WIKIPEDIA

Möller–Trumbore intersection algorithm

The **Möller–Trumbore ray-triangle intersection algorithm**, named after its inventors Tomas Möller and Ben Trumbore, is a fast method for calculating the intersection of a ray and a triangle in three dimensions without needing precomputation of the plane equation of the plane containing the triangle.[1] Among other uses, it can be used in computer graphics to implement ray tracing computations involving triangle meshes.[2]

Contents

C++ Implementation

The following is an implementation of the algorithm in C++:

```
bool RayIntersectsTriangle(Vector3D rayOrigin,
                           Vector3D rayVector,
                           Triangle* inTriangle,
                           Vector3D& outIntersectionPoint)
{
    const float EPSILON = 0.0000001;
    Vector3D vertex0 = inTriangle->vertex0;
    Vector3D vertex1 = inTriangle->vertex1;
    Vector3D vertex2 = inTriangle->vertex2;
    Vector3D edge1, edge2, h, s, q;
    float a,f,u,v;
    edge1 = vertex1 - vertex0;
    edge2 = vertex2 - vertex0;
    h = rayVector.crossProduct(edge2);
    a = edge1.dotProduct(h);
    if (a > -EPSILON && a < EPSILON)
        return false;
    f = 1/a;
    s = rayOrigin - vertex0;
    u = f * (s.dotProduct(h));
    if (u < 0.0 || u > 1.0)
        return false;
    q = s.crossProduct(edge1);
    v = f * rayVector.dotProduct(q);
    if (v < 0.0 || u + v > 1.0)
        return false;
    // At this stage we can compute t to find out where the intersection point is on the line.
    float t = f * edge2.dotProduct(q);
    if (t > EPSILON) // ray intersection
    {
        outIntersectionPoint = rayOrigin + rayVector * t;
        return true;
    }
    else // This means that there is a line intersection but not a ray intersection.
        return false;
}
```

See also

- Badouel intersection algorithm
- MATLAB version (http://www.mathworks.com/matlabcentral/fileexchange/33073) of this algorithm (highly vectorized)
- Baldwin-Weber ray-triangle intersection algorithm (http://jcgt.org/published/0005/03/03/)
- Schlick–Subrenat algorithm[3] for ray-quadrilateral intersection

Links

- Fast Minimum Storage Ray-Triangle Intersection (http://webserver2.tecgraf.puc-rio.br/~mgattass/cg/trbRR/Fast%20MinimumStorage%20RayTriangle%20Intersection.pdf)
- Optimizations on the basic algorithm by Möller & Trumbore (https://github.com/erich666/jgt-code/tree/master/Volume_02/Number_1/Moller1997a), code from *journal of graphics tools*

References

1. Möller, Tomas; Trumbore, Ben (1997). "Fast, Minimum Storage Ray-Triangle Intersection". *Journal of Graphics Tools*. **2**: 21–28. doi:10.1080/10867651.1997.10487468 (https://doi.org/10.1080/10867651.1997.10487468).
2. "Ray-Triangle Intersection" (http://www.lighthouse3d.com/tutorials/maths/ray-triangle-intersection). lighthouse3d. Retrieved 2017-09-10.
3. *Ray Intersection of Tessellated Surfaces: Quadrangles versus Triangles (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.44.6955)*, Schlick C., Subrenat G. Graphics Gems 1993

Retrieved from "https://en.wikipedia.org/w/index.php?title=Möller–Trumbore_intersection_algorithm&oldid=831822834"