Intersections of Rays and Triangles (3D)

by Dan Sunday

The intersection of the most basic geometric primitives was presented in the Algorithm 5 about Intersections of Lines and Planes. We will now extend those algorithms to include 3D triangles which are common elements of 3D surface and polyhedron models. We only consider transversal intersections where the two intersecting objects do not lie in the same plane. Ray and triangle intersection computation is perhaps the most frequent nontrivial operation in computer graphics rendering using ray tracing. Because of its importance, there are several published algorithms for this problem (see: [Badouel, 1990], [Moller & Trumbore, 1997], [O'Rourke, 1998], [Moller & Haines, 1999]). We present an improvement of these algorithms for ray (or segment) and triangle intersection. We also give algorithms for triangle-plane and triangle-triangle intersection.

Intersection of a Ray/Segment with a Plane

Assume we have a ray **R** (or segment **S**) from P_0 to P_1, and a plane $\mathcal{P}$ through V_0 with normal **n**. The intersection of the parametric line **L**: $P(r) = P_0 + r(P_1 - P_0)$ and the plane $\mathcal{P}$ occurs at the point $P(r_I)$ with parameter value:

$$r_I = \frac{\mathbf{n} \bullet (V_0 - P_0)}{\mathbf{n} \bullet (P_1 - P_0)}$$

When the denominator $\mathbf{n} \bullet (P_1 - P_0) = 0$, the line **L** is parallel to the plane $\mathcal{P}$, and thus either does not intersect it or else lies completely in the plane (whenever either P_0 or P_1 is in $\mathcal{P}$). Otherwise, when the denominator is nonzero and r_I is a real number, then the ray **R** intersects the plane $\mathcal{P}$ only when $r_I \geq 0$. A segment **S** intersects $\mathcal{P}$ only if $0 \leq r_I \leq 1$. In all algorithms, the additional test $r_I \leq 1$ is the only difference for a segment instead of a ray.

Intersection of a Ray/Segment with a Triangle

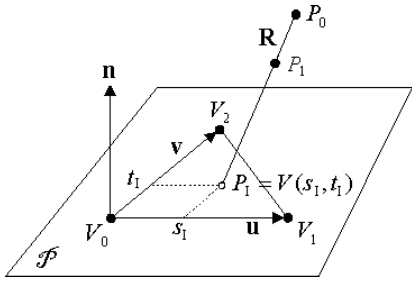
Consider a ray **R** (or a segment **S**) from P_0 to P_1, and a triangle **T** with vertices V_0, V_1 and V_2. The triangle **T** lies in the plane $\mathcal{P}$ through V_0 with normal vector $\mathbf{n} = (V_1 - V_0) \times (V_2 - V_0)$. To get the intersection of **R** (or **S**) with **T**, one first determines the intersection of **R** (or **S**) and $\mathcal{P}$. If it does not intersect, then it also does not intersect **T** and we are done. However, if they intersect in the point $P_I = P(r_I)$, we need to determine if this point is inside the triangle **T** for there to be a valid intersection.

There are a number of ways to test for the inclusion of a point inside a 3D planar triangle. The algorithms given by [Badouel, 1990] and [O'Rourke, 1998] project the point and triangle onto a 2D coordinate plane where inclusion is tested. To implement these algorithms, one must choose a projection coordinate plane which avoids a degenerate projection. This is done by excluding the coordinate which has the largest component in the plane normal vector **n** [Synder & Barr, 1987]. The intent is to reduce the 3D problem to a simpler 2D problem which has an efficient solution. However, there is a small overhead involved in selecting and applying the projection function. The algorithm of [Moller-Trumbore, 1997] (MT) does not project into 2D, and finds a solution using direct 3D computations. Testing with some complex models shows that the MT algorithm is faster than the one by Badouel.

We present here an alternate method that also uses direct 3D computations to determine inclusion, avoiding the projection onto a 2D coordinate plane. As a result, the code is cleaner and more compact. Like [Moller-Trumbore, 1997], we use the parametric equation of $\mathcal{P}$ relative to **T**, but derive a different method of solution which computes the ***parametric coordinates*** of the intersection point in the plane. The parametric plane equation (see: Planes in Algorithm 4) is given by:

$$\mathrm{V}(s,t) = \mathrm{V}_0 + s(\mathrm{V}_1 - \mathrm{V}_0) + t(\mathrm{V}_2 - \mathrm{V}_0) = \mathrm{V}_0 + s\mathbf{u} + t\mathbf{v}$$

where s and t are real numbers, and $\mathbf{u} = V_1 - V_0$ and $\mathbf{v} = V_2 - V_0$ are edge vectors of **T**. Then, $P = V(s,t)$ is in the triangle **T** when $s \geq 0$, $t \geq 0$, and $s + t \leq 1$. So, given P_I, one just has to find the (s_I, t_I) coordinate for it, and then check these inequalities to verify inclusion in **T**. Further, $P = V(s,t)$ is on an edge of **T** if one of the conditions $s = 0$, $t = 0$, or $s + t = 1$ is true (each condition corresponds to one edge). And, the three vertices are given by: $V_0 = V(0,0)$, $V_1 = V(1,0)$, and $V_2 = V(0,1)$.



To solve for s_I and t_I, we apply the method described in Algorithm 4 for Barycentric Coordinate Computation using the "***generalized perp operator*** on $\mathbf{P}$ " that we defined there. Then, with $\mathbf{w} = P_I - V_0$, which is a vector in $\mathbf{P}$ (that is, $\mathbf{n} \cdot \mathbf{w} = 0$), we solve the equation: $\mathbf{w} = s\mathbf{u} + t\mathbf{v}$ for s and t. The final result is:

$$s_I = \frac{(\mathbf{u}\cdot\mathbf{v})(\mathbf{w}\cdot\mathbf{v}) - (\mathbf{v}\cdot\mathbf{v})(\mathbf{w}\cdot\mathbf{u})}{(\mathbf{u}\cdot\mathbf{v})^2 - (\mathbf{u}\cdot\mathbf{u})(\mathbf{v}\cdot\mathbf{v})}$$

$$t_I = \frac{(\mathbf{u}\cdot\mathbf{v})(\mathbf{w}\cdot\mathbf{u}) - (\mathbf{u}\cdot\mathbf{u})(\mathbf{w}\cdot\mathbf{v})}{(\mathbf{u}\cdot\mathbf{v})^2 - (\mathbf{u}\cdot\mathbf{u})(\mathbf{v}\cdot\mathbf{v})}$$

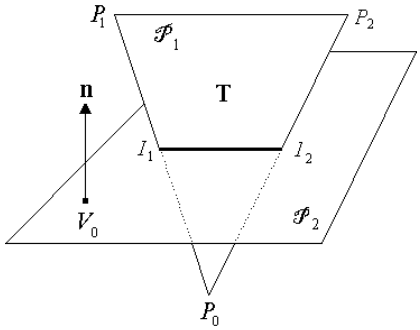
which has only 5 distinct dot products. We have arranged terms so that the two denominators are the same and only need to be calculated once.

This solution yields a straightforward ray/segment-triangle intersection algorithm (see our implementation: intersect3D_RayTriangle()). Based on a count of the operations done up to the first rejection test, this algorithm is a bit less efficient than the MT algorithm, although we have not done any runtime performance comparisons. However, the MT algorithm uses two cross products whereas our algorithm uses only one, and the one we use computes the normal vector of the triangle's plane, which is needed to compute the line parameter r_I. But, when the normal vectors have been precomputed and stored for all triangles in a scene (which is often the case), our algorithm would not have to compute this cross product at all. But, in this case, the MT algorithm would still compute two cross products, and be less efficient than our algorithm. So, the preferred algorithm depends on the application.

Intersection of a Triangle with a Plane

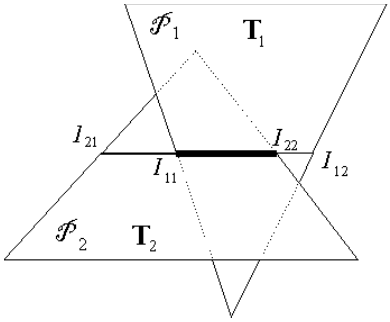
Consider a triangle **T** with vertices P_0, P_1 and P_2 lying in a plane $\mathbf{P}_1$ with normal $\mathbf{n}_1$. Let $\mathbf{P}_2$ be a second plane through the point V_0 with the normal vector $\mathbf{n}_2$. Unless they are parallel, the two planes $\mathbf{P}_1$ and $\mathbf{P}_2$ intersect in a line **L**, and when **T** intersects $\mathbf{P}_2$ it will be a segment contained in **L**. When **T** does not intersect $\mathbf{P}_2$ all three of its vertices must strictgly lie on the same side of the $\mathbf{P}_2$ plane. On the other hand, when **T** does intersect $\mathbf{P}_2$, one point of **T** must be on one side of (or on) $\mathbf{P}_2$ and the other two be on the other side of (or on) $\mathbf{P}_2$. We gave a test for which side of a plane a point is on (by using the signed distance from the point to the plane) in Algorithm 4 for Distance of a Point to a Plane. In fact, to determine the sideness of a point P, one only has to find the sign of $\mathbf{n} \cdot (P - V_0) = 0$.

Suppose that P_0 is on one side of $\mathbf{P}_2$ and that P_1 and P_2 are on the other side. Then the two segments P_0P_1 and P_0P_2 intersect $\mathbf{P}_2$ in two points I_1 and I_2 which are on the intersection line of $\mathbf{P}_1$ and $\mathbf{P}_2$. Then, the segment I_1I_2 is the intersection of triangle **T** and the plane $\mathbf{P}_2$.



Intersection of a Triangle with a Triangle

Consider two triangles $\mathbf{T}_1$ and $\mathbf{T}_2$. They each lie in a plane, respectively $\mathbf{P}_1$ and $\mathbf{P}_2$, and their intersection must be on the line of intersection **L** for the two planes. Let the intersection of $\mathbf{T}_1$ and $\mathbf{P}_2$ be the segment $\mathbf{S}_1 = I_{11}I_{12}$, and the intersection of $\mathbf{T}_2$ and $\mathbf{P}_1$ be $\mathbf{S}_2 = I_{21}I_{22}$. If either $\mathbf{S}_1$ or $\mathbf{S}_2$ doesn't exist (that is, one triangle does not intersect the plane of the other), then $\mathbf{T}_1$ and $\mathbf{T}_2$ do not intersect. Otherwise their intersection is equal to the intersection of the two segments $\mathbf{S}_1$ and $\mathbf{S}_2$ on the line **L**. This can be easily computed by projecting them onto an appropriate coordinate axis, and determining their intersection on it.



Implementations

Here are some sample "C++" implementations of these algorithms.

```
// Copyright 2001 softSurfer, 2012 Dan Sunday
// This code may be freely used and modified for any purpose
// providing that this copyright notice is included with it.
// SoftSurfer makes no warranty for this code, and cannot be held
// liable for any real or imagined damage resulting from its use.
// Users of this code must verify correctness for their application.


// Assume that classes are already given for the objects:
//    Point and Vector with
//        coordinates {float x, y, z;}
//        operators for:
//            == to test  equality
//            != to test  inequality
//            (Vector)0 =  (0,0,0)         (null vector)
//            Point   = Point ± Vector
//            Vector =  Point - Point
//            Vector =  Scalar * Vector    (scalar product)
//            Vector =  Vector * Vector    (cross product)
//    Line and Ray and Segment with defining  points {Point P0, P1;}
//        (a Line is infinite, Rays and  Segments start at P0)
//        (a Ray extends beyond P1, but a  Segment ends at P1)
//    Plane with a point and a normal {Point V0; Vector  n;}
//    Triangle with defining vertices {Point V0, V1, V2;}
//    Polyline and Polygon with n vertices {int n;  Point *V;}
//        (a Polygon has V[n]=V[0])
//===================================================================


#define SMALL_NUM   0.00000001 // anything that avoids division overflow
// dot product (3D) which allows vector operations in arguments
#define dot(u,v)   ((u).x * (v).x + (u).y * (v).y + (u).z * (v).z)



// intersect3D_RayTriangle(): find the 3D intersection of a ray with a triangle
//    Input:  a ray R, and a triangle T
//    Output: *I = intersection point (when it exists)
//    Return: -1 = triangle is degenerate (a segment or point)
//             0 =  disjoint (no intersect)
//             1 =  intersect in unique point I1
//             2 =  are in the same plane
int
intersect3D_RayTriangle( Ray R, Triangle T, Point* I )
{
    Vector    u, v, n;              // triangle vectors
    Vector    dir, w0, w;           // ray vectors
    float     r, a, b;              // params to calc ray-plane intersect

    // get triangle edge vectors and plane normal
    u = T.V1 - T.V0;
    v = T.V2 - T.V0;
    n = u * v;              // cross product
    if (n == (Vector)0)             // triangle is degenerate
        return -1;                  // do not deal with this case

    dir = R.P1 - R.P0;              // ray direction vector
    w0 = R.P0 - T.V0;
    a = -dot(n,w0);
    b = dot(n,dir);
    if (fabs(b) < SMALL_NUM) {     // ray is  parallel to triangle plane
        if (a == 0)                 // ray lies in triangle plane
            return 2;
        else return 0;              // ray disjoint from plane
    }

    // get intersect point of ray with triangle plane
```

```
    r = a / b;
    if (r < 0.0)                    // ray goes away from triangle
        return 0;                   // => no intersect
    // for a segment, also test if (r > 1.0) => no intersect

    *I = R.P0 + r * dir;            // intersect point of ray and plane

    // is I inside T?
    float    uu, uv, vv, wu, wv, D;
    uu = dot(u,u);
    uv = dot(u,v);
    vv = dot(v,v);
    w = *I - T.V0;
    wu = dot(w,u);
    wv = dot(w,v);
    D = uv * uv - uu * vv;

    // get and test parametric coords
    float s, t;
    s = (uv * wv - vv * wu) / D;
    if (s < 0.0 || s > 1.0)         // I is outside T
        return 0;
    t = (uv * wu - uu * wv) / D;
    if (t < 0.0 || (s + t) > 1.0)  // I is outside T
        return 0;

    return 1;                       // I is in T
}
```

References

Didier Badouel, "An Efficient Ray-Polygon Intersection" in Graphics Gems (1990)

Francis Hill, "The Pleasures of 'Perp Dot' Products" in Graphics Gems IV (1994)

Tomas Moller & Eric Haines, "Intersection Test Methods" in Real-Time Rendering (3rd Edition) (2008)

Tomas Moller & Ben Trumbore, "Fast Minimum Storage Ray-Triangle Intersection" in J. Graphics Tools (**jgt**) 2(1), 21-28 (1997)

Joseph O'Rourke, "Segment-Triangle Intersection" in Computational Geometry in C (2nd Edition) (1998)

J.P. Snyder and A.H. Barr, "Ray Tracing Complex Models Containing Surface Tessellations", ACM Comp Graphics 21, (1987)