Using a ray tracing method with polygonal databases, we must define a fast algorithm to compute ray–polygon intersection. The following algorithm is quite similar to but faster than the barycentric approach described in Snyder and Barr (1987).

The goal of the algorithm is first to determine if a ray goes through the polygon, and then to determine the coordinates of the intersection point and parameters, to localize this point with respect to the polygon's vertices. These parameters are used to compute the interpolated normal at this point, and can be used also to compute the entry of a texture map.

First Step: Intersecting the Embedding Plane

This step is common with the other intersection algorithms but can be presented again. A polygon is described by its vertices \( V_i \) \( (i \in \{0, \ldots, n - 1\}, n \geq 3) \). Let \( x_i, y_i, \) and \( z_i \) the coordinates of the vertex \( V_i \). The normal of the plane containing the polygon, \( \mathbf{N} \), is computed with the cross product:

\[
\mathbf{N} = \overrightarrow{V_0 V_1} \times \overrightarrow{V_0 V_2}.
\]

For each point \( P \) of the plane, the quantity \( P \cdot \mathbf{N} \) is constant. This constant value is computed by the dot product \( d = -\mathbf{V_0} \cdot \mathbf{N} \). The implicit
representation of the plane,
\[ \mathbf{N} \cdot \mathbf{P} + d = 0, \]  
(1)
is computed once, and then stored in the polygon description.

Let the parametric representation of the ray be
\[ r(t) = \mathbf{O} + \mathbf{D}t. \]  
(2)
The evaluation of the parameter \( t \) corresponding to the intersection point can be obtained using the equations (1) and (2):
\[ t = -\frac{d + \mathbf{N} \cdot \mathbf{O}}{\mathbf{N} \cdot \mathbf{D}}. \]  
(3)
This calculation requires 12 floating operations and three tests:
- If polygon and ray are parallel (\( \mathbf{N} \cdot \mathbf{D} = 0 \)), the intersection is rejected.
- If the intersection is behind the origin of the ray (\( t \leq 0 \)), the intersection is rejected.
- If a closer intersection has been already found for the ray (\( t > t_{ray} \)), the intersection is rejected.

**Second Step: Intersecting the Polygon**

A parametric resolution is now presented. This solution is based on triangles. If a polygon has \( n \) vertices (\( n > 3 \)), it will be viewed as a set of \( n - 2 \) triangles. For this reason, the resolution is restricted to convex polygons. The point \( \mathbf{P} \) (see Fig. 1) is given by
\[ \overrightarrow{V_0P} = \alpha \overrightarrow{V_0V_1} + \beta \overrightarrow{V_0V_2}. \]  
(4)
The point \( \mathbf{P} \) will be inside the triangle \( (\triangle V_0 V_1 V_2) \) if
\[ \alpha \geq 0, \beta \geq 0, \text{ and } \alpha + \beta \leq 1. \]
Equation (4) has three components:
\[
\begin{align*}
x_P - x_0 &= \alpha (x_1 - x_0) + \beta (x_2 - x_0) \\
y_P - y_0 &= \alpha (y_1 - y_0) + \beta (y_2 - y_0) \\
z_P - z_0 &= \alpha (z_1 - z_0) + \beta (z_2 - z_0). 
\end{align*}
\]  
(5)
A solution exists and is unique. To reduce this system, we wish to project the polygon onto one of the primary planes, either $xy$, $xz$, or $yz$. If the polygon is perpendicular to one of these planes, its projection onto that plane will be a single line. To avoid this problem, and to make sure that the projection is as large as possible, we find the dominant axis of the normal vector and use the plane perpendicular to that axis. As in Snyder and Barr (1987), we compute the value $i_0$,

$$i_0 = \begin{cases} 
0 & \text{if } |N_x| = \max(|N_x|, |N_y|, |N_z|) \\
1 & \text{if } |N_y| = \max(|N_x|, |N_y|, |N_z|) \\
2 & \text{if } |N_z| = \max(|N_x|, |N_y|, |N_z|) 
\end{cases}$$

Consider $i_1$ and $i_2$ ($i_1$ and $i_2 \in \{0, 1, 2\}$), the indices different from $i_0$. They represent the primary plane used to project the polygon. Let $(u, v)$ be the two-dimensional coordinates of a vector in this plane; the coordinates of $\overrightarrow{V_0P}$, $\overrightarrow{V_0V_1}$, and $\overrightarrow{V_0V_2}$, projected onto that plane, are

$$u_0 = P_{i_1} - V_{0_{i_1}} \quad u_1 = V_{1_{i_1}} - V_{0_{i_1}} \quad u_2 = V_{2_{i_1}} - V_{0_{i_1}}$$

$$v_0 = P_{i_2} - V_{0_{i_2}} \quad v_1 = V_{1_{i_2}} - V_{0_{i_2}} \quad v_2 = V_{2_{i_2}} - V_{0_{i_2}}$$

Equations 5 then reduce to

$$\begin{cases} 
  u_0 = \alpha u_1 + \beta u_2 \\
  v_0 = \alpha v_1 + \beta v_2 
\end{cases}$$
The solutions are
\[ \alpha = \frac{\det\begin{pmatrix} u_0 & u_2 \\ v_0 & v_2 \end{pmatrix}}{\det\begin{pmatrix} u_1 & u_2 \\ v_1 & v_2 \end{pmatrix}} \quad \text{and} \quad \beta = \frac{\det\begin{pmatrix} u_1 & u_0 \\ v_1 & v_0 \end{pmatrix}}{\det\begin{pmatrix} u_1 & u_2 \\ v_1 & v_2 \end{pmatrix}}. \]

The interpolated normal from the point \( P \) may be computed by
\[ \mathbf{N}_P = (1 - (\alpha + \beta)) \mathbf{N}_0 + \alpha \mathbf{N}_1 + \beta \mathbf{N}_2. \]

**Pseudo-Code for a Ray – Triangle Intersection**

\[ \begin{align*}
O: & \text{ point; } \quad \text{Origin of the ray} \\
D: & \text{ vector; } \quad \text{Direction of the ray} \\
P: & \text{ point; } \quad \text{Intersection point} \\
V: & \text{ array[0..2] of point; } \quad \text{Polygon vertices}
\end{align*} \]

\[ \begin{align*}
P & \leftarrow O + Dt; \\
i_1 \text{ and } i_2 & \text{ are in the polygon description.} \\
u_0 & \leftarrow P[i_1] - V[0][i_1]; \\
v_0 & \leftarrow P[i_2] - V[0][i_2]; \\
u_1 & \leftarrow V[1][i_1] - V[0][i_1]; \\
u_2 & \leftarrow V[2][i_1] - V[0][i_1]; \\
v_1 & \leftarrow V[1][i_2] - V[0][i_2]; \\
v_2 & \leftarrow V[2][i_2] - V[0][i_2]; \\
\text{if } u_1 & \text{ is } 0:
\begin{align*}
& \text{then } \beta \leftarrow u_0/u_2; \\
& \text{if } 0 \leq \beta \leq 1:
\begin{align*}
& \text{then } \alpha \leftarrow (v_0 - \beta \cdot v_2)/v_1; \\
& \text{else } \beta \leftarrow (v_0 \cdot u_1 - u_0 \cdot v_1)/(v_2 \cdot u_1 - u_2 \cdot v_1); \\
& \text{if } 0 \leq \beta \leq 1:
\end{align*}
\end{align*}
\end{align*} \]

The values \( \alpha \) and \( \beta \) are the interpolation parameters.

return (\( \alpha \geq 0 \) and \( \beta \geq 0 \) and \( \alpha + \beta \leq 1 \))

*See also* Efficient Generation of Sampling Jitter Using Look-up Tables (64); Fast Line–Edge Intersections on a Uniform Grid (29); Transforming Axis-Aligned Bounding Boxes (548)

*See* Appendix 2 for C Implementation (735)