Volume Graphics (2005)
E. Gröller, I. Fujishiro (Editors)

Robust Generation of Signed Distance Fields from Triangle Meshes

J. Andreas Bærentzen†

Informatics and Mathematical Modelling, Technical University of Denmark

Abstract

A new method for robust generation of distance fields from triangle meshes is presented. Graphics hardware is used to accelerate a technique for generating layered depth images. From multiple layered depth images, a binary volume and a point representation are extracted. The point information is then used to convert the binary volume into a distance field.

The method is robust and handles holes, spurious triangles and ambiguities. Moreover, the method lends itself to boolean operations between solids.

Since a point cloud as well as a signed distance is generated, it is possible to extract an iso-surface of the distance field and fit it to the point set. Using this method, one may recover sharp edge information. Examples are given where the method for generating distance fields coupled with mesh fitting is used to perform boolean and morphological operations on triangle meshes. [volume graphics] [layered depth image] [distance field]

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling

1. Introduction

An interesting trend in computer graphics is the use of the volume representation and volumetric techniques in a widening range of applications. For instance, volumes are used for applications such as collision detection [GBF03], shape filtering [TW03], and interactive shape modeling [Bær02].

Many of these applications rely on signed distance fields [Gib98, OF02], i.e. volumes where each voxel contains the shortest distance to a watertight surface. The distance is negative if the voxel is inside and positive if it is outside. Signed distance fields are, for instance, commonly used to initialize the level set method [Set99b, OF02] which is a volumetric technique for tracking a deforming interface. The level set method and its variations have many applications in computer graphics and computer vision [TW03, Bær02, Set99b, OP03].

For may reasons, triangle meshes are the prevailing representation of 3D geometry. Hence, it is important to be able to convert triangles meshes to signed distance fields. This problem is more difficult than the problem of computing unsigned distance fields. An unsigned distance field can be computed trivially by finding for each voxel the distance to each triangle and picking the shortest. Most researchers subsequently use some form of scan conversion to compute the sign [JS01]. Recently, it has been shown that if angle-weighted normals are stored at each vertex, edge, and face, one may use the normal stored at the closest feature to compute whether a given point is inside or outside the mesh [BA05].

Unfortunately, these methods require that the surface is watertight, i.e. a closed two-manifold, and in many cases this requirement is not met; triangle meshes with a wide variety of degeneracies are often encountered. Typically the meshes have holes, but a mesh may also fail to be two-manifold for other reasons, e.g. because it contains a self-intersection.

† e-mail: jab@imm.dtu.dk

Here, a novel technique which converts a triangle mesh to a distance field is proposed. The method works on meshes which may have holes or self-intersections, and it is possible to perform boolean operations on the meshes as a part of the conversion. The method works by generating a set of *layered depth images* from a number of directions and by voting amongst these LDIs to decide whether a given voxel is inside or out. This idea was proposed by Nooruddin and Turk [NT03], but it is extended and improved here, and new applications are demonstrated. The major contributions of this paper are

- A technique for generating LDIs based on graphics hardware and LDI processing which allows for boolean operations on solids.

- A robust procedure which produces distance fields from these LDIs.

- The LDI representation subsumes a point representation of the surface. Applications where an iso-surface is fitted to this cloud are shown.

The LDI representation and processing is discussed in Section 3, and the conversion from LDI to voxel grid is described in Section 3.2 with details on how to compute and correct distances in Section 3.3 and Section 3.4, respectively. In Section 4 results are discussed. Applications involving mesh fitting are demonstrated in Section 5. There are some remarks about the implementation in Section 6, and conclusions are drawn in Section 7. Related work is discussed in the next section, but first a word on nomenclature: In the following, the word *voxel* will be used to denote a point in a 3D lattice whereas the word cell is reserved for the cubic element framed by eight voxels and their connecting lines.

2. Related Work

The present method is most directly influenced by the work of Nooruddin and Turk [NT03] who first used multiple "deep" z-buffers for voxelization and also noted that this was essentially the same as layered depth images. Layered depth images were introduced by Shade et al. [SGHS98] in the context of image based rendering.

While the work described in the present paper has many similarities to the work of Nooruddin et al. [NT03], there are also fundamental differences. Precise surface points are recorded and used to generate distance values whereas Nooruddin et al. generate LDIs at a higher resolution, filter, and then downsample to get a scalar volume, but as noted by Gibson [Gib98], a great deal of supersampling has to be used to achieve results comparable to a distance map. In their extension of Nooruddin and Turk's work, Kolb et al. also generate distance fields [KJ01], but the distances are distances along the direction of projection used to generate the LDI. Kolb et al. also use graphics hardware, but they use a simpler method than ours which allows for only a single entry and exit point along a given ray (i.e. there are at most two layers in the LDI). An alternative to the voting technique used here and in [NT03, KJ01] was proposed by Ju [Ju04]. In his paper, Ju observes that the surface consisting of faces dual to the edges intersected in a scan converted grid must be closed. If it is not, there is a boundary cycle, and he introduces an elegant method which detects and patches these cycles. While the voting approach works very well, this method could also have been used.

There is also another volumetric approach for hole closing due to Davis et al. who propose to use volumetric diffusion [DMGL02] to extend known boundary information across hole regions.

In this paper, a technique for GPU implemented depth peeling similar to the one proposed by Everitt [Eve01] is used to extract *all* visual layers of the object, thus creating an LDI representation. Depth peeling is otherwise mostly used for order independent transparency [Eve01]. Another application is the generation of point impostors which is discussed by Bærentzen [Bær].

Several other authors have used graphics hardware as a part of a voxelization process. For instance, Liao et al. [LF02] voxelize CSG hierarchies using an approach where the volume is divided into a set of slices that are processed iteratively on the graphics card. This method generates only binary volumes, and the same is true of the method by Passalis et al. [GPT04], although their work is more similar to the present paper since they also use depth buffers in the voxelization process. Graphics hardware has also been used to accelerate Mauch's *characteristics scan conversion* method [Mau03] for distance field generation [CS03]. A more generic approach with a number of optimizations is due to Sud et al. [SOM04]. Hoff et al. used graphics hardware to generate Voronoi diagrams in 2D and 3D [KEHKL*99]. His work was inspired by a brief mention in the OpenGL Programming Guide [WND98].

One alternative to the approach presented here is to use a technique for fixing the polygonal mesh first and then generate a distance field from the cleaned mesh. That could be done directly using the method in [Ju04], except that its dual contouring does not necessarily produce a two-manifold surface which is required as input to non-robust distance field algorithms. Hence, a different algorithm would need to be used for the final mesh generation. Another alternative would be to generate an implicit function containing the polygonal surface as its zero set and then compute the distance to that zero set. Such a method has recently been proposed by Shen et al. [SOS04]. However, for the precise purpose of generating distance fields, the present method is arguably more direct and simpler than either alternative, and it is certainly faster than the latter method.

Another related area is the construction of distance volumes from range images which can be understood as single layer depth images. For instance, Hoppe et al. [HDD*92],

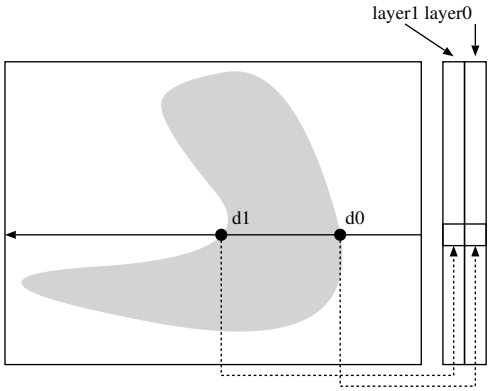

Figure 1: *A layered depth image. The first intersection at depth d0 is recorded in layer 0 and the second intersection is recorded in layer 1 of the LDI.*

Wheeler et al. [WSI98] and Frisken et al. [FRP02] all generate distance fields from range scans. However, the problem is not exactly the same since range scans have an uncertainty associated with the physical scanning process. In addition, surface normal information generally has to be inferred.

Finally, Hoppe has described a technique for mesh optimization [HDD*93]. This method has inspired the one used in Section 5.

3. From Layered Depth Images to Signed Distance Fields

A layered depth image (LDI) is a stack of images where each pixel in each image contains a depth value. A pixel in a depth image generated from a given direction contains the distance from the near plane of the projection to a surface in the observed object. The depth image stored in the first layer contains the depths to the first (the visible) surface. The second layer contains the distances to the layer obscured only by the visible layer, etc. The basic idea is illustrated in Figure 1. One may also construe an LDI as a run-length encoded volume: A first-layer pixel contains the length of a span from the front plane to the object, and the first and second layers together define a span which is interior to the object, provided that the LDI has been generated from a closed manifold object.

Layered Depth Images can be generated by a method for rendering, i.e. ray casting or rasterization. However, it is possible to generate LDIs using the graphics card if a method called depth peeling [Eve01] is used. Two depth buffers are used to perform depth peeling. The first layer is created by rendering and using the first depth buffer to reject fragments behind the first visual layer. The contents of the first depth buffer is then moved to the auxiliary depth buffer. During the second rendering, the first depth buffer works as before, but

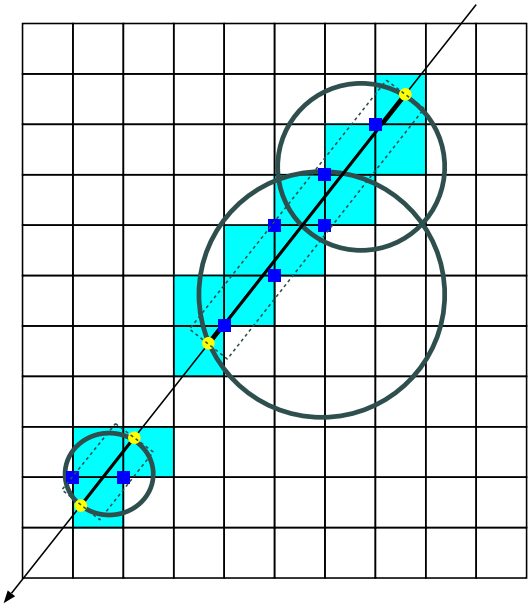

Figure 2: *Illustration of the grid traversal. The long arrow indicates the direction of a ray corresponding to runs extracted from an LDI pixel. After the two interior intersections have been removed, two spans remain; The entry and exit points of these spans are shown as filled circles. All grid points at the corners of cells pierced by the ray are marked if they project onto the LDI pixel and lie between the entry and exit points (the projected area is demarcated by the dotted lines).*

the auxiliary depth buffer is used to reject anything on or in front of the first surface. In this way, the second visual layer is obtained. This process is repeated until nothing is drawn.

Apart from the depth values it is possible to extract any number of attributes for each layer. However, in this context, only depth values and normals need to be extracted, and this can be done in one pass per layer. The triangle face normals are drawn to the colour channel and both colour and depth are read back from the framebuffer after each pass. Like Nooruddin and Turk [NT03], I render from the 13 directions defined by face midpoints, edge midpoints, and vertices. For more details on the implementation of the LDI generation, see Section 6 and [BA05].

3.1. Extracting Spans

The goal of the next step of the algorithm is to extract a set of spans. A span is a line segment between an intersection point where a ray enters and an intersection point where the same ray exits an object. The ray in question is the one passing through the centre of an LDI pixel in the direction of projection, i.e. the ray that would have been used if ray casting had been used for LDI creation.

To extract these spans, consider the depth values of each LDI pixel. Each depth value corresponds to an intersection of the ray and a surface. From the sign of the dot product of the ray direction and the normal associated with each depth value, it is possible to determine whether the ray is entering or exiting. These intersection points are processed in order, using a counter to keep track of whether we are inside or outside.

Initially, the counter is zero, and it is incremented when entering and decremented when exiting. Only intersections which correspond to counter changes from 0 to 1 or 1 to 0 are kept. Thus, if an object consists of two intersecting objects as shown in Figure 2, only intersections on the union are recorded. It is not assumed that the objects processed are manifold. Hence, it is possible that the result is an uneven number of intersections, e.g. due to a ray which exits an object through a hole. As suggested in [NT03] the entire set of intersections is discarded in this case. Since multiple LDIs are used, it will be possible to fill in the missing parts from another LDI. For valid rays, the output is a set of spans where each span contains two intersection points (an entry and an exit point) and their respective surface normals.

The method is clearly very similar to the one used in [NT03], but there are important differences. Since normal information is used, it is possible to handle multiple intersecting objects. The same is not true of the parity count method used in [NT03] where it is implicitly assumed that intersections alternately correspond to entry and exit points. Nooruddin and Turk solve the problem of objects that intersect by only using the first and last value (ray stabbing) but this has obvious limitations. On the other hand, the present method requires that the triangles have normals that are consistently oriented. This requirement seems justified, since if normals cannot be consistently assigned, it is questionable whether it makes sense to generate a signed distance field.

3.2. Voxelization

The next task is to convert the LDI representation to a voxel grid. This is split into two tasks: First, we count for each voxel the number of LDIs that consider it to be interior. Subsequently, distances are computed for each voxel as explained in the next section.

Previous authors [NT03, KJ01] have elected to perform the counting by projecting each voxel into each LDI to pull information from the LDIs. Here the opposite is done. For each span produced by the method above, the volume is traversed along the line segment connecting its end points. This choice is motivated by two objections against the projection-based method: First of all, many voxels are empty, but would still have to be projected into all LDIs to ascertain this fact. Moreover, it would be necessary to introduce a new intermediate LDI representation.

For each span, the cells pierced by the line segment between the entry point and the exit point are enumerated (the enumerated cells are shown shaded in Figure 2). For each voxel which is a corner of an enumerated cell, it is tested whether the voxel projects onto the depth pixel under consideration and whether the voxel lies between the entry and exit point. If the voxel belongs to this rectangular parallelepiped (shown as a dotted rectangle in Figure 2), it is considered seen, and the reference count is increased for that voxel (shown as a filled box).

The enumeration is done using the six-connected 3DDDA algorithm proposed by Cohen-Or and Kaufman [CoK97]. However, due to its use of integer arithmetic, this algorithm is constrained to produce lines that begin and end on grid points. Using floating point arithmetic, on the other hand, would make it very hard to guarantee that the discrete ray terminates in the correct cell. A solution is to use integer arithmetic but to specify the end points of the line on a finer grid. This can be done with few changes to the algorithm and no loss of efficiency in the central loop.

It should be argued that voxels which project onto an LDI pixel are not missed by the above procedure. Observe that only a voxel which is a corner of a cell pierced by a given line can be closer to that line than one voxel unit. Since the lines associated with spans pass through the center of the LDI pixel and the point farthest from the center of the pixel is at a distance of $\sqrt{\frac{1}{2}} < 1$ from the center, one may conclude that all voxels projecting onto the pixels belong to cells pierced by the line connecting the two intersection points.

When all spans have been processed, all voxels are revisited, and those with a reference count strictly greater than half the number of LDIs are marked as interior. Exterior voxels are assigned distance $\sqrt{3}$ and interior voxels are assigned distance $-\sqrt{3}$.

It is pertinent to ask whether the result of this voxel labeling is always what one would expect. While it is difficult to rule out that some contrived input could produce counterintuitive results, it should be noted that since the spans begin and end in intersection points, the interior voxels must lie within the convex hull of the input triangles.

3.3. Distance Estimation

At this point a distance estimate is computed for all voxels that belong to cells containing an intersection point. An estimate of the distance can be computed by projecting the voxel onto the plane defined by the position and normal of the closest intersection point. Let the voxel be at $\mathbf{p}$ and the closest point at $\mathbf{p}_0$ with normal $\mathbf{n}$. The distance is simply computed using

$$d = \mathbf{n} \cdot (\mathbf{p} - \mathbf{p}_0) \ . \tag{1}$$

For each voxel that belongs to a cell containing at least one point, the distance is estimated. This is done by finding the intersection point closest to the voxel and applying (1).

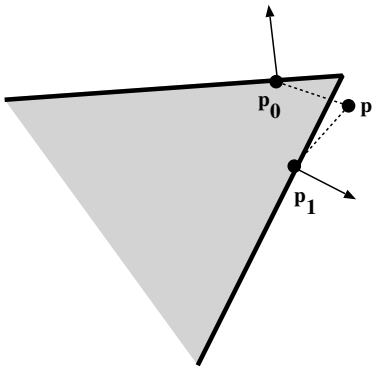

Figure 3: *The voxel* **p** *is erroneously classified as being outside since it is behind the plane of* $\mathbf{p}_0$*, its closest point.*

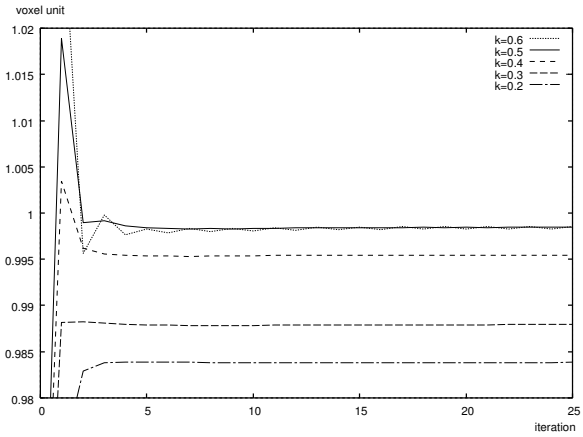

Figure 4: *The optimal damping was found empirically by observing the effect on the average gradient length. At* $k = 0.5$ *the gradient length appears to converge to a value close to 1.*

However, the distance is not necessarily correct; in fact, even the sign may be wrong, if the closest intersection point was not sampled from the closest surface. This problem is illustrated in Figure 3. A simple idea is to use the fact that voxels have already been assigned values corresponding to whether they are interior ($-\sqrt{3}$) or exterior ($\sqrt{3}$). If the sign of the existing voxel value agrees with the sign of the distance estimate, the distance estimate is assigned to the voxel. Unfortunately, this method is too crude: The exterior, interior labelling is not exact either.

However, there is a remedy: If the initial volume, initialized to distance values of $\pm\sqrt{3}$, is smoothed, a more plausible volume is obtained (the zero level iso-surface contains less pronounced staircase artefacts) and it seems more likely that the distance estimates will agree with the smoothed voxel values. Moreover, this can be iterated, producing the loop illustrated in pseudocode below:

```
compute_distance(Volume V, PointSet P)
{
  L: list of (voxel pos, voxel value)
  for each point p in P
  {
    c = cell containing p;
    For each voxel v belonging to c
      if v has not been visited
      {
        d = distance_est(v);
        L.add_to_list(v,d);
      }
  }
  for 15 iterations
  {
    Smooth(V);
    for each element (v,d) of L
    {
      if(sign(d) == sign(V[v]))
        V[v] = d;
    }
  }
```

As shown above, the distance is estimated once for all voxels adjacent to cells carrying points. For each iteration, the estimated distance value is assigned to a voxel if the signs agree. The smoothing is performed only on voxels that have a six-neighbour with a differing sign, and the smoothing is performed simply by taking a weighted average of the voxel and its six neighbours.

In areas where there is an interface between interior and exterior due to the closing of a hole, the surface will automatically be smoothed. In areas where there is information from intersection points, many voxels will be assigned estimated distances in each iteration and thus prevent the surface from drifting.

3.4. Correcting and Extending the Distance Field

The volume produced by the distance estimation above will contain many voxels with correct distances but also areas (near holes in the original surface) where the voxel values are produced only by the smoothing. This means that the volume is smooth, but not all voxel values correspond to distances.

To ameliorate this problem, a final step of reestimating the distance values is performed. If we assume that the smoothed voxel values depend approximately linearly on the distance to the surface, we can simply divide the voxel value d with the length of the gradient to get a distance estimate. More precisely, we can estimate the distance

$$d_{est} = \frac{d}{\|\nabla d\|}$$

where d_{est} is the new estimated distance value based on the current voxel value d. To understand this intuitively, think of a step of the Newton-Raphson root finding method. How-

ever, simply dividing distances by the length of the gradient leads to oscillating distance values. A good solution is to recompute the distance value d using a damping factor k

$$d \leftarrow (1-k)d + k\frac{d}{\|\nabla d\|} \tag{2}$$

where the optimal k is close to 0.5 as witnessed by Figure 4. Ten iterations appears to be sufficient.

The gradient must be computed using upwind differences, i.e. for a given voxel the derivative estimate depends only on neighbours that are either closer to the surface or on the other side. This is a technique commonly used when implementing level set methods. In that context the rationale is that upwinding prevents blending information from colliding fronts [Set99b]. Here it is necessary for a practical reason, namely that the transition region, in general, consists only of voxels which belong to cells that are intersected by the surface.

Note that by design, this correction step ensures that the voxel values are distances in the sense that if we estimate the gradient its lenght will be close to one. However, some diffusion can also be introduced near sharp edges and corners which are not well represented in the distance field. This issue is discussed in the next section.

After hole smoothing and distance correction, the only remaining problem is that distances have only been computed in a narrow range. For many applications, e.g. morphological operations, a broader range of distances is required. Using the fast marching method [Set99b] it is easy to extend a distance field to a set maximum limit. It is worthwhile using the high accuracy version [Set99a]. For a discussion of the implementation of the fast marching method on 3D lattices, the reader is referred to [Bær01].

4. Results

Table 1 contains timings performed on a variety of models. Clearly, the voxelization (Section 3.2) and the distance computation (Section 3.3) are much slower than LDI generation (Section 3) and distance correction (Section 3.4) steps. It is also apparent that voxelization is quite output sensitive; the time it takes is highly dependent on the volume of the produced model. The easiest way to evaluate the result is by tessellating the zero-level isocontour of the distance field. Tessellated iso-contours of all the models in Table 1 are shown in Figure 6. Notice that the mermaid model (Figure 6a1) contains both missing polygons, spurious polygons and ambiguity, i.e. in some places the same surface is represented by two different patches. The model is a scan of a figurine performed with a partially defect scanner. Using the presented method, the spurious parts are removed and the holes are filled when the distance field shown in Figure 6a2 is generated. However, the input is so bad that some artefacts are unavoidable in regions where the algorithm must, essentially, choose between the two surfaces.

error	Torus		Cube	
	uncorr.	corrected	uncorr.	corrected
max	0.328534	0.230868	1.16095	1.96335
min	-0.724344	-0.113973	-1.53404	-1.53404
avg	0.00950544	0.0028777	0.0164621	0.0313922

Table 2: *Error measurements. For the two synthetic models, Torus and Cube, the maximum, minimum and average errors were computed. The error measurements were performed both with and without the correction procedure discussed in Section 3.4.*

The hole in the bunny is also smoothly filled, and although some diffusion is evident, there is little difference between the zero-level iso-surface of the distance field representing the bunny and the original.

Two of the models, a deformed torus and a noisy cube, are known to be closed two-manifold models. This means that it is possible to compute the distance using the method presented in [BA05]. Since the method in [BA05] produces exact distances for closed two-manifolds, it can be used to generate ground truth for an evaluation of the results produced by the method in the present paper. The results are shown in Table 2. Note that the average error is the average of the absolute error. In the case of the torus model which is smooth, the max and min errors are both less than one voxel unit, and the average error is below one hundredth of a voxel unit. The errors are somewhat larger for the cube. The reason is that the cube model contains many sharp edges and corners where the distance is less reliably estimated than near the smooth surface of the torus model.

The purpose of the correction step is to change voxel values so that they reflect the actual distance to the zero-level isosurface. However, as mentioned, this correction also introduces a bit of diffusion which shifts the location of the isosurface. In the case of the cube this effect is noticeable which is why the corrected distance values are actually worse than the uncorrected. In the case of the torus, the procedure is purely beneficial.

5. Mesh Generation and Optimization

For completeness, I briefly describe the methods used for isosurface tessellation and mesh optimizaiton. To tessellate isosurfaces of the distance field, a method which is essentially a non–table driven marching cubes [LC87] is employed. Like in MC, vertices are placed on the zero crossings along the grid line between two cells. However, the connectivity of the mesh is computed by dualizing the mesh produced when cubes are associated with all interior voxels and the faces shared by interior voxels are removed. If two such cubes share an edge, the surfaces are separated to ensure two-manifold connectivity. The desire to construct a mesh

Info				timings (seconds)			
Name	# Faces	Vol. size	# Layers	LDI generation	Voxelization	Dist. computation	Dist. correction
Torus	4096	256	8	1.4	14.4	34.3	0.8
Cube	768	256	14	2.2	40.9	39.4	1.4
Buddha	1087716	400	20	14.8	64.1	125	2.6
Bunny	69451	256	18	2.6	18.4	34.6	0.8
Mermaid	90812	300	21	6	20.9	84.3	1.5

Table 1: *Model information and timings: For each model, the number of polygons, the size of the volume generated, and the maximum number of LDI layers is stated in the left half of the table. In the right half is a list of timings of the various parts of the algorithm.*

with two-manifold connectivity is also the reason why dual contouring is not employed [JLSW02].

Once vertices have been connected, the resulting polygonal mesh is triangulated by iteratively splitting polygons. When several choices are possible, I split along the edge whose midpoint is closest to the isovalue. Finally, all edges that connect two faces of the same cube are collapsed if the mesh is still a valid manifold after the collapse. Consequently, most voxels contribute precisely one vertex, which causes almost all sliver-triangles to disappear.

The conversion of a triangle mesh to the distance field representation will remove small features and soften sharp edges. However, when a mesh is reconstructed from the distance field, it can be refitted to the intersection points generated in the LDI creation process. A method for mesh optimization has been described by Hoppe et al. [HDD*93]. Hoppe et al. use an outer loop consisting of two steps: First, an energy minimization that optimizes a given mesh to the point cloud. Second, the mesh connectivity is changed to further reduce the energy. I use a method that is similar in structure but differs in many particulars. The optimization loop consists of the following steps:

1. The intersection points from LDI generation are projected onto the mesh using ray casting in the point-normal direction.

2. For each vertex, compute its optimal position by finding the point that minimizes the sum of squared distances to all of the plane equations (point, normal pairs) represented by the points that project onto the triangles in its one ring.

3. An energy minimization step is then performed. The vertex is moved towards its optimal position, but a spring term is used to temper the motion.

4. The mesh is improved, by (a) splitting long edges, (b) removing bad triangles (needles and caps [BK01]), (c) edge flipping to minimize max angles (peformed only if the dihedral angle is very low). When performing these mesh changes, it is important to change the geometry as little as possible.

The mesh optimization briefly outlined above is very effective at finding the features represented in the point cloud. However, a vertex close to a feature, say an edge, is only drawn to that feature if points on both sides of the feature have been projected onto the triangles in its one ring. This means that the point cloud should be dense with respect to the mesh, and in the applications discussed below, a resolution of twice the volume resolution has been used in LDI generation to ensure a sufficient number of points.

5.1. Boolean Operations

The intersection counting scheme introduced in Section 3 removes the interior parts of intersecting or self-intersecting solids, in effect computing the union for each pixel in the LDI. The principle is simple: The counter keeps track of how many surfaces have been crossed, i.e. how many solids the ray is simultaneously inside. If there are two objects in the scene, counter changes from 1 to 2 correspond to going from one solid and into their intersection. Thus, intersection can be implemented by issuing intersection points only when the counter goes from 1 to 2 or vice versa. If there are more than two objects, this method will produce the union of pairwise intersections. Clearly, this generalizes to the intersection of multiple solids.

It is also possible to compute differences. A difference is just the intersection of one object with the complement of the object being removed. One initializes the counter to 1 and inverts the sense of the normals of the object being removed. Examples of intersection and difference are shown in Figure 5. Note that the sharp edge introduced by the boolean operation were made fuzzy by the conversion to distance field and reintroduced by the mesh fitting procedure.

5.2. Morphological Operations

It is well known that morphological operations can be performed easily on distance volumes. Simply thresholding a distance field corresponds to a dilation or erosion [Jon01]. By computing first a dilation and then an erosion, one may

perform a morphological closing [Ser82]. Here a slightly different scheme is used. The model shown in Figure 5 (center right) was dilated slightly by thresholding at a small positive value in order to eliminate the canals. The points were then offset by the same amount in the normal direction. The interpolated distance value at the offset position must clearly be identical to the threshold used for dilation if the offset point lies on the dilation. If this is not true for a given point, it should be removed before eroding. Thus, the points inside the canals were removed. Erosion was performed simply by shrinking the mesh. After that, fitting was used to match the original model as closesly as possible. Without the point removal, the fitting process would have failed.

6. Implementation Notes

The method was implemented in C++ using OpenGL, and all tests were carried out on a Linux PC equipped with a 3GHz Intel Pentium IV, 1 GB RAM and a Geforce FX5900 XT graphics card.

The depth buffer precision used is 24 bits. Each component of the normal is stored as an eight bit value in the colour channel of the frame buffer. Entities were converted to floating points during the `glReadPixels` calls made to acquire depth and normal information.

Depth peeling is performed using a very simple class with the interface shown below

```
class DepthPeeler {
  // ...
  public:
  DepthPeeler(int width, int height);
  void disable_depth_test2();
  void enable_depth_test2();
  void read_back_depth();
};
```

Occlusion queries were used to detect when to stop the generation of new LDI layers.

OpenGL samples at pixel centres. This means that the intersection spans would pass through the centre of cells. To ensure that the intersection points are as close as possible to the grid points, all intersection points are shifted by $-(0.5, 0.5, 0.5)$.

7. Conclusions and Future Work

A straightforward method for generating distance fields from triangle meshes has been presented. The method does not produce exact distance fields, but the results indicate that the average error is quite low and the zero-level iso-surfaces produced by the method correspond well with the input. Moreover, the method has been shown to produce reasonable distance fields even from a very degenerate model.

A central idea is to extract and utilize as much geometric information as possible. In fact, even after an iso-surface has been produced from the generated distance fields, the intersection points from the layered depth images can be used to reconstruct sharp edges and corners. It has been shown that with this post-processing, boolean and morphological operations can be performed on triangle meshes producing new triangle meshes as output.

There is one particular avenue of future research which looks especially promising, namely an even greater use of the existing voxel value when updating voxel values. Currently, the sign of the existing voxel value is used to decide whether to accept or reject the estimated distance value. One might estimate several likely distances and use the stored distance value and gradient to choose between these estimates.

The simple smoothing used to propagate distance information could also be replaced by a more judicious approach, possibly using the Eikonal equation [Set99a].

References

[BA05] BÆRENTZEN J. A., AANÆS H.: Signed distance computation using the angle weighted pseudo-normal. *Transactions on Visualization and Computer Graphics 11*, 3 (May/June 2005), 243–253.

[BK01] BOTSCH M., KOBBELT L.: A robust procedure to eliminate degenerate faces from triangle meshes. In *Vision, Modeling, Visualization 2001 Proceedings* (2001).

[Bær] BÆRENTZEN J. A.: Hardware-accelerated point generation and rendering of point-based impostors. *Journal of Graphics Tools*. To appear.

[Bær01] BÆRENTZEN J. A.: *ON THE IMPLEMENTATION OF FAST MARCHING METHODS FOR 3D LATTICES*. Tech. Rep. IMM-REP-2001-13, DTU.IMM, 2001. http://www.imm.dtu.dk/~jab/publications.html.

[Bær02] BÆRENTZEN J. A.: Interactive modelling of shapes using the level-set method. *International Journal on Shape Modeling 8*, 2 (December 2002), 79–97.

[CoK97] COHEN-OR D., KAUFMAN A. E.: 3D line voxelization and connectivity control. *IEEE Computer Graphics and Applications 17*, 6 (November/December 1997).

[CS03] C. SIGG R. PEIKERT M. G.: Signed distance transform using graphics hardware. In *Proceedings of IEEE Visualization '03* (2003), IEEE Computer Society Press, pp. 83–90.

[DMGL02] DAVIS J., MARSCHNER S., GARR M., LEVOY M.: Filling holes in complex surfaces using volumetric diffusion. In *Proceedings of the First International Symposium on 3D Data Processing, Visualization, and Transmission* (june 2002).

[Eve01] EVERITT C.: Interactive order-independent transparency. Technical report, NVIDIA Corporation, May 2001. Available at http://www.nvidia.com/., 2001.

[FRP02] FRISKEN S., RONALD PERRY M.: Efficient estimation of 3d euclidean distance fields from 2d range images. In *Proceedings of IEEE Symposium on Volume Visualization 2002* (2002).

[GBF03] GUENDELMAN E., BRIDSON R., FEDKIW R. P.: Nonconvex rigid bodies with stacking. *ACM Transactions on Graphics 22*, 3 (2003), 871–878.

[Gib98] GIBSON S. F.: Using distance maps for accurate surface representation in sampled volumes. In *Proceedings of IEEE Symposium on Volume Visualization* (October 1998), Spencer S., (Ed.).

[GPT04] G. PASSALIS I. K., THEOHARIS T.: Efficient hardware voxelization. In *Proceedings of Computer Graphics International* (June 2004), pp. 374–377.

[HDD*92] HOPPE H., DEROSE T., DUCHAMP T., MCDONALD J., STUETZLE W.: Surface reconstruction from unorganized points. *Computer Graphics 26*, 2 (1992), 71–78.

[HDD*93] HOPPE H., DEROSE T., DUCHAMP T., MCDONALD J., STUETZLE W.: Mesh optimization. *Proc ACM SIGGRAPH 93 Conf Comput Graphics and Proceedings of the ACM SIGGRAPH '93 Conference on Computer Graphics* (1993), 19–25.

[JLSW02] JU T., LOSASSO F., SCHAEFER S., WARREN J.: Dual contouring of hermite data. *ACM Transactions on Graphics 21*, 3 (2002), 339–346.

[Jon01] JONES M. W.: Facial reconstruction using volumetric data. In *Proceedings of the Vision Modeling and Visualization Conference 2001 (VMV-01)* (Berlin, Nov. 21–23 2001), Ertl T., Girod B., Niemann G. G. H.,, Seidel H.-P., (Eds.), Aka GmbH, pp. 135–150.

[JS01] JONES M. W., SATHERLEY R.: Shape representation using space filled sub-voxel distance fields. In *Proceedings of International Conference on Shape Modelling and Applications* (May 2001).

[Ju04] JU T.: Robust repair of polygonal models. *ACM Trans. Graph. 23*, 3 (2004), 888–895.

[KEHKL*99] KENNETH E. HOFF I., KEYSER J., LIN M., MANOCHA D., CULVER T.: Fast computation of generalized Voronoi diagrams using graphics hardware. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (1999), ACM Press/Addison-Wesley Publishing Co., pp. 277–286.

[KJ01] KOLB A., JOHN L.: Volumetric model repair for virtual reality applications. *Proceedings of Eurographics 2001* (2001). Short presentations.

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics 21*, 3 (July 1987), 163–169.

[LF02] LIAO D., FANG S.: Fast volumetric csg modeling using standard graphics system. In *Symposium on Solid Modeling and Applications* (2002), pp. 204–211.

[Mau03] MAUCH S.: *Efficient Algorithms for Solving Static Hamilton-Jacobi Equations*. PhD thesis, Caltech, 2003.

[NT03] NOORUDDIN F., TURK G.: Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics 9*, 2 (April–June 2003), 191–205.

[OF02] OSHER S. J., FEDKIW R. P.: *Level Set Methods and Dynamic Implicit Surfaces*, 1st ed. Springer Verlag, November 2002.

[OP03] OSHER S., PARAGIOS N. (Eds.): *Geometric Level Set Methods in Imaging, Vision, and Graphics*. Springer, 2003.

[Ser82] SERRA J.: *Image Analysis and Mathematical Morphology*, vol. 1. Academic Press, 1982.

[Set99a] SETHIAN J. A.: Fast marching methods. *SIAM Review 41*, 2 (1999), 199–235.

[Set99b] SETHIAN J. A.: *Level Set Methods and Fast Marching Methods*, second ed. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 1999.

[SGHS98] SHADE J. W., GORTLER S. J., HE L.-W., SZELISKI R.: Layered depth images. *Computer Graphics 32*, Annual Conference Series (1998), 231–242.

[SOM04] SUD A., OTADUY M. A., MANOCHA D.: Difi: Fast 3d distance field computation using graphics hardware. *Comput. Graph. Forum 23*, 3 (2004), 557–566.

[SOS04] SHEN C., O'BRIEN J. F., SHEWCHUK J. R.: Interpolating and approximating implicit surfaces from polygon soup. In *Proceedings of ACM SIGGRAPH 2004* (Aug. 2004), ACM Press.

[TW03] TASDIZEN T., WHITAKER R.: Anisotropic diffusion of surface normals for feature preserving surface reconstruction. In *Proceedings of Fourth International Conferenceon 3-D Imaging and Modeling* (October 2003), pp. 353–360.

[WND98] WOO M., NEIDER J., DAVIS T.: *OpenGL Programming Guide*, second ed. Addison Wesley, 1998.

[WSI98] WHEELER M. D., SATO Y., IKEUCHI K.: Consensus surfaces for modeling 3d objects from multiple range images. In *ICCV '98: Proceedings of the Sixth International Conference on Computer Vision* (1998), IEEE Computer Society, p. 917.