The entire test starts with six determinant tests, and the first three share the first arguments, so there is a lot to gain in terms of shared computations. In principle, the determinant can be computed using many smaller 2×2 subdeterminants (see Section A.3.1), and when these occur in more than one 4×4 determinant, the computations can be shared. There is code on the web for this test [467], and it is also possible to augment the code to compute the actual line segment of intersection.

If the triangles are coplanar, they are projected onto the axis-aligned plane where the areas of the triangles are maximized (see Section 16.9). Then, a simple two-dimensional triangle-triangle overlap test is performed. First, test all closed edges (i.e., including endpoints) of T_1 for intersection with the closed edges of T_2. If any intersection is found, the triangles intersect. Otherwise, we must test whether T_1 is totally contained in T_2 or vice versa. This can be done by performing a point-in-triangle test (see Section 16.8) for one vertex of T_1 against T_2, and vice versa.

Robustness problems may arise when the triangles are nearly coplanar or when an edge is nearly coplanar to the other triangle (especially when the edge is close to an edge of the other triangle). To handle these cases in a reasonable way, a user-defined constant, `EPSILON` (ϵ), can be used.[12] For example, if $|[\mathbf{p}_2, \mathbf{q}_2, \mathbf{r}_2, \mathbf{p}_1]| < \epsilon$, then we move $\mathbf{p}_1$ so that $[\mathbf{p}_2, \mathbf{q}_2, \mathbf{r}_2, \mathbf{p}_1] = 0$. Geometrically, this means that if a point is "close enough" to the other triangle's plane, it is considered to be on the plane. The same is done for the points of the other triangle, as well. The source code does not handle degenerate triangles (i.e., lines and points). To do so, those cases should be detected first and then handled as special cases.

16.12 Triangle/Box Overlap

This section presents an algorithm for determining whether a triangle intersects an axis-aligned box. Such a test can be used to build voxel-spaces, test triangles against boxes in collision detection, and test polygons against canonical view volumes (see Section 4.6), and thus potentially eliminate the need for calls to clipping and lighting routines, etc.

Green and Hatch [441] present an algorithm that can determine whether an arbitrary polygon overlaps a box. Akenine-Möller [11] developed a faster method that is based on the separating axis test (page 731), and which we present here.

We focus on testing an axis-aligned bounding box (AABB), defined by a center $\mathbf{c}$, and a vector of half lengths, $\mathbf{h}$, against a triangle $\Delta \mathbf{u}_0 \mathbf{u}_1 \mathbf{u}_2$. To simplify the tests, we first move the box and the triangle so that the box is centered around the origin, i.e., $\mathbf{v}_i = \mathbf{u}_i - \mathbf{c}$, $i \in \{0, 1, 2\}$. This

[12] For floating point precision, $\epsilon = 10^{-6}$ works for "normal" data.

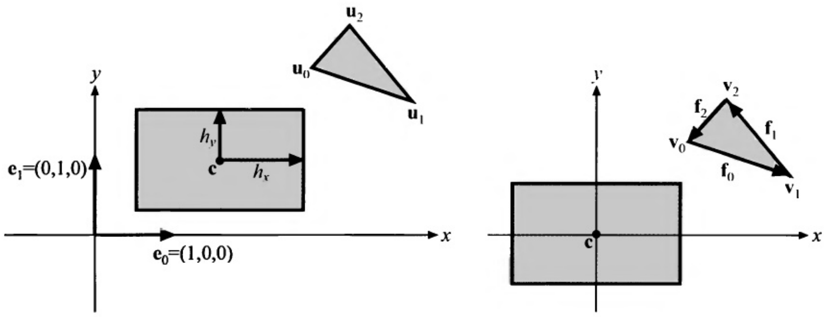

Figure 16.18. Notation used for the triangle-box overlap test. To the left, the initial position of the box and the triangle is shown, while to the right, the box and the triangle have been translated so that the box center coincides with the origin.

translation and the notation used is shown in Figure 16.18. To test against an oriented box, we would first rotate the triangle vertices by the inverse box transform, then use the test here.

Based on the separating axis test (SAT), we test the following 13 axes:

1. [3 tests] $\mathbf{e}_0 = (1,0,0)$, $\mathbf{e}_1 = (0,1,0)$, $\mathbf{e}_2 = (0,0,1)$ (the normals of the AABB). In other words, test the AABB against the minimal AABB around the triangle.

2. [1 test] $\mathbf{n}$, the normal of $\Delta\mathbf{u}_0\mathbf{u}_1\mathbf{u}_2$. We use a fast plane/AABB overlap test (see Section 16.10.1), which tests only the two vertices of the box diagonal whose direction is most closely aligned to the normal of the triangle.

3. [9 tests] $\mathbf{a}_{ij} = \mathbf{e}_i \times \mathbf{f}_j$, $i, j \in \{0,1,2\}$, where $\mathbf{f}_0 = \mathbf{v}_1 - \mathbf{v}_0$, $\mathbf{f}_1 = \mathbf{v}_2 - \mathbf{v}_1$, and $\mathbf{f}_2 = \mathbf{v}_0 - \mathbf{v}_2$, i.e., edge vectors. These tests are very similar and we will only show the derivation of the case where $i = 0$ and $j = 0$ (see below).

As soon as a separating axis is found the algorithm terminates and returns "no overlap." If all tests pass, i.e., there is no separating axis, then the triangle overlaps the box.

Here we derive one of the nine tests, where $i = 0$ and $j = 0$, in Step 3. This means that $\mathbf{a}_{00} = \mathbf{e}_0 \times \mathbf{f}_0 = (0, -f_{0z}, f_{0y})$. So, now we need to project the triangle vertices onto $\mathbf{a}_{00}$ (hereafter called $\mathbf{a}$):

$$\begin{aligned}
p_0 &= \mathbf{a} \cdot \mathbf{v}_0 = (0, -f_{0z}, f_{0y}) \cdot \mathbf{v}_0 = v_{0z}v_{1y} - v_{0y}v_{1z}, \\
p_1 &= \mathbf{a} \cdot \mathbf{v}_1 = (0, -f_{0z}, f_{0y}) \cdot \mathbf{v}_1 = v_{0z}v_{1y} - v_{0y}v_{1z} = p_0, \\
p_2 &= \mathbf{a} \cdot \mathbf{v}_2 = (0, -f_{0z}, f_{0y}) \cdot \mathbf{v}_2 = (v_{1y} - v_{0y})v_{2z} - (v_{1z} - v_{0z})v_{2y}.
\end{aligned} \tag{16.33}$$

Normally, we would have had to find $\min(p_0, p_1, p_2)$ and $\max(p_0, p_1, p_2)$, but fortunately $p_0 = p_1$, which simplifies the computations. Now we only need to find $\min(p_0, p_2)$ and $\max(p_0, p_2)$, which is significantly faster because conditional statements are expensive on modern CPUs.

After the projection of the triangle onto $\mathbf{a}$, we need to project the box onto $\mathbf{a}$ as well. We compute a "radius," r, of the box projected on $\mathbf{a}$ as

$$r = h_x|a_x| + h_y|a_y| + h_z|a_z| = h_y|a_y| + h_z|a_z|, \qquad (16.34)$$

where the last step comes from that $a_x = 0$ for this particular axis. Then, this axis test becomes

$$\texttt{if(}\min(p_0, p_2) > r \texttt{ or } \max(p_0, p_2) < -r\texttt{) return false;} \qquad (16.35)$$

Code is available on the web [11].

16.13 BV/BV Intersection Tests

A closed volume that totally contains a set of objects is (in most situations) called a *bounding volume* (BV) for this set. The purpose of a BV is to provide simpler intersection tests and make more efficient rejections. For example, to test whether or not two cars collide, first find their BVs and test if these overlap. If they do not, then the cars are guaranteed not to collide (which we assume is the most common case). We then have avoided testing each primitive of one car against each primitive of the other, thereby saving computation.

Bounding volume hierarchies are often part of the foundation of collision detection algorithms (see Chapter 17). Four bounding volumes that are commonly used for this purpose are the sphere, the *axis-aligned bounding box* (AABB), the *discrete oriented polytope* (k-DOP), and the *oriented*

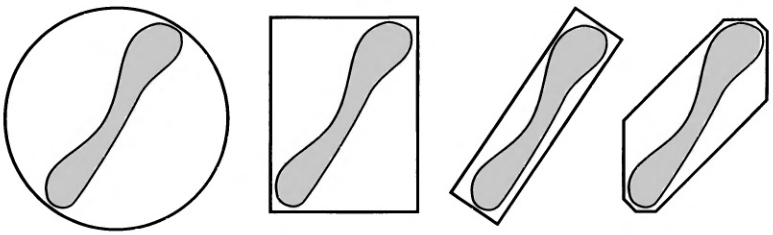

Figure 16.19. The efficiency of a bounding volume can be estimated by the "empty" volume; the more empty space, the worse the fit. A sphere (left), an AABB (middle left), an OBB (middle right), and a k-DOP (right) are shown for an object, where the OBB and the k-DOP clearly have less empty space than the others.