Ray-Polygon and Ray-Quadric Intersection Testing

We discuss convex polygons first, then concave polygons, and finally, quadric surfaces.

Intersecting a Ray with a Convex Polygon

Given a convex polygon with n vertices (x_i, y_i, z_i) for $0 \le i < n$, and a ray $\mathbf{p} + t\mathbf{d}$, where $|\mathbf{d}| = 1$ and $t > 0$, we want to determine if the ray intersects the polygon. We break the problem down into two steps:

1. Find intersection of ray with the plane of the polygon.
2. Test if this intersection point is inside the polygon.

Finding the Plane Equation of a Polygon

A plane is described by an equation[1] of the form $Ax + By + Cz + D = 0$. The three coefficients (A, B, C) are the plane's normal. We'll assume that the normal is unit length ($A^2 + B^2 + C^2 = 1$) to simplify later calculations.

The normal vector can be calculated in several ways. The simplest is to compute the cross product of the vectors along two consecutive edges of the polygon. This will work fine in most cases, but if the edges happen to be colinear or zero-length, then the computed normal will be zero-length. A better approach is to look at all of the vertices, not just three of them. If we compute the areas of the orthographic projection of the polygon onto the yz, zx, and xy planes then we can use these areas directly for A, B, and C.

Those areas are:

$$A = \sum_{i=0}^{n-1} (y_i - y_{i+1})(z_i + z_{i+1}) \quad = 2 \times \text{area of projection onto } yz \text{ plane}$$

$$B = \sum_{i=0}^{n-1} (z_i - z_{i+1})(x_i + x_{i+1}) \quad = 2 \times \text{area of projection onto } zx \text{ plane}$$

$$C = \sum_{i=0}^{n-1} (x_i - x_{i+1})(y_i + y_{i+1}) \quad = 2 \times \text{area of projection onto } xy \text{ plane}$$

The indexing here should be taken modulo n.

[1]Many of the equations for planes can also be found in Foley et. al sections 11.1.3 and 15.10.1.

Digression: The formulas for A and B are easily derived from the formula for C by cyclic permutation of the symbols x, y, z and A, B, C. That is, to get the formula for A from the formula for C, or to get the formula for B from the formula for A, replace symbols with the cyclic rules: $x \to y \to z \to x$ and $A \to B \to C \to A$. If we use the cyclic permutation mnemonic and resist the urge to re-order zx into xz, there's no need to flip the sign of B, as suggested by Foley (p. 477).

Foley's figure 11.6 illustrates nicely how this 2-D polygon area formula works: it's summing the signed area of trapezoids defined by each polygon edge and one of the axes. Note that these polygon area formulas work even if the polygon is concave. If the polygon is convex, each coefficient will be positive if the projected polygon is counterclockwise, negative if it is clockwise.

Once we've calculated the normal (A, B, C) as above, we should normalize it to unit length (for later). Then we can calculate the offset D simply by substituting any vertex into the plane equation and solving for D:

$$D = -(Ax_0 + By_0 + Cz_0)$$

These coefficients A, B, C, and D can be precomputed and stored in the polygon data structure before ray casting begins.

Ray-Plane Intersection

Intersecting a ray with a plane is easy: we substitute the parametric equation for the ray:

$$(x, y, z) = (p_x, p_y, p_z) + t(d_x, d_y, d_z)$$

into the implicit equation for the plane, yielding:

$$Ap_x + Bp_y + Cp_z + t(Ad_x + Bd_y + Cd_z) + D = 0$$

and solving for t:

$$t = -\frac{Ap_x + Bp_y + Cp_z + D}{Ad_x + Bd_y + Cd_z}$$

If the denominator is zero, that tells us that the ray is parallel to the plane. If $t \leq 0$ then the ray (strictly speaking, the ray is the portion of the line for which $t > 0$) does not intersect the plane, and we need not continue further.

If $t > 0$ then we go on to check whether the intersection point $\mathbf{p} + t\mathbf{d}$ is inside the polygon.

3-D Point-in-Polygon Testing

There are (at least) two ways to test if a 3-D point is inside a 3-D polygon (where the point lies on the plane of the polygon). One can work in 3-D, or one can project the problem down to 2-D and work there. Relative to the second method, the first method requires less code, but it is slower.

Method 1: Work in 3-D

If we construct a plane for each edge of the polygon which is tangent to both the edge and the plane's normal and perpendicular to the polygon, then a point in the plane of the polygon is inside the convex polygon if and only if it is on the same side of each of these planes as the polygon itself.

To be specific, let the vertices be $\mathbf{v}_i = (x_i, y_i, z_i)$ for $i = 0..n-1$, the signed normal to the polygon be $\mathbf{n} = (A, B, C)$, where $\mathbf{n}$ points to the side of the polygon from which the vertices appear counterclockwise, and the test point be $\mathbf{x} = (x, y, z)$. The plane for edge i satisfying the conditions above has equation $(\mathbf{x} - \mathbf{v}_i) \cdot \mathbf{e}_i = 0$, where $\mathbf{e}_i = (\mathbf{v}_{i+1} - \mathbf{v}_i) \times \mathbf{n}$. The vector $\mathbf{e}_i$ is perpendicular to both edge i and the plane's normal, and it contains the edge. Since $\mathbf{e}_i$ points away from the polygon (verify this using the right hand rule for cross products), a point will be on the same side of this plane as the polygon if $(\mathbf{x} - \mathbf{v}_i) \cdot \mathbf{e}_i < 0$. Consequently, a point in the plane of the polygon is inside the polygon if and only if $(\mathbf{x} - \mathbf{v}_i) \cdot \mathbf{e}_i < 0$ for all i. These tests are geometrically equivalent to testing if the point $\mathbf{x}$ is inside the intersection of the halfspaces defined by each of the planes.

For briefest code, $\mathbf{e}_i$ can be computed on the fly[2], but to speed things up, $\mathbf{e}_i$ can be precomputed and saved before ray casting begins. One could optimize a bit more by expanding the test expression like so: $(\mathbf{x} - \mathbf{v}_i) \cdot \mathbf{e}_i = \mathbf{x} \cdot \mathbf{e}_i - \mathbf{v}_i \cdot \mathbf{e}_i$ and precomputing $\mathbf{v}_i \cdot \mathbf{e}_i$.

Method 2: Project to 2-D

The second method projects everything to 2-D. This will be faster on most machines, but it involves more conditionals and more code.

We want to orthographically project into either the yz, zx, or xy plane, selecting the one that has the projection of greatest area. But recall that A, B, and C are proportional to these areas! So we simply test which of $|A|$, $|B|$, or $|C|$ is largest:

if $|A|$ is largest, then project into yz: let $u_i = y_i$ and $v_i = z_i$.
if $|B|$ is largest, then project into zx: let $u_i = z_i$ and $v_i = x_i$.
if $|C|$ is largest, then project into yz: let $u_i = x_i$ and $v_i = y_i$.

and we project the test point into 2-D the same way, yielding a point (u, v). We've now reduced 3-D point-in-polygon testing to 2-D point-in-polygon testing.

Note that we can pre-compute, for each polygon, which of the three projection planes to use.

2-D Point-in-Polygon Testing

We want to test if a point (u, v) is contained in a 2-D polygon with n vertices (u_i, v_i). We'll assume that the polygon is convex so we can use a simpler algorithm.

---

[2] The calculation you're doing in this case is $(\mathbf{x} - \mathbf{v}_i) \cdot \big((\mathbf{v}_{i+1} - \mathbf{v}_i) \times \mathbf{n}\big)$, which is called the *triple scalar product* of vectors $\mathbf{v}_{i+1} - \mathbf{v}_i$, $\mathbf{n}$, and $\mathbf{x} - \mathbf{v}_i$. It is the signed volume of the parallelepiped with those vectors as edges. It is also the determinant of the 3×3 matrix with those vectors as rows.

A convex polygon can be represented by an intersection of half-spaces. Let's call the line segment between vertices i and $i+1$ "edge i". The equation of the line containing edge i can be written in the form $e_i(u,v) = 0$, where $e_i(u,v) = a_i u + b_i v + c_i$. The coefficients a_i, b_i, and c_i are easily found from the equation for a line through two points (u_i, v_i) and (u_{i+1}, v_{i+1}). The signs of a_i, b_i, and c_i can be chosen so that $e_i(u,v)$ is negative for any point inside the polygon (you many need to flip their signs if the polygon is clockwise). The polygon is then the intersection of the halfspaces $e_i(u,v) < 0$ for $i = 0 \ldots n-1$. To test if the point is in the polygon, we loop over all n edges evaluating e_i at that point. If any $e_i \geq 0$ then the point is outside the polygon, but if $e_i < 0$ for all edges then it's inside.

The coefficients a_i, b_i, and c_i can be precomputed, for each polygon, for each edge.

Debugging Tip for Method 2

To debug an implementation of ray-polygon intersection using method 2, it is advisable to do a standalone test of the trickiest part, 2-D point-in-polygon testing. An easy, helpful way to test is to do simple scan conversion of a single polygon something like this:

```
<read in polygon vertices from command line or standard input>
for (y=20-1; y>=0; y--) {
    for (x=0; x<20; x++)
        if (inside_polygon(polygon,x+.5,y+.5)) printf("@");
        else printf(":");
    printf("\n");
}
```

and test it on polygons with coordinates in the range $[0, 20]$, both counterclockwise and clockwise.

Intersecting a Ray with a Concave Polygon

For a planar, concave polygon, the approach is the same: we first intersect the ray with the plane of the polygon, just as before, then we test if the point is inside the polygon. Point-in-concave-polygon testing is more complex than point-in-convex-polygon testing because a concave polygon cannot be represented as an intersection of halfspaces. Instead of covering the algorithm in detail here, I recommend the articles: "An Incremental Angle Point in Polygon Test" by Kevin Weiler, and "Point in Polygon Strategies" by Eric Haines, in *Graphics Gems IV*, Paul Heckbert, ed., Academic Press, 1994.

Intersecting a Ray with a Quadric Surface

Quadric surfaces are second degree algebraic surfaces[3] of the form

$$ax^2 + 2bxy + 2cxz + 2dx + ey^2 + 2fyz + 2gy + 2hz^2 + 2iz + j = 0$$

They include spheres, ellipsoids, paraboloids, hyperboloids of one and two sheets, hyperbolic paraboloids, cones, cylinders, and several degenerate forms including planes.

The general equation can be rewritten in matrix notation as

$$\begin{pmatrix} x & y & z & 1 \end{pmatrix} \begin{pmatrix} a & b & c & d \\ b & e & f & g \\ c & f & h & i \\ d & g & i & j \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = 0$$

(expand this out to convince yourself that this is the same equation). Or, if we let $\mathbf{x} = (x, y, z, 1)^T$, where superscript "T" represents matrix transpose, and $\mathbf{Q}$ is the 4×4 matrix above, then the quadric equation can be written concisely as $\mathbf{x}^T\mathbf{Q}\mathbf{x} = 0$.

To intersect a ray $\mathbf{x} = \mathbf{p} + t\mathbf{d}$ with a quadric surface[4], we simply substitute $\mathbf{x}$ into the matrix equation above. Expanding out, we get

$$\begin{aligned} 0 = \mathbf{x}^T\mathbf{Q}\mathbf{x} &= (\mathbf{p} + t\mathbf{d})^T\mathbf{Q}(\mathbf{p} + t\mathbf{d}) \\ &= (\mathbf{d}^T\mathbf{Q}\mathbf{d})t^2 + (2\mathbf{d}^T\mathbf{Q}\mathbf{p})t + (\mathbf{p}^T\mathbf{Q}\mathbf{p}) \end{aligned}$$

This is a quadratic equation in t. If it has two real roots, then the ray's line intersects the surface twice, if it has one real root of multiplicity two then the ray grazes the surface, and if it has two complex roots (negative discriminant) then the ray misses the surface. If there are real roots, their signs should be tested, since they are at or behind the origin of the ray if $t \leq 0$.

The normal to the quadric at any point $\mathbf{x}$ on the surface is the gradient of the quadric expression: $\mathbf{n} = \nabla(\frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x}) = \mathbf{Q}\mathbf{x}$.

The expressions above can be optimized for restricted classes of quadric surfaces such as spheres, cylinders, etc.[5]

[3] See also section 6-4 of *Mathematical Elements for Computer Graphics*, by David Rogers and Alan Adams, McGraw-Hill, 1990.

[4] Now **x**, **p**, and **d** are homogeneous 4-vectors. The homogeneous coordinates of **x** and **p** is 1 since they are points and the homogeneous coordinate of **d** is 0, since it's a vector.

[5] Foley et. al section 15.10.1 has formulas for ray-sphere intersection. For more, see "Writing a Ray Tracer" by Paul Heckbert, and "Essential Ray Tracing Algorithms" by Eric Haines, in *Introduction to Ray Tracing*, Andrew Glassner, ed., Academic Press, 1989, And see "A Survey of Ray-Surface Intersection Algorithms" by Pat Hanrahan, in the same book, for intersection formulas and algorithms for other surface types besides polygons and quadrics.