AABB of a Transformed AABB

David Eberly
Geometric Tools, LLC
http://www.geometrictools.com/

Created: September 24, 2005
Last Modified: February 12, 2008

Contents

1 Two Dimensions **2**

This document describes how to compute the axis-aligned bounding box (AABB) of another AABB, which is transformed by a rotation and translation (applied in that order). The transformed AABB is an oriented bounding box (OBB). The goal is to quickly compute the minimum and maximum values in each dimension for the OBB without having to transform *all* the vertices and search them for the extrema. The construction applies to general dimensions, but the 2D and 3D cases are provided first for motivation.

1 Two Dimensions

An AABB in two dimensions is the set of points (x, y), where $x_{\min} \le x \le x_{\max}$ and $y_{\min} \le y \le y_{\max}$. The minimum and maximum x and y values are, of course, known quantities. The point in the AABB is written in vector form as

$$\mathbf{P} = \begin{bmatrix} x \\ y \end{bmatrix}$$

The rotation is a matrix R and the translation is a vector $\mathbf{T}$, say

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}, \quad \mathbf{T} = \begin{bmatrix} a \\ b \end{bmatrix}$$

where θ is the angle of rotation. The transformed AABB point is

$$\mathbf{Q} = R\mathbf{P} + \mathbf{T} = \begin{bmatrix} x\cos\theta - y\sin\theta + a \\ x\sin\theta + y\cos\theta + b \end{bmatrix}$$

The four vertices of the AABB are

$$\begin{aligned} \mathbf{P}_0 &= (x_{\min}, y_{\min}) \\ \mathbf{P}_1 &= (x_{\max}, y_{\min}) \\ \mathbf{P}_2 &= (x_{\min}, y_{\max}) \\ \mathbf{P}_3 &= (x_{\max}, y_{\max}) \end{aligned}$$

The four transformed vertices are those of an OBB and are

$$\begin{aligned} \mathbf{Q}_0 &= (x_{\min}\cos\theta - y_{\min}\sin\theta + a, x_{\min}\sin\theta + y_{\min}\cos\theta + b) \\ \mathbf{Q}_1 &= (x_{\min}\cos\theta - y_{\max}\sin\theta + a, x_{\min}\sin\theta + y_{\max}\cos\theta + b) \\ \mathbf{Q}_2 &= (x_{\max}\cos\theta - y_{\min}\sin\theta + a, x_{\max}\sin\theta + y_{\min}\cos\theta + b) \\ \mathbf{Q}_3 &= (x_{\max}\cos\theta - y_{\max}\sin\theta + a, x_{\max}\sin\theta + y_{\max}\cos\theta + b) \end{aligned}$$

A naive method for computing the extreme values in x and y is to search the components directly. The pseudocode is

```
void GetAABB (Point2 min, Point2 max, Matrix2 rot, Point2 trn,
```

```
    Point2& newmin, Point2& newmax)
{
    Real cs = rot[0][0], sn = rot[1][0];
    Real x, y;

    newmin.x = newmax.x = min.x * cs - min.y * sn + trn.x;
    newmin.y = newmax.y = min.x * sn + min.y * cs + trn.y;

    x = min.x * cs - max.y * sn + trn.x;
    y = min.x * sn + max.y * cs + trn.y;
    if (x < newmin.x) { newmin.x = x; } else if (x > newmax.x) { newmax.x = x; }
    if (y < newmin.y) { newmin.y = y; } else if (y > newmax.y) { newmax.y = y; }

    x = max.x * cs - min.y * sn + trn.x;
    y = max.x * sn + min.y * cs + trn.y;
    if (x < newmin.x) { newmin.x = x; } else if (x > newmax.x) { newmax.x = x; }
    if (y < newmin.y) { newmin.y = y; } else if (y > newmax.y) { newmax.y = y; }

    x = max.x * cs - max.y * sn + trn.x;
    y = max.x * sn + max.y * cs + trn.y;
    if (x < newmin.x) { newmin.x = x; } else if (x > newmax.x) { newmax.x = x; }
    if (y < newmin.y) { newmin.y = y; } else if (y > newmax.y) { newmax.y = y; }
}
```

This algorithm requires 16 multiplications, 16 additions (or subtractions), and at least 6 comparisons but at most 12 comparisons.

Noticing that the extrema calculations are dependent on the signs of `cs` and `sn`, the following pseudocode has fewer operations and comparisons.

```
void GetAABB (Point2 min, Point2 max, Matrix2 rot, Point2 trn,
    Point2& newmin, Point2& newmax)
{
    Real cs = rot[0][0], sn = rot[1][0];

    if (cs >= 0)
    {
        if (sn >= 0)
        {
            newmin.x = min.x * cs - max.y * sn + trn.x;
            newmin.y = min.x * sn + min.y * cs + trn.y;
            newmax.x = max.x * cs - min.y * sn + trn.x;
            newmax.y = max.x * sn + max.y * cs + trn.y;
        }
        else
        {
            newmin.x = min.x * cs - min.y * sn + trn.x;
            newmin.y = max.x * sn + min.y * cs + trn.y;
            newmax.x = max.x * cs - max.y * sn + trn.x;
            newmax.y = min.x * sn + max.y * cs + trn.y;
        }
    }
    else
    {
        if (sn >= 0)
        {
            newmin.x = max.x * cs - max.y * sn + trn.x;
            newmin.y = min.x * sn + max.y * cs + trn.y;
            newmax.x = min.x * cs - min.y * sn + trn.x;
            newmax.y = max.x * sn + min.y * cs + trn.y;
        }
        else
        {
            newmin.x = max.x * cs - min.y * sn + trn.x;
            newmin.y = max.x * sn + max.y * cs + trn.y;
            newmax.x = min.x * cs - max.y * sn + trn.x;
            newmax.y = min.x * sn + min.y * cs + trn.y;
        }
    }
}
```

This code requires 8 multiplications, 8 additions, and 2 comparisons.