;
}
```
Memory Considerations

- Nodes are stored in a single array.
- Each left child is stored immediately to the right of its parent.
- The `skip` field stores the number nodes in the subtree (the number of nodes to skip if the overlap test is negative).
Ray Cast on an AABB Tree

- Returns the primitive that is stabbed by the ray earliest.
- Requires a line-segment-versus-box test.
- Each stabbed primitive shortens the ray.
- Traverse the AABB tree testing the AABB closest to the ray’s source first.
Line-segment vs. Box Test

- Use a SAT testing the box’s three principal axes and the three cross products of principal axes and the line segment.
- If the line segment is almost parallel to an axis then the cross product is close to zero and the SAT may return false negatives!!!
Line-segment Box Test (cont’d)

- The cross product rejection test has the form

\[ | r_z \times c_y - r_y \times c_z | > | r_z | \times e_y + | r_y | \times e_z, \]

where \( r \) is the ray direction, and \( c \) and \( e \) are resp. the center and extent of the AABB.

- The rounding noise in the lhs can get greater than the rhs, resulting in a false negative!!
Line-segment Box Test (cont’d)

● The lhs suffers from cancellation, if $c$ is at some distance from the origin.

● The solution is to add an upper bound for the rounding noise to the rhs, which is

$$\max(|r_z \ast c_y|, |r_y \ast c_z|) \varepsilon,$$

Here, $\varepsilon$ is the machine epsilon of the number type.
Shape Casting on an AABB Tree

- Similar to ray casting but now we need to find the primitive that is hit by a convex shape.
- Perform a ray cast on the Minkowski sum of the node’s AABB and the shape’s box:

\[ [a, b] - [c, d] = [a - d, b - c] \]
AABB Tree Construction

- AABB Trees are typically constructed top down.
- Bottom-up construction may yield good trees, but takes a lot of processing.
- Start with a set of AABBs of the primitives.
Top-down Construction

- Compute the AABB of the set of AABBs. This is the root node’s volume.
- Split the set using a plane. The plane is chosen according to some heuristic.
- AABBs that straddle the plane are added to the dominant side. (AABBs of the two sets may overlap.)
- Repeat until all sets contain one AABB.
Median Split Heuristic

- Compute the bounding box of the set of AABB center points.
- Choose the plane that splits the box in half along the longest axis.
Median Split Heuristic No Good

- Median splits may not carve out empty space really well.
Better Splitting Heuristic

- Off-center splits may do better.
Surface Area Heuristic

- Find the splitting plane that minimizes

$$\text{SA}(\text{AABB}_{\text{left}}) \cdot N_{\text{left}} + \text{SA}(\text{AABB}_{\text{right}}) \cdot N_{\text{right}}$$

- Here, SA(AABB) is the surface area of the box, and N is the number of primitives.
Surface Area Heuristic (cont’d)

- Determining the best SAH splitting plane can be computationally expensive.
- Sufficiently-good splitting planes can be found quickly by using a binning technique:
  - Evaluate a pre-defined set of splitting planes, and pick the best one.
Surface Area Heuristic (cont’d)

- Group primitives per cell and compute the AABB of the group.
- Compute $\text{AABB}_{\text{left}}$ and $\text{AABB}_{\text{right}}$ for each splitting plane from these cell AABBs, and compute their SA.
Top-down Construction Demo
Updating AABB Trees

- AABB trees can be updated rather than reconstructed for deformable meshes.
- First recompute the AABBs of the leaves.
- Work your way back to the root: A parent’s box is the AABB of the children’s boxes.
- For extreme deformations reconstruction is better and may not be that slow [Wald].
Compressed AABB Trees [Gomez]

- Only 6 of the 12 faces of the child AABBs differ from the parent’s faces.
- Pack the children paired, and only store the coordinates of these new inner faces.
- For each of the 6 inner faces store a bit to denote which child it belongs to.
Compressed AABB Trees [Gomez]

- Inner faces are shared among children.
- Sharing need not be even.
Compressed AABB Trees [Gomez]

- Encode each of the 6 inner face coordinates by a single byte.
- The byte represents the coordinate as a fraction of the parent’s AABB.
- Lower bounds are rounded down. Upper bounds are rounded up.
- This reduces the memory footprint from 56 to 16 bytes per primitive.
Boxtree [Zachmann]

- Since the set of primitives is split along the longest axis, only one of each child’s faces will differ significantly from its parent’s.

This face is close to the box’s parent

This face differs significantly from the box’s parent.
Boxtree [Zachmann]

- Store only the coordinate for the inner faces (similar to k-d tree.)
- The other coordinates are inherited from the parent box.
Boxtree [Zachmann]

The Boxtree (aka Bounding-Interval Hierarchy) has a few benefits over traditional AABB trees:

- Smaller memory footprint.
- Slightly faster build times.
- Faster traversal times due to the fact that the number of axes in the SAT can be further reduced.
- Empty space is captured less greedily, so YMMV!
References


References


Thank You!

My pursuits can be traced on:

- Web: http://www.dtecta.com
- Twitter: @dtecta
- Or just mail me...