

CS-277

experimental haptics

"Collision Detection"

Christopher Sewell


## Outline

- Why do we need collision detection? What kinds of collision tests do we need?
  - Including motivational videos!
- Collision tests for primitives
  - Ray/segment to triangle
  - Triangle to triangle
- Hierarchical Bounding Volumes
  - Intuition
  - Building a hierarchy
  - Using two HBVs to search for a collision
  - Implementing a HBV
    - Choosing a bounding volume
    - Sphere trees, Axis-aligned bounding boxes, Oriented Bounding Boxes
- Alternatives to HBVs
  - Spatial Partitioning
  - Minkowski Differences and Proximity Queries
- Collision Response
- 6DOF Collision Detection and Response
- Resources

## Why do we need collision detection? (1)

- Line segment to triangular mesh collision checker for god-object/proxy

- create a segment between god-object and goal position.
- search for a collision between segment and environment.
  - if no collision occurs, move god-object to goal position and exit.
  - if collision occurs, move god-object to collision point.
- project goal position onto plane constraint (goal position).
- create a segment between god-object and new goal position'.
- search for a collision between segment and environment.
  - if no collision occurs, move god-object to goal position' and exit.
  - if collision occurs, move god-object to collision point'.
- project goal position onto plane constraint (goal position)'.
- create a segment between god-object and new goal position''.
- search for a collision between segment and environment.
  - if no collision occurs, move god-object to goal position'' and exit.
  - if collision occurs, move god-object to collision point''.

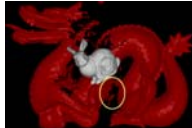


- god-object is now constrained at a single point.
- exit.

## Why do we need collision detection? (2)

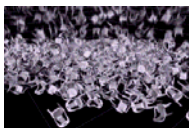
- Triangular mesh to triangular mesh collision checker for dynamic virtual environments

Important for a number of practical applications, such as



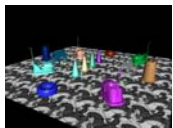
Throwing bunnies at dragons

Naga Govindaraju, Stephane Redon, Ming C. Lin and Dinesh Manocha



Dropping 12,000 lawn chairs

Doug James and Dinesh Pai




Bumper Cars

Kenneth E. Hoff III, Andrew Zafarakis, Ming Lin, Dinesh Manocha

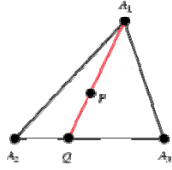
## Primitive Tests

- Regardless of other optimizations, we will usually still need to be able to perform the primitive geometrical tests
  - Does a given line segment intersect a given triangle?
  - Does a given triangle intersect another given triangle?
- Because of their importance in graphics, much work has been done in optimizing segment (or ray) to triangle and triangle to triangle collision tests
  - Ray - triangle intersections are important in ray-tracing algorithms
  - Triangle-triangle intersections are important for animated scenes
- A ray-triangle collision checker can be used for segment-triangle collision checking
  - If there is no intersection with the ray (with its origin at one of the segment's endpoints), there is no intersection with the segment
  - If there is an intersection, and it returns the intersection point closest to the ray's origin, then there is an intersection with the segment if and only if the distance from the origin to the collision point is less than or equal to the distance between the segment endpoints (actually compare squared distances to avoid costly square roots)
- The following primitive tests were published by Tomas Moller (as used in CHAI)



## Primitive Test : Segment - Triangle (1)

Barycentric coordinates: The barycentric coordinates of a point P in a triangle are masses  $t_1, t_2, t_3$  placed at the vertices  $v_1, v_2, v_3$  such that P is the geometric centroid of the masses



$$t_3 \overline{QA_3} = t_2 \overline{QA_2}$$

$$t_1 \overline{PA_1} = (t_2 + t_3) \overline{PQ}$$

$$t_1 + t_2 + t_3 = 1$$

To calculate the barycentric coordinates for a point P, first find the intersection of the line through  $A_1$  and P with the edge between  $A_2$  and  $A_3$ , and determine  $t_2$  and  $t_3$  as the relative masses needed to balance that edge at fulcrum Q. Then calculate  $t_1$  as the mass needed at  $A_1$  to balance a mass  $t_2+t_3$  at Q with fulcrum P. Then normalize the  $t$ 's so that they sum to one. (The balancing masses can be computed just as ratios of lengths.)

"Fast, minimum storage ray-triangle intersection test", Thomas Moller, Journal of Graphics Tools, Vol. 2 Issue 1, 1997, pp. 21-28.

## Primitive Test : Segment - Triangle (2)

Parametric equation of a ray ( $t \geq 0$ ) with origin  $O$  and direction  $D$

$$R(t) = O + tD$$

Parametric equation of a triangle using barycentric coordinates ( $u \geq 0, v \geq 0, u+v \leq 1$ )

$$T(u, v) = (1-u-v)V_0 + uV_1 + vV_2$$

To find where these intersect, just set them equal

$$O + tD = (1-u-v)V_0 + uV_1 + vV_2$$

Yielding the system of three linear equations

$$\begin{bmatrix} -D & V_1 - V_0 & V_2 - V_0 \end{bmatrix} \begin{bmatrix} t \\ u \\ v \end{bmatrix} = O - V_0$$

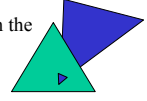
By Cramer's Rule,

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{\begin{vmatrix} -D & V_1 - V_0 & V_2 - V_0 \end{vmatrix}} \begin{bmatrix} \begin{vmatrix} O - V_0 & V_1 - V_0 & V_2 - V_0 \\ -D & O - V_0 & V_2 - V_0 \\ -D & V_1 - V_0 & O - V_0 \end{vmatrix} \\ \begin{vmatrix} O - V_0 & V_1 - V_0 & V_2 - V_0 \\ -D & O - V_0 & V_2 - V_0 \\ -D & V_1 - V_0 & O - V_0 \end{vmatrix} \\ \begin{vmatrix} O - V_0 & V_1 - V_0 & V_2 - V_0 \\ -D & O - V_0 & V_2 - V_0 \\ -D & V_1 - V_0 & O - V_0 \end{vmatrix} \end{bmatrix}$$

To calculate the determinates, you can use the formula  $[A \ B \ C] = (A \times B) \cdot C$   
If  $0 \leq u, 0 \leq v$ , and  $u+v \leq 1$ , there is an intersection. To get the Cartesian coordinates of the intersection point, just substitute  $t$  into the parametric ray equation.

## Primitive Test : Triangle – Triangle (1)

- 1) Compute plane equation of second triangle
- 2) Return false if all vertices of first triangle are on the same side of second triangle's plane
- 3) Compute plane equation of first triangle
- 4) Return false if all vertices of second triangle are on the same side of first triangle's plane
- 5) Compute line of intersection between two planes
- 6) Compute the intervals of each triangle along this line
- 7) Return true if the two intervals overlap; otherwise return false



"A fast triangle-triangle intersection test", Thomas Moller, Journal of Graphics Tools, Vol. 2 Issue 2, 1997, pp. 25-30.

## Primitive Test : Triangle – Triangle (2)

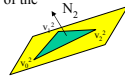
To find the plane  $N_2 \cdot X + d_2 = 0$  for the second triangle,

compute the normal by taking the cross product of two of the edges

$$N_2 = (V_1^2 - V_0^2) \times (V_2^2 - V_0^2)$$

And the constant by substituting a point on the plane (one of the vertices) back into the plane equation

$$d_2 = -N_2 \cdot V_0^2$$



- 1) Compute plane equation of second triangle
- 2) Return false if all vertices of first triangle are on the same side of second triangle's plane
- 3) Compute plane equation of first triangle
- 4) Return false if all vertices of second triangle are on the same side of first triangle's plane
- 5) Compute line of intersection between two planes
- 6) Compute the intervals of each triangle along this line
- 7) Return true if the two intervals overlap; otherwise return false

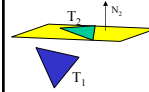
## Primitive Test : Triangle – Triangle (3)

We can compute the signed distance of each vertex on the first triangle  $V_i^1, i=1..3$ , from the plane of the second triangle by substituting them into the plane equation

$$d_{V_i^1} = N_2 \cdot V_i^1 + d_2$$

$$\text{If } \text{sgn}(d_{V_1^1}) = \text{sgn}(d_{V_2^1}) = \text{sgn}(d_{V_3^1})$$

then all vertices of the first triangle are on the same side of the second triangle's plane, and there can be no intersection



- 1) Compute plane equation of second triangle
- 2) Return false if all vertices of first triangle are on the same side of second triangle's plane
- 3) Compute plane equation of first triangle
- 4) Return false if all vertices of second triangle are on the same side of first triangle's plane
- 5) Compute line of intersection between two planes
- 6) Compute the intervals of each triangle along this line
- 7) Return true if the two intervals overlap; otherwise return false

## Primitive Test : Triangle – Triangle (4)

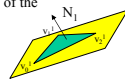
To find the plane  $N_1 \cdot X + d_1 = 0$  for the first triangle,

compute the normal by taking the cross product of two of the edges

$$N_1 = (V_1^1 - V_0^1) \times (V_2^1 - V_0^1)$$

And the constant by substituting a point on the plane (one of the vertices) back into the plane equation

$$d_1 = -N_1 \cdot V_0^1$$



- 1) Compute plane equation of second triangle
- 2) Return false if all vertices of first triangle are on the same side of second triangle's plane
- 3) Compute plane equation of first triangle
- 4) Return false if all vertices of second triangle are on the same side of first triangle's plane
- 5) Compute line of intersection between two planes
- 6) Compute the intervals of each triangle along this line
- 7) Return true if the two intervals overlap; otherwise return false

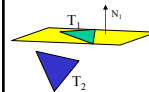
## Primitive Test : Triangle – Triangle (5)

As we've seen before, we can compute the signed distance of each vertex on the second triangle  $V_i^2, i=1..3$ , from the plane of the first triangle by substituting them into the plane equation

$$d_{V_i^2} = N_1 \cdot V_i^2 + d_1$$

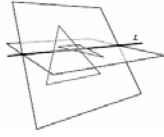
$$\text{If } \text{sgn}(d_{V_1^2}) = \text{sgn}(d_{V_2^2}) = \text{sgn}(d_{V_3^2})$$

then all vertices of the second triangle are on the same side of the first triangle's plane, and there can be no intersection



- 1) Compute plane equation of second triangle
- 2) Return false if all vertices of first triangle are on the same side of second triangle's plane
- 3) Compute plane equation of first triangle
- 4) Return false if all vertices of second triangle are on the same side of first triangle's plane
- 5) Compute line of intersection between two planes
- 6) Compute the intervals of each triangle along this line
- 7) Return true if the two intervals overlap; otherwise return false

## Primitive Test : Triangle – Triangle (6)



Unless the triangles are coplanar (in which case all  $d_{V_i}$  will be 0), the planes of the triangles will intersect in a line with parametric equation

$$L = O + tD$$

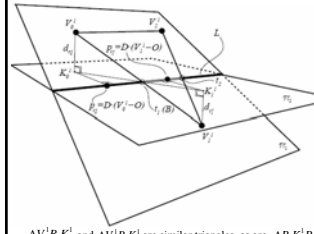
The direction of the line,  $D$ , is orthogonal to both planes' normals and so can be computed as

$$D = N_1 \times N_2$$

$O$  is a point on the line but will not need to be computed

- 1) Compute plane equation of second triangle
- 2) Return false if all vertices of first triangle are on the same side of second triangle's plane
- 3) Compute plane equation of first triangle
- 4) Return false if all vertices of second triangle are on the same side of first triangle's plane
- 5) Compute line of intersection between two planes
- 6) Compute the intervals of each triangle along this line
- 7) Return true if the two intervals overlap; otherwise return false

## Primitive Test : Triangle – Triangle (7)



Consider the first triangle; computing the interval for the second triangle is exactly analogous

Project each vertex onto the line  $L = O + tD$  by computing  $P_{V_i} = O + D(D \cdot (V_i - O))$  for  $i=1,3$ .

These projected points are points on  $L$  with  $t = P_{V_i} - O \cdot (V_i - O)$ . We are only concerned about the relative positions of the projected points, so we can ignore  $O$ .

The projections of the vertices of the first triangle onto the second triangle's plane are  $K_1$  (need not be computed)

Let  $V_0$  and  $V_2$  be the two vertices on the same side of  $L$ ; we want to find  $B_1$ , the intersection of  $V_0V_2$  with  $L$ , and  $B_2$ , the intersection of  $V_1V_2$  with  $L$ .

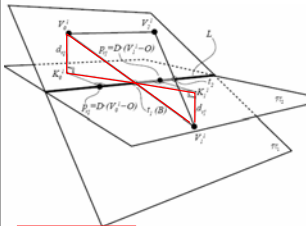
$\Delta V_0B_1K_0$  and  $\Delta V_1B_1K_1$  are similar triangles, as are  $\Delta B_1K_0P_0$  and  $\Delta B_1K_1P_1$ . Since  $B_1$  is on  $L$ ,  $B_1 = O + t_1D$ .

Therefore, 
$$\frac{t_1 - P_{V_0}}{P_{V_1} - P_{V_0}} = \frac{d_{V_0}}{d_{V_1} - d_{V_0}} \quad \text{Solving for } t_1, \quad t_1 = P_{V_0} + \left( P_{V_1} - P_{V_0} \right) \frac{d_{V_0}}{d_{V_1} - d_{V_0}}$$

- 1) Compute plane equation of second triangle
- 2) Return false if all vertices of first triangle are on the same side of second triangle's plane
- 3) Compute plane equation of first triangle
- 4) Return false if all vertices of second triangle are on the same side of first triangle's plane
- 5) Compute line of intersection between two planes
- 6) Compute the intervals of each triangle along this line
- 7) Return true if the two intervals overlap; otherwise return false

In the same way, for  $B_2$ , we get a  $t_2$  and now have the interval where the first triangle intersects  $L$ .

## Primitive Test : Triangle – Triangle (7)



Consider the first triangle; computing the interval for the second triangle is exactly analogous

Project each vertex onto the line  $L = O + tD$  by computing  $P_{V_i} = O + D(D \cdot (V_i - O))$  for  $i=1,3$ .

These projected points are points on  $L$  with  $t = P_{V_i} - O \cdot (V_i - O)$ . We are only concerned about the relative positions of the projected points, so we can ignore  $O$ .

The projections of the vertices of the first triangle onto the second triangle's plane are  $K_1$  (need not be computed)

Let  $V_0$  and  $V_2$  be the two vertices on the same side of  $L$ ; we want to find  $B_1$ , the intersection of  $V_0V_2$  with  $L$ , and  $B_2$ , the intersection of  $V_1V_2$  with  $L$ .

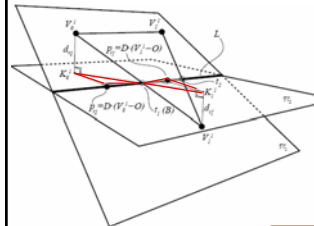
$\Delta V_0B_1K_0$  and  $\Delta V_1B_1K_1$  are similar triangles, as are  $\Delta B_1K_0P_0$  and  $\Delta B_1K_1P_1$ . Since  $B_1$  is on  $L$ ,  $B_1 = O + t_1D$ .

Therefore, 
$$\frac{t_1 - P_{V_0}}{P_{V_1} - P_{V_0}} = \frac{d_{V_0}}{d_{V_1} - d_{V_0}} \quad \text{Solving for } t_1, \quad t_1 = P_{V_0} + \left( P_{V_1} - P_{V_0} \right) \frac{d_{V_0}}{d_{V_1} - d_{V_0}}$$

- 1) Compute plane equation of second triangle
- 2) Return false if all vertices of first triangle are on the same side of second triangle's plane
- 3) Compute plane equation of first triangle
- 4) Return false if all vertices of second triangle are on the same side of first triangle's plane
- 5) Compute line of intersection between two planes
- 6) Compute the intervals of each triangle along this line
- 7) Return true if the two intervals overlap; otherwise return false

In the same way, for  $B_2$ , we get a  $t_2$  and now have the interval where the first triangle intersects  $L$ .

## Primitive Test : Triangle – Triangle (7)



Consider the first triangle; computing the interval for the second triangle is exactly analogous

Project each vertex onto the line  $L = O + tD$  by computing  $P_{V_i} = O + D(D \cdot (V_i - O))$  for  $i=1,3$ .

These projected points are points on  $L$  with  $t = P_{V_i} - O \cdot (V_i - O)$ . We are only concerned about the relative positions of the projected points, so we can ignore  $O$ .

The projections of the vertices of the first triangle onto the second triangle's plane are  $K_1$  (need not be computed)

Let  $V_0$  and  $V_2$  be the two vertices on the same side of  $L$ ; we want to find  $B_1$ , the intersection of  $V_0V_2$  with  $L$ , and  $B_2$ , the intersection of  $V_1V_2$  with  $L$ .

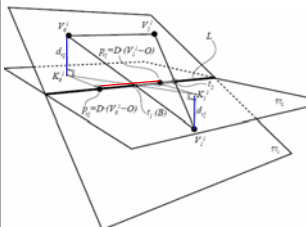
$\Delta V_0B_1K_0$  and  $\Delta V_1B_1K_1$  are similar triangles, as are  $\Delta B_1K_0P_0$  and  $\Delta B_1K_1P_1$ . Since  $B_1$  is on  $L$ ,  $B_1 = O + t_1D$ .

Therefore, 
$$\frac{t_1 - P_{V_0}}{P_{V_1} - P_{V_0}} = \frac{d_{V_0}}{d_{V_1} - d_{V_0}} \quad \text{Solving for } t_1, \quad t_1 = P_{V_0} + \left( P_{V_1} - P_{V_0} \right) \frac{d_{V_0}}{d_{V_1} - d_{V_0}}$$

- 1) Compute plane equation of second triangle
- 2) Return false if all vertices of first triangle are on the same side of second triangle's plane
- 3) Compute plane equation of first triangle
- 4) Return false if all vertices of second triangle are on the same side of first triangle's plane
- 5) Compute line of intersection between two planes
- 6) Compute the intervals of each triangle along this line
- 7) Return true if the two intervals overlap; otherwise return false

In the same way, for  $B_2$ , we get a  $t_2$  and now have the interval where the first triangle intersects  $L$ .

## Primitive Test : Triangle – Triangle (7)



Consider the first triangle; computing the interval for the second triangle is exactly analogous

Project each vertex onto the line  $L = O + tD$  by computing  $P_{V_i} = O + D(D \cdot (V_i - O))$  for  $i=1,3$ .

These projected points are points on  $L$  with  $t = P_{V_i} - O \cdot (V_i - O)$ . We are only concerned about the relative positions of the projected points, so we can ignore  $O$ .

The projections of the vertices of the first triangle onto the second triangle's plane are  $K_1$  (need not be computed)

Let  $V_0$  and  $V_2$  be the two vertices on the same side of  $L$ ; we want to find  $B_1$ , the intersection of  $V_0V_2$  with  $L$ , and  $B_2$ , the intersection of  $V_1V_2$  with  $L$ .

$\Delta V_0B_1K_0$  and  $\Delta V_1B_1K_1$  are similar triangles, as are  $\Delta B_1K_0P_0$  and  $\Delta B_1K_1P_1$ . Since  $B_1$  is on  $L$ ,  $B_1 = O + t_1D$ .

Therefore, 
$$\frac{t_1 - P_{V_0}}{P_{V_1} - P_{V_0}} = \frac{d_{V_0}}{d_{V_1} - d_{V_0}} \quad \text{Solving for } t_1, \quad t_1 = P_{V_0} + \left( P_{V_1} - P_{V_0} \right) \frac{d_{V_0}}{d_{V_1} - d_{V_0}}$$

- 1) Compute plane equation of second triangle
- 2) Return false if all vertices of first triangle are on the same side of second triangle's plane
- 3) Compute plane equation of first triangle
- 4) Return false if all vertices of second triangle are on the same side of first triangle's plane
- 5) Compute line of intersection between two planes
- 6) Compute the intervals of each triangle along this line
- 7) Return true if the two intervals overlap; otherwise return false

In the same way, for  $B_2$ , we get a  $t_2$  and now have the interval where the first triangle intersects  $L$ .

## Primitive Test : Triangle – Triangle (8)

Assuming  $t_1^1 < t_2^1$  and  $t_1^2 < t_2^2$ ,

```

if (  $t_2^1 < t_2^2$  ) or (  $t_2^2 < t_1^1$  )
    return false
else
    return true

```

- 1) Compute plane equation of second triangle
- 2) Return false if all vertices of first triangle are on the same side of second triangle's plane
- 3) Compute plane equation of first triangle
- 4) Return false if all vertices of second triangle are on the same side of first triangle's plane
- 5) Compute line of intersection between two planes
- 6) Compute the intervals of each triangle along this line
- 7) Return true if the two intervals overlap; otherwise return false

## Primitive Test : Triangle – Triangle (9)

- If triangles are coplanar ( $d_{vi} = 0$  for  $i=1..3$ )
  - Test for intersection between any edge of first triangle with any edge of second triangle (total 9 tests)
    - Compute intersection between two lines  $O_1+t_1D_1$  and  $O_2+t_2D_2$  by setting them equal and solving for  $t_1$  and  $t_2$  by creating a system of two equations from the x and y components
    - Determine if intersection point is within both edge line segments
    - If any edges intersect, the triangles intersect
  - Test if first triangle is entirely within second triangle or vice versa
    - Test if each vertex of first triangle is inside second triangle
    - One way is to find barycentric coordinates for each vertex of the first triangle with respect to the second; if  $t_1$ ,  $t_2$ , and  $t_3$  are all positive, the point is inside the triangle
    - If any vertex of one triangle is within the other, the triangles intersect
- If neither of the above finds an intersection, the triangles do not intersect



## Brute Force

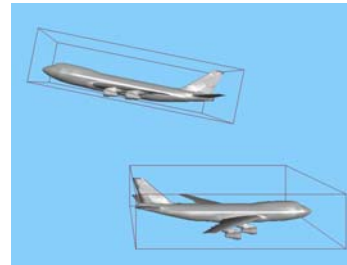
- Segment – mesh
  - for each triangle
    - if (segment intersects triangle)
      - return true
  - return false
- Mesh - mesh
  - for each triangle in first mesh
    - for each triangle in second mesh
      - if (triangle intersects triangle)
        - > return true
    - return false
- Works, but...
  - Segment-triangle way too slow for 1 kHz haptics (for mesh with 100,000 triangles, up to  $3 \times 100,000 \times 1000$  collision checks per second)
  - Triangle-triangle way too slow for anything real-time (for two meshes each with 100,000 triangles, up to  $100,000 \times 100,000$  primitive collision checks per iteration)
  - Worst performance in common case of no intersections

## Hierarchical Bounding Volumes : Intuition (1)



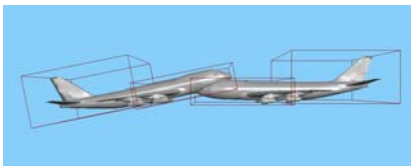
Do we *really* need to test all pairs of triangles from the two meshes here to determine that the planes do not intersect?

## Hierarchical Bounding Volumes : Intuition (2)



**No!** If we precompute a bounding volume for each mesh, and the bounding volumes do not intersect (easy test), then return false.

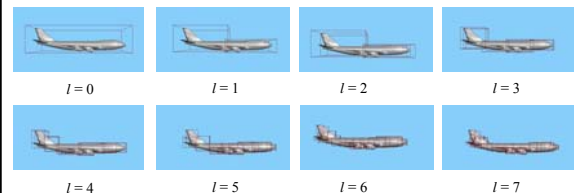
## Hierarchical Bounding Volumes : Intuition (3)



Here, the bounding boxes of the two airplanes intersect (and in fact the meshes intersect, tragically), but do we really need to be testing the triangles in the tail of one airplane against the triangles in the tail of the other?

What if we had a second level with two bounding boxes for each mesh? Then we would know only to test pairs of triangles in the front halves of the planes since only those boxes intersect, resulting in  $\sim (n/2) \times (m/2) = (nm/4)$  instead of  $nm$  collisions (75% savings)

## Axis-Aligned Bounding Boxes



Big idea: hierarchical tree of bounding volumes

You can easily apply rigid-body transformations to the BVHs

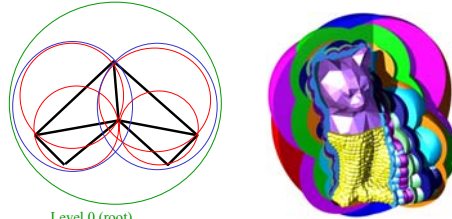
Shown here are the first 8 levels of a bounding volume tree in CHAI using axis-aligned boxes as the bounding volumes

## Bounding Spheres



You can use many many different types of shapes for bounding volumes  
We will discuss trade-offs shortly

## Overview of Building a Hierarchy



Level 0 (root)  
Level 1  
Level 2 (leaves)

- Top-down – Start with root enclosing all triangles; each node then recursively partition triangles between its children
- Bottom-up – Start with a leaf node for each triangle, then at each level merge children into a parent

## Using Two Bounding Volume Hierarchies to Search for Collision (1)



BVH of object 1      BVH of object 2



In general, the two trees may be of different sizes

Based on an example by Jean-Claude Latombe

## Using Two Bounding Volume Hierarchies to Search for Collision (2)

Search tree

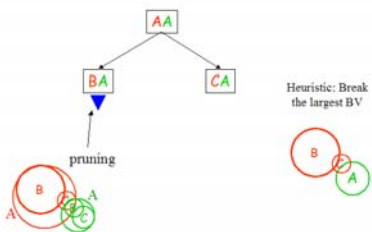
AA

Heuristic: Break the largest BV



Based on an example by Jean-Claude Latombe

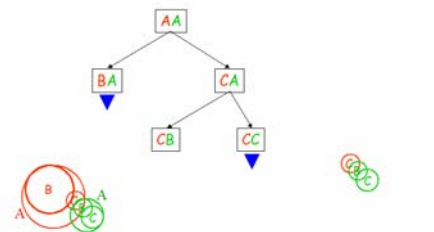
## Using Two Bounding Volume Hierarchies to Search for Collision (3)



Heuristic: Break the largest BV

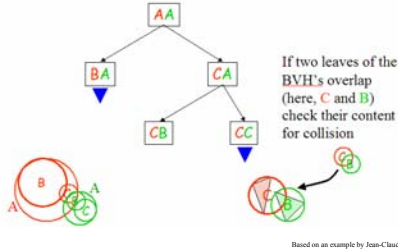
Based on an example by Jean-Claude Latombe

## Using Two Bounding Volume Hierarchies to Search for Collision (4)

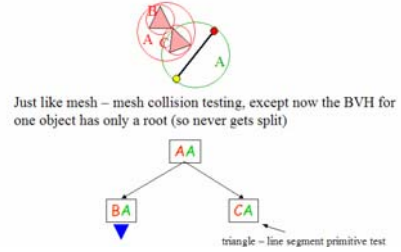


Based on an example by Jean-Claude Latombe

## Using Two Bounding Volume Hierarchies to Search for Collision (5)

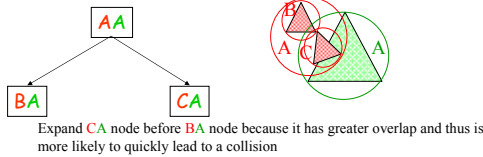


## Using Bounding Volume Hierarchies for Segment – Mesh Collision Testing



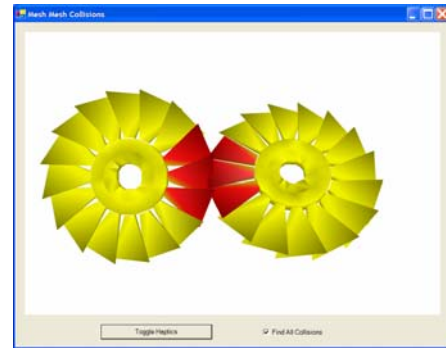
## Analyzing the Algorithm

- If objects have little or no overlap, search tree will be pruned early, resulting in little work
- If you only need to know if there is a collision or not, you can explore the search tree “greedily”, choosing to preferentially expand the nodes with greatest bounding volume overlap, and may quickly find a pair of leaves with intersecting triangles and terminate



- For some applications, such as the god-object, you need to find closest collision (to old god-object), so you can't terminate after finding one collision

## CHAI “Mesh-Mesh Collisions” Example



## Mesh Collisions Implementation (1)

```
class cCollisionStack {
public:
    cCollisionSpheresSphere* node1;
    cCollisionSpheresSphere* node2;
    cMesh* mesh1;
    cMesh* mesh2;
    float dist;
};

int cCollisionMeshMesh::collisionSearch(cCollisionSpheresSphere* a_node1, cCollisionSpheresSphere* a_node2,
    cMesh* a_mesh1, cMesh* a_mesh2, std::vector<triangle>* aa_tril, std::vector<triangle>* aa_trig)
{
    // Initialize priority queue, inserting a node containing the roots of the two sphere trees
    cCollisionStack collision_stack;
    collision_stack.node1 = a_node1; collision_stack.mesh1 = a_mesh1;
    collision_stack.node2 = a_node2; collision_stack.mesh2 = a_mesh2;
    collision_stack.dist = 0.0;
    std::priority_queue<cCollisionStack> stack;
    stack.push(collision_stack);

    int hit = 0;
    while (!((stack.empty()) && ((hit == 0) || (m_findAll == 1))))
    {
        cCollisionStack cur_struct = stack.top();
        stack.pop();

        // Get global coordinates of the two trees' centers
        cVector3d center1 = cMul(cur_struct.mesh1->getGlobalRot(), cur_struct.node1->getCenter());
        cVector3d center2 = cMul(cur_struct.mesh2->getGlobalRot(), cur_struct.node2->getCenter());
        center1.add(cur_struct.mesh1->getGlobalPos());
        center2.add(cur_struct.mesh2->getGlobalPos());

        // If the distance between the centers is greater than the sum of the radii,
        // there can be no intersection between these two trees
        float distsq = (float)(cDistanceSq(center1, center2));
        float radius_sum = (float)(cur_struct.node1->getRadius() + cur_struct.node2->getRadius());
        radius_sum = radius_sum*radius_sum;
        if (distsq > radius_sum)
            continue;
    }
}
```

## Mesh Collisions Implementation (2)

```
// If both nodes are leaves, we need to do a triangle-triangle primitive test
// between the triangles associated with these leaves
if ((cur_struct.node1->isLeaf()) && (cur_struct.node2->isLeaf()))
{
    cCollisionSpheresLeaf* leaf1 = (cCollisionSpheresLeaf*)cur_struct.node1;
    cCollisionSpheresLeaf* leaf2 = (cCollisionSpheresLeaf*)cur_struct.node2;
    cCollisionSpheresTri* prim1 = (cCollisionSpheresTri*)leaf1->m_prim;
    cCollisionSpheresTri* prim2 = (cCollisionSpheresTri*)leaf2->m_prim;
    int test = primitiveTest(prim1->getOriginal(), prim2->getOriginal());

    // If these triangles do intersect, return them
    if (test == 1)
    {
        m_lastContact1 = prim1->getOriginal();
        m_lastContact2 = prim2->getOriginal();
        a_tril.push_back(m_lastContact1);
        a_tril.push_back(m_lastContact2);
        hit = 1;
        if (!m_findAll) return 1;
    }
    continue;
}
```

## Mesh Collisions Implementation (3)

```
// If the first node is a leaf and the second is an internal node, recurse by
// comparing the leaf to the left and to the right subtrees of the internal node
if ((cur_struct.node1->isleaf()) && !(cur_struct.node2->isleaf()))
{
    // Compare leaf node 1 to left child of node 2
    cCollisionSphereNode* internal2 = (cCollisionSphereNode*)cur_struct.node2;
    cCollisionStack next_struct;
    next_struct.node1 = internal2->m_left;
    next_struct.node2 = cur_struct.node1;
    next_struct.mesh1 = cur_struct.mesh2;
    next_struct.mesh2 = cur_struct.mesh1;

    // Get the overlap between these two spheres
    cVector3d c2 = cMul(cur_struct.mesh2->getGlobalRot(), internal2->m_left->getCenter());
    c2.add(cur_struct.mesh2->getGlobalPos());
    float r = (float)(cur_struct.mesh1->getRadius() + internal2->m_left->getRadius());
    next_struct.dist = cDistance(c2, center1) - r*r;
    stack.push(next_struct);

    // Compare leaf node 1 to right child of node 2
    next_struct.node1 = internal2->m_right;
    next_struct.node2 = cur_struct.node1;
    next_struct.mesh1 = cur_struct.mesh2;
    next_struct.mesh2 = cur_struct.mesh1;

    // Get the overlap between these two spheres
    c2 = cMul(cur_struct.mesh2->getGlobalRot(), internal2->m_right->getCenter());
    c2.add(cur_struct.mesh2->getGlobalPos());
    r = (float)(cur_struct.mesh1->getRadius() + internal2->m_right->getRadius());
    next_struct.dist = cDistance(c2, center1) - r*r;
    stack.push(next_struct);

    continue;
}
}
```

## Mesh Collisions Implementation (4)

```
// If the second node is a leaf and the first is an internal node, recurse by
// comparing the leaf to the left and to the right subtrees of the internal node
if (!(cur_struct.node2->isleaf()) && (cur_struct.node1->isleaf()))
{
    // Compare leaf node 2 to left child of node 1
    cCollisionSphereNode* internal1 = (cCollisionSphereNode*)cur_struct.node1;
    cCollisionStack next_struct;
    next_struct.node1 = internal1->m_left;
    next_struct.node2 = cur_struct.node2;
    next_struct.mesh1 = cur_struct.mesh1;
    next_struct.mesh2 = cur_struct.mesh2;

    // Get the overlap between these two spheres
    cVector3d c1 = cMul(cur_struct.mesh1->getGlobalRot(), internal1->m_left->getCenter());
    c1.add(cur_struct.mesh1->getGlobalPos());
    float r = (float)(cur_struct.node2->getRadius() + internal1->m_left->getRadius());
    next_struct.dist = cDistance(c1, center2) - r*r;
    stack.push(next_struct);

    // Compare leaf node 2 to right child of node 1
    next_struct.node1 = internal1->m_right;
    next_struct.node2 = cur_struct.node2;
    next_struct.mesh1 = cur_struct.mesh1;
    next_struct.mesh2 = cur_struct.mesh2;

    // Get the overlap between these two spheres
    c1 = cMul(cur_struct.mesh1->getGlobalRot(), internal1->m_right->getCenter());
    c1.add(cur_struct.mesh1->getGlobalPos());
    r = (float)(cur_struct.node2->getRadius() + internal1->m_right->getRadius());
    next_struct.dist = cDistance(c1, center2) - r*r;
    stack.push(next_struct);

    continue;
}
}
```

## Mesh Collisions Implementation (5)

```
// Otherwise, both nodes are internal nodes, and we will want to recurse by breaking the larger into its
// children, so swap nodes if necessary so that the larger one is the first one
if (cur_struct.node1->getRadius() < cur_struct.node2->getRadius())
{
    cCollisionSphereSphere* temp = cur_struct.node1; cMesh* temp_mesh = cur_struct.mesh1;
    cur_struct.node1 = cur_struct.node2; cur_struct.mesh1 = cur_struct.mesh2;
    cur_struct.node2 = temp; cur_struct.mesh2 = temp_mesh;
}

// Recurse by comparing the second internal node to the left and to the right subtrees
// of the first internal node
cCollisionSphereNode* internal1 = (cCollisionSphereNode*)cur_struct.node1;

// Compare node 2 to left child of node 1
cCollisionStack next_struct;
next_struct.node1 = internal1->m_left; next_struct.mesh1 = cur_struct.mesh1;
next_struct.node2 = cur_struct.node2; next_struct.mesh2 = cur_struct.mesh2;

// Get the overlap between these two spheres
cVector3d c1 = cMul(cur_struct.mesh1->getGlobalRot(), internal1->m_left->getCenter());
c1.add(cur_struct.mesh1->getGlobalPos());
float r = (float)(cur_struct.node2->getRadius() + internal1->m_left->getRadius());
next_struct.dist = cDistance(c1, center2) - r*r;
stack.push(next_struct);

// Compare node 2 to right child of node 1
next_struct.node1 = internal1->m_right; next_struct.mesh1 = cur_struct.mesh1;
next_struct.node2 = cur_struct.node2; next_struct.mesh2 = cur_struct.mesh2;

// Get the overlap between these two spheres
c1 = cMul(cur_struct.mesh1->getGlobalRot(), internal1->m_right->getCenter());
c1.add(cur_struct.mesh1->getGlobalPos());
r = (float)(cur_struct.node2->getRadius() + internal1->m_right->getRadius());
next_struct.dist = cDistance(c1, center2) - r*r;
stack.push(next_struct);

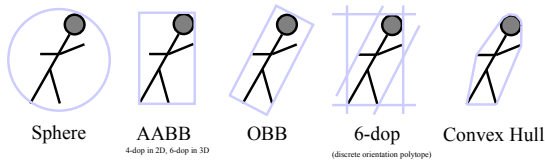
} m_lastContact1 = 0; m_lastContact2 = 0; return hit;
}
```

## Implementing a BVH

- Choose a type of bounding volume
  - Spheres
  - Axis-aligned bounding boxes
  - Oriented bounding boxes
  - Many others
- Build the hierarchy (preprocessing)
  - How to create a bounding volume from primitives or from children nodes
  - How to partition primitives among nodes
- Check for collisions (real-time)
  - How to check for collision between two bounding volumes?
  - Already described primitive – primitive tests for leaves
- Maintaining the hierarchy (as object deforms or changes topology)



## Choosing a Type of Bounding Volume



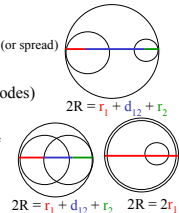
increasing complexity & tightness of fit (earlier pruning)

decreasing cost of overlap tests and BVH updates

Based on an example by Ming Lin

## Building a Sphere Tree Top-Down (CHAI Implementation; many other possibilities)

- How to partition primitives among children
  - Find minimum and maximum value of a triangle center along each coordinate axis (x,y,z) among all triangles bounded by this node
  - Find the coordinate index with the largest range (max to min)
  - Sort the primitives according to the coordinate with largest range (or spread)
  - Put first half in left subtree and second half in right subtree
- How to create a bounding volume from a triangle (leaf nodes)
  - Build the circumscribing sphere for the triangle (see next slide)
  - Using one edge as a diameter may sometimes give smaller sphere
  - Calculate radius as largest distance from the center to a vertex
- How to create a bounding volume from children nodes (internal nodes)
  - If one sphere is entirely contained within the other, set parent to the larger one
  - Otherwise
    - $radius_{parent} = (radius_{left-child} + distance(center_{left-child}, center_{right-child}) + radius_{right-child}) / 2$
    - Place the center of the parent's sphere along the line segment between children's centers



## Building a Circumsphere for a Triangle

- Circumcenter is intersection of perpendicular bisectors of edges
- Get the normal of the triangle's plane by crossing two edges
- For each of two edges, get the direction of the perpendicular bisector by crossing the edge with the plane normal, and the midpoint of the edge by averaging the two vertices
- Find the intersection of the two lines defined by the perpendicular bisectors to get the sphere center:

– For two lines,  $\mathbf{p}_1 + s\mathbf{d}_1$  and  $\mathbf{p}_2 + t\mathbf{d}_2$ , the intersection is at

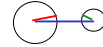
$$s = \frac{|\mathbf{p}_2 - \mathbf{p}_1 \cdot \mathbf{d}_2 \cdot \mathbf{d}_1 \times \mathbf{d}_2|}{\|\mathbf{d}_1 \times \mathbf{d}_2\|^2}$$



- Substitute this value of  $s$  into  $\mathbf{p}_1 + s\mathbf{d}_1$  to get the intersection point
- All vertices should be equidistant from this center, and this distance is the sphere's radius

## Checking for Collisions between Spheres

- Two spheres intersect if and only if the distance between their centers is less than (or equal to) the sum of their radii
- Easy and fast!



$d_{12} > r_1 + r_2$   
No intersection



$d_{12} < r_1 + r_2$   
Intersection

## Building an AABB Tree Top-Down (CHAI Implementation; many other possibilities)

- How to partition primitives among children
  - Find minimum and maximum value of a triangle center along each coordinate axis ( $x,y,z$ ) among all triangles bounded by this node
  - Find the coordinate index with the largest range (max to min) or spread
  - Sort the primitives according to the coordinate with largest range
  - Put first half in left subtree and second half in right subtree
- How to create a bounding volume from a triangle (leaf nodes)
  - Along each coordinate axis, the bounding box has an edge from the minimum value on that axis of any vertex of the triangle to the maximum value on that axis of any vertex of the triangle
- How to create a bounding volume from children nodes (internal nodes)
  - Along each coordinate axis, the bounding box for a node has an edge from the minimum value on that axis of any triangle in its subtree to the maximum value on that axis of any triangle in its subtree



## Checking for Collisions between Axis-Aligned Bounding Boxes

- There is an intersection if and only if there is an overlap along the  $x$ ,  $y$ , and  $z$  axes

For each coordinate axis  $i=x,y,z$  and box  $j=1,2$ , let  $i^j < i^j$

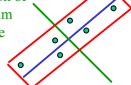
```

if (X2^1 < X1^2) or (X2^2 < X1^1)
    return false
if (Y2^1 < Y1^2) or (Y2^2 < Y1^1)
    return false
if (Z2^1 < Z1^2) or (Z2^2 < Z1^1)
    return false
return true
    
```

## Building an Oriented Bounding Box

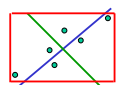
- Which way to orient the box?
- Axes should be directions of maximum and minimum spread (which are always orthogonal to each other)
- Think about projecting points onto the axis and determining "spread"
- These directions can be calculated as the eigenvectors of the  $3 \times 3$  covariance matrix  $C$  of the original points

Direction of minimum variance



Good box

Direction of maximum variance

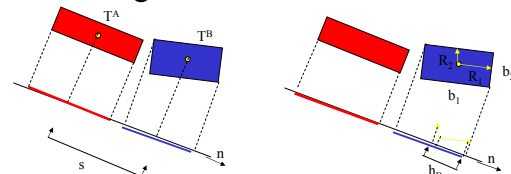


Bad box

$$\mu = \frac{1}{3n} \sum_{i=1}^n (a^i + b^i + c^i) \quad C_{jk} = \frac{1}{3n} \left( \sum_{i=1}^n (a_i^j - \mu)(a_i^k - \mu) + (b_i^j - \mu)(b_i^k - \mu) + (c_i^j - \mu)(c_i^k - \mu) \right)$$

where  $a^i, b^i, c^i$  are the vertices of the  $i^{\text{th}}$  of  $n$  triangles;  $j,k=1..3$

## Checking for Collisions between OBBs



Since the box centers project to interval midpoints, the distance between interval midpoints is

$$s = |(T^A - T^B) \cdot n|$$

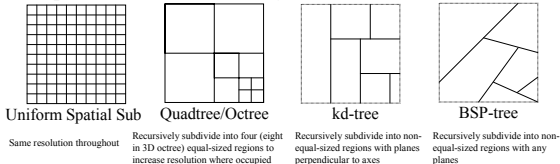
The half-length of the projected interval is equal to the sum of the projected box axis images

$$h_B = b_1 |R_1^B \cdot n| + b_2 |R_2^B \cdot n| + b_3 |R_3^B \cdot n|$$

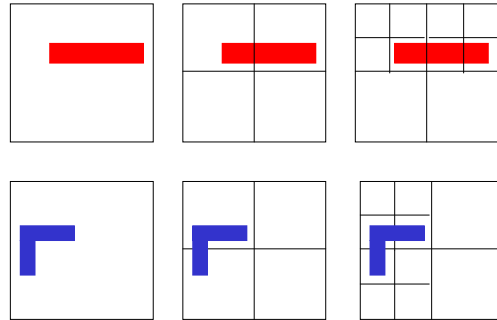
- Boxes do not intersect if a "separating axis" can be found
- Just as for spheres, the boxes are separated on an axis if and only if the projected midpoint distance  $s$  is greater than the sum of the interval half-lengths ("radiuses"),  $s > h_A + h_B$
- According to the Separating Axis Theorem, we need only consider projecting onto lines perpendicular to a face from either box or perpendicular to an edge from each
- In general, there could be  $2(\# \text{ faces}) + (\# \text{ edges})^2$  axes to test. OBBs have 3 edge directions and 3 face normals, so we need only repeat this interval test for  $2^3 + 3^2 = 15$  axes

## Alternative to BVH : Spatial Partitioning

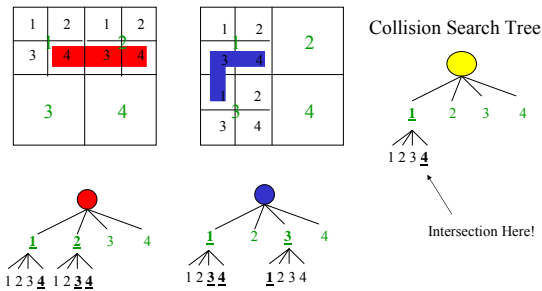
- Divide the entire environment into partitions, and represent the partitions occupied by each object
- If it is found that two objects occupy the same partition, the primitives of those objects in that partition must be tested against each other for collision
- “Space centric” versus “object centric” for BVH



## Building Quadtrees / Octrees



## Collision Detection with Quadtrees / Octrees



## Voxel Map

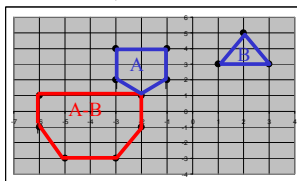
- A voxel map is a 3D spatial partition
- Usually a uniform spatial subdivision



## Minkowski Differences

- The Minkowski Difference of two sets of points A and B is

$$\{a - b \mid a \in A, b \in B\}$$

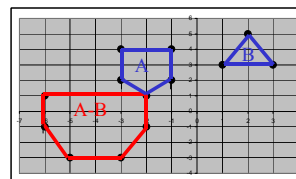


- A and B intersect if and only if the Minkowski Difference contains 0 (since  $a - b = 0$  if and only if there is a common point such that  $a = b$ )
- For convex polyhedra, we could compute the boundaries of the Minkowski difference by calculating  $a - b$  for all pairs of vertices, but this would take  $O(m * n)$  time just like brute force

## Proximity Queries

- How close are A and B? In other words, what is the shortest distance between any point in A to any point in B?
- Useful for collision avoidance, among many other things

$$\text{dist}(A, B) = \min_{a \in A, b \in B} \|a - b\|_2 = \min_{c \in \text{MinkowskiDiff}(A, B)} \|c\|_2$$



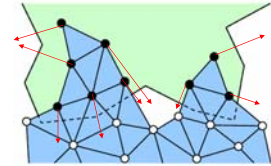
In this example, the minimum norm point on A-B is (-2,0) with norm 2. The closest points on A and B are (-1,3) and (1,3), which are 2 apart.

## Using Minkowski Differences for Convex Polyhedra Proximity Queries

- GJK is an algorithm that computes, in expected linear time, the minimum norm point of a set of points without requiring an explicit representation of the set as input; it only requires that points from the set with extreme values in given directions be supplied on demand
- If we ran GJK on a point set  $P = \text{Minkowski-Difference}(A,B)$ , we would get the distance between A and B
- We can't efficiently calculate P explicitly, but we can calculate the extreme value in a given direction  $-x$  as  $p_{-x} = a_{-x} - b_x$  (using convex hulls of A and B) (subscript x means the point extreme in direction x)
- Thus we can calculate the distance between two convex polyhedra A and B in expected linear time

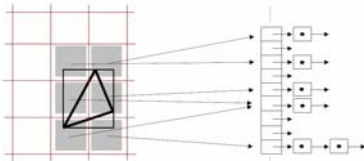
## Collision Response

- What do you do if you find that your objects do collide?
- Depends on
  - Deformable or rigid body
  - 3DOF or 6DOF feedback
  - Through what surface penetration occurred
- Should be “self-consistent”



## Teschner's Collision Detection

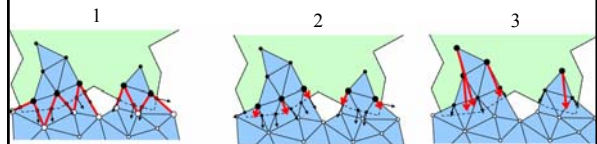
- Divide the world into grid cells
- At each time step
  - Hash all vertices into a spatial hash table
  - For each tetrahedron
    - Find all grid cells in its axis-aligned bounding box
    - Check each of these cells in the hash table to see if there is a vertex there
    - If so, perform primitive collision test
- Imperfect accuracy (can miss edge-edge collisions), but very fast



M. Teschner, B. Heidelberger, M. Mueller, D. Pomanets, M. Gross, "Optimized Spatial Hashing for Collision Detection of Deformable Objects," Proc. Vision, Modeling, Visualization VMV03, Munich, Germany, pp. 47-54, Nov. 2003.

## Teschner's Collision Response

1. After finding colliding (black) and non-colliding (white) points using collision detection algorithm, find “border edges” connecting a black and a white point, and calculate intersection points (red) and surface normals
2. Approximate penetration depth and direction for all border points by averaging adjacent intersection points and normals
3. Propagate penetration depths and directions to all other colliding points by averaging values at adjacent nodes



B. Heidelberger, M. Teschner, R. Keiser, M. Mueller, M. Gross, "Consistent Penetration Depth Estimation for Deformable Collision Response," Proc. Vision, Modeling, Visualization VMV04, Stanford, USA, pp. 339-346, Nov. 2004.

## Voxel Point Shell

- Test for intersections between surface points (point shell) of 3D tool and 3D spatial partition (voxel map) of environment
- Each intersection contributes a force proportional to vector from tool point along reversed surface normal to intersection with a tangent plane through voxel center
- Virtual coupling computes a force and torque by coupling haptic handle with tool via 6D spring and damper system

$$F_{spring} = k_T d - b_T v$$

$$\tau_{spring} = k_R \theta - b_R \omega$$

with  $k_T$ ,  $b_T$ ,  $k_R$ , and  $b_R$  the spring translational and rotational stiffness and velocity;  $\theta$  the equivalent-axis angle, and  $v$  and  $\omega$  the object's relative linear and angular velocity

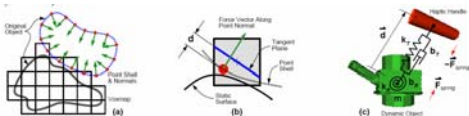


Figure 1. (a) Point-voxel collision detection, (b) Tangent plane force model, (c) Virtual coupling model

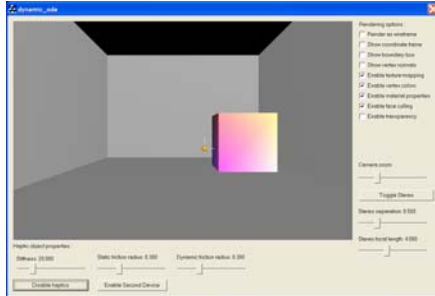
McNeely, W., Putterbaugh, K., and Troy, J., "Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling", Proc. ACM SIGGRAPH, pp. 401-408, Aug. 1999.

## Open Dynamics Engine

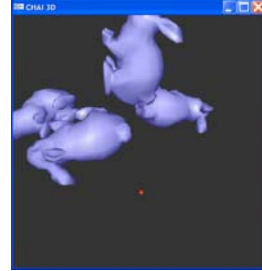
- Free, open-source library for dynamic simulations involving rigid bodies
- <http://www.ode.org>
- Written primarily by Russell Smith
- Used in two CHAI examples
  - dynamic\_ode
  - dynamic\_meshes



## CHAI “Dynamic ODE” Example



## CHAI “Dynamic Meshes” Example



## ODE Basics

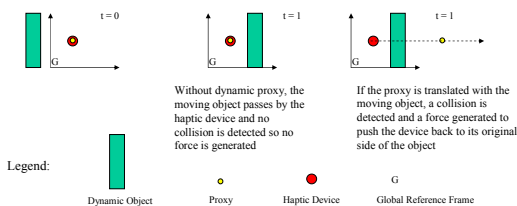
- Linking in the libraries
  - Add ode.lib import library to dependencies
  - Put ode.dll in path
  - #include ode.h
- Basic ODE constructs and terms
  - The *world* is a collection of *bodies*, each of which has a rotation, translation, mass, inertia, etc.
  - The *space* is a collection of *geometries*, each of which has a shape (cube, triangular mesh, etc.)
  - Each *geometry* is associated with a *body*
  - The *joint group* is a collection of *joints* (hinge, slider, universal, fixed, etc.) that connect *bodies*
  - When ODE detects a collision between *geometries*, it calls a user-defined callback function that creates a *contact joint* between the colliding *geometries* to handle collision response

## Using ODE with CHAI

- Simple (though not necessarily efficient) approach is to have two parallel worlds
  - ODE world handles collisions between objects, collision response, and rigid body dynamics
  - CHAI world handles rendering and interactions with haptic tool
- In each iteration of the render loop, have ODE compute a simulation step, and re-sync the rotations and translations of CHAI objects to those computed for their corresponding ODE geometries
- Add external forces to ODE world based on tool interactions computed by CHAI

## Dynamic Proxies

- Create separate dynamic proxies for each CHAI object, since the proxy must be computed in a moving object’s local reference frame to avoid passing through it



## ODE Implementation Examples (1)

```

// Copy data from cMesh to arrays to give an ODE geometry
VertexCount = mesh->getNumVertices();
Vertices = new float[VertexCount*3];
IndexCount = 3*mesh->getNumTriangles();
Indices = new int[IndexCount];
for (i=0; i<mesh->getNumVertices(); i++)
{
    Vertices[3*i] = mesh->getVertex(i)->getPos().x;
    Vertices[3*i+1] = mesh->getVertex(i)->getPos().y;
    Vertices[3*i+2] = mesh->getVertex(i)->getPos().z;
}
for (i=0; i<mesh->getNumTriangles(); i++)
{
    Indices[3*i] = mesh->getTriangle(i)->getIndexVertex0();
    Indices[3*i+1] = mesh->getTriangle(i)->getIndexVertex1();
    Indices[3*i+2] = mesh->getTriangle(i)->getIndexVertex2();
}

// Create an ODE geometry with this data
dTriMeshDataID new_tmdata = dGeomTriMeshDataCreate();
dGeomTriMeshDataBuildSingle(new_tmdata, #Vertices[0],
                             #Vertices[1], #Vertices[2],
                             #Indices[0], #Indices[1], #Indices[2],
                             #Indices[3], #Indices[4], #Indices[5]);
dGeomID geom = dCreateTriMesh(space, new_tmdata, 0, 0, 0);
dGeomSetData(geom, new_tmdata);

// Associate the geometry with the body, and set the mass
dGeomSetBody(geom,body);
dBodySetMass(body,sm);
    
```

## ODE Implementation Examples (2)

### Syncing poses for ODE and CHAI meshes

```
const dReal* odePosition = dGeomGetPosition(geom);
const dReal* odeRotation = dGeomGetRotation(geom);
cMatrix3d chaiRotation;
chaiRotation.set(
  odeRotation[0],odeRotation[1],odeRotation[2],
  odeRotation[4],odeRotation[5],odeRotation[6],
  odeRotation[8],odeRotation[9],odeRotation[10]);
mesh->setRot(chaiRotation);
mesh->setPos(
  odePosition[0],odePosition[1],odePosition[2]);
mesh->computeGlobalPositions();
```

### Taking a simulation step

```
dSpaceCollide (space,0,&nearCallback);
dWorldStepFast1 (ode_world,0.05, 5);
for (int j = 0; j < dSpaceGetNumGeoms(space); j++)
  dSpaceGetGeom(space, j);
dJointGroupEmpty (contactgroup);
```

### Adding a force to a body at a position

```
dBodyAddForceAtPos(body,fx,fy,fz,x,y,z);
```

### Collision Callback Function

```
static void nearCallback(void *data, dGeomID o1, dGeomID o2)
{
  // exit if the two bodies are connected by a joint
  dBodyID b1 = dGeomGetBody(o1);
  dBodyID b2 = dGeomGetBody(o2);
  if (b1 && b2 &&
      dAreConnectedExcluding(b1,b2,dJointTypeContact)) return;
  dContact contact[MAX_CONTACTS];
  for (int i=0; i<MAX_CONTACTS; i++)
  {
    contact[i].surface.mode = dContactBounce | dContactSoftCFM;
    contact[i].surface.mu = dInfinity;
    contact[i].surface.mu2 = 0;
    contact[i].surface.bounce = 0.1;
    contact[i].surface.bounce_vel = 0.1;
    contact[i].surface.soft_cfm = 0.01;
  }
  if (int numc = dCollide (o1,o2,MAX_CONTACTS,&contact[0].geom,
                          sizeof(dContact)))
  {
    dMatrix3 RI; dRSetIdentity (RI);
    const dReal ss[3] = {0.02,0.02,0.02};
    for (i=0; i<numc; i++)
    {
      dJointID c = dJointCreateContact (ode_world,
                                       contactgroup, contact+i);
      dJointAttach (c,b1,b2);
    }
  }
}
```

## Resources

- Collision detection is a major field of research in computer graphics; there's a lot out there
- Many libraries available on the web, often including rigid body dynamics
  - Open Dynamics Engine (ODE), <http://www.ode.org>
  - I-COLLIDE, V-COLLIDE, RAPID, and others, <http://www.cs.unc.edu/~lin/code.html>