Scan Conversion of Signed Distance Fields

Kenny Erleben and Henrik Dohlmann*

Department of Computer Science, University of Copenhagen, Denmark

Abstract

Fast and robust signed distance field computation is often either a performance bottleneck due to high resolution fields or nearly impossible due to degeneracies in input meshes. Thus, it can be tedious and very time consuming to obtain a signed distance field to be used for collision detection in for instance a physical based animation, motion planing, or geometry processing.

Sign leaking problems in scan conversion methods may result in erroneous signed distance fields for even perfect two-manifold meshes. We present solutions for the sign leaking problems. The major contribution is the robust handling of errors caused by overlapping bounded volumes of neighboring features.

Our method is simple to implement, and has a tradeoff between performance and quality. Further the method is robust in the sense that it handles the sign problems of previous work.

The novelty lies in representing the narrow-band as a decomposition of tetrahedra, a shell mesh. We provide numerous examples and comparisons on different methods for generating the narrow-band shell.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation;

Keywords: Tetrahedra Mesh, GPGPU, Scan Conversion, Distance Field

1 Introduction

Signed distance fields are very attractive in computer graphics and related fields. Often they are used for collision detection in cloth animation [Bridson et al. 2003], multibody dynamics [Guendelman et al. 2003], deformable objects [Fisher and Lin 2001], also mesh generation [Molino et al. 2003; Persson and Strang 2004], motion planning [Hoff, III et al. 1999], and sculpting [Bærentzen 2001].

In all of these applications a signed distance field is represented as a regular sampling of the closest distance to the surface of an object. Usually the convention of using negative values inside the object and positive values outside the object is applied. There does exist adaptive distance fields [Frisken et al. 2000], but we will not consider these in this paper.

The first problem of using distance fields is to actually compute them. There are several issues involved here: The object surface is in most cases modeled as a polygonal model by an animator or obtained by some means of scanning or segmentation. In all cases one often has to deal with a polygonal model having holes, flipped surfaces, overlapping faces, and much worse. This is often termed inconsistent meshes [Bischoff et al. 2005]. Inconsistent meshes are unpleasant, mostly because it is not always meaningful, what is inside or outside.

The computational complexity is also often problematic. A naïve implementation on CPU can take hours, even days, to complete for high-resolution grids (256^3 resolution or greater).

Thus the practicalities in obtaining a signed distance field is often overwhelming, and it is these problems that is the focus of this paper.

The brute force approach to computing distance fields can be described as: For each grid node compute the closest distance to the faces in a polygonal model. Acceleration techniques do exist, such as only querying grid nodes against a bounding volume hierarchy or reversing the iteration to iterate over bounding volumes around faces. These previous methods used a two-pass strategy to resolve the sign issue. In [Aanæs and Bærentzen 2003] angle weighted pseudo-normals was used to determine the correct sign, thus allowing for a single pass only.

A straightforward parallelization of the naïve approach is possible by reversing the order of iteration, that is for each face compute the distance to all grid nodes. This was done in [Hoff, III et al. 1999]. Here the authors mesh the distance function of a vertex, edge, or face, and render it directly to the depth buffer. For volumes this is done in a slice by slice manner, and the distance field is read back from the depth buffer. Any distance metric can be used, but signs are not handled. The simplicity of the method is attractive, although it requires tessellation of elliptical cones and hyperboloid sheets in 3D. Obviously the tessellation causes discretization errors in the distance computation, but the errors can be controlled. This approach is henceforth termed distance meshing.

Scan conversion algorithms using the GPU have become quite popular. Here various external regions is scan converted which bounds the space of points lying closer to a geometric feature, than any other geometric feature. These methods require the construction of bounded volumes that is scan-converted in a slice by slice manner. For each grid node being rendered (voxel), a distance value is being computed. In [Mauch 2003] the characteristic scan conversion (CSC) algorithm was presented. Here three different kinds of Characteristic Polyhedra is used: A prism (for faces), a cone (for vertices), and a wedge (for edges). Conceptually easy to understand it is not very clear how the curved surfaces of the cones and wedges should be tessellated. To avoid aliasing, the polyhedra was enlarged, however the author did not describe the possible errors in the computations, caused by grid nodes getting caught on the wrong side of the surface. This artifact is described in detail in Section 2.

In [Sigg et al. 2003] an optimized GPU version of CSC is presented, together with a more aggressive scan-conversion method, named Prism Scan. Here prisms are constructed for faces only, thus reducing the number of bounded volumes that need to be scan converted. Also a novel fragment program is presented for computing the signed distances of the rasterized grid nodes.

Prism Scan suffers from the same sign problems as CSC, since only the face plane are used to determine the sign, explained in de-

*{kenny,henrikd}@diku.dk

tail in Section 2. These sign errors may seem very innocent since they only occur rarely for small narrow-bands and smooth curved objects. However if narrow-band size is increased and objects with sharp ridges and valleys are scan converted, the sign errors immediately blows up as huge areas of discontinuities where the wrong side of the surface is leaked into the other side.

Both Prism Scan and CSC are limited by a user specified narrow-band size, unlike the distance meshing approach which is capable of computing a full grid.

Both these methods relies on the input surface mesh to be a perfect two-manifold. Working with real-world models this is often not the case and one must often resort to some kind of mesh reconstruction [Nooruddin and Turk 2003]. In this paper we will take the stand point that meshes might be ugly and may cause errors in the signed distance field.

In [Sud et al. 2004] several performance improvements for computing distance fields on graphics hardware are presented. The main two contributions is a culling method based on occlusion queries and a conservative clamping computation based on the spatial coherency of the distance field. Although distance meshing was used in this paper, the method generalizes to scan-conversion algorithms as well, here the conservative clamping can be used to control the size of the narrow-band parameter.

To summarize, methods for computing distance fields on graphics hardware falls into two different approaches: distance meshing or scan conversion of bounded volumes. In [Hsieh and Tai 2005] a hybrid of these two approaches is presented for the 2D case.

Other approaches involve solving the Eikonal equation using for instance a two stage fast marching method [Sethian 1999b]. First one marches from the surface out, then from the surface in. In order to be efficient these methods rely on a good (fast) heap implementation. Besides, one have to seed the fast marching method by computing the distance values on nodes lying just next to the surface. Whereas the scan conversion methods described above compute exact signed distance fields, fast marching methods have a discretization error of order $O(1)$. Although [Sethian 1999a] presents a higher order accurate version.

There are other ways for dealing with the computation of distance fields, such as Danielsson's distance field algorithm [Danielsson 1980]. Here a four pass scan method is used to propagate distance information on a regular 2D grid. The method can be extended to 3D, and has some resemblance with the fast marching method.

We will not discuss CPU based algorithms any further since it is our goal to exploit graphics hardware to obtain a sufficient performance.

In this paper we will present a novel scan conversion approach. Our approach combines the novel fragment program from Prism Scan with the pseudo-normal method. Hereby, we avoid any sign errors of the previous scan conversion algorithms.

Our approach uses a tetrahedral shell [Erleben and Dohlmann 2004; Erleben et al. 2005], meaning only tetrahedra volumes are considered. This allows the usage of a fast tetrahedron slicer to compute cross sections. Slicing tetrahedra are well known from volume visualization and are extremely efficient. Thus it is cheaper to compute cross-sections from scratch, and render these, than doing a 3D scan conversion of more complex prisms, cones, or wedges.

We will present and discuss several methods for computing tetrahedra shells.

- First we will discuss a shell generation method based on simple extrusion along vertex angle weighted pseudo normals, which will be combined with a convex hull computation [Barber et al. 1996], and an enlargement to handle aliasing. The convex hull computation will ensure consistency and removal of degeneracies.

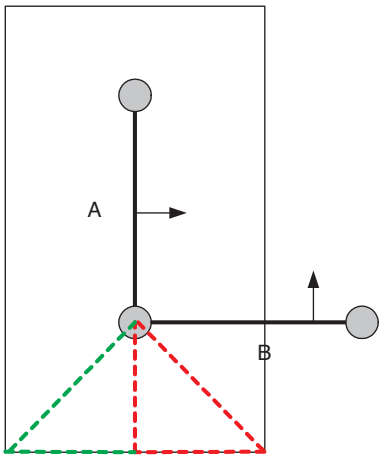


Figure 1: Plane test sign error. The figure shows a cross section of a polygonal model focused on two faces, A and B. Shown together with the bounded region around A, in which the signed distance is calculated. In the dashed red area, the points are closest to A, and the sign will therefore become positive. This is clearly wrong. The points in the dashed red region is located inside the object, so the sign should have been negative. The sign will be correct in the green dashed region. If the planes alone are used to determine the distance, then the distance will be wrong in both the red and green dashed areas.

- Then we will extend the simple method with the extrusion algorithm from the thin tetrahedral shell mesh [Erleben and Dohlmann 2004] combined with a simple face normal extrusion technique to avoid leaking. The benefit over the more simple approach is a more tight fitting shell with less overlapping tetrahedra, thus implying fewer rasterized grid nodes.

- Finally an oriented bounding box (OBB) fitting method is presented, which is simple and easy to implement. In comparison with the other methods, it may have large overlapping regions, even regions expanding far beyond the wanted narrow-band.

The first two shell creation methods, we present, rely on the ability to compute the angle weighted vertex pseudo normals, the last shell creation method makes no assumption on the mesh whatsoever and can be used for unstructured meshes with all kinds of degeneracies.

Note any kind of tetrahedral shell generation method could be used in our scan conversion method, e.g. the adaptive thin tetrahedral shell mesh [Erleben et al. 2005]. This allow for a tradeoff between simplicity of creation and efficiency of scan-conversion.

The correct sign computation in the fragment program relies on the angle weighted pseudo normals of both vertices and edges. If these cannot be computed correctly, then there is no guarantee that the method will compute the proper sign of the distance field.

We have organized our paper as follows: In Section 2 we describe the leaking problems and their sources. Hereafter we present our method in Section 3 and our results in Section 4. Finally we conclude in Section 5.

2 Leaking

In characteristic scan conversion (CSC) and Prism Scan a plane test is used to determine the sign of the distance function, as illustrated in Figure 1. As seen in the figure, this may lead to incorrect computation of the sign. In the case of CSC this becomes even worse, because the characteristic polyhedra are enlarged to avoid aliasing.

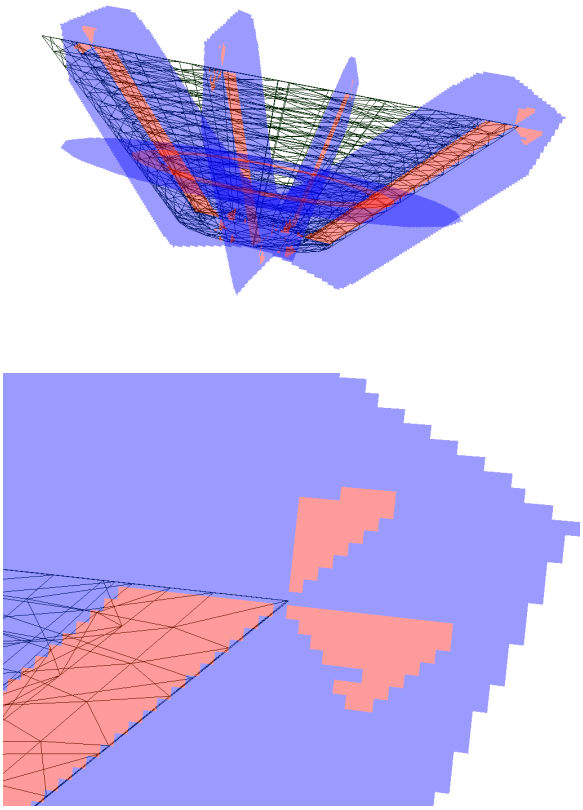


Figure 2: Real life example of leaking due to the plane test problem. Note that the red color have leaked into the blue color. Red is negative and blue is positive. Figure 7 shows the result using our method.

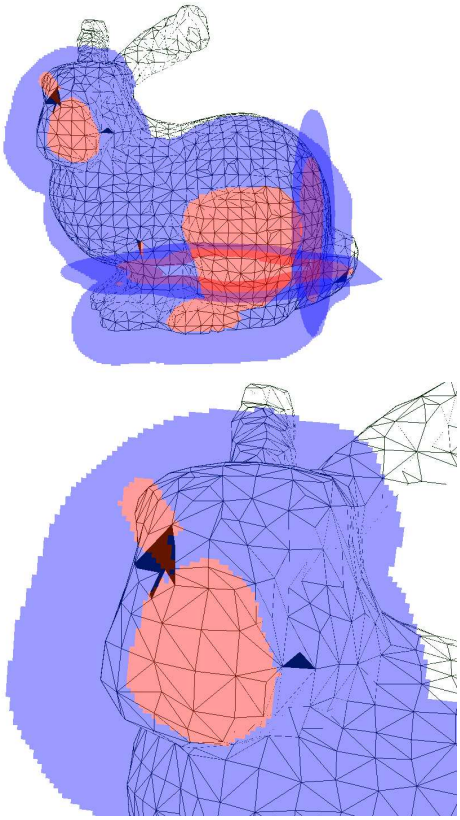


Figure 3: Real life example of leaking by the folding problem. Note that red color is leaking into the blue color. Red is negative and blue is positive. The triangles shaded in black are folded.

Thus for the face case, the distance of grid nodes outside the face-Voronoi region are also computed wrt. the face plane. This means that the dashed lines will produce grid nodes with distances close to zero inside Voronoi regions of neighboring faces. Prism Scan performs a case analysis of grid nodes in enlarged regions and will only suffer from a wrong sign computation. Figure 2 show real life examples of these problems.

Requiring a mesh to be a two-manifold is not a sufficient condition to avoid sign problems. If a surface mesh contains folds, then the orientation of a triangle face can be flipped. Thus, if scan-conversion algorithms are used, then the results depend on the scan order. If the flipped face is scan converted first, then it will result in wrong sign computations. This creates a strange leaking effect. Figure 4 illustrates the mesh-topology of a fold, and Figure 3 shows a real life example.

The final source for leaking problems is due to construction of bounding volumes representing the narrow-band. This is illustrated in Figure 5. Here, the narrow-band shells have different widths on opposite sides of a thin region. This causes the inside region of one side to extend beyond the outside region on the opposite side. In Figure 6 a real life example is shown.

Our method the Tetrahedra (T4) GPU scan method is capable of handling the leaking by plane test and construction problems as shown in Figure 7.

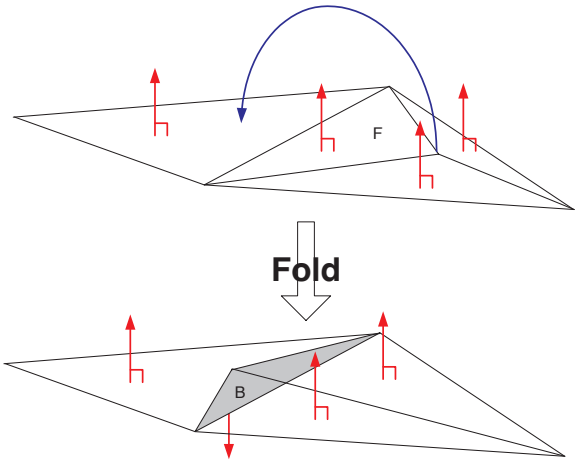


Figure 4: A planar mesh is folded, such that the back side B of a triangle is turning outside instead of the front side F.

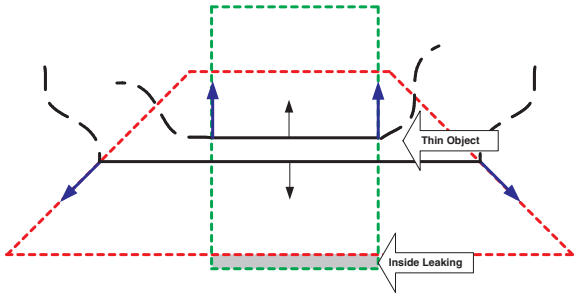


Figure 5: Cross section of a mesh with thin structure, shown together with their bounding regions. The top face has a bounding region shown in green, and the bottom face has a bounding region shown in red. For this configuration, the bounding region of the top face extrudes below the bounding region of the bottom face. This results in the small grey area, wherein the distance becomes negative.

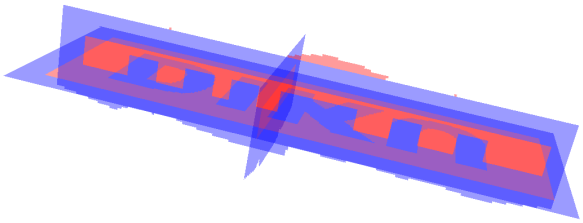


Figure 6: Real life example of leaking by construction problem. Observe the red area on the wrong side of the blue area. Figure 7 shows the result using our method.

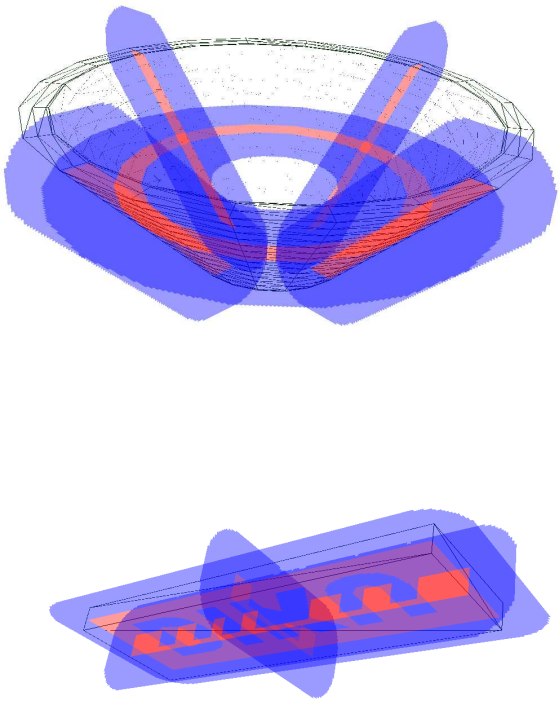


Figure 7: Example showing how our T4 GPU scan method handles the real-life examples from Figure 2 and 6. Notice that no leaking is present.

3 The T4 GPU Scan Method

We call our method the tetrahedron (T4) GPU scan method. In the following we will give an overview of our method.

Given a surface of an object as a collection of triangles, we compute the signed distance within a user specified narrow-band. First we generate tetrahedra in such a way that a single tetrahedron is related to a single triangle face. Note that several tetrahedra can be related to the same triangle face. Tetrahedra are generated while iterating over the triangular faces. Figure 8 illustrates the tetrahedra shell creation in pseudo code. A tetrahedron bounds a region of space containing a subset of grid nodes. For each of these grid nodes, the closest distance to the related triangle face is computed, and the sign is determined using pseudo-normals.

In order to determine the grid nodes lying inside a tetrahedron we move a z-plane in the direction of the positive z-axis. At each z-slice of the regular grid, we halt the z-plane and find the cross sections between tetrahedra and the z-plane. We have adopted a simple sweep-line [de Berg et al. 1997] algorithm to quickly find all tetrahedra that intersects the z-plane. As an alternative one could use the occlusion query method from [Sud et al. 2004].

Having found the cross-sections we render these and use a GPU fragment program to compute the signed distances. Before moving on to the next z-slice of the regular grid, we read back the computed distance values from the frame-buffer, and store it in an internal data structure.

Figure 9 shows the overall steps of the T4 GPU scan method. Note that the shell creation could be done during the scan conversion, which will minimize storage usage. However, in our implementation we have chosen to keep the shell creation as a separate stage for better modularity of the implementation.

Our shell creation methods presented in Section 3.3, 3.4, and 3.5 have linear time complexity, $O(n)$ in the number of triangle faces n, because while iterating once over the triangle faces a fixed number of tetrahedra is generated for each triangle face. The initialization of the sweep-line ie. the z-sorting of the tetrahedra have $O(nlgn)$ time complexity, although the actual scan-conversion can be expected to have linear complexity in the number of generated tetrahedra.

In the following subsections we will describe the details of the individual steps. In Section 3.1 we describe an efficient method for computing the cross-section of a tetrahedron and a z-plane. In Section 3.2 we describe how to compute the signed distance values in a fragment program. In section 3.3 we describe a simple approach to shell creation, which is extended in Section 3.4. Finally in Section 3.5 another shell creation approach is described.

3.1 Computing Cross Section of a Tetrahedron

We use tetrahedra as the underlying primitive that bounds the mesh. To scan convert the distance field, we therefore need an efficient

```
algorithm create-shell()
  L = empty list
  for each face F
    generate tetrahedra of F
    add tetrahedra to L
  next F
  sort L in increasing min. z-values
end algorithm
```

Figure 8: Pseudo code for tetrahedra shell generation.

```
algorithm T4-GPU-scan()
  for z = min z plane to max z plane
    set S = {t in L, and intersects z}
    for tetrahedra t in S
      find cross-section with z
      render cross section
    next t
    read back distance values
  next z
end algorithm
```

Figure 9: Pseudo code for our tetrahedra (T4) GPU scan conversion method.

way to slice a tetrahedron with a plane. This method is inspired by [Bærentzen 2005].

To calculate a cross section of a tetrahedron, the four points of the tetrahedron is sorted by increasing z-value. This allows for a very simple algorithm to find the number of intersections and to create polygons to be processed by the fragment program.

Consider Figure 10. If the z-plane under consideration is below the lowest point in the tetrahedron, then there will be no intersections. Similar, if the z-plane is above the highest point in the tetrahedron, then there will be no intersections.

There are only three topologically distinct ways a z-plane can actually slice the tetrahedron:

A: The z-plane lies below p_1. In this case the plane cuts the lines p_0p_3, p_0p_1, and p_0p_2.

B: The z-plane lies between p_1 and p_2. In this case the plane cuts the lines p_0p_3, p_1p_3, p_1p_2, and p_0p_2.

C: The z-plane lies above p_2. In this case the plane cuts the lines p_0p_3, p_1p_3, and p_2p_3.

In case B, the polygon will always be convex. This can be seen by drawing all the possible configurations of a tetrahedron and consider the order, in which the plane cuts the four lines.

Cases, where the tetrahedron is only sliced in one point or along a line, has no area and should not be considered. The above algorithm ensures this never happens.

The polygons might be either clockwise or counter-clockwise, so a post-process might be necessary to ensure a proper orientation. However, the T4 GPU Scan method does not need this property.

3.2 Computing the Sign using Angle Weighted Pseudo Normals

A novel fragment program was introduced in [Sigg et al. 2003], which calculated the distance to a triangle. Here we give a description of the case analysis used to determine the distance, together

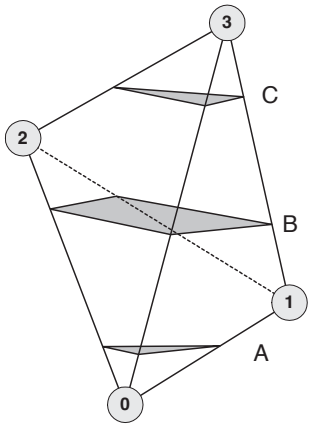


Figure 10: The possible topological different slicings of a tetrahedron.

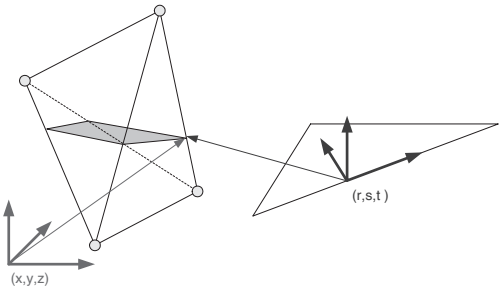


Figure 11: Local triangle frame for a triangle in the mesh and a related cross section.

with our extension that calculates the correct sign using the angle weighted pseudo normals.

Each triangle on the mesh is encased in a bounded volume. Each bounding volume consists of tetrahedra. The triangle is used to create a local triangle frame consisting of vectors $\vec{r}$, $\vec{s}$ and $\vec{t}$, as shown in Figure 11. The coordinates of the slice of the tetrahedron is converted to the local triangle frame and send to the GPU as texture coordinates.

The triangle on the mesh is analyzed to produce three lengths: the height called h, the length from the origin of the triangle frame to vertex $\vec{v}_1$ called a, and the length from the origin to vertex $\vec{v}_0$ called b. See Figure 12. Further, the six angle weighted pseudo normals, $\vec{n}_{v_0}$, $\vec{n}_{v_1}$, and $\vec{n}_{v_2}$ for the vertices, and $\vec{n}_{e_0}$, $\vec{n}_{e_1}$, and $\vec{n}_{e_2}$ for the edges, are calculated and transformed to the local triangle frame using a rotation matrix constructed from unit column vectors, as shown in (1).

$$\vec{n}' = \begin{bmatrix} \frac{\vec{a}}{||\vec{a}||} & \frac{\vec{h}}{||\vec{h}||} & \frac{\vec{n}}{||\vec{n}||} \end{bmatrix}^T \vec{n}, \qquad (1)$$

where $\vec{n}'$ is the transformed normal of $\vec{n}$. These pseudo normals and the three lengths are sent to the GPU as texture coordinates.

On the GPU, the first thing that happens is a reduction of the problem to the half-plane, where $r \geq 0$. That is, if the r-coordinate is negative, we flip the data such that $r = -r$, $a = b$, $\vec{n}_{v_1} = \vec{n}_{v_0}$, and $\vec{n}_{e_1} = \vec{n}_{e_2}$. This reduces the further analysis considerably.

Next, the r'-, and s'-coordinates is constructed from the r- and s-coordinates, and a case analysis is performed according to regions shown in Figure 13. From the case analysis, the distance to the closest feature can be computed, and the corresponding pseudo-normal can be determined. The sign of point $\vec{p}$ can be computed using the pseudo normal of the closest feature, $\vec{n}(\vec{c})$, and some point, $\vec{c}$, on

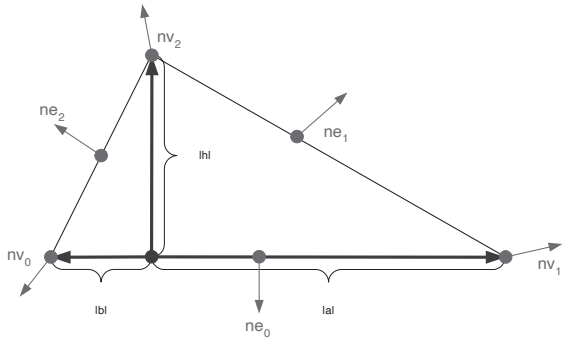


Figure 12: Local triangle with lengths a, b and h, and pseudo normals.

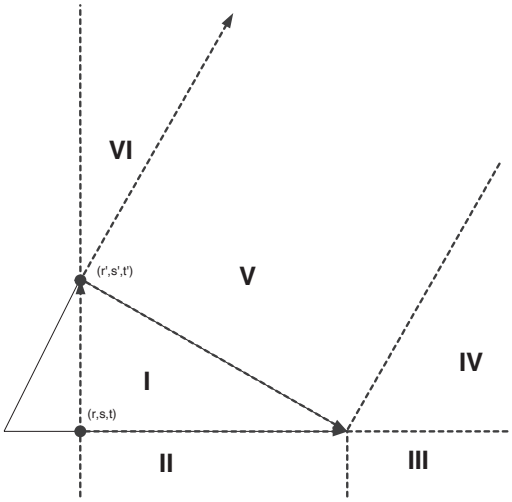


Figure 13: Regions used in the case analysis for the triangle.

the closest feature, as

$$d = \vec{n}(\vec{c}) \cdot (\vec{p} - \vec{c}), \tag{2}$$

as described in [Bærentzen and Aanæs 2005].

3.3 Angle Weighted Vertex Pseudo Normal Shell

For each triangular face we extrude the user-specified narrow-band distance, ε, outward and inward along vertex normals, which generates 6 points. Then we compute the convex hull to get a convex mesh completely covering and enclosing the triangular face, as shown in Figure 14. Hereafter we use the center of the convex mesh as apex for each generated tetrahedron and the triangular faces of the convex hull as bases of the generated tetrahedra. If non-triangular faces are found, then we use a simple ear-clipping algorithm [O'Rourke 1998] to tessellate these into triangular faces.

Note that this will in general not generate a connected tetrahedra mesh. In some cases tetrahedra generated from one face do not connect nicely with tetrahedra generated from neighboring faces.

This is either because, the quadrilateral faces of the generated prism in Figure 14 are not necessarily planar, or because the extrusion may cross over and create a swallow tail as shown in Figure 15.

In both cases using the convex hull of the extruded vertices ensures a conservative coverage; it also guarantees that no gaps will occur between the bounded region of the convex hull and the

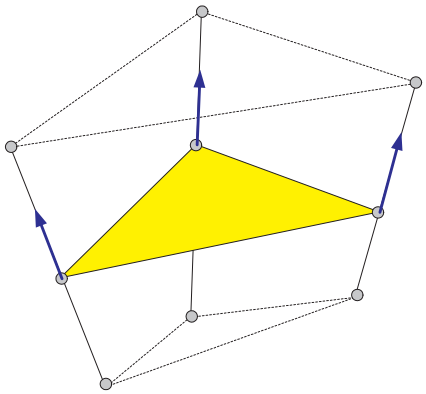


Figure 14: Convex hull of pseudo normal extruded vertices.

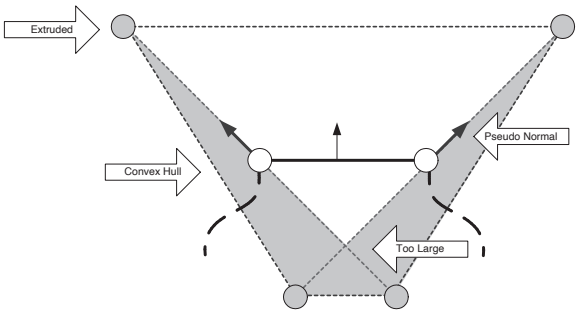


Figure 15: Swallow-tail extrusion making shell region of a single face too large. The too large region is illustrated in grey.

bounded regions of hulls from neighboring faces. The drawback is that grid nodes belonging to overlapping regions will be scan-converted more than once, causing a slight performance degradation.

Aliasing artifacts from rasterization of the sliced cross-sections may cause empty voxels inside the narrow-band. Working with floating point arithmetics may lead to numerical imprecision and truncation errors. Thus, even if the tetrahedra generated from two neighboring faces are perfectly meeting along the shared edge of the faces, then truncation and imprecision can lead to small voids. Furthermore, obscure faces could result in oblong tetrahedra, which would generate slivers when rendered. In conclusion, empty regions are unavoidable unless we do something extra.

The method of choice in the past have been to enlarge the polyhedra being scan-converted. However past methods did not recognize the leaking problems caused by the enlargement. Besides, enlargement have an inherent scale dependency of mesh size versus grid spacing, thus leading to an element of parameter tuning. Conservative rasterization [Hasselgren et al. 2005] ensures no aliasing effects and have no element of parameter tuning. The only drawback is a computational penalty, due to extra geometry processing in the vertex program pipeline. Another problem with the pseudo normal based shell creation method is that it may result in a leaking problem, due to the way the shell mesh is constructed.

3.4 Thin Tetrahedral Shell

The shell creation method in the previous section suffers from the swallow tail problem. This causes the shell around a single face too become too large, thus causing large overlapping regions with neighboring faces. This is illustrated in Figure 15. To remedy

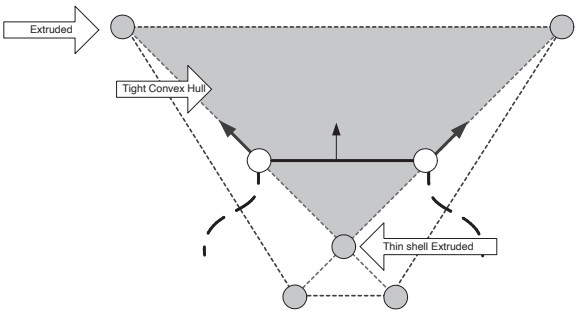


Figure 16: Using thin-shell extrusion lengths result in a more tight fitting convex hull, illustrated by the grey area.

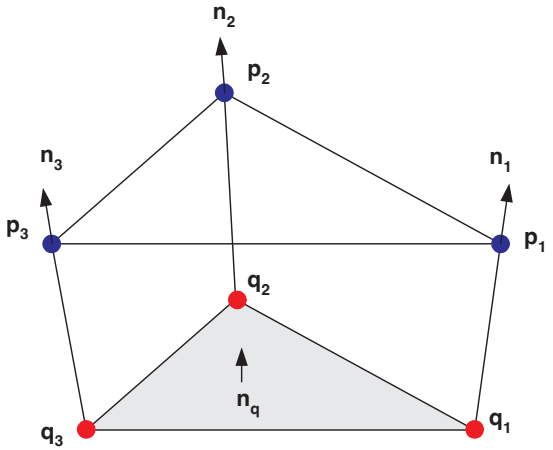


Figure 17: The six corner points defining a prism, and vectors yielding extrusion directions.

this problem we could try to keep the shell region of a single face as tight as possible, e.g. by using the thin-shell extrusion length method in [Erleben and Dohlmann 2004], the details of which will be described later.

Using the thin-shell extrusion length method we compute three inward extruded vertices and three outward extruded vertices, for a total of 6 vertex positions which is passed along to the convex hull algorithm before doing the tetrahedra tessellation. Again convex hull and apex construction method can be applied as in the pseudo normal shell method. Note that if the extrusion length is limited, then the coordinates of some of the 6 extruded vertices will be the same. Figure 16 illustrates the thin-shell idea.

Given a triangle consisting of three vertices $\vec{p}_1$, $\vec{p}_2$, and $\vec{p}_3$, with corresponding three unit direction vectors (we use the angle weighted normals) $\vec{n}_1$, $\vec{n}_2$, and $\vec{n}_3$, indicating the extrusion line direction, Then the inward extruded prism is defined by the six corner points $(\vec{p}_1, \vec{p}_2, \vec{p}_3)$ and $(\vec{q}_1, \vec{q}_2, \vec{q}_3)$ where:

$$\vec{q}_1(\varepsilon) = \vec{p}_1 - \vec{n}_1\varepsilon \tag{3}$$
$$\vec{q}_2(\varepsilon) = \vec{p}_2 - \vec{n}_2\varepsilon \tag{4}$$
$$\vec{q}_3(\varepsilon) = \vec{p}_3 - \vec{n}_3\varepsilon. \tag{5}$$

The extrusion length is given by $\varepsilon > 0$. Notation is illustrated in Figure 17. Similarly, the corner points of the outward extrusion can be found by flipping the extrusion direction vectors.

By requiring ε to be strictly positive, all generated prisms will have non-zero volume. We therefore seek a robust way to determine an upper bound on ε, such that the prism will be valid.

The direction of the normal of the extruded face, $\vec{n}_q$, can be found from $\vec{q}_1$, $\vec{q}_2$, and $\vec{q}_3$, using the cross-product:

$$\vec{n}_q(\varepsilon) = (\vec{q}_2(\varepsilon) - \vec{q}_1(\varepsilon)) \times (\vec{q}_3(\varepsilon) - \vec{q}_1(\varepsilon)). \tag{6}$$

This is a second order polynomial in ε,

$$\vec{n}_q(\varepsilon) = \vec{a}\varepsilon^2 + \vec{b}\varepsilon + \vec{c}, \tag{7}$$

where

$$\vec{a} = (\vec{n}_1 - \vec{n}_2) \times (\vec{n}_1 - \vec{n}_3) \tag{8}$$
$$\vec{b} = (\vec{p}_2 - \vec{p}_1) \times (\vec{n}_1 - \vec{n}_3) + (\vec{n}_1 - \vec{n}_2) \times (\vec{p}_3 - \vec{p}_1) \tag{9}$$
$$\vec{c} = (\vec{p}_2 - \vec{p}_1) \times (\vec{p}_3 - \vec{p}_1). \tag{10}$$

Observe that $\vec{c} \neq \vec{0}$, since its magnitude is equal to twice the area of the triangle being extruded.

To ensure we avoid a swallow-tail, the dot product of the direction of the normal of the extruded face, $\vec{n}_q$, with the vectors, $\vec{n}_1$, $\vec{n}_2$, and $\vec{n}_3$, must always be positive. That is $\vec{n}_1 \cdot \vec{n}_q(\varepsilon) > 0$, $\vec{n}_2 \cdot \vec{n}_q(\varepsilon) > 0$, and $\vec{n}_3 \cdot \vec{n}_q(\varepsilon) > 0$. This yields the following system of constraints,

$$\begin{bmatrix} \vec{n}_1 \cdot \vec{a} & \vec{n}_1 \cdot \vec{b} & \vec{n}_1 \cdot \vec{c} \\ \vec{n}_2 \cdot \vec{a} & \vec{n}_2 \cdot \vec{b} & \vec{n}_2 \cdot \vec{c} \\ \vec{n}_3 \cdot \vec{a} & \vec{n}_3 \cdot \vec{b} & \vec{n}_3 \cdot \vec{c} \end{bmatrix} \begin{bmatrix} \varepsilon^2 \\ \varepsilon \\ 1 \end{bmatrix} > 0. \tag{11}$$

We solve for the smallest positive ε fulfilling the system of constraints. That is, each row represents the coefficient of a second order polynomial in ε, thus for each row we find the two roots of the corresponding polynomial. The three rows yields a total of 6 roots. If no positive root exist, then $\varepsilon = \infty$, otherwise ε is set equal to the smallest positive root.

In fact, the tree dot-product constraints ensure that no neighboring prism will intersect each other, nor will the prism turn its inside out (ie. flipping the extruded face opposite the original face).

The thin-shell extrusion length creation method can lead to a leaking artifact, when creating shells for thin objects. This is illustrated in Figure 5. Computing extrusion lengths along vertex normals only guarantee that we reach the outer boundary of the narrow-band along the vertex normals. Everywhere else the computed narrow-band will have less extent than along the vertex normals.

To minimize the chance of leaking due to differences in pseudo normal angles and trying to make the narrow-band evenly thick, we could extend the current shell creation method with more extruded vertices. We have chosen to make an outward and inward extrusion of the face vertices along the face normal, thus passing a total of 12 extruded points to the convex hull algorithm. We term this heuristic "face-offsetting".

The idea of face-offsetting is illustrated in Figure 18. Face-offsetting do result in lesser tight shell region around the face. Thus increasing overlap with shell region of neighboring faces. This causes more grid nodes to be scan converted.

3.5 OBB Shell

Using the longest edge, $\vec{e}$, and the orthogonal height vector, $\vec{h}$, a tight fitting rectangle can be placed in the face plane of the triangle. Hereafter the rectangle is enlarged by the user-specified narrow-band size, ε. Finally the four vertices of the rectangle are extruded ε-distance outward and inward along the face normal, $\vec{n}$, in order to produce an enclosing OBB around the triangle face. Figure 19 illustrates the steps involved. The OBB can be directly decomposed into 5 tetrahedra. This is very simple to implement and nearly impossible to get wrong. It does ensure a complete coverage of the narrow-band, although large parts may stick outside or overlap. Thus simplicity comes at a performance degradation.

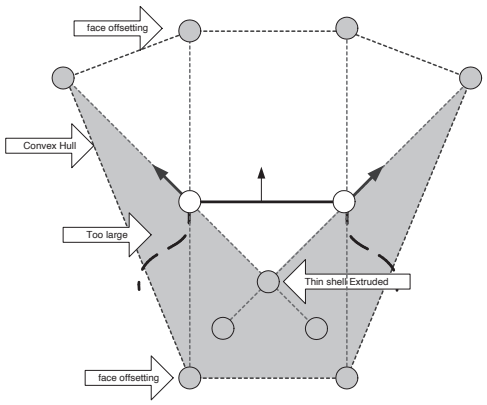


Figure 18: Using face-offsetting to minimize chance of leaking and creating a more evenly thick narrow-band. Grey area show the redundant region.

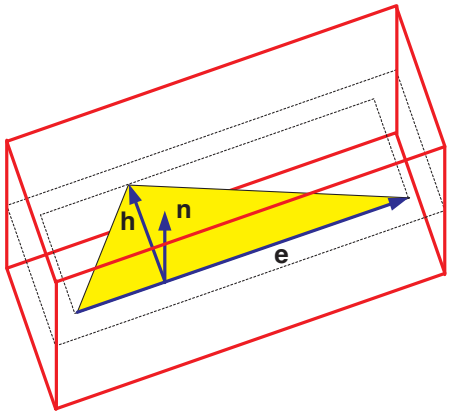


Figure 19: Fitting an OBB around a triangle face.

4 Results

All measurements were performed on a 2.4 GHz P4, with 4GB RAM, running Gentoo Linux. The graphics card installed is a Geforce 6800GT with 256MB RAM. We used a narrow-band size corresponding to 10% of the maximum mesh extend.

We have plotted performance measurements in Figures 20, 21, 22, 23, 24, 25, and 26. As expected all figures shows linear complexity that scales with mesh sizes.

Figures 22, 23, 24, and 25 show the CPU time overhead for the 4 different configurations. A clear bottleneck in our implementation is the lookup operations in our tetrahedra mesh. Next most expensive operation is surface mesh lookup of vertex coordinates and normals.

Figure 26 shows the GPU time overhead. It shows that the fragment program is computationally most expensive and out-weights the frame-buffer read back for large mesh sizes.

In Figure 27 we have shown a few of our signed distance field results using OBB shell creation method. Left column shows the sign computation. Middle column shown the signed distance field. Right column has the mesh super-imposed. Note that no leaking is present, and that the signed distance field appears smooth everywhere.

Figure 28 shows the different narrow-bands obtained using the different shell creation methods. Here it is clearly seen that pseudo-normal extrusion, thin-shell extrusion, and face-offsetting creates a somewhat jagged narrow-band. The OBB creation method clearly yields the best quality, however it is also the one with worst scan conversion performance as seen in Figure 20. This is due to the too large OBBs extending far beyond the narrow-band size and having large overlaps.

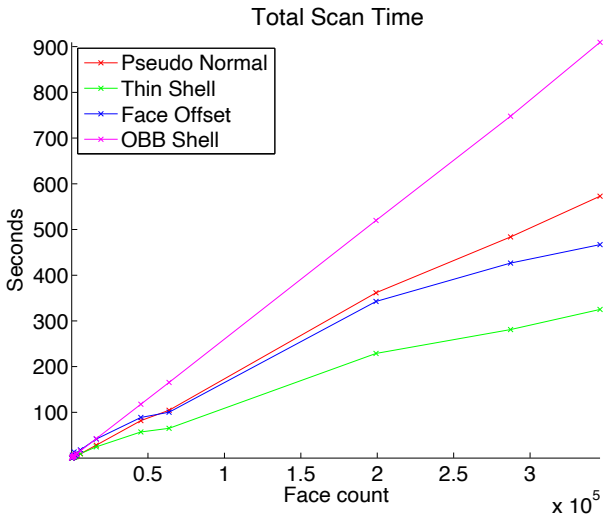


Figure 20: Time used totally.

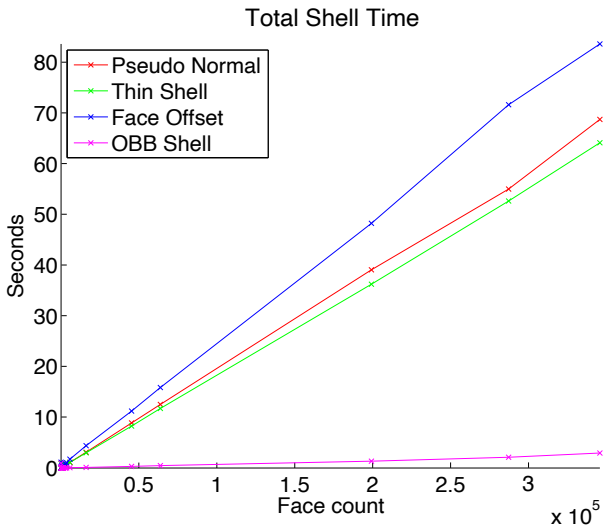


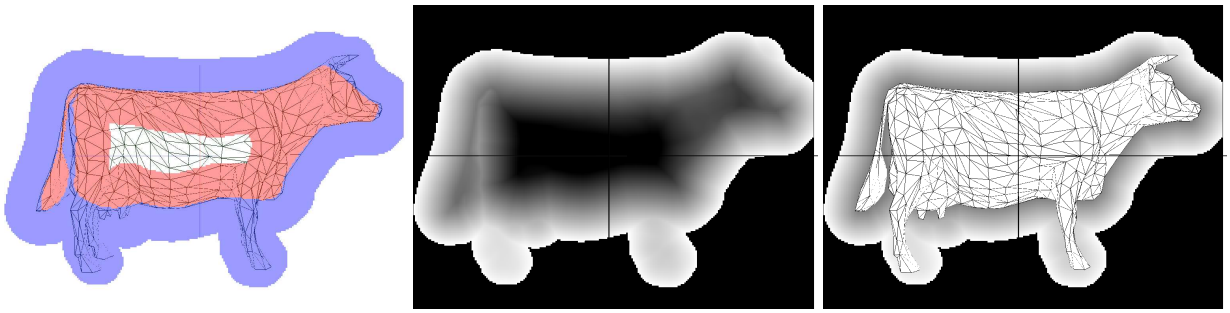
Figure 21: Time used to create the shell.

5 Conclusion

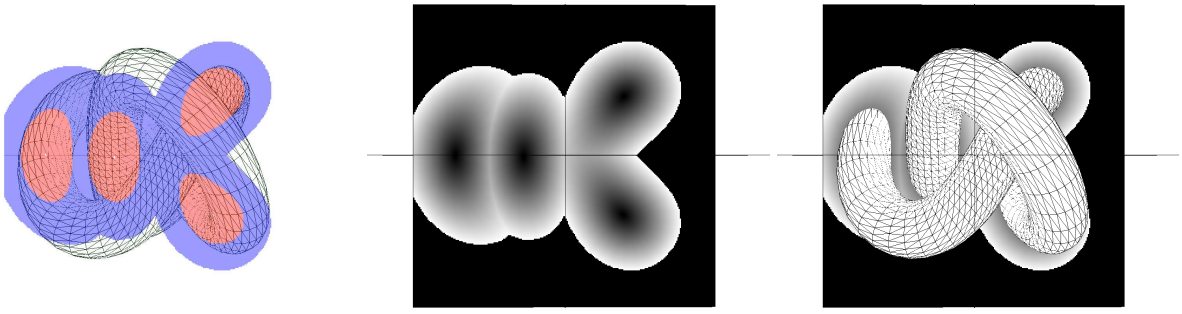
We have presented an approach for scan conversion of signed distance fields.

- It is based on a single type of simple geometry: a tetrahedron.
- It uses pseudo normals to handle correct sign computations.

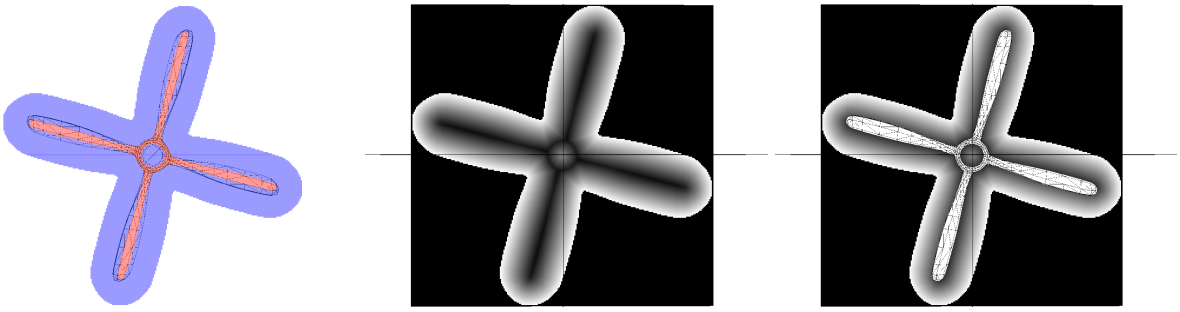
We have presented several shell generation methods and discussed drawbacks and benefits. They are all simple to understand easy to implement. All put together our work yields a robust, simple, and efficient system for computing signed distance fields.



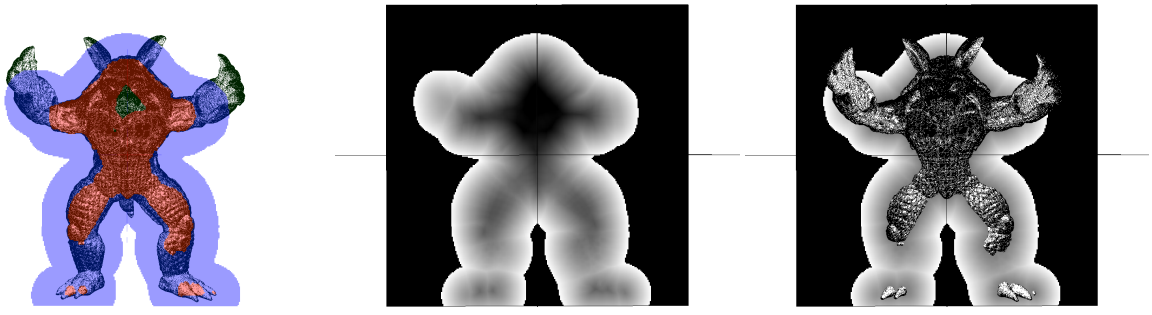
(a) Cow



(b) Knot

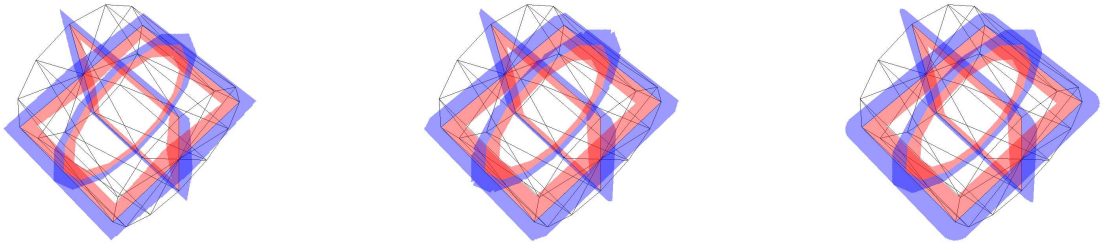


(c) Propeller



(d) armadillo

Figure 27: Sign verification and Signed Distance Field Results. Red is negative and blue is positive.



(a) Pseudo normal (b) Thin shell with face offset (c) OBB shell

Figure 28: Differences in shell creation method illustrated using a cylinder. Red is negative and blue is positive.

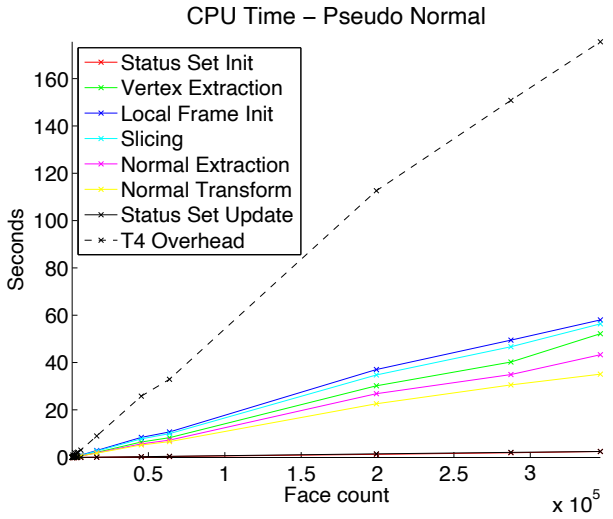


Figure 22: Time used to process the geometry on the CPU for pseudo normal extrusion and anti-aliasing.

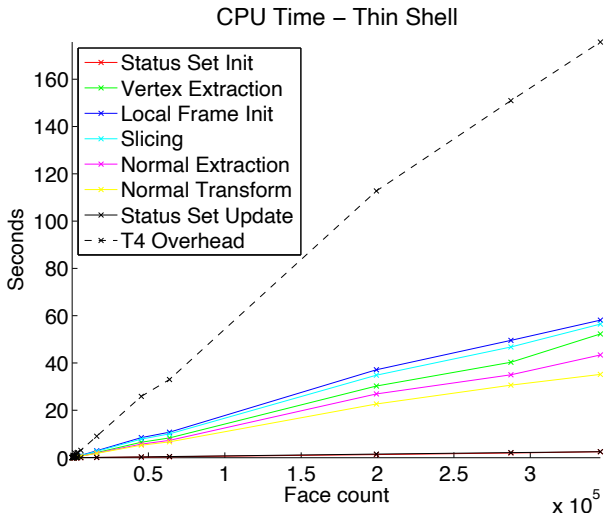


Figure 23: Time used to process the geometry on the CPU for thin shell extrusion length.

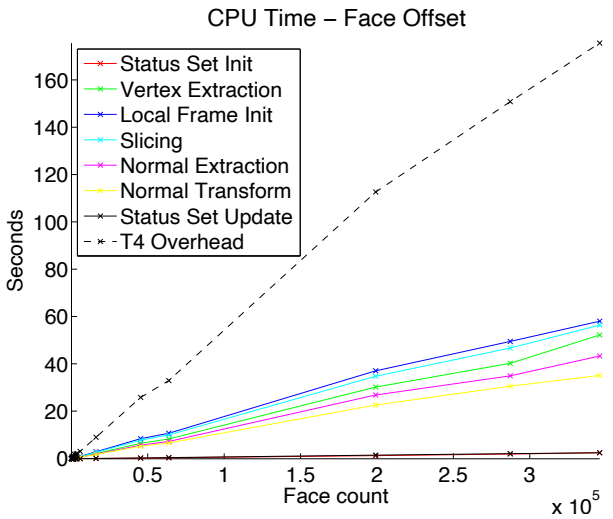


Figure 24: Time used to process the geometry on the CPU for the addition of face offsetting.

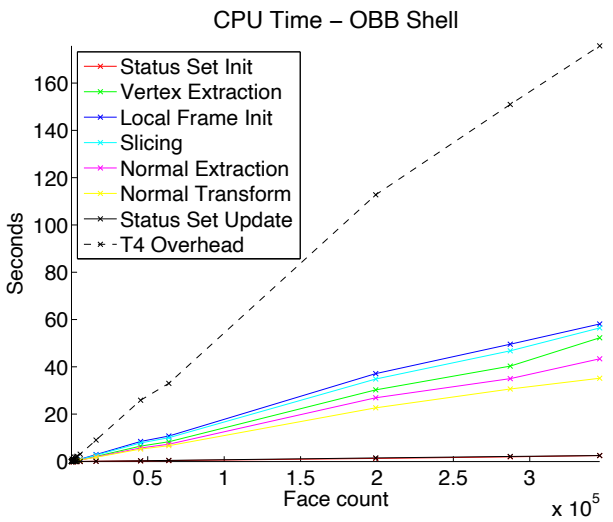


Figure 25: Time used to process the geometry on the CPU for the OBB shell creation method.

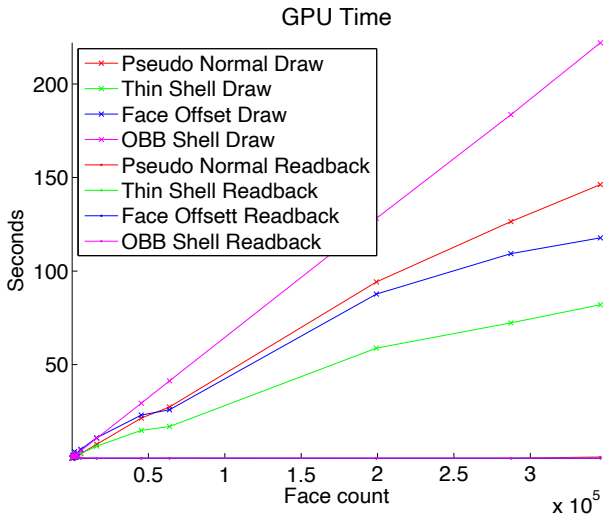


Figure 26: Time used to process the geometry on the GPU.

There is still room for improvements in this work. Faster methods of generating tight fitting tetrahedral shell meshes could boost the performance.

Although we have presented a shell generation method not relying on pseudo normals, the fragment program does need the pseudo normals. This may be an disadvantage for several degenerate meshes, that have redundant vertices creating open boundaries, which meet, but are not topologically connected. Other than that, the method is capable of handling open boundaries, even overlapping faces. Future work could focus on the dependence on pseudo normals, for instance by an algorithm capable of computing meaningful pseudo normals for degenerate meshes.

Besides, we have shown that folding cannot be handled with pseudo-normals. We speculate that a solution to folding problems requires a two-pass method. This is left as future work.

References

AANÆS, H., AND BÆRENTZEN, J. A. 2003. Pseudo–normals for signed distance computation. In *Proceedings of VISION, MODELING, AND VISUALIZATION*.

BÆRENTZEN, J. A., AND AANÆS, H. 2005. Signed distance computation using the angle weighted pseudo-normal. *Transactions on Visualization and Computer Graphics 11*, 3 (June), 243–253.

BÆRENTZEN, J. A. 2001. *Manipulation of Volumetric Solids, with application to sculpting*. PhD thesis, IMM, Technical University of Denmark. BMP 08-0011-311.

BÆRENTZEN, J. A., 2005. Personal communication.

BARBER, C. B., DOBKIN, D. P., AND HUHDANPAA, H. 1996. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw. 22*, 4, 469–483.

BISCHOFF, S., PAVIC, D., AND KOBBELT, L. 2005. Automatic restoration of polygon models. *ACM Trans. Graph. 24*, 4, 1332–1352.

BRIDSON, R., MARINO, S., AND FEDKIW, R. 2003. Simulation of clothing with folds and wrinkles. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, 28–36.

DANIELSSON, P. E. 1980. Euclidean distance mapping. *Computer Graphics and Image Processing 14*, 227–248.

DE BERG, M., VAN KREVELD, M., OVERMARS, M., AND SCHWARZKOPF, O. 1997. *Computational Geometry, Algorithms and Applications*. Springer-Verlag.

ERLEBEN, K., AND DOHLMANN, H. 2004. The thin shell tetrahedral mesh. In *Proceedings of DSAGM*, S. I. Olsen, Ed., 94–102.

ERLEBEN, K., DOHLMANN, H., AND SPORRING, J. 2005. The adaptive thin shell tetrahedral mesh. *Journal of WSCG*, 17–24.

FISHER, S., AND LIN, M. C. 2001. Deformed distance fields for simulation of non-penetrating flexible bodies. In *Proceedings of the Eurographic workshop on Computer animation and simulation*, Springer-Verlag New York, Inc., 99–111.

FRISKEN, S. F., PERRY, R. N., ROCKWOOD, A. P., AND JONES, T. R. 2000. Adaptively sampled distance fields: a general representation of shape for computer graphics. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 249–254.

GUENDELMAN, E., BRIDSON, R., AND FEDKIW, R. 2003. Nonconvex rigid bodies with stacking. *ACM Transaction on Graphics, Proceedings of ACM SIGGRAPH*.

HASSELGREN, J., AKENINE-MÖLLER, T., AND OHLSSON, L. 2005. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. NVIDIA Corporation, ch. Conservative Rasterization on the GPU.

HOFF, III, K. E., KEYSER, J., LIN, M., MANOCHA, D., AND CULVER, T. 1999. Fast computation of generalized voronoi diagrams using graphics hardware. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 277–286.

HSIEH, H.-H., AND TAI, W.-K. 2005. A simple gpu-based approach for 3d voronoi diagram construction and visualization. *Simulation Modelling Practice and Theory 13*, 8, 681–692.

MAUCH, S. 2003. *Efficient Algorithms for Solving Static Hamilton-JAcobi Equations*. PhD thesis, California Institue of Technology.

MOLINO, N., BRIDSON, R., TERAN, J., AND FEDKIW, R. 2003. A crystalline, red green strategy for meshing highly deformable objects with tetrahedra. In *International Meshing Roundtable*, vol. 12, 103–114.

NOORUDDIN, F. S., AND TURK, G. 2003. Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics 9*, 2, 191–205.

O'ROURKE, J. 1998. *Computational Geometry in C*, 2nd ed. Cambridge University Press.

PERSSON, P.-O., AND STRANG, G. 2004. A simple mesh generator in matlab. *SIAM Review 46*, 2 (June), 329–345.

SETHIAN, J. A. 1999. Fast marching methods. *SIAM Rev. 41*, 2, 199–235.

SETHIAN, J. A. 1999. *Level Set Methods and Fast Marching Methods. Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge University Press. Cambridge Monograph on Applied and Computational Mathematics.

SIGG, C., PEIKERT, R., AND GROSS, M. 2003. Signed distance transform using graphics hardware. In *Proceedings of IEEE Visualization*, IEEE Computer Society Press, Seattle, WA, USA, 83–90.

SUD, A., OTADUY, M. A., AND MANOCHA, D. 2004. DiFi: Fast 3d distance field computation using graphics hardware. In *Proc. of Eurographics*, M.-P. Cani and M. Slater, Eds., vol. 23.