*This is article from The Graphics Codex by Morgan McGuire. The full app contains 12 chapters of lecture notes for a physically based rendering course and 225 encylopedia-like reference articles. Buy the web or iOS edition for only $10 at http://graphicscodex.com.*

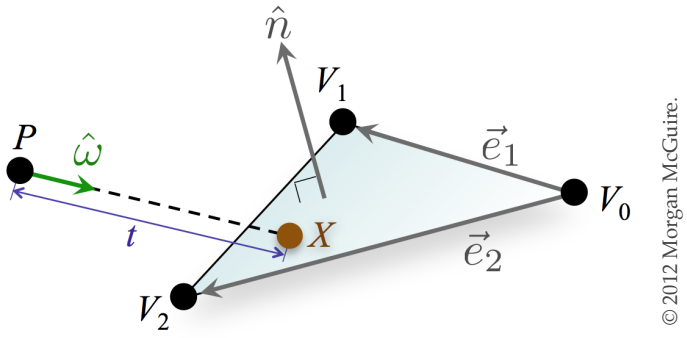
---

Pseudo | **C++** | GLSL

```
/* If ray P + tw hits triangle V[0], V[1], V[2], then the function returns true, stores the barycentric coordinates in b[], and stores the distance to the intersection in t. Otherwise returns false and the other output parameters are undefined. */
bool rayTriangleIntersect(const Point3& P, const Vector3& w, const Point3 V[3], float b[3], float& t) {
    // Edge vectors
    const Vector3& e_1 = V[1] - V[0];
    const Vector3& e_2 = V[2] - V[0];

    // Face normal
    const Vector3& n = e_1.cross(e_2).direction();

    const Vector3& q = w.cross(e_2);
    const float a = e_1.dot(q);

    // Backfacing or nearly parallel?
    if ((n.dot(w) >= 0) || (abs(a) <= eps)) return false;

    const Vector3& s = (P - V_0) / a;
    const Vector3& r = s.cross(e_1);

    b[0] = s.dot(q);
    b[1] = r.dot(w);
    b[2] = 1.0f - b[0] - b[1];

    // Intersected outside triangle?
    if ((b[0] < 0.0f) || (b[1] < 0.0f) || (b[2] < 0.0f)) return false;
```

```
14        t = e_2.dot(r);
15        return (t >= 0.0f);
16    }
```

[ RTR3 p750]




## Implementations

G3D Ray::intersectionTime

G3D CollisionDetection::collisionTimeForMovingPointFixedTriangle

G3D Triangle::intersect G3D Tri::Intersector::operator()

Mitsuba Triangle::rayIntersect PBRT Triangle::Intersect

## Discussion

The first return value of the function is true iff ray $P + t\hat{\omega}$ intersects triangle $(V_0, V_1, V_2)$. If the intersection exists, it is at:

$$X = P + t\hat{\omega} = \sum_{i=0}^{2} (b[i]\ V_i)$$

The barycentric weights $b$ can be applied to texture coordinates, vertex normals, and other per-vertex attributes to interpolate them bilinearly across the triangle to the intersection point.

The epsilon ($\varepsilon$, `eps`) constant must be a small non-negative number, roughly on the order of $10^{-7}$ for 32-bit floating point. If it is too small then the $\vec{s}$ vector will become infinite due to underflow. If it is too large then glancing intersections will be missed.

It is also a common practice (not shown here) to accept any

intersection where the barycentric coordinates are greater than a very small negative value. This slightly increases the size of each triangle by an amount proportional to its edge lengths and ensures that rays do not pass through edges between adjacent triangles in a mesh due to limited numerical precision.

## Symbols

| *Symbol* | *Type* | *Description* | *Ref* |
|---|---|---|---|
| $S(\vec{v})$ | $\mathbb{S}^n$ | Normalize $\vec{v} \in \mathbb{R}^n$ by projecting it onto $\mathbb{S}^n$. | [nrmlz] |

## References


[RTR3]

***Real-Time Rendering 3rd Edition***
Tomas Akenine-Möller, Eric Haines, and Naty Hoffman
p. 1045, A. K. Peters, Ltd., Natick, MA, USA, 2008.