

- Cross product as matrix multiplication

My equation editor broke down, so please refer to:

http://en.wikipedia.org/wiki/Cross_product#Conversion_to_matrix_multiplication

Visual Interactive Simulation, Fall 2011

Overview

- Representing geometric primitives
 - Real-time collision detection by Christer Ericsson is a useful reference (see literature page on course web).
- Intersection test/find for convex polyhedra
 - Separating axis theorem
- Intersection find for OBB's (boxes) and how to get basic credits without implementing it...

Spheres

Representation independent of orientation

- Center point
- Radius
- Rotation (not needed for intersections, but for rigid body rotations).

```
Struct Sphere {  
    Point c;          //Center point  
    Float r;          //Radius  
}
```

4 floats, 16 bytes.

Sphere-Sphere (1/2)

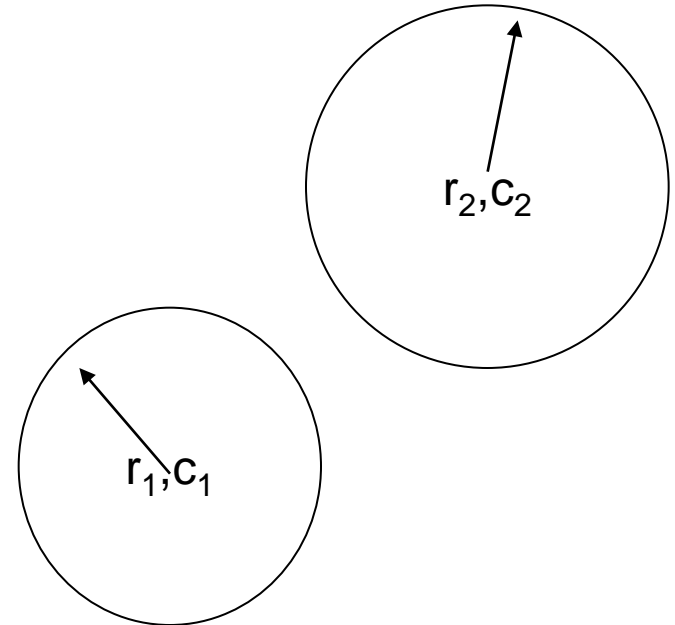
Intersection test

Compare distance with sum of radii:

$$|\mathbf{c}_1 - \mathbf{c}_2| > (r_1 + r_2)$$

Square root is expensive so we test,

$$(\mathbf{c}_1 - \mathbf{c}_2)^2 > (r_1 + r_2)^2$$



Sphere-Sphere (2/2)

Intersection find

Contact point, contact normal, penetration.

Contact normal when overlapping

$$\mathbf{n} = \frac{\mathbf{c}_2 - \mathbf{c}_1}{|\mathbf{c}_2 - \mathbf{c}_1|}$$

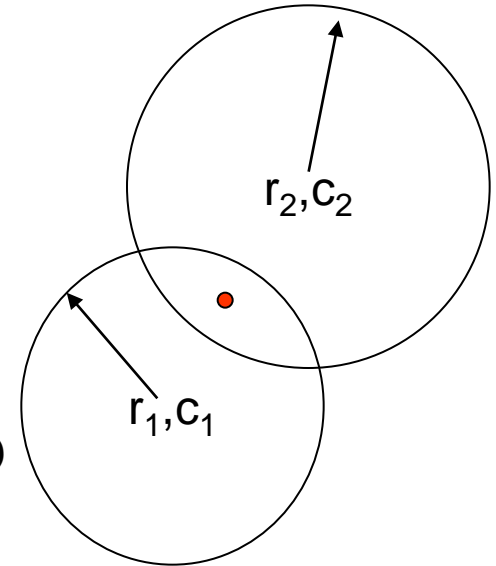
Contact point \mathbf{P} found by weighting with radii (don't use mid point - find the point that gives symmetric penetration)

$$\frac{|\mathbf{P} - \mathbf{c}_1|}{|\mathbf{P} - \mathbf{c}_2|} = \frac{r_1}{r_2}$$

Solve for \mathbf{P} ...

Penetration,

$$d = |\mathbf{c}_2 - \mathbf{c}_1| - r_2 - r_1$$



Boxes

Possible representations:

- All 8 vertices
- 6 planes
- Pair of parallel planes
- 1 corner vertex + 3 edge vectors (common in math)
- Center point, orientation matrix, 3 half-edge lengths (common in rigid body simulation):

```
Struct OBB {           //Oriented bounding box
    Point c;           //Box center
    Vector u[3];       //Local x,y,x axis
    Vector e;          //Positive half-edge lengths
}
```

15 floats.

- Lower memory if two edge vectors are stored and third is computed from cross-product (but more expensive cpu-wise (typical ps2 coding...))
- Often called "Oriented Bounding Box" (OBB) (since it is often used for wrapping up objects in subsections of space for broad phase collision detection)

Planes and half-spaces

Representations

- 3 points not on a straight line (forming a triangle on the plane)
- 1 normal and 1 point on the plane
- 1 normal and 1 distance from the origin

```
Struct Plane {  
    Vector n;    // Plane normal. Points x on the plane  
                // satisfy Dot(n,x) = d  
    float d;    // d = dot(n,p) for a given point p on  
                // the plane  
}
```

4 floats.

OBB-Plane I (1/1)

Test all vertices/corners against plane, $i=0..7$,

- Vector between vertex p_i and a point X (any X , e.g. $d \cdot n$) in the plane.
- Project this vector in the normal direction and measure penetration. If there is a penetration – there is an intersection!
- To summarize, penetration is given by,
$$d_i = n \cdot (p_i - X)$$

OBB-Plane II (1/2) (test)

Let a plane be given by $(\mathbf{n} \odot \mathbf{X}) = d$ for all points \mathbf{X} on the plane.
Test intersection using the *separating axis test*.

“Two convex curves will not curve around each other. Thus, if separated there will be a gap into which a plane can be inserted. A separating axis is a line perpendicular to this plane. Objects projected on this axis give non-overlapping intervals if objects do not intersect.”

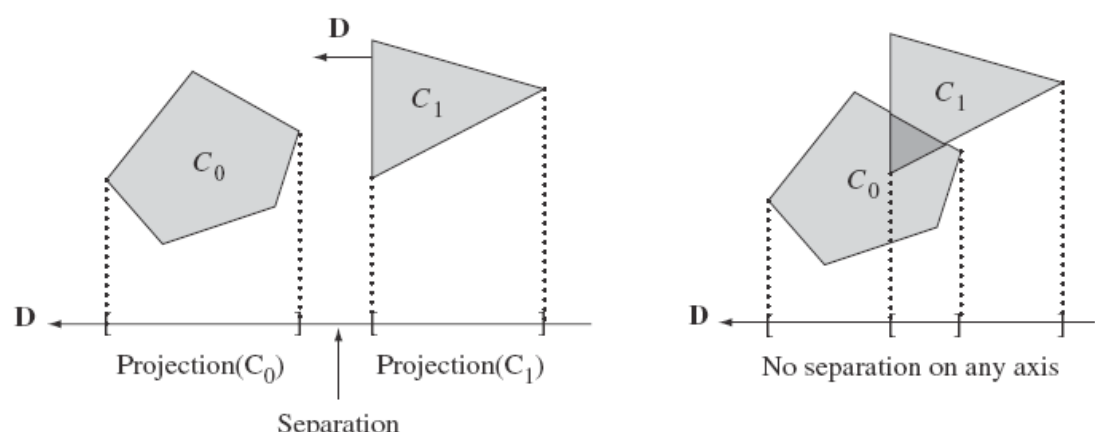
For OBB-Plane, this separating plane is parallel to the plane, and the separating axis is parallel to the normal.

Only the axis parallel to the plane normal \mathbf{n} need to be tested. Any line parallel to \mathbf{n} will actually do, so choose the line L that goes through the origin of the box.

OBB-Plane II (2/2) (test)

```
// Test if OBB b intersects Plane p
int TestOBBPlane(OBB b, Plane p)
{
    // Compute the projection interval radius
    // of b onto  $L(t) = b.c + t * p.n$ 
    float r = b.e[0]*Abs(Dot(p.n, b.u[0])) +
              b.e[1]*Abs(Dot(p.n, b.u[1])) +
              b.e[2]*Abs(Dot(p.n, b.u[2]));
    // Compute distance of box center from plane
    float s = Dot(p.n, b.c) - p.d;
    // Intersection occurs when distance s
    // falls within  $[-r, +r]$  interval
    return Abs(s) <= r;
}
```

Separating axes for convex polyhedra

- Separating axis. A hyperplane separates the polyhedra, and an axis normal to this plane is the separating axis. Projections of extreme points (vertices) on this axis show if the polyhedra intersect or not.
 - Test hyperplanes parallel with a face. Axis is the normal to the face
 - Test on axes that are cross products of pairs of edges.
 - If a hyperplane exists we have a *rejection* i.e. Boolean false in test!
- 
- Thus, it might make sense to do this in the right order (computationally and statistically optimized).
 - Naive implementation: $O((N_0 + N_1)^2)$ testing all possible projections.
 - A smarter implementation would just test min-max intervals. Worst case is then: $O(2N_0N_1)$ (bad but better).
 - For really *large* systems we can improve by using the *Dobkin-Kirkpatrick hierarchy* with logarithmic complexity (though much more overhead, since we need a special data hierarchy for it – this also has logarithmic complexity, but comes with a big pre factor).

Box-Box intersection test/find

- Box is defined by a center point (C), three coordinate axes (U), and three half edge lengths (e).
- Obviously the axes are normals to the faces of the box.
- Vertices are given by (σ are 8 different permutations of ± 1):

$$P = C + \sigma_0 e_0 U_0 + \sigma_1 e_1 U_1 + \sigma_2 e_2 U_2$$

- If blindly testing all 6 face normals of one box with those of the second (6+6), and also perform all the (12x12) edge-edge pair tests, we end up doing 156 tests.

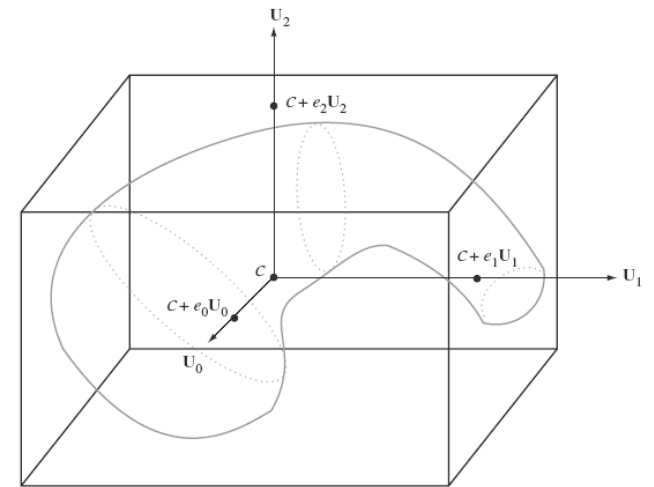


Figure 5.24 An OBB with center point C , coordinate axis directions U_0 , U_1 , and U_2 , and extents e_0 , e_1 , and e_2 along the coordinate axes. The object bounded by the box is shown in gray.

Box-Box intersection test/find

- However, three pairs of faces are parallel (special for box!)
- Only three edge directions are unique (and equal to face normals)
- Thus we have $3+3=6$ face normals and $3*3=9$ edge-edge pairs, in total 15 potential tests.
- A vertex must be an extreme point, and thus we try to find the vertex the maximizes:

$$\mathbf{D} \cdot (\mathcal{P} - \mathcal{Q}) = \mathbf{D} \cdot (\mathcal{C} - \mathcal{Q}) + e_0\sigma_0\mathbf{D} \cdot \mathbf{U}_0 + e_1\sigma_1\mathbf{D} \cdot \mathbf{U}_1 + e_2\sigma_2\mathbf{D} \cdot \mathbf{U}_2$$

- This can be sorted out analytically to find the right combinations of σ 's and therefore we can precompute exactly what projection intervals we should look at.
- In summary: Two boxes are separated if, with respect to some axis, the sum of their projected radii is less than the distance between the projection of their center points
- And we can make a table out of that...

Box-Box intersection test/find

- The projected interval tests for testing box (C_0, A, a) with box (C_0, A, a) , where $\alpha_i = \Delta \cdot A_i$ and $\beta_i = \Delta \cdot B_i$, for $\Delta = C_1 - C_0$

Table 5.1 Potential Separating Directions for OBBs and Values for r_0 , r_1 , and r

D	r_0	r_1	r
A_0	a_0	$b_0 c_{00} + b_1 c_{01} + b_2 c_{02} $	$ \alpha_0 $
A_1	a_1	$b_0 c_{10} + b_1 c_{11} + b_2 c_{12} $	$ \alpha_1 $
A_2	a_2	$b_0 c_{20} + b_1 c_{21} + b_2 c_{22} $	$ \alpha_2 $
B_0	$a_0 c_{00} + a_1 c_{10} + a_2 c_{20} $	b_0	$ \beta_0 $
B_1	$a_0 c_{01} + a_1 c_{11} + a_2 c_{21} $	b_1	$ \beta_1 $
B_2	$a_0 c_{02} + a_1 c_{12} + a_2 c_{22} $	b_2	$ \beta_2 $
$A_0 \times B_0$	$a_1 c_{20} + a_2 c_{10} $	$b_1 c_{02} + b_2 c_{01} $	$ c_{10}\alpha_2 - c_{20}\alpha_1 $
$A_0 \times B_1$	$a_1 c_{21} + a_2 c_{11} $	$b_0 c_{02} + b_2 c_{00} $	$ c_{11}\alpha_2 - c_{21}\alpha_1 $
$A_0 \times B_2$	$a_1 c_{22} + a_2 c_{12} $	$b_0 c_{01} + b_1 c_{00} $	$ c_{12}\alpha_2 - c_{22}\alpha_1 $
$A_1 \times B_0$	$a_0 c_{20} + a_2 c_{00} $	$b_1 c_{12} + b_2 c_{11} $	$ c_{20}\alpha_0 - c_{00}\alpha_2 $
$A_1 \times B_1$	$a_0 c_{21} + a_2 c_{01} $	$b_0 c_{12} + b_2 c_{10} $	$ c_{21}\alpha_0 - c_{01}\alpha_2 $
$A_1 \times B_2$	$a_0 c_{22} + a_2 c_{02} $	$b_0 c_{11} + b_1 c_{10} $	$ c_{22}\alpha_0 - c_{02}\alpha_2 $
$A_2 \times B_0$	$a_0 c_{10} + a_1 c_{00} $	$b_1 c_{22} + b_2 c_{21} $	$ c_{00}\alpha_1 - c_{10}\alpha_0 $
$A_2 \times B_1$	$a_0 c_{11} + a_1 c_{01} $	$b_0 c_{22} + b_2 c_{20} $	$ c_{01}\alpha_1 - c_{11}\alpha_0 $
$A_2 \times B_2$	$a_0 c_{12} + a_1 c_{02} $	$b_0 c_{21} + b_1 c_{20} $	$ c_{02}\alpha_1 - c_{12}\alpha_0 $

Box-Box intersection test/find

- And, we can code this down...
- Code is given on the course web
- Some comments:
 - In general, do the tests in the given order
 - Three orthogonal axes tested first – covers a large portion of space
 - The representation is such that body A is transformed to the origin and therefore tests on A are cheaper (50%) than tests on B (the following three cases).
 - R and Abs(R) can be done after the first three tests to avoid unnecessary square roots.
 - In many applications boxes are highly aligned most of the time
 - Choose to do the most relevant tests first to get an early rejection!
 - Statistically (it lies of course!) you will only miss 5-7% of the rejections if you skip the last 9 tests completely!
 - If an edge from box A is parallel with an edge of box B, the cross product (case 7-15) will be zero. Whatever you project onto this axis will be zero, so projections on both sides in the inequality will be zero and you compare $0 > 0$. This is ill-behaved numerically and you can get the wrong answer! Add a small ϵ to make your test *conservative*, or check cross product and give it special treatment!
I.e. project back to face case!

Box-Box intersection find

- Ok, cool, now we know how to do an intersection test. Find then?
- Potentially:
 - face-face, face-edge, face-vertex, edge-edge, edge-vertex, vertex-vertex
 - Normals are given from the face normals, and edge-edge cross products
 - Vertex-vertex, vertex-edge, vertex-face: contact point is a vertex
 - Edge-edge: contact line (rare) or point, but reduced to contact point
 - Edge-face: contact-line, reduced to points
 - Face-face: convex polygon, reduced to points
- If there is an intersection, we need to find which of the above represents the end of the projection interval, and from this a reduced contact set.
- Bouma, Vanecek 1993; Moravanzky, Terdiman GPG4 2004; Baraff SIGGRAPH notes 2003;
- Kenny Erleben PhD Thesis, 2005, points out rather serious and overlooked problems with the most common algorithms, and suggests improvements.
- Many collision detection libraries do not report finds, and are therefore useless (almost) in rigid body simulation. Some engines report points incrementally.

Box-Box intersection find

- Finding a reduced contact set:
 - Only vertex-face or edge-edge (point when not parallel)
 - If edge-face \rightarrow edge end point/vertex if it is contained in the face and edge-edge intersection if the edge overlaps and edge of the face
 - If face-face \rightarrow only use vertices of one face contained in the other face, or edge-edge intersection, one edge from each face

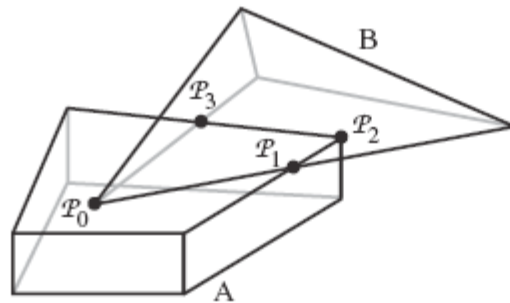


Figure 5.2 The reduced contact set for two convex polyhedra A and B.

Box-box intersection find

- Strategy
 - Find axis with minimum overlap
 - If intersection: loop over cases and sort overlaps
 - Treat edge-edge last so we pick face cases before..
 - Contact normal = separation axis
 - Penetration depth = overlap size

Improvements for OBB-OBB suggested by Erleben (see thesis)

- First detect if vertices generate contacts and flag them
- Next look for edge-edge crossings. No new contact points if both end-points are flagged!
- The end-points must lie on opposite sides.

Intersect test/find in lab project

- You may borrow the ODE or Bullet OBB-OBB test for your basic credit points (read open source licenses!).
- You'll get bonus credits if you implement your own OBB-OBB intersection test/find.
- You must be able to *explain* how the axis separation test works and principles for how to compute a contact set!
- Always VISUALIZE your contacts if you want to understand what you're doing right and wrong...