```
    for (int k = p.numVerts, i = 0, j = k - 1; i < k; j = i, i++) {
        float t;
        Point q;
        // Test if edge (p.v[j], p.v[i]) intersects s
        if (IntersectRaySphere(p.v[j], p.v[i] - p.v[j], s, t, q) && t <= 1.0f)
            return 1;
    }
    // Test if the orthogonal projection q of the sphere center onto m is inside p
    Point q = ClosestPtPointPlane(s.c, m);
    return PointInPolygon(q, p);
}
```

As an optimization, for steps 2 and onward it is possible to project the sphere and polygon into the principal plane where the polygon has the largest area and treat the problem as the 2D test of a circle against a polygon. This reduces the overall number of arithmetic operations required for the test.

### 5.2.9 Testing AABB Against Triangle

The test of a triangle $T$ intersecting a box $B$ can be efficiently implemented using a separating-axis approach ([Eberly01], [Akenine-Möller01]). There are 13 axes that must be considered for projection:

1. Three face normals from the AABB
2. One face normal from the triangle
3. Nine axes given by the cross products of combination of edges from both

As before, as soon as a separating axis is found the test can immediately exit with a "no intersection" result. If all axes are tested and no separating axis is found, the box and the triangle must be intersecting. It has been suggested that the most efficient order in which to perform these three sets of tests is 3-1-2 [Akenine-Möller01].

The same 13 axis tests apply to both OBBs and AABBs. However, for an AABB (because the local axes of the box are known) some optimizations can be made to speed up the runtime calculations required for the test. Here, only the AABB test is presented, but to better illustrate the similarities — as well as to facilitate the AABB-specific optimizations — the AABB is assumed to be given in a form commonly used for OBBs. That is, by a center $C$; local axes $\mathbf{u}_0 = (1, 0, 0)$, $\mathbf{u}_1 = (0, 1, 0)$, and $\mathbf{u}_2 = (0, 0, 1)$; and extents $e_0$, $e_1$, and $e_2$. The triangle it is tested against is given by points $V_0 = (v_{0x}, v_{0y}, v_{0z})$, $V_1 = (v_{1x}, v_{1y}, v_{1z})$, and $V_2 = (v_{2x}, v_{2y}, v_{2z})$.

The three face normals ($\mathbf{u}_0$, $\mathbf{u}_1$, $\mathbf{u}_2$) from the AABB are trivially tested by computing the AABB of the triangle and testing $B$ and the AABB of the triangle for overlap. If the two AABBs do not intersect, neither do $B$ and $T$. Testing the axis parallel to the triangle face normal corresponds to testing if the AABB intersects the plane of the triangle. As this test was described in Section 5.2.3, it is not further elaborated on here. Remaining is testing the nine axes corresponding to the cross products of the three edge directions from $B$ and the three edge directions from $T$.

As with most separating-axis tests, the computations are simplified by moving one object (the symmetrical one, if present) to the origin. Here, the box center is moved to align with the origin, resulting in the following variable updates: $V_0 \leftarrow V_0 - C$, $V_1 \leftarrow V_1 - C$, $V_2 \leftarrow V_2 - C$, and $C \leftarrow (0, 0, 0)$. Let the triangle edges be given by $\mathbf{f}_0 = V_1 - V_0 = (f_{0x}, f_{0y}, f_{0z})$, $\mathbf{f}_1 = V_2 - V_1 = (f_{1x}, f_{1y}, f_{1z})$, and $\mathbf{f}_2 = V_0 - V_2 = (f_{2x}, f_{2y}, f_{2z})$. The nine axes considered as separating axes can then be specified as $\mathbf{a}_{ij} = \mathbf{u}_i \times \mathbf{f}_j$. Because $\mathbf{u}_0$, $\mathbf{u}_1$, and $\mathbf{u}_2$ have a simple form, these axes simplify as follows:

$$\begin{aligned}
\mathbf{a}_{00} &= \mathbf{u}_0 \times \mathbf{f}_0 = (1,0,0) \times \mathbf{f}_0 = (0, -f_{0z}, f_{0y}) \\
\mathbf{a}_{01} &= \mathbf{u}_0 \times \mathbf{f}_1 = (1,0,0) \times \mathbf{f}_1 = (0, -f_{1z}, f_{1y}) \\
\mathbf{a}_{02} &= \mathbf{u}_0 \times \mathbf{f}_2 = (1,0,0) \times \mathbf{f}_2 = (0, -f_{2z}, f_{2y}) \\
\mathbf{a}_{10} &= \mathbf{u}_1 \times \mathbf{f}_0 = (0,1,0) \times \mathbf{f}_0 = (f_{0z}, 0, -f_{0x}) \\
\mathbf{a}_{11} &= \mathbf{u}_1 \times \mathbf{f}_1 = (0,1,0) \times \mathbf{f}_1 = (f_{1z}, 0, -f_{1x}) \\
\mathbf{a}_{12} &= \mathbf{u}_1 \times \mathbf{f}_2 = (0,1,0) \times \mathbf{f}_2 = (f_{2z}, 0, -f_{2x}) \\
\mathbf{a}_{20} &= \mathbf{u}_2 \times \mathbf{f}_0 = (0,0,1) \times \mathbf{f}_0 = (-f_{0y}, f_{0x}, 0) \\
\mathbf{a}_{21} &= \mathbf{u}_2 \times \mathbf{f}_1 = (0,0,1) \times \mathbf{f}_1 = (-f_{1y}, f_{1x}, 0) \\
\mathbf{a}_{22} &= \mathbf{u}_2 \times \mathbf{f}_2 = (0,0,1) \times \mathbf{f}_2 = (-f_{2y}, f_{2x}, 0)
\end{aligned}$$

Recall that the projection radius of a box with respect to an axis $\mathbf{n}$ is given by

$$r = e_0 \left| \mathbf{u}_0 \cdot \mathbf{n} \right| + e_1 \left| \mathbf{u}_1 \cdot \mathbf{n} \right| + e_2 \left| \mathbf{u}_2 \cdot \mathbf{n} \right|.$$

In the case of $\mathbf{n} = \mathbf{a}_{00}$, this simplifies to

$$\begin{aligned}
&r = e_0 \left| \mathbf{u}_0 \cdot \mathbf{a}_{00} \right| + e_1 \left| \mathbf{u}_1 \cdot \mathbf{a}_{00} \right| + e_2 \left| \mathbf{u}_2 \cdot \mathbf{a}_{00} \right| \Leftrightarrow \\
&r = e_0 \left| 0 \right| + e_1 \left| -\mathbf{a}_{00z} \right| + e_2 \left| \mathbf{a}_{00y} \right| \Leftrightarrow \\
&r = e_1 \left| f_{0z} \right| + e_2 \left| f_{0y} \right|.
\end{aligned}$$

Similarly, simple expressions for the AABB projection radius are easily obtained for the remaining $\mathbf{a}_{ij}$ axes. In all cases, the projection interval of the box is simply $[-r, r]$.

Projecting $T$ onto a given axis $\mathbf{n}$ results in the projection interval $\left[\min(p_0, p_1, p_2), \max(p_0, p_1, p_2)\right]$, where $p_0$, $p_1$, and $p_2$ are the distances from the origin to the projections of the triangle vertices onto $\mathbf{n}$. For $\mathbf{n} = \mathbf{a}_{00}$ this gives:

$$\begin{aligned}
p_0 &= V_0 \cdot \mathbf{a}_{00} = V_0 \cdot (0, -f_{0z}, f_{0y}) = -v_{0y}f_{0z} + v_{0z}f_{0y} = -v_{0y}(v_{1z} - v_{0z}) + v_{0z}(v_{1y} - v_{0y}) \\
&= -v_{0y}v_{1z} + v_{0z}v_{1y} \\
p_1 &= V_1 \cdot \mathbf{a}_{00} = V_1 \cdot (0, -f_{0z}, f_{0y}) = -v_{1y}f_{0z} + v_{1z}f_{0y} = -v_{1y}(v_{1z} - v_{0z}) + v_{1z}(v_{1y} - v_{0y}) \\
&= v_{1y}v_{0z} - v_{1z}v_{0y} = p_0 \\
p_2 &= V_2 \cdot \mathbf{a}_{00} = V_2 \cdot (0, -f_{0z}, f_{0y}) = -v_{2y}f_{0z} + v_{2z}f_{0y} = -v_{2y}(v_{1z} - v_{0z}) + v_{2z}(v_{1y} - v_{0y})
\end{aligned}$$

If the projection intervals $[-r, r]$ and $\left[\min(p_0, p_1, p_2), \max(p_0, p_1, p_2)\right]$ are disjoint for the given axis, the axis is a separating axis and the triangle and the AABB do not overlap.

For this axis, $\mathbf{n} = \mathbf{a}_{00}$, it holds that $p_0 = p_1$ and the projection interval for the triangle simplifies to $\left[\min(p_0, p_2), \max(p_0, p_2)\right]$. The triangle projection intervals simplify in the same manner for all nine projection axes, as they all contain one zero component. Here, the AABB and triangle projection intervals are therefore disjoint if $\max(p_0, p_2) < -r$ or $\min(p_0, p_2) > r$. If min() and max() operations are available as native floating-point instructions on the target architecture, an equivalent expression that avoids one (potentially expensive) comparison is $\max(-\max(p_0, p_2), \min(p_0, p_2)) > r$. The latter formulation is especially useful in an SIMD implementation (see Chapter 13 for more on SIMD implementations).

The following code fragment illustrates how this test can be implemented.

```
int TestTriangleAABB(Point v0, Point v1, Point v2, AABB b)
{
    float p0, p1, p2, r;

    // Compute box center and extents (if not already given in that format)
    Vector c = (b.min + b.max) * 0.5f;
    float e0 = (b.max.x - b.min.x) * 0.5f;
    float e1 = (b.max.y - b.min.y) * 0.5f;
    float e2 = (b.max.z - b.min.z) * 0.5f;

    // Translate triangle as conceptually moving AABB to origin
    v0 = v0 - c;
    v1 = v1 - c;
    v2 = v2 - c;

    // Compute edge vectors for triangle
    Vector f0 = v1 - v0,  f1 = v2 - v1, f2 = v0 - v2;

    // Test axes a00..a22 (category 3)
    // Test axis a00
    p0 = v0.z*v1.y - v0.y*v1.z;
    p2 = v2.z*(v1.y - v0.y) - v2.z*(v1.z - v0.z);
    r = e1 * Abs(f0.z) + e2 * Abs(f0.y);
    if (Max(-Max(p0, p2), Min(p0, p2)) > r) return 0;  // Axis is a separating axis

    // Repeat similar tests for remaining axes a01..a22
    ...

    // Test the three axes corresponding to the face normals of AABB b (category 1).
    // Exit if...
    // ... [-e0, e0] and [min(v0.x,v1.x,v2.x), max(v0.x,v1.x,v2.x)] do not overlap
    if (Max(v0.x, v1.x, v2.x) < -e0 || Min(v0.x, v1.x, v2.x) > e0) return 0;
    // ... [-e1, e1] and [min(v0.y,v1.y,v2.y), max(v0.y,v1.y,v2.y)] do not overlap
    if (Max(v0.y, v1.y, v2.y) < -e1 || Min(v0.y, v1.y, v2.y) > e1) return 0;
    // ... [-e2, e2] and [min(v0.z,v1.z,v2.z), max(v0.z,v1.z,v2.z)] do not overlap
    if (Max(v0.z, v1.z, v2.z) < -e2 || Min(v0.z, v1.z, v2.z) > e2) return 0;

    // Test separating axis corresponding to triangle face normal (category 2)
    Plane p;
    p.n = Cross(f0, f1);
    p.d = Dot(p.n, v0);
    return TestAABBPlane(b, p);
}
```

Note that there are robustness issues related to the tests of categories 2 (in computing the face normal for a degenerate or oversized triangle) and 3 (the cross product of two parallel edges giving a zero vector). See Section 5.2.1.1 for a discussion of what has to be done to cover these cases in a fully robust implementation.

The topic of triangle-AABB overlap testing is discussed in [Voorhies92]. A test of arbitrary polygons against an AABB is given in [Green95].

### 5.2.10 Testing Triangle Against Triangle

Many algorithms have been suggested for detecting the intersection of two triangles *ABC* and *DEF*. The most straightforward test is based on the fact that in general when two triangles intersect either two edges of one triangle pierce the interior of the other or one edge from each triangle pierces the interior of the other triangle (Figure 5.19).