

Fast Processing of Triangle Meshes using Triangle Fans

Eric Galin
LIRIS - CNRS
Université Claude Bernard Lyon 1
eric.galin@liris.cnrs.fr

Samir Akkouche
LIRIS - CNRS
Université Claude Bernard Lyon 1
samir.akkouche@liris.cnrs.fr

Abstract

This paper presents a technique for decomposing a triangulated model into a set of triangle fans. We show that the triangle fan representation improves the performance of several fundamental geometric algorithms operating over triangle meshes. We present two accelerated algorithms, one for computing the nearest intersection between a ray and a fan of triangles, and one for computing the Euclidean distance from a point to a fan of triangles. We demonstrate the effectiveness of the triangle fan representation in several applications, including collision detection, implicit surface modeling and fast ray-tracing and clustered back-face culling.

Keywords: ray triangle-mesh intersection, Euclidean distance computation, triangle fan decomposition.

1. Introduction

Because of their mathematical simplicity and flexibility, triangle meshes are the most widely used representation for modeling three dimensional objects. Triangle strips have been extensively used for accelerated rendering of complex triangulated models. Triangle strips are the more efficient as their size increases. Therefore creating long triangle strips from a set of triangles has been an active research field. Whenever possible, a single strip representation of a triangle mesh enables a vast variety of geometric and topological algorithms to be applied to this model. Since it is NP-complete to test whether a given triangulated model can be represented as a single triangle strip, many heuristics and techniques have been proposed to partition models into as few strips as possible [6, 21, 20, 9].

In contrast, triangle fans have not received as much attention, mostly because it is not possible to create large triangle fans from general triangulated models [16]. Unlike long triangle strips which have a vast coverage of the surface of the model, triangle fans lend themselves for some local and optimized geometric algorithms. In this paper, we demonstrate

Victory 49078 Triangles
9747 Fans

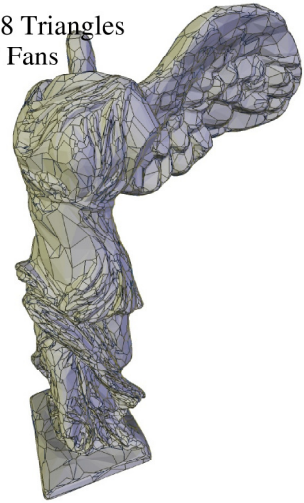


Figure 1. Some accelerations provided by decomposing the Victory model into triangle fans. Ray tracing and distance computations drops by almost 40% and clustered back face culling speeds up rendering time by more than 50%

that decomposing a triangle mesh model into a set of compact triangle fans creates an efficient representation of the mesh. Triangle fans are not only an efficient representation for fast rendering. We show that algorithms such as the intersection between a ray and a triangle fan, or the Euclidean distance computation can be accelerated significantly by reusing some intermediate factors shared by neighbouring triangles during the computations.

Our main contributions are as follows. Section 2 presents a short overview of some related work. In Section 4, we present an optimized algorithm for computing the nearest intersection between a ray a fan of triangles. In Section 4, we address the computation of the Euclidean distance from a point to a triangle fan. Section 5 presents a simple and

efficient method for automatically decomposing a triangulated model into an optimized set of triangle fans. Eventually, Section 6 presents some applications and the corresponding timings.

2. Related work

Ray tracing 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 [4, 11]. An alternative approach that need not compute nor store the plane equation or the normal of the triangle was proposed in [15].

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, many triangles still exist at the leaf nodes of those structures [22]. 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 [22]. 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 [2]. Those methods are particularly adapted for processing rectangular parametric patches discretized into triangles.

Our efficient intersection algorithm between a triangle fan and a ray mostly relates to those last two techniques, and takes advantage of the triangle fan structure to combine several optimizations efficiently.

Euclidean distance computation Computing the Euclidean distance from a point in space to a complex polygonal shape is another fundamental problem in many applications. Several techniques have been proposed. A brute force approach evaluates the minimum distance to all the polygons of the shape. Another technique consists in decomposing an arbitrary shape into a set of convex elements as the distance to a convex shape may be computed efficiently [8]. Voronoï decomposition of space [3] may be used to report, for every point in space, the closest element on the shape. Octrees have also been proposed to index the polygonal shape and permit efficient querying for the closest point [18].

Another approach consists in creating a hierarchy of bounding volumes on the polygonal mesh itself so as to

rapidly discard many triangles in the mesh and perform the exact distance computation on a reduced set of elements. Multiresolution hierarchies of bounding volumes [12, 10] have been successfully applied for fast point to polygonal mesh distance computation.

In this paper, we create a hierarchy of bounding volumes as presented in [12] but over triangle fans instead of triangles, which reduces the depth of the tree. We propose a fast algorithm for computing the Euclidean distance from a point to a triangle fan, which incrementally builds upon the efficient point to triangle algorithm presented in [5].

3. Intersection algorithm

Let $\{\mathbf{p}_0, \dots, \mathbf{p}_n\}$ denote the vertex list of a triangle fan, where \mathbf{p}_0 denotes the center vertex (Figure 2). 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} , $k \in [1, n - 1]$.

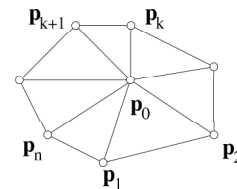


Figure 2. Notations for a triangle fan

Our method incrementally builds upon the algorithm presented in [15]. 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. Although a complete description of our method is presented in [7], we present this algorithm in details to make this paper self contained.

3.1. Ray triangle fan intersection

Let us briefly recall the algorithm presented in [15]. 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 $\mathbf{e}_k = \mathbf{p}_k - \mathbf{p}_0$ and $\mathbf{e}_{k+1} = \mathbf{p}_{k+1} - \mathbf{p}_0$ 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 [15]:

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{|-\mathbf{d} \mathbf{e}_k \mathbf{e}_{k+1}|} \begin{pmatrix} |\mathbf{s} \mathbf{e}_k \mathbf{e}_{k+1}| \\ |-\mathbf{d} \mathbf{s} \mathbf{e}_{k+1}| \\ |-\mathbf{d} \mathbf{e}_k \mathbf{s}| \end{pmatrix}$$

Unlike [15], 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:

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{(\mathbf{e}_k \wedge \mathbf{e}_{k+1}) \cdot \mathbf{d}} \begin{pmatrix} -(\mathbf{e}_k \wedge \mathbf{e}_{k+1}) \cdot \mathbf{s} \\ (\mathbf{s} \wedge \mathbf{d}) \cdot \mathbf{e}_{k+1} \\ -(\mathbf{s} \wedge \mathbf{d}) \cdot \mathbf{e}_k \end{pmatrix}$$

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.

3.2. 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 3).

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.

3.3. Overall algorithm

The overall algorithm may be split into two parts: the first step that evaluates the constant terms, whereas the second step iterates over the triangles of the triangle fan.

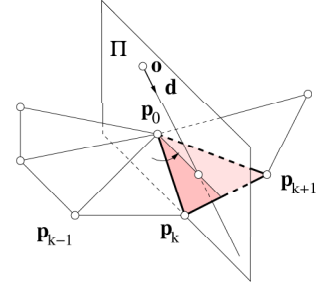


Figure 3. Intersection between the plane containing the ray and the center vertex, and a triangle fan

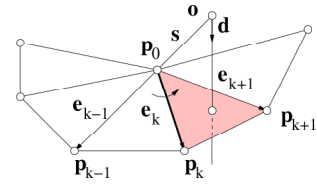


Figure 4. Intersection between a ray and a triangle fan

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 4).

The first 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 $\mathbf{a} = \mathbf{p}_1 - \mathbf{p}_0$ 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 the triangle T_k 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 :

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

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 \mathbf{b} to \mathbf{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 \mathbf{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 [15] which makes the algorithm extremely stable.

4. Euclidean distance computation

We compute the minimum Euclidean distance between a point \mathbf{p} and a triangle fan by iteratively evaluating the minimum distance between \mathbf{p} and the triangles T_k as presented in [5]. As for the ray intersection algorithm, we optimize the overall process by factorizing the computation of constant terms and also reusing computations for two consecutive triangles that share a common edge.

4.1. Point to triangle distance computation

Let us briefly recall the algorithm presented in [5]. Let $\mathbf{s} = \mathbf{p}_0 - \mathbf{p}$, the minimum distance between a point \mathbf{p} and a triangle T_k may be written as the minimization of the quadratic function $f(u, v) = \|\mathbf{p}(u, v) - \mathbf{p}\|^2$ over the triangle domain $\Omega = \{(u, v) : u \in [0, 1], v \in [0, 1], u + v \leq 1\}$:

$$f(u, v) = \mathbf{e}_k^2 u^2 + 2(\mathbf{e}_k \cdot \mathbf{e}_{k+1}) uv + \mathbf{e}_{k+1}^2 v^2 + 2(\mathbf{e}_k \cdot \mathbf{s}) u + 2(\mathbf{e}_{k+1} \cdot \mathbf{s}) v + \mathbf{s}^2$$

Since $f(u, v)$ is a continuously differentiable function, the minimum occurs either at an interior point (u_0, v_0) of Ω where $\nabla f(u_0, v_0) = (0, 0)$ or at a point on the boundary of Ω . A complete description of the algorithm may be found in [5], we simply recall the fundamental steps that are important for optimizing the process for triangle fans:

1. First, the algorithm evaluates the coefficients of the quadratic function $f(u, v)$ and computes the coordinates (u_0, v_0) of the orthogonal projection of \mathbf{p} onto the plane of the triangle.

2. The coordinates (u_0, v_0) are used to classify the point \mathbf{p} against the Voronoï decomposition of space of the triangle (Figure 5). The squared Euclidean distance is computed as the distance to the plane of the triangle, the edges or the vertices according to this classification.

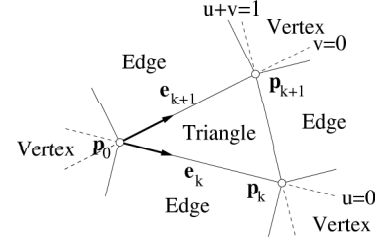


Figure 5. Parameterization of the triangle and the Voronoï decomposition of space

This algorithm enables us to perform the following optimizations:

1. The constant coefficients such as the squared distance $\mathbf{s}^2 = \|\mathbf{p} - \mathbf{p}_0\|^2$ are computed once and for all. Moreover, \mathbf{s}^2 may be used as an upper bound of the minimum Euclidean distance everywhere in the algorithm.
2. We can share the computations of the edge \mathbf{e}_k as well as the evaluation of the dot products \mathbf{e}_k^2 and $\mathbf{e}_k \cdot \mathbf{s}$ between two consecutive triangles T_k in the triangle fan.

4.2. Fast triangle rejection test

When computing the coefficients of the quadratic form, it is possible to avoid computing the exact Euclidean distance by evaluating if the points of the triangle T_k are further away from \mathbf{p} than the center vertex \mathbf{p}_0 of the triangle fan.

If $\mathbf{s} \cdot \mathbf{e}_k > 0$ and $\mathbf{s} \cdot \mathbf{e}_{k+1} > 0$, then the triangle T_k lies in the positive half space separated by the plane orthogonal to \mathbf{s} and passing through \mathbf{p}_0 (Figure 6). In that case, the nearest point of T_k to \mathbf{p} is \mathbf{p}_0 . Therefore, we may skip further computations for this triangle and proceed to the next one. Note that this test is inexpensive since the scalar products $\mathbf{s} \cdot \mathbf{e}_k$ and $\mathbf{s} \cdot \mathbf{e}_{k+1}$ are needed later to compute the coefficients of the quadratic function.

4.3. Overall algorithm

Let \mathbf{a} and \mathbf{b} denote the two vectors that will be used to store the edge vectors of the triangle T_k . Let x_a denote the dot product between the edge \mathbf{a} and \mathbf{s} , and r_a denote the

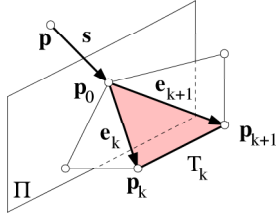


Figure 6. Skipping the Euclidean distance computation by checking the orientation of edge vectors e_k and e_{k+1} with respect to vector s

squared length of the edge \mathbf{a} (similar notations apply to edge \mathbf{b}). The overall algorithm may be split into two parts: a first step that evaluates the constant terms, followed by an iteration over the triangles of the triangle fan. The first step may be outlined as follows:

1. Compute $\mathbf{s} = \mathbf{p} - \mathbf{p}_0$, and compute an upper bound of the squared distance $r = \mathbf{s}^2$.
2. Compute the edge vector $\mathbf{a} = \mathbf{p}_1 - \mathbf{p}_0$, and the dot products $x_a = \mathbf{s} \cdot \mathbf{a}$ and $r_a = \mathbf{s}^2$.

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 $x_b = \mathbf{s} \cdot \mathbf{b}$.
2. If x_a and x_b are both positive, then skip the exact distance computation and go to Step 4.
3. Otherwise, compute the coefficients of the quadric function $f(u, v)$, evaluate the coordinates (u_0, v_0) and compute the exact Euclidean distance r_k between \mathbf{p} and the triangle T_k as presented in [5]. Update r if $r_k < r$.
4. Loop to Step 1, reusing the previously computed values by assigning \mathbf{b} to \mathbf{a} and x_b to x_a and r_b to r_a respectively.

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 \mathbf{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 length of edges \mathbf{e}_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.

5. Creating triangle fans from a mesh model

In this section, we address the decomposition of a triangulated model into a set of triangle fans. Since triangles fans

are the more efficient as their number of vertices is high, our method aims at creating a reduced set of triangle fans with a high valence. Moreover, we wish to avoid the creation of degenerate triangles fans, *i.e.* with only one triangle, whenever possible.

Our technique proceeds as follows. Let \mathcal{M} denote a closed triangle mesh. Starting from a given candidate vertex in this mesh, denoted as \mathbf{c} , we create a first triangle fan centered at \mathbf{c} , remove the corresponding triangles from \mathcal{M} and create a list of open edges, denoted as \mathcal{L} .

We select new candidate vertices in the one ring neighbourhood of the open edge front according to the local topology of the mesh so as to avoid as much as possible the creation of degenerate triangle fans, and favour the creation of large triangle fans. We progressively remove the triangles of the triangles fans from the mesh until all the triangles have been processed.

Note that if the initial mesh is not closed, we skip the creation of the first triangle fan and directly proceed with the list of open edges of \mathcal{L} . The overall algorithm may be outlined as follows:

1. If the triangle mesh has open edges, initialize the list of boundary vertices \mathcal{L} with the corresponding vertices, otherwise select a vertex in the mesh, create a triangle fan and initialize the \mathcal{L} with the resulting open edges.
2. While \mathcal{L} is not empty
 - 2.1 Find the best candidate vertex by analysing the list of boundary vertices \mathcal{L} .
 - 2.2 Create the triangle fan centered at \mathbf{c} , remove the triangles of the triangle fan from the mesh, and update the list \mathcal{L} .

In the next paragraphs, we present a detailed description of the criteria used for selecting candidate vertices as well as the triangle fan decomposition algorithm.

5.1. Notations

Let $v(\mathbf{c})$ denote the connectivity of a vertex \mathbf{c} and $r(\mathbf{c})$ denote the number of open edges in the one ring neighbourhood of \mathbf{c} . Let $t(\mathbf{c})$ denote the number of remaining degenerate triangles fans that would remain by creating a triangle fan centered at \mathbf{c} (Figure 7).

5.2. Selecting candidate vertices

Candidate boundary vertices \mathbf{c} with a low valence are processed first. Therefore, we rely on a priority function, denoted as $f(\mathbf{c})$, which rates the priority of the candidate vertices \mathbf{c} of the mesh.

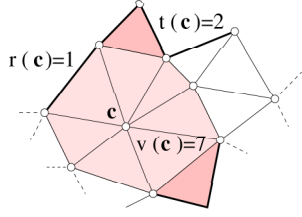


Figure 7. Notations for a candidate vertex c : the triangle fan (dark blue) contains $v(c) = 7$ triangles, is adjacent to $r(c) = 1$ boundary edge and generates $t(c) = 2$ degenerate triangle fans (light blue)

Valence two Let p denote a boundary vertex with valence $v(p) = 2$. To avoid degenerate triangle fans, we create a triangle fan centered either at vertex a or b , where vertices a and b refer to the neighbour vertices of p (Figure 8).

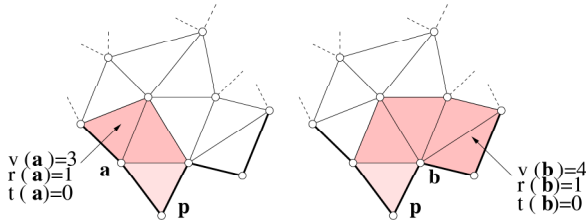


Figure 8. Two different triangle fans that can be created to from either vertex a of b to prevent the creation of a degenerate triangle fan

The choice is made by comparing the priorities $f(a)$ and $f(b)$ and selecting the vertex with the highest priority.

Valence three Boundary vertices with $v(p) = 3$ are slightly more difficult to handle. Let a , b and c denote the three neighbors vertices of p . We select one of those three vertices as a center of the next triangle fan by comparing their priorities $f(a)$, $f(b)$ and $f(c)$ (Figure 9).

Higher valences boundary vertices with a higher valence are processed in the same way as valence three vertices. We select one of the n neighbour vertices p_i by comparing their priorities $f(p_i)$ and choosing the vertex with the highest priority.

5.3. Marking vertices

We classify candidate vertices according to the following linear function:

$$f(c) = \alpha v(c) + \beta r(c) - \gamma t(c)$$

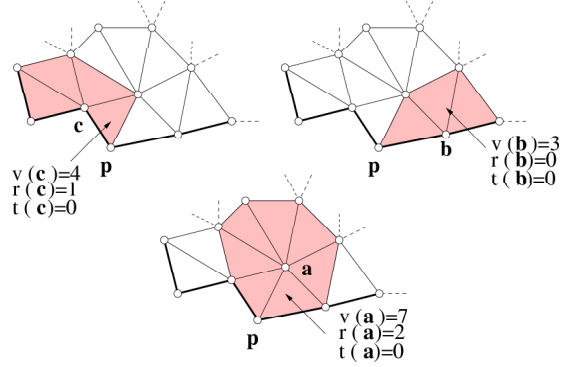


Figure 9. Three different ways of creating a triangle fan that includes a boundary vertex p with valence $v(p) = 3$

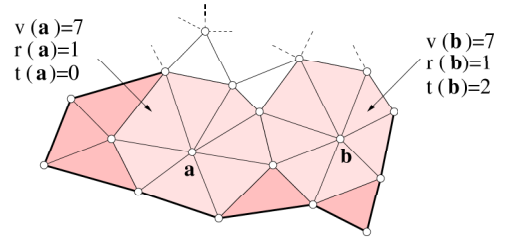


Figure 10. Small triangle fan and degenerate triangles created by processing vertices with a high valence first

The coefficients α and β weight the influence of connectivity and the importance of contact between triangle fans. The coefficient γ is used to penalize the creation of degenerate triangles fans. If $\beta = 0$, the vertices will be classified only against their connectivity *i.e.*, vertices with a high valence will be processed first whereas vertices with a low valence will be processed at the end of the algorithm. This approach tends to create a large number of small and even degenerate triangle fans (Figure 11).

In contrast, if $\alpha = 0$, then vertices that are close to an open edge, *i.e.* close to the border of the progressively decimated triangle mesh, are handled first. This approach privileges the creation of triangle fans that share several boundary edges with previously created triangle fans. Therefore, this method creates smaller triangle fans and tends to avoid the creation of small or degenerate triangle fans (Figure 10).

Eventually, we lower the priority of a candidate vertex the more so as it creates some degenerate triangle fans. Therefore, the coefficient γ penalizes such candidate triangle fans. Our own experiments demonstrated that the fol-

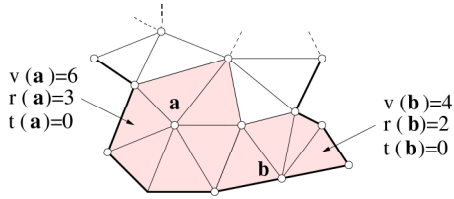


Figure 11. Processing vertices with a high number of open edges first avoids the creation of degenerate triangles

lowing coefficients $\alpha = 2$, $\beta = 7$ and $\gamma = 11$ work well in the general case.

6. Applications

In this section, we demonstrate the effectiveness of the triangle fan decomposition in several applications, including distance field computation, implicit surface modeling, fast ray-tracing and clustered backface culling.

The previous algorithms have been implemented in C++ as member functions of a triangle fan class. 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.

6.1. Distance fields

One application of our fast point to triangle fan algorithm is to efficiently compute the distance field surrounding triangle mesh models, which is interesting for accelerating the initialization of level set models [17].

Instead of creating a hierarchy of bounding volumes directly over the triangles of the mesh as presented in [12], we first decompose the mesh into a set of triangle fans and compute the hierarchy of bounding volumes over this reduced set of surface elements. This approach not only benefits from our accelerated point to triangle fan Euclidean distance algorithm, but also from a simplification of the tree which is built over fewer elements and therefore has a smaller depth.

Table 1 reports timings for computing the 64^3 distance field map of several complex mesh models. The number of triangles, as well as the number of triangle fans is reported in Figure 12. Timings demonstrate that triangle fans speed up the overall process by 25% to 40%.

Algorithm	Triangles	Triangle Fans
Horse	25.47	17.67
Bunny	34.70	25.75
Isis	41.73	25.71
Feline	39.49	30.47
Dragon	51.19	32.33
Aphrodite	59.98	40.45

Table 1. Timings (in seconds) for computing A 64^3 distance field for some complex models

6.2. Hybrid Tree modeling

Another application is the acceleration of distance queries to complex mesh skeletal elements in the Hybrid Tree modeling system [1]. The Hybrid Tree combines implicit surfaces and triangle meshes into a unified framework by selecting the most appropriate representation of a model according to the editing operation. Blending two triangle mesh models requires that they should be converted into an implicit surface representation. Many Euclidean distance computations from points to the mesh models are needed to evaluate the shape of the resulting surface.

By converting the triangle meshes into a hierarchy of triangles fans, we speed up blending operations by 25% to 40%, depending on the complexity of the two argument models.

6.3. Ray tracing

The decomposition of a triangle mesh into triangle fans also enables us to speed up ray tracing applications. Instead of creating a hierarchy of triangles, we first decompose the input models into triangle fans and create a hierarchy of this reduced set of surface element.

We have compared our triangle fan intersection method with the implementation of the reference algorithm proposed in [15].

Table 2 presents timings for ray tracing several models. The bounding box hierarchy for triangles stored 5 triangles per leaf node, whereas the hierarchy of triangle fans stored 1 triangle fan per leaf node so as to make a fair comparison between the two techniques (recall from Table 1 that the average number of triangles per triangle fan is 5). The number of triangles as well as the corresponding number of triangle fans is reported in Figure 12. Timings demonstrate that using triangle fans speeds up intersections by almost 40%.

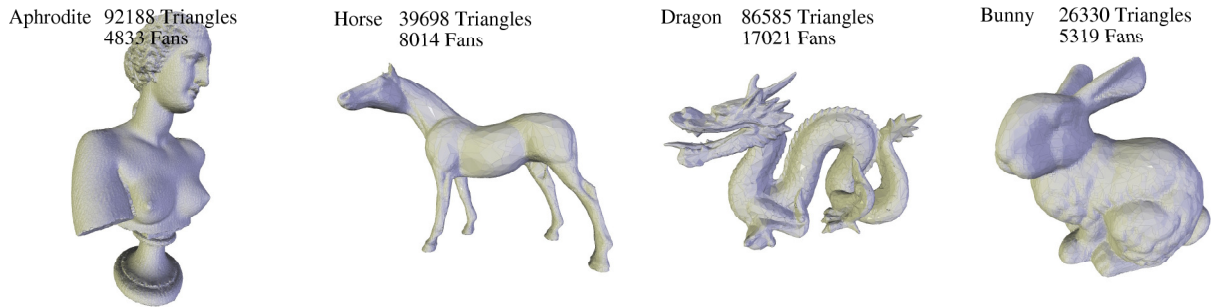


Figure 12. Some models used to test our ray triangle fan intersection algorithm

Algorithm	Triangles	Triangle Fans
Bunny	42.26	24.41
Horse	69.51	40.42
Victory	88.23	53.05
Dragon	125.78	68.91

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

6.4. Clustered backface culling

The triangle fan representation of a mesh also enables us to perform a low level albeit simple and efficient clustered backface culling [19, 13] to speed up the rendering of complex models.

Since we are dealing with triangle fans, the algorithm for computing the cone of normals presented in [19] may be simplified. Let \mathbf{n}_0 denote the vector computed as the average of the normals \mathbf{n}_k of the triangles T_k . The angle of the cone of normals is defined as:

$$\cos(\alpha) = \min_{i \in [1..n-1]} \mathbf{n}_i \cdot \mathbf{n}_0$$

If $\alpha \geq \pi/2$, then the triangle fan cannot be culled. Otherwise, let \mathbf{d} denote the normalized vector from the center of the triangle fan \mathbf{p}_0 to the eye point. The triangle fan is backfacing if:

$$\mathbf{n}_0 \cdot \mathbf{d} \leq \sin(\alpha)$$

We have implemented this rejection test, and observed a 30% speed up in rendering when rejecting backfacing triangle fans, at almost no CPU cost.

7. Conclusion

We have presented an method for decomposing a triangle mesh into a set of triangle fans. We have proposed accelerated algorithms for computing the intersection between

a ray and a triangle fan, and for computing the Euclidean distance from a point to a triangle fan. Those algorithms significantly speed up computations by factoring constant terms and reusing intermediate factors when iterating over the neighbouring triangles of the fan. Those fundamental algorithms can be used in a vast variety of applications to speed up point to mesh distance computations and ray mesh intersection queries. Moreover, triangle fans can be used as a low level primitive for fast clustered backface culling.

In the near future, we plan to investigate multiresolution triangle fan representations. By hierarchically collapsing a whole triangle fan into a single vertex, we hope to be able to create multiresolution representations with continuous levels of detail.

References

- [1] R. Allègre, A. Barbier, E. Galin and S. Akkouche. A Hybrid Shape Representation for Free-Form Modelling. *Proceedings of Shape Modeling International*, 7–18, 2004.
- [2] J. Amanatides and K. Choi. Ray Tracing Triangular Meshes. *Proceedings of the Eighth Western Computer Graphics Symposium*, 43–52, 1997.
- [3] F. Aurenhammer. Voronoï Diagrams: a Survey of a Fundamental Geometric Data Structure. *ACM Computing Surveys*, **23**, 345–405, 1991.
- [4] D. Badouel. An Efficient Ray-Polygon Intersection. *Graphics Gems*, Academic Press, 390–393, 1990.
- [5] P. Schneider and D. Eberly. Geometric Tools for Computer Graphics. *Morgan Kaufmann*, ISBN 1-55860-594-0, 2003.
- [6] F. Evans, S. Skiena and A. Varshney. Optimizing triangle strips for fast rendering. *Proceedings of IEEE Visualization*, 319–326, 1996.
- [7] E. Galin. Fast Ray-Triangle Fan Intersection. Submitted to *Journal of Graphic Tools*.
- [8] E.G. Gilbert, D.W. Johnson and S.S. Keerthi. A Fast Procedure for Computing the Distance Between Complex Objects in Three Dimensional Space. *IEEE Journal of Robotics and Automation*, **4** (2), 1988.
- [9] M. Gopi and D. Eppstein. Single-Strip Triangulation of Manifolds with Arbitrary Topology. *Proceedings of Eurographics*, **23** (3), 371–379, 2004.

- [10] A. Gueziec. MeshSweeper: Dynamic Point-to-Polygonal-Mesh Distance and Applications. *IEEE Transactions on Visualization and Computer Graphics*, **7**(1), 47–61, 2001.
- [11] E. Haines. Point in Polygon Strategies. *Graphics Gems IV*, Academic Press, 24–46, 1994.
- [12] D. Johnson and E. Cohen. A Framework for Efficient Minimum Distance Computation. *Proceedings of Conference on Robotics and Automation*, 3678–3683, 1998.
- [13] S. Kumar, D. Manocha, B. Garrett, and M. Lin. Hierarchical Back-Face Culling. *Proceedings of the 7th Eurographics Workshop on Rendering*, 231–240, 1996.
- [14] S. Mauch. A fast algorithm for computing the closest point and distance transform. *SIAM Journal on Scientific Computing*, 2003.
- [15] T. Möller and B. Trumbore. Fast, Minimum Storage Ray-Triangle Intersection. *Journal of Graphic Tools*, **2**(1), 21–28, 1997.
- [16] T. Möller and E. Haines. Real-Time Rendering. *A.K. Peters*, 2nd edition, ISBN 1568811829.
- [17] K. Museth, D. Breen, R. Whitaker and A. Barr. Level Set Surface Editing Operators. *ACM Transactions on Graphics*, **21**(3), 330–338, 2002.
- [18] H. Samet. The design and Analysis of Spatial Data Structures. *Addison-Wesley*, 1989.
- [19] L. Shirmun, S. Abi-Ezzi. The Cone of Normals Technique for Fast Processing of Curved Patches. *Proceedings of Eurographics*, **12**(3), 261–272, 1993.
- [20] A. Stewart. Tunneling for Triangle Strips in Continuous Level of Detail Meshes. *Proceedings of Graphics Interface*, 91–100, 2001.
- [21] X. Xiang, M. Held and J. Mitchell. Fast and effective stripification of polygonal surface models. *Proceedings of Symposium on Interactive 3D Graphics*, 71–78, 1999.
- [22] 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.