Fast Ray-Triangle Fan Intersection

Eric Galin

LIRIS, CNRS, Université Claude Bernard Lyon 1, France

Abstract

This paper presents an algorithm for computing the nearest intersection between a ray a fan of triangles. Our approach speeds up computations by rapidly rejecting non intersecting triangles and reusing some intermediate factors shared by neighbouring triangles. Source code is available online.

1. Introduction

Because of their mathematical simplicity and flexibility, triangle meshes are the most widely used representation for modeling three dimensional objects. Therefore, computing the intersection between a set of triangles and a ray is an important problem in many applications.

One approach consist in computing the intersection between the ray and the triangle's plane and then testing if the intersection point lies inside the edges [2, 3]. An alternative approach that need not compute nor store the plane equation or the normal of the triangle was proposed in [4].

Triangles may be organized in some three dimensional structures such as Bounding Box hierarchies, Binary Space Partitioning trees or Octrees so as to avoid unnecessary intersection tests whenever possible. In practice, because of memory limits and especially for large complex scenes including millions of triangles, hundreds of triangles still exist at the leaf nodes of those structures. Therefore, there is still a need for techniques that compute the intersection between a ray and a set of triangles efficiently.

Most techniques process every triangle independently and do not take advantage of the shared information between neighbouring triangles. An efficient ray-triangle mesh intersection algorithm was presented in [5]. The proposed method defines the ray as two intersecting planes and relies on a simple shared vertex location scheme to reject most nonintersecting triangles at an early step in the algorithm. Several techniques have been proposed to factor the computations for two triangles sharing a common edge [1]. Those methods are particularly adapted for processing rectangular parametric patches discretized into triangles.

This paper presents an efficient algorithm to compute the intersection between a ray and a set of triangles organized into a fan structure. Our method incrementally builds upon the algorithm presented in [4]. Optimizations are provided so as to take advantage of shared information between neighbouring triangles sharing a common edge and a common center vertex.

2. Intersection algorithm

Let $\{\mathbf{p}_0,...,\mathbf{p}_n\}$ denote the vertex list of a triangle fan, where $\mathbf{p}_0$ denotes the center vertex (Figure 1). We compute the intersection between a ray and a triangle fan by iteratively computing the nearest intersection between the ray and the set of triangles T_k whose vertices will be denoted as $\mathbf{p}_0$, $\mathbf{p}_k$ and $\mathbf{p}_{k+1}$, $k \in [1, n-1]$.

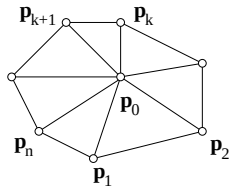

Figure 1: *Notations for a triangle fan*

To optimize the overall process, we not only factorize the computation of constant terms, but also reuse computations for two consecutive triangles that share a common edge.

2.1. Ray triangle fan intersection

Let us briefly recall the algorithm presented in [4]. A point $\mathbf{p}(u,v)$ on a triangle T_k is given by the parametric equation:

$$\mathbf{p}(u,v) = \mathbf{p}_0 + u(\mathbf{p}_k - \mathbf{p}_0) + v(\mathbf{p}_{k+1} - \mathbf{p}_0)$$

The barycentric coordinates (u,v) should fulfil $u \geq 0$, $v \geq 0$ and $u+v \leq 1$. Let $\mathbf{r}(t) = \mathbf{o} + \mathbf{d}t$ denote the parametric equation of a ray. Computing the intersection between a ray and the triangle is equivalent to $\mathbf{r}(t) = \mathbf{p}(u,v)$. Let $\mathbf{e}_1 = \mathbf{p}_k - \mathbf{p}_0$ and $\mathbf{e}_2 = \mathbf{p}_{k+1} - \mathbf{p}_0$ denote the edge vectors. Let $\mathbf{s} = \mathbf{o} - \mathbf{p}_0$ denote the translation vector to the origin of the ray. The solution to the previous equation is obtained by Cramer's rule using four determinants as described in [4]:

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{|-\mathbf{d}\,\mathbf{e}_1\,\mathbf{e}_2|} \begin{pmatrix} |\mathbf{s}\,\mathbf{e}_1\,\mathbf{e}_2| \\ |-\mathbf{d}\,\mathbf{s}\,\mathbf{e}_2| \\ |-\mathbf{d}\,\mathbf{e}_1\,\mathbf{s}| \end{pmatrix}$$

Unlike [4], we rewrite the determinant so that constant terms independent of the edge vectors should appear. From linear algebra, we know that the determinant $|\mathbf{a}\,\mathbf{b}\,\mathbf{c}|$ may be rewritten as:

$$|\mathbf{a}\,\mathbf{b}\,\mathbf{c}| = (\mathbf{a} \wedge \mathbf{b}) \cdot \mathbf{c} = -(\mathbf{a} \wedge \mathbf{c}) \cdot \mathbf{b} = (\mathbf{c} \wedge \mathbf{b}) \cdot \mathbf{a}$$

Therefore, we have:

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{(\mathbf{e}_1 \wedge \mathbf{e}_2) \cdot \mathbf{d}} \begin{pmatrix} -(\mathbf{e}_1 \wedge \mathbf{e}_2) \cdot \mathbf{s} \\ (\mathbf{s} \wedge \mathbf{d}) \cdot \mathbf{e}_2 \\ -(\mathbf{s} \wedge \mathbf{d}) \cdot \mathbf{e}_1 \end{pmatrix}$$

This formulation enables us to perform the following optimizations in the computation of the intersection between a ray and the set of triangles organized in a fan structure:

- We compute both the translation vector $\mathbf{s} = \mathbf{o} - \mathbf{p}_0$ and the constant cross product $\mathbf{n} = \mathbf{s} \wedge \mathbf{d}$ once and for all the triangles;
- We can share the computation of the edges $\mathbf{e}_k = \mathbf{p}_k - \mathbf{p}_0$ as well as the term $(\mathbf{s} \wedge \mathbf{d}) \cdot \mathbf{e}_k$ between two consecutive triangles.

2.2. Fast triangle rejection test

When testing the intersection between the ray and all the triangles T_k of the triangle fan, it is possible to reject some non intersecting triangles easily by classifying the vertices $\mathbf{p}_k$ against a plane.

Let Π denote the plane that contains both the ray and the center vertex $\mathbf{p}_0$. The normal of the plane is simply defined as $\mathbf{n} = \mathbf{s} \wedge \mathbf{d} = (\mathbf{o} - \mathbf{p}_0) \cdot \mathbf{d}$. The equation of the plane is defined by the equation $\Pi(\mathbf{p}) = (\mathbf{p} - \mathbf{p}_0) \cdot \mathbf{n} = 0$. A triangle T_k does not intersect Π if and only if the expressions $\Pi(\mathbf{p}_k)$ and $\Pi(\mathbf{p}_{k+1})$ have the same sign, *i.e.*, if the two vertices $\mathbf{p}_k$ and $\mathbf{p}_{k+1}$ are on the same side of the plane (Figure 2).

Therefore, the ray may intersect the triangle T_k only if the plane intersects T_k. Since the computation of the normal vector $\mathbf{n} = \mathbf{s} \wedge \mathbf{d}$, the edge vectors $\mathbf{e}_k = \mathbf{p}_k - \mathbf{p}_0$ and $\mathbf{n} \cdot \mathbf{e}_k$ are already needed in the computation of the parameters u, v and t, we can compare the signs of $\Pi(\mathbf{p}_k)$ and $\Pi(\mathbf{p}_{k+1})$ as an inexpensive and efficient rejection test.

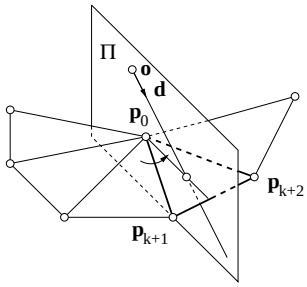

Figure 2: *Intersection between the plane containing the ray and the center vertex, and a triangle fan*

2.3. Overall algorithm

The overall algorithm may be split into two parts: a pre-processing step that evaluates the constant terms, followed by an iteration over the triangles of the triangle fan.

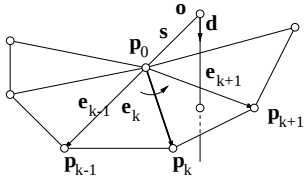

Figure 3: *Intersection between a ray and a triangle fan*

Let $\mathbf{a}$ and $\mathbf{b}$ denote two vectors that will be used to store edge vectors. Let x_a and x_b two scalars that will store the dot product between the edges $\mathbf{a}$ and $\mathbf{b}$ and $\mathbf{n}$ (Figure 3).

The pre-processing step may be outlined as follows:

1. Compute the constant translation vector $\mathbf{s} = \mathbf{o} - \mathbf{p}_0$ and evaluate the cross product with ray direction $\mathbf{n} = \mathbf{s} \wedge \mathbf{d}$.
2. Compute the edge vector $\mathbf{a} = \mathbf{p}_k - \mathbf{p}_0$ and the dot product $x_a = \mathbf{n} \cdot \mathbf{a}$.

The second step of the algorithm iterates over the triangles T_k for all $k \in [1, n-1]$ as follows:

1. Evaluate the edge vector $\mathbf{b} = \mathbf{p}_{k+1} - \mathbf{p}_0$ and the dot product $x_b = \mathbf{n} \cdot \mathbf{b} = (\mathbf{s} \wedge \mathbf{d}) \cdot (\mathbf{p}_{k+1} - \mathbf{p}_0)$;
2. If x_a and x_b have the same sign, then there is no intersection between the ray and the triangle so go to Step 4.
3. Compute the normal of the triangle $\mathbf{n}_k = \mathbf{a} \wedge \mathbf{b}$ and evaluate the determinant $\delta = \mathbf{n}_k \cdot \mathbf{d}$.

3.1 If determinant is close to zero, then the ray lies in the plane of triangle and no intersection occurs.

3.2 Otherwise, set $\alpha = 1/\delta$ and calculate u_k and v_k parameters as well as the intersection depth t_k:

$$u_k = -\alpha x_b \qquad v_k = \alpha x_a \qquad t_k = -\alpha(\mathbf{n}_k \cdot \mathbf{s})$$

The intersection is valid if and only if $u_k \geq 0$, $v_k \geq 0$ and $u_k + v_k \leq 1$.

4. Loop to Step 1, reusing the previously computed values by assigning **b** to **a** and x_b to x_a respectively.

The nearest intersection is obtained by selecting the minimum of positive valid intersection depths.

Some aspects of this algorithm deserve special attention. The computation of the edge vectors can be performed on a pre-processing step, with edges $\mathbf{e}_k = \mathbf{p}_k - \mathbf{p}_0$, $k \in [1,n]$ stored in place of vertices **p** in the structure of the triangle fan. This is possible if the vertex locations are not required for other computations or shared between triangles.

The normal $\mathbf{n}_k = \mathbf{e}_k \wedge \mathbf{e}_{k+1}$ of the triangle T_k may also be stored in the data structure at the expense of an extra memory usage, which is a classical memory-speed trade-off.

To ensure numerical stability, the test that eliminates parallel rays must compare the determinant to a small interval around 0. We use the same constant $\varepsilon = 10^{-5}$ as suggested in [4] which makes the algorithm extremely stable.

3. Implementation and performance

The previous algorithms have been implemented in C++ as member functions of a two triangle fan classes. We tried not only to reduce the overall number of operations in the worst case scenario, but also to identify specific cases that can be processed very efficiently. The algorithms are robust and do not involve divisions by small numbers which could produce floating point overflows. Timings were performed on a Pentium IV 2.4GHz workstation using `g++ -O -s` as compiler command and options.

Algorithm	$+$	$\times$	$/$	?
Möller [4]	$24n$	$27n$	n	$6n$
Triangle Fan	$11+13n$	$9+18n$	n	$7n$
Stored Normals	$8+7n$	$9+12n$	n	$7n$

Table 1: *Number of operations involved in the ray triangle fan intersection algorithm in the worst case*

We have compared our method with the implementation of the reference algorithm proposed in [4]. The total number of operations performed in the worst case are reported in Table 1. The notation n refers to the number of triangles in the triangle fan. The computational cost drops significantly if the edge vectors $\mathbf{e}_k$ and the normal vectors $\mathbf{n}_k$ are stored into the triangle fan structure.

Table 2 presents timings for ray tracing several models. The number of triangles as well as the corresponding number of triangle fans is reported in Figure 4. Timings demonstrate that using triangle fans speeds up intersections by almost 40%.

Algorithm	Bunny	Horse	Victory	Dragon
Möller [4]	42.26	69.51	88.23	125.78
Triangle Fan	33.66	54.98	69.70	102.74
Edges Normals	24.41	40.42	53.05	68.91

Table 2: *Timings (in seconds) for ray tracing some complex models*

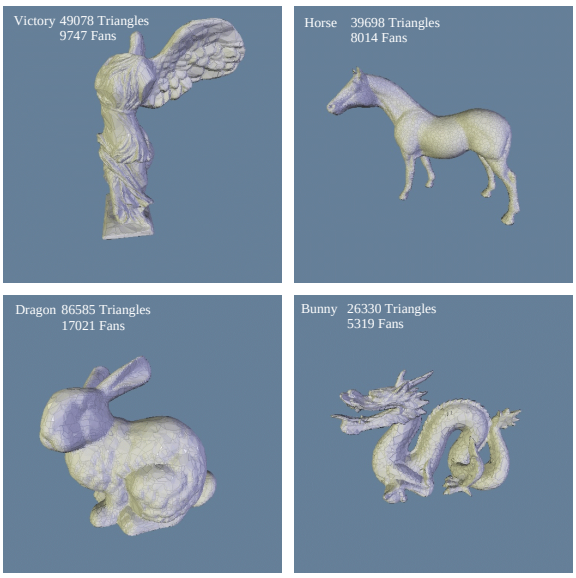

Figure 4: *Some models used to test our ray triangle fan intersection algorithm*

4. Conclusion

We have presented an algorithm for computing the intersection between a ray and a triangle fan efficiently. This algorithm significantly speeds up computations by factoring constant terms and reusing intermediate factors when iterating over the neighbouring triangles of the fan.

Web information

A $C++$ implementation of the algorithm and tests are available online at:

`http://liris.cnrs.fr/~egalin/Triangles`.

References

[1] J. Amanatides and K. Choi. Ray Tracing Triangular Meshes. *Proceedings of the Eighth Western Computer Graphics Symposium*, 43–52, 1997.

[2] D. Badouel. An Efficient Ray-Polygon Intersection. *Graphics Gems*, Academic Press, 390–393, 1990.

[3] E. Haines. Point in Polygon Strategies. *Graphics Gems IV*, Academic Press, 24–46, 1994.

[4] T. Möller and B. Trumbore. Fast, Minimum Storage Ray-Triangle Intersection. *Journal of Graphic Tools*, **2**(1), 21–28, 1997.

[5] Z.-Y. Xu, Z.-S. Tang and L. Tang. An Efficient Rejection Test for Ray Triangle Mesh Intersection. *Journal of Software*, **14**(10), 1787–1795, 2003.