Notes on Cuboid-Cuboid Collision Detection

Paul Nathan 01/09/2014

This collision detection method shall be limited to the special case of:

- right rectangular prisms (cuboids)
  - lying flat in the x-y plane
  - having rotations only about the z-axis

Such a cuboid can be fully specified by its four vertices as seen in plan view ((x, y) coordinates), as well as the z coordinates of its lower and upper surfaces. This specification requires only 10 values per cuboid, compared to the 24 that would be required if specifying the (x, y, z) coordinate of each vertex.

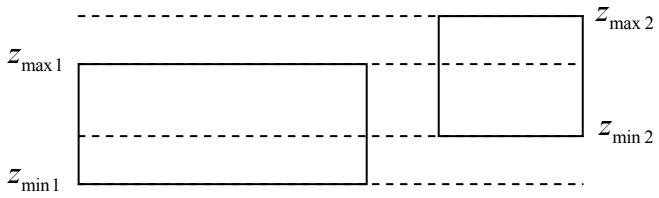
Collision detection strategy:

1. Z test. Check whether cuboid 1 lies within the z range of cuboid 2 and vice versa (in case of complete containment of one cuboid in the other). If true, then there may be a collision, proceed to step 2. This step essentially reduces the problem to two-dimensions
2. Bounding circle test. Test whether the bounding circle of cuboid 1 (plus some specified minimum safe distance, if desired) intersects the bounding circle of cuboid 2. If true, then these cuboids warrant full collision detection check, proceed to step 3
3.
   a. Edge-edge intersection tests. Check for intersection of each edge of cuboid 1 against each edge of cuboid 2. 16 tests at most. Exit as soon as collision occurs else continue to b. This test will detect all types of intersection except for complete containment of one cuboid inside the other
   b. Vertex-edge signed distance tests. For each vertex of cuboid 1, check its signed distance from each edge of cuboid 2. If all returned values are negative, then that vertex must be inside cuboid 2, therefore collision. If still no collision, carry out this test for each vertex of cuboid 2 on the edges of cuboid 1. 32 tests at most. This test covers complete containment of one cuboid inside the other.
4. If 3 returns no collision, then return the minimum separation distance between the cuboids.

Details:

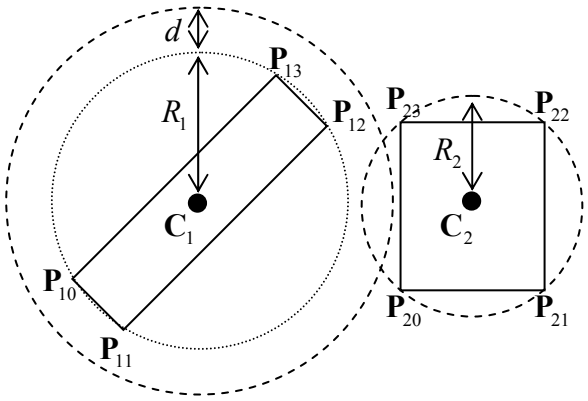
1. Z test:

(Side view)



If $z_{min\,2} \leq z_{min\,1} \leq z_{max\,2}$ OR $z_{min\,2} \leq z_{max\,1} \leq z_{max\,2}$ then proceed to step 2

2. Bounding circle test:



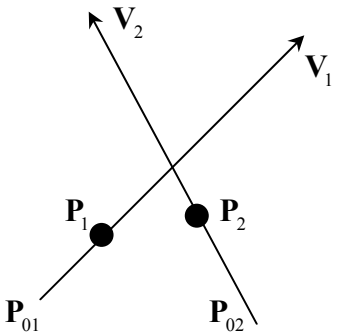
$$\mathbf{C}_i = \tfrac{1}{4}\left(\mathbf{P}_{i0} + \mathbf{P}_{i1} + \mathbf{P}_{i2} + \mathbf{P}_{i3}\right)$$
$$R_i = \left\|\mathbf{P}_{i0} - \mathbf{C}_i\right\|$$

If $\left(\mathbf{C}_2 - \mathbf{C}_1\right)\bullet\left(\mathbf{C}_2 - \mathbf{C}_1\right) \le \left(R_1 + R_2 + d\right)^2$ then proceed to step 3

3.

a. Edge-edge intersection test:

In the present 2D simplification, all lines are co-planar and so an overlap in plan view must be an intersection. Consider the generic example of two edges, $\mathbf{V}_1$ and $\mathbf{V}_2$, where the edge vectors of a cuboid are given by $\mathbf{V}_i = \mathbf{P}_{(i+1)\bmod 4} - \mathbf{P}_i$ with $i = 0\ldots3$



$$\mathbf{P}_1 = \mathbf{P}_{01} + s_1\mathbf{V}_1$$
$$\mathbf{P}_2 = \mathbf{P}_{02} + s_2\mathbf{V}_2$$

The condition for intersection is simply $\mathbf{P}_1 = \mathbf{P}_2$

$$\mathbf{P}_{01} + s_1\mathbf{V}_1 = \mathbf{P}_{02} + s_2\mathbf{V}_2$$

This must now be solved for s_1 and s_2, the distance parameters [0…1] along the edge from the start point $\mathbf{P}_{01}$ and $\mathbf{P}_{02}$ respectively. Firstly, eliminate s_2 by taking the cross-product of both sides with $\mathbf{V}_2$, then solve for s_1

$$\mathbf{P}_{01}\times\mathbf{V}_2 + s_1\mathbf{V}_1\times\mathbf{V}_2 = \mathbf{P}_{02}\times\mathbf{V}_2 + s_2\cancel{\mathbf{V}_2\times\mathbf{V}_2}$$

$$s_1 = \frac{\left(\mathbf{P}_{02} - \mathbf{P}_{01}\right)\times\mathbf{V}_2}{\mathbf{V}_1\times\mathbf{V}_2}$$

Similarly for s_2

$$s_2 = \frac{(\mathbf{P}_{02} - \mathbf{P}_{01}) \times \mathbf{V}_1}{\mathbf{V}_1 \times \mathbf{V}_2}$$

If $\mathbf{V}_1 \times \mathbf{V}_2 = 0$ then the edges are parallel

If $(\mathbf{P}_{02} - \mathbf{P}_{01}) \times \mathbf{V}_2 \neq 0$ then the edges are parallel non-intersecting. No collision.

Else the edges are collinear

If $0 \leq (\mathbf{P}_{02} - \mathbf{P}_{01}) \bullet \mathbf{V}_1 \leq \mathbf{V}_1 \bullet \mathbf{V}_1$ OR $0 \leq (\mathbf{P}_{02} - \mathbf{P}_{01}) \bullet \mathbf{V}_2 \leq \mathbf{V}_2 \bullet \mathbf{V}_2$ then the edges are overlapping. Collision.

Else the edges are disjoint. No collision.

Else

If $0 \leq s_1 \leq 1$ AND $0 \leq s_2 \leq 1$ then the edges intersect. Collision.

Else no collision.

Note the use of the 2D cross-product which outputs a scalar, not a vector. For 2D vectors $\mathbf{A} = \begin{pmatrix} A_x & A_y \end{pmatrix}$ and $\mathbf{B} = \begin{pmatrix} B_x & B_y \end{pmatrix}$ the 2D cross product is as follows

$$\mathbf{A} \times \mathbf{B} = \begin{vmatrix} A_x & A_y \\ B_x & B_y \end{vmatrix} = A_x B_y - A_y B_x$$

b. Vertex-edge signed distance tests:
   i. For vertex *i* of cuboid 1 and edge *j* of cuboid 2, check the projection of the vertex is on the edge with the condition $0 \leq (\mathbf{P}_{1i} - \mathbf{P}_{2j0}) \bullet \mathbf{V}_{2j} \leq \mathbf{V}_{2j} \bullet \mathbf{V}_{2j}$. If true then continue to step ii, else cycle through remaining edges. If no positive result, test vertices of cuboid 2 against edges of cuboid 1. If still no positive result, omit step ii.
   ii. For vertex *i* of cuboid 1 and outward directed unit normal of edge *j* of cuboid 2, check the signed distance of the vertex from the edge $\hat{\mathbf{n}}_{2j} \bullet (\mathbf{P}_{1i} - \mathbf{P}_{2j}) \leq 0$. If true for all $j = 0 \ldots 3$ then vertex *i* of cuboid 1 is inside cuboid 2. Collision. Otherwise continue checking each vertex, and then also the vertices of cuboid 2 against the edges of cuboid 1. The signed distance itself, only if positive, should be stored for obtaining the minimum separation distance in the case of no collision.
   iii. If no collision, compute the squared distances between all of the vertices, $(\mathbf{P}_{1i} - \mathbf{P}_{2j}) \bullet (\mathbf{P}_{1i} - \mathbf{P}_{2j})$, for all *i* and *j* and return the minimum value. Take the square root of this single value, add it to the list of (positive) signed distances previously obtained and return the minimum value. Otherwise, if collision, return 0 for the minimum separation distance.

The unit normal vector of an edge *i* can be found by taking the 3D cross product of the edge vector with the unit z-axis vector and normalising the result

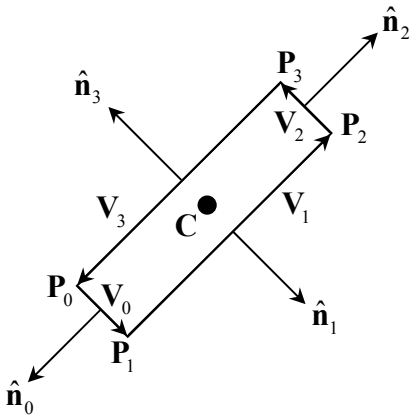
$$\hat{\mathbf{n}}_i = \frac{\mathbf{V}_i \times \hat{\mathbf{z}}}{\left\| \mathbf{V}_i \times \hat{\mathbf{z}} \right\|}$$

Alternatively, the use of the 3D cross product can be avoided by making use of the cuboid's centroid (already computed earlier) as follows

$$\mathbf{Q}_i = \mathbf{C} - \mathbf{P}_i$$

$$\mathbf{n}_i = \frac{(\mathbf{Q}_i \bullet \mathbf{V}_i)\mathbf{V}_i}{\mathbf{V}_i \bullet \mathbf{V}_i} - \mathbf{Q}_i$$

$$\hat{\mathbf{n}}_i = \frac{\mathbf{n}_i}{\|\mathbf{n}_i\|}$$



This method works by finding the projection of the radius vector $\mathbf{Q}_i$ onto the associated edge *i* and then subtracting off the radius vector to be left with only the component normal to the edge, directed away from the centroid (i.e. outward positive).

Both of these methods for computing the edge unit normal vectors require the vertices of the cuboid projection to be wound in an anti-clockwise manner. Supposing a randomly order set of vertices, this can be achieved as follows:

- Starting with the first vertex in the list (index 0), find the next closest vertex and place it at the next index in the list. Repeat for all remaining vertices (except the last one, as by process of elimination there is no need), excluding the previously sorted vertices from the distance check. This procedure guarantees that the vertices are now representing the convex hull of the cuboid, without any internal crossing. However it does not guarantee anti-clockwise winding. To check the winding direction:
    - Calculate signed area *A* of the cuboid projection using the 2D cross-product $A = \mathbf{V}_i \times \mathbf{V}_{(i+1) \bmod 4}$ for any *i*. If the returned value is negative, the winding is clockwise and so the order of the vertices needs to be reversed.

With floating point implementation on computer, instead of testing equality with zero it is good practice to use a tolerance ε, such as the single precision machine epsilon value. So, in practice the condition $A \leq x \leq B$ would be implemented as $A - \varepsilon < x < B + \varepsilon$. This ensures that finite precision truncation error does not cause the equality case to be missed.