Vision, Modeling, and Visualization (2011)
Peter Eisert, Konrad Polthier, and Joachim Hornegger (Eds.)

Fast and Efficient 3D Chamfer Distance Transform for Polygonal Meshes

Michael Martinek [1], Roberto Grosso [1] and Günther Greiner[1]

[1]University of Erlangen-Nuremberg, Germany

Abstract

We present an efficient GPU-based method to perform 3D chamfer distance transform (CDT) in a wavefront scheme. In this context, we also introduce a binary voxelization algorithm which provides the initial boundary condition for the CDT. The voxelization method is capable of both, surface and solid voxelization, allowing for the computation of unsigned distance fields for arbitrary polygonal meshes and signed distances for models with orientable surfaces. Our method is trimmed on speed rather than accuracy. It works with simple chamfer metrics such as the Manhattan and chessboard distance and requires only a single rendering pass per distance layer. Due to the wavefront scheme, a propagation can be stopped if a required number of distance layers is reached. However, even a complete distance field can be computed in the magnitude of 10^{-3} seconds including the preprocessing voxelization step. This allows for a use in real-time applications such as path planning or proximity computations. We demonstrate the application of our method for the latter.

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

1. Introduction

Considering a volumetric domain around one or more 3D objects, a distance field contains the minimum distance to all geometric primitives for each point in the domain. 3D distance fields play an important role in many areas of computer graphics, computer vision and related areas. Typical applications include surface representation [FPRJ00, KBSS01, BC08], collision or proximity computations [GBF03, SGGM06, MRS08], medial axis estimation and skeletonization [Mon68, ZT99, BSB*00, FLM03], CSG operations [BC01, NDS04] or the computation of Voronoi diagrams [HKL*99, SKW09].

Depending on the type of application, different distance metrics can be used. The most meaningful metric is the Euclidean norm, however, it is also by far the most expensive one in terms of computation. A common alternative is to approximate the Euclidean distance by metrics, which can be propagated locally, i.e. where the distance information at a voxel can be computed from the values of its neighbors. This principle is commonly known as chamfer distance transform.

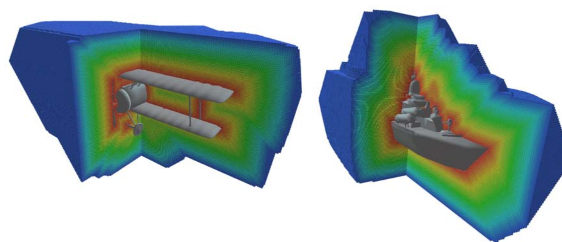

Figure 1: *These examples show 30 discrete depth layers using the chessboard metric around complex objects in a 256^3 voxel grid. The entire process, including voxelization, took just 2.3 ms on an Nvidia GTX 280.*

The propagation can be either done in a sweeping or a wavefront scheme. In the first one, the propagation moves from one corner of the domain to the opposite corner in a voxel-by-voxel manner, typically requiring a forward and a backward pass to complete the distance transform. Another possibility is a wavefront-scheme, where the distance propagates from the surfaces in the direction of increasing dis-

© The Eurographics Association 2011.

DOI: 10.2312/PE/VMV/VMV11/121-128

delivered by EUROGRAPHICS DIGITAL LIBRARY www.eg.org diglib.eg.org

tances. Distance fields can be further classified to be signed or unsigned. If the objects in the scene have orientable surfaces, distance values can be given a sign, depending on whether the measured point is inside or outside an object.

In this paper, we introduce an efficient framework, which propagates either a signed or an unsigned distance field in a wavefront scheme. Our method either works directly on voxelized data, or on arbitrary polygonal meshes since it also includes a binary GPU-voxelization algorithm to produce the initial zero-set of the distance field. After this process, the voxels are stored in 2D textures on the GPU and are directly used in this form for the propagation of the distance field. We combine all voxels in the scene having the same distance value in a common distance layer which is rendered and stored in a separate 2D texture. For each new layer, we employ a single rendering pass to obtain the respective neighborhood information by means of logical bit-operations in a fragment shader. Since we process one discrete distance layer at a time, we use simple chamfer-metrics such as the Manhattan or chessboard metric, which only consist of integer values. These metrics are only rough approximations of true Euclidean distances but can be propagated in a highly efficient way. Our method is thus designed for applications which require high speed rather than accurate Euclidean distances. We demonstrate the applicability of our distance field for real-time 3D proximity computations in a subsequent section.

The contributions of our work are the following:

i) A GPU voxelization method for arbitrary 3D models which is capable of performing real-time solid and boundary voxelization.
ii) A method to perform a chamfer distance transform with simple metrics such as the chessboard and Manhattan metric in a wavefront scheme.

Furthermore, we present a framework which combines the entire process of distance transformation for arbitrary polygonal meshes, including the pre-processing voxelization in a consistent data structure on the GPU. In fact, one could also use other voxelization methods in order to produce the initial boundary condition for our distance transform, but this would hamper the performance of the entire process. In our framework, the interfaces between the two steps are consistently adapted which means that no data conversions or expensive GPU-CPU transfers are necessary between the computation of the zero-set of the distance field and the propagation of the distance transform.

2. Related Work

Since our framework also includes voxelization, we briefly describe some approaches which are closely related to our proposed voxelization method.

2.1. GPU-Voxelization

One of the first methods using GPU-rasterization was introduced by Fang and Chen [FC00]. In their approach, cutting slices are moved in z-direction with a constant step size in a front-to-back order and the geometry is rasterized for each new slice. The step size is equal to the z-resolution of the voxel grid so the resulting frame buffer data becomes one slice of the voxelization. The method requires as many rendering passes as there are voxels along the z-dimension for a complete voxelization.

Dong et al. [DCB*04] were the first to render voxels directly to a 2D texture using the bits of the RGBA-components of each texel to store the voxels. They also use three orthogonal perspectives and only process triangles which project the maximum area along the current perspective. However, since they attempt to store the entire voxel grid in a single texture, the x- and y-dimensions of the texture do not naturally correspond to the dimensions of the voxel grid, making it cumbersome to reconstruct the true 3D structure of the voxel domain from the 2D texture for further processing.

Eisemann and Decoret [ED08] exploited further improvements of graphics hardware to perform solid voxelization in a single rendering pass. They use 32 bits per color channel, allowing them to store 128 voxels in a single texel. In addition, they use multiple render targets to process up to 1024^3 voxels in a single pass and perform a solid voxelization by means of simple XOR blending operations. However, the method only works for watertight models and since they only use a single perspective, the voxelization easily misses surfaces which are parallel to the viewing direction.

The voxelization method provided in this paper eliminates these drawbacks at a minimum of extra costs and is therefore perfectly suitable to provide the boundary condition of 3D distance transforms for arbitrary polygonal meshes.

2.2. 3D Distance Fields

Because of its widespread field of applications, 3D distance fields have been studied for decades. An exhaustive overview of techniques is provided by Jones et al. [JBS06]. Methods can be basically classified into ones which directly compute an exact Euclidean distance field and ones which determine an approximation by propagating discrete distance information across neighboring voxels (distance transforms). The type of input data conforms with this classification. While exact methods require the objects to be represented as triangle meshes, discrete methods work on voxelized input objects.

2.2.1. Exact/Direct Distance Computation

If an exact Euclidean distance field is required, one would have to compute the distance of each sample point to each

© The Eurographics Association 2011.

geometric primitive and choose the distance to the closest primitive as value at the respective point. Since this brute-force strategy is very expensive for meshes with a large number of triangles, the distance field computation is often accelerated by graphics hardware.

Hoff et al [HKL*99] compute generalized Voronoi diagrams (GVD) by generating a polygonal mesh for each site, representing the site's distance function. When rendering such a distance mesh, the rasterizer provides all distances across the mesh by interpolation. A 3D distance field is obtained by rendering the volume slice-wise for each geometric primitive.

A similar approach to compute distance fields is the Characteristics/Scan-Conversion Algorithm introduced by Mauch [Mau03] and efficiently implemented by Sigg et al. [SPG03]. In this method, the relevant volume for the distance field computation at each triangle is reduced by a polyhedron containing the Voronoi cell of the respective triangle. Sud et al. [SOM04] further improve this technique by exploiting spatial coherence between slices. All these methods suffer from the fact that the distance function of certain sites is non-linear and thus, the distance meshes need to be finely tessellated to approximate these non-linear functions. Sud et al. [SGGM06] overcome this problem by interpolating distance vectors rather than just the distance values.

2.2.2. Distance Transforms

The complexity of direct methods depends on the number of triangles in the scene which makes them computationally expensive for complex scenes. In contrast, distance transforms generate a surface boundary condition and distance information is propagated to the rest of the volume. General overviews and analyses of 3D distance transforms can be found in [Bor96, SB02, JBS06].

If only distance values are propagated, the DT is denoted as chamfer DT and was first introduced by Rosenfeld and Pfaltz [RP66]. A distance matrix is used which indicates the local distance values around a voxel. An example of various 3x3x3 matrices is shown in Figure 2. An analysis on optimality of chamfer distance matrices is provided in [BM98].

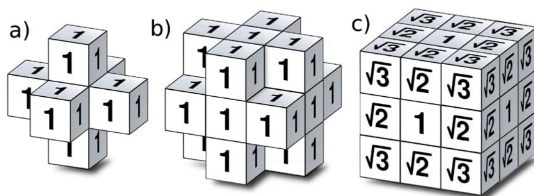

Figure 2: *Chamfer distance matrices for a) Manhattan (aka city-block) b) chessboard and c) Euclidean 3x3x3 metric. The distance values correspond to the respective center voxel.*

This idea was extended by the concept of vector distance transforms (VDT) [Dan80], where not only the distance is propagated, but also the relative coordinates of the nearest surface point. Breen at al. [BMW00] propose a wavefront scheme of a VDT and an efficient GPU implementation for 2D Images is presented by Schneider et al. [SKW09].

Other methods which perform distance transforms on the GPU include the jump-flooding algorithm [RT06] and the fast hierarchical algorithm [CK07], which also provides a control mechanism between speed and accuracy.

While these approaches aim at approximating the Euclidean distance as good as possible, our method is on the other side of the trade-off-scale between speed and accuracy. It is designed for applications which require fast proximity computations even for complex scenes in real time without requiring exact Euclidean distances.

3. Data Structures and Notations

Our method operates on a binary, cubic 3D voxel grid V of size N^3. Let be a binary word consisting of N bits and $[z]$ is the z-th bit. We denote this binary word as *voxel lane* and formulate the voxel grid V as 2D array of voxel lanes $_{x,y}$:

$$V = \{ \ _{x,y} | x, y \in [0..N-1]\}$$

A single voxel $v_{x,y,z} \in \{0,1\}$ can either be accessed by $V[x,y,z]$ or by $_{x,y}[z]$. The introduction of voxel lanes is due to the data structure we use to store an N^3 voxel grid, which is a 2D texture of size $N \times N$. Each texel stores one voxel lane which can be processed in parallel on GPU hardware. Since we can store 32 bits per color channel, we can store a voxel lane of size 128 in each texel, so a 128^3-grid can be stored in a single texture (cf. [ED08]).

It is easily possible to extend the resolution in x- and y-directions, however, in order to increase the z-resolution, we have to perform a slicing approach. Thereby, a voxel grid of size $(k \cdot 128)^3$ is stored in k textures with size $k \cdot 128 \times k \cdot 128$ respectively and each voxel lane $_{x,y}$ contains $k \cdot 128$ bits which are distributed over the k texture slices.

4. Solid- and Boundary Voxelization of 3D Objects

The first step of any distance transform is to obtain the initial boundary condition. Therefore, we transform the objects in the scene into an N^3 voxel grid V such that a voxel $V[x,y,z]$ is '1' if it contains a surface of the mesh and '0' otherwise. In case of solid voxelization, the voxels lying in the interior of an object are set to '1'. We begin with the description of the boundary voxelizer.

4.1. Boundary Voxelization

Determining the required N^3 voxel grid V from a single perspective is a trivial task on the GPU. We simply render the

© The Eurographics Association 2011.

scene with an $N \times N$-viewport and each voxel lane $_{x,y}$ is processed by a fragment shader. This shader quantizes the incoming z-value into the range $[0,..,N-1]$ and sets the respective bit in the voxel lane. It is important that these operations are performed with the depth-test disabled. However, the resulting voxel grid is very sparse at surfaces almost parallel to the viewing direction since the rasterizer misses large areas of these surfaces. This is illustrated in Figure 3.

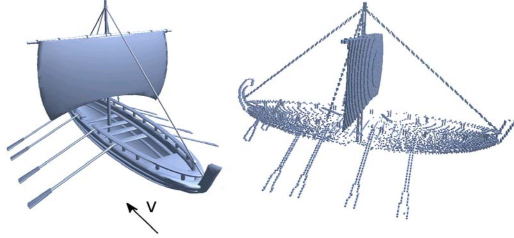

Figure 3: *Problem of voxelization from only one viewing direction v: The surfaces which are almost parallel to v are only sparsely voxelized.*

To remedy this problem, we add information from other perspectives. We use 3 orthogonal directions and render all of them in a single pass using multiple render targets. This results in three individual voxel grids V_f, V_l and V_t, one from each of the perspectives *front, left* and *top*. In order to compose them into a common voxel grid, two of them have to be rotated so that the corresponding 3D data matches. We denote the rotated version of a voxel grid V as $\hat{V}$. However, this rotation is not trivial due to the fact that voxels are stored in the bits of 2D textures.

Using the front-direction as reference, we have to determine rotated version of each voxel lane $\hat{}^{\,l}_{x,y}$ and $\hat{}^{\,t}_{x,y}$ to combine them with the voxel lane $^{f}_{x,y}$. In order to achieve that these voxel lanes represent the same portion of the 3D voxel space, the following identities between the rotated version $\hat{}^{\,l}_{x,y}$ and the original lane $^{l}_{x,y}$ must hold:

$$\hat{}^{\,l}_{i,j}[k] = {}^{l}_{j,k}[i] \quad i,j,k = 0,..,N-1.$$

Analog for the top perspective:

$$\hat{}^{\,t}_{i,j}[k] = {}^{t}_{k,i}[j] \quad i,j,k = 0,..,N-1$$

As we can see, a rotated voxel lane requires an extraction of a single bit from N different voxel lanes as well as the composition of these bits to a binary word with N bits. However, this operation can be done in parallel for each voxel lane in a fragment shader. The render target for this pass is an $N \times N$ quad and the textures containing the voxel grids V_f, V_l and V_t are provided to the shader, which then performs the rotation steps and the combination of all three perspectives using a logical OR. The final voxel grid V is therefore obtained by

$$V = V_f \lor \hat{V}_l \lor \hat{V}_t. \tag{1}$$

4.2. Solid Voxelization

If objects in the scene have well-defined interior and exterior regions, we can assign solid voxels to the interior parts which will later be used to identify signed distances. While various solid voxelization algorithms [DCB*04, ED08] require 100% watertightness of the models, we also allow for objects which consist of solid as well as non-solid parts. An example for such an objects is the potted plant from Figure 4. While the pot is solid, the leafs are modeled as surfaces.

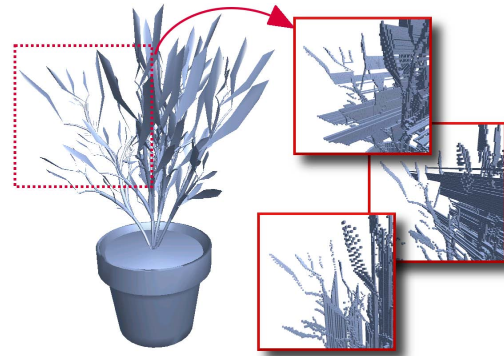

Figure 4: *Each individual perspective can show misleading volume due to the fact that the leafs of the plant are modeled as surfaces.*

In the first step of our algorithm, we render the scene without z-tests and quantize the depth of incoming fragments in the bits of the respective voxel lane. In contrast to the boundary voxelization, we distinguish between front- and back-facing polygons and write the respective voxels in two separate grids, a front-grid V^f and a back-grid V^b. This operation is done for each perspective in a single pass.

In a second pass, these grids are handed to a fragment shader which renders an $N \times N$ buffer, producing the voxel lanes of the solid voxel grid V^s. The shader detects all intervals $[a,b]$ between bits of a front-lane $^{f}_{x,y}$ and a back-lane $^{b}_{x,y}$ for which $^{f}_{x,y}[a] = {}^{b}_{x,y}[b] = 1$ and additionally $^{f}_{x,y}[i] = {}^{b}_{x,y}[i] = 0 \quad \forall i \in [a+1, b-1]$. The bits inside such an interval are defined as solid voxels and are written into the resulting solid voxel lane $^{s}_{x,y}$. This process is illustrated in Figure 5.

Yet, we have separate data from three different perspectives and have to combine them into a common voxel grid. However, the combination of solid voxels cannot be done in the same way as the combination of the boundary voxels. As we can see in Figure 4, the fact that some parts of the objects are not solid yields to misleading volume. In contrast to the boundary voxels, where the goal was to add missing information from other perspectives, we now want to subtract information. This is possible since the misleading

© The Eurographics Association 2011.

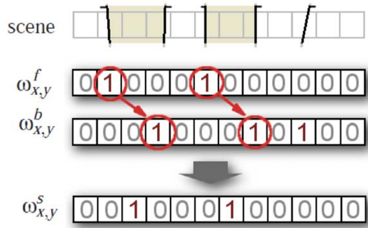

Figure 5: *Determining solid voxels for a single lane. From top to bottom: oriented surfaces along a discretized voxel lane, corresponding front-lane* ^{f} *and back-lane* ^{b}*, resulting solid voxel lane* ^{s}.

volumes are view-dependent. A solid voxel is thus only defined where it occurs in all three perspectives. These voxels are obtained by combining the solid voxel grid V_f^s from the front-perspective with the rotated grids $\hat{V}_l^s$ and $\hat{V}_t^s$ from the left- and top-perspective using a logical AND-operation. The formula for the entire solid voxel grid V^s thus renders to:

$$V = V_f^s \wedge \hat{V}_l^s \wedge \hat{V}_t^s$$

The solid voxelization implicitly provides a boundary voxelization by combining the front-grid V^f and the back-grid V^b for each perspective and then combining the results in the same way as in Equation 1. Figure 6 shows results of our voxelization method while performance analyses are provided subsequently in Section 6.

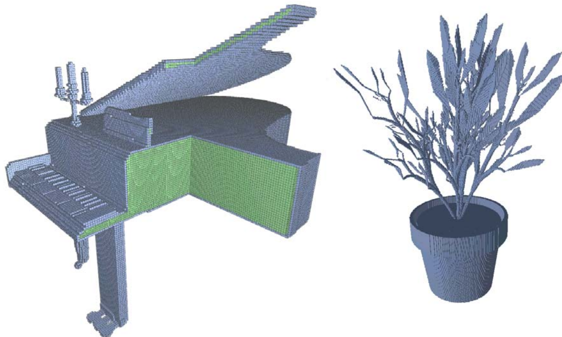

Figure 6: *The object on the left was processed with a resolution of* 256^3 *voxels. Cutting planes are added to illustrate solid voxels (green). The object on the right is voxelized in a* 512^3 *grid.*

5. 3D Distance Transform

We will now present a simple and easy-to-implement distance transform which is based on the voxelization from the previous section. The distance value at each voxel is obtained from the values of neighboring voxels, which classifies our method as chamfer distance transform. Traditional algorithms loop through the entire N^3 voxel grid in a forward, and again in a backward pass in order to obtain a full distance field. In contrast to this $O(N^3)$ method, we perform a wavefront scheme which computes a distance layer by looping through the voxel lanes $_{i,j}$ which is $O(N^2)$.

5.1. Distance Propagation

As mentioned in the introduction, we only use chamfer metrics which produce integer distances. These include the distance masks from Figure 2 a) and b), the Manhattan and the chessboard distance. These matrices define a local $3\times3\times3$ neighborhood around a center voxel, which always has the matrix entry '0'. The neighboring entries denote the distance value to that center.

Since we only use matrices containing zeros and ones, we denote a matrix M as

$$M = \{m[i,j,k] \in \{0,1\} \mid i,j,k = -1,0,1\}$$

Furthermore we pool all voxels having the distance n in a separate binary voxel grid V_n. The layer V_0 is the original voxel grid as obtained from the boundary voxelization described in section 4.1. Each new layer V_{n+1} can be computed from the previous layers $V_n, V_{n-1}, .., V_0$ in the following way:

$$V_{n+1}[x,y,z] = \Big(\bigvee_{i,j,k} V_n[x+i,y+j,z+k]\cdot m[i,j,k]\Big) \wedge \neg \bigvee_{l=0}^{n} V_l[x,y,z] \quad (2)$$

where i,j,k = -1,0,1. The final OR-concatenation of all previous layers is necessary to avoid an overwriting of voxels which are already occupied by shorter distances. This guarantees that the layers are disjoint and each voxel can be unambiguously assigned a distance value. In order to compute a single layer, equation 2 would have to be applied for all voxels in the domain, i.e. N^3 times. This sounds bad since traditional chamfer methods require two loops through the N^3 grid to compute an entire distance field.

However, the clue to our method lies in the fact that we can reduce the number of operations per layer to $O(N^2)$ due to an efficient implementation. Since a voxel lane $_{x,y}$ is compactly stored as a series of unsigned integers, we can process an entire lane using only simple bit-shifts and logical operations on these integers in a fragment shader. The details about this efficient method are provided in the following.

5.2. Implementation

The data from the voxelization, i.e. the zero-set V_0 of the distance field, is residing on the GPU in form of one or more 2D textures as described in section 3. This data structure allows us to implement the propagation scheme from equation 2 in a highly efficient way. In each new layer V_{n+1}, a voxel is set if

© The Eurographics Association 2011.

i) it is not set in any of the previous layers V_i, $i = 0,..,n$ and *ii)* at least one neighbor defined by the respective distance matrix is set in the previous layer V_n.

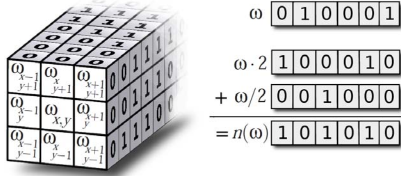

Figure 7: *left: 8-neighborhood of a voxel lane $_{x,y}$. Right: computation of a voxel lane $n(\ _{x,y})$ by simply adding bit-shifted versions of $_{x,y}$.*

Instead of performing these tests on each individual voxel, we process an entire voxel lane $^{n+1}_{x,y}$ of the new layer V_{n+1} at once. Therefore, we consider the eight neighbors around a corresponding voxel lane $^{n}_{x,y}$ from the previous layer V_n as shown in Figure 7 left. In addition, we define a neighborhood operation $n(\ )$ on a voxel lane, which produces a voxel lane containing the left and the right neighbors of each bit in the original lane (cf. Figure 7 right). This operation is simply obtained by combining bit-shifted versions of the binary word :

$$n(\ ) = \ 2 \cdot \ + \frac{1}{2} \ = \frac{5}{2} \qquad (3)$$

For the chessboard metric, we additionally require a neighborhood operation $n'(\ )$ which also contains the original bits. This is simply done by adding to equation 3, so $n'(\ ) = \ 7/2 \cdot$. However, the binary word is in practice distributed over various RGB-components and texture slices. In the implementation, we have to take care that possible over/underflowing bits are handed over to the next/previous component.

The first part of computing a new voxel lane $^{n+1}$ using the Manhattan metric is to compose the voxel lanes $n(\ ^{n}_{x,y})$, $^{n}_{x+1,y}$, $^{n}_{x-1,y}$, $^{n}_{x,y+1}$ and $^{n}_{x,y-1}$ with a logical OR. The remaining operation is the one corresponding to the second part of equation 2, i.e the test whether the respective voxel is already set in one of the previous layers. To do this efficiently, we use an accumulation grid V_A which contains the concatenation of all previous layers and which is updated in each computation of a new layer. The second OR-concatenation of 2 can thereby be substituted by $V_A[x,y,z]$. We denote the voxel lanes of the accumulation grid as ^{A}.

The entire operation which has to be performed for each voxel lane $^{n+1}_{x,y}$ in order to obtain a new distance layer V_{n+1} using the Manhattan metric is:

$$^{n+1}_{x,y} = \left(\frac{5}{2}\ ^{n}_{x,y} \vee \ ^{n}_{x+1,y} \vee \ ^{n}_{x-1,y} \vee \ ^{n}_{x,y+1} \vee \ ^{n}_{x,y-1}\right) \wedge \neg \ ^{A}_{x,y}$$

In case of the chessboard metric, we additionally have to consider the edge-neighbors. The corresponding formula renders to:

$$^{n+1}_{x,y} = \left(\frac{5}{2}\ ^{n}_{x,y} \vee \frac{7}{2}\ ^{n}_{x+1,y} \vee \frac{7}{2}\ ^{n}_{x-1,y} \vee \frac{7}{2}\ ^{n}_{x,y+1} \vee \frac{7}{2}\ ^{n}_{x,y-1} \vee \ ^{n}_{x+1,y+1} \vee \ ^{n}_{x+1,y-1} \vee \ ^{n}_{x-1,y+1} \vee \ ^{n}_{x-1,y-1}\right) \wedge \neg \ ^{A}_{x,y}$$

Note that we could also implement a metric which considers the full 26-neighborhood around a voxel as distance '1' by simply applying the multiplication of $7/2$ to all of the 8 neighboring voxel lanes. However, the chessboard metric is more accurate since vertex neighbors have an Euclidean distance of $\sqrt{3}$, which is better approximated by 2 than by 1.

If a solid voxelization was applied prior to the distance propagation, a sign can be trivially assigned to each distance by testing whether the respective voxel is set in the solid voxel grid V^S.

6. Results

This section provides performance analyses of the individual steps of our framework and demonstrates the application of our distance transform for proximity computations. The implementation was done in OpenGL and the evaluation was performed using an Nvidia GTX 280 graphics card.

6.1. Performance

Although voxelization is a pre-processing step for 3D distance transforms, it is also crucial that this step is fast in order to allow applications for an interactive re-initialization of the zero-set in highly dynamic scenes. This goal is met by our voxelization scheme as illustrated in Table 1.

Triangles	Res.	B.Vox.	S.Vox.
10,000	128^3	0.31	0.85
	256^3	1.4	2.3
	512^3	6.8	10.4
100,000	128^3	0.7	1.3
	256^3	2.1	3.5
	512^3	8.5	13.4
300,000	128^3	2.4	2.9
	256^3	5.5	6.7
	512^3	15.1	16.9

Table 1: *Time in ms for the boundary and the solid voxelization depending on the number of triangles and resolution of the voxel grid.*

The first step of the voxelization, rendering the scene from 3 different perspectives, is the only one in the pipeline which depends on the complexity of the scene. All subsequent steps, including those of the distance transform, are

© The Eurographics Association 2011.

performed on voxel grids and thus only depend on the voxel resolution. The solid voxelization needs additional effort for the detection of solid intervals as described in Section 4.2, which becomes the bottleneck for scenes with only few triangles. In more complex scenes, the first step is the dominant factor of the run-times.

Besides the fast computation, our voxelization method also works very robust for objects which are not entirely watertight. This property is not achieved by the previous approaches mentioned in Section 2.1.

The propagation of the distance field directly builds upon the output of the voxelization step without requiring any data transfers or conversions. The performance of the distance transform is shown in table 2 and a visualization of distance layers around 3D objects can be seen in figure 1.

Res.	Single Layer	Complete DF	Memory
128^3	0.019	2.43	262 KB
256^3	0.031	7.93	2.1 MB
512^3	0.053	27.13	16.76 MB

Table 2: *Run-time (in ms) of the distance transform for a single layer and an entire distance field. The last column shows the memory consumption of a single layer.*

We illustrate numbers for a single layer as well as a complete distance transform requiring a worst-case computation of N distance layers for an $N \times N$ grid. However, the worst-case scenario is just theoretical and very unlikely to happen in real applications. In fact, some applications only require a distance field in a certain range around an object so the propagation can be stopped if this range is compassed. The complexity of the algorithm then reduces to MN^2, where M is the number of required distance layers.

A major problem of propagating a large number of distance layers in a high-resolution voxel grid is memory consumption. The last column of Table 2 shows the required memory of a single distance layer on the GPU. Since a voxel layer is rendered in the bits of 2D textures, the information is stored very compactly, nevertheless, we have to employ new textures for each discrete distance value. This is also a key issue why we restrict the distance to integer metrics. Theoretically, we could also apply our framework to chamfer metrics such as the Euclidean 3x3x3 from Figure 2 c) by introducing $\sqrt{2}$- and $\sqrt{3}$-layers. However, the number of different distance values rapidly increases away from a surface which makes the method infeasible for non-integer metrics.

6.2. Post-Processing Data Conversion

It is suitable to transform the distance information into a true 3D structure for further processing. Evaluating the distance value at one specific voxel is cumbersome for an application if the distance information resides in multiple 2D textures. In the worst case, all textures have to be traversed in order to determine the layer having a '1' on the corresponding position.

Transforming the data into a 3D array can be trivially done on the CPU. However, data transfer between the GPU and CPU is time-critical for real-time applications. The conversion can also be done directly on the GPU by rendering into the slices of a 3D texture, which requires N passes for an N^3 voxel grid. In the $i-th$ render pass, the $i-th$ bit of each voxel lane is extracted and multiplied by the corresponding layer index. If the signed distance is required an additional multiplication with -1 has to be applied if the corresponding solid voxel is set. After this process, a distance value can be directly accessed by a single texture lookup. The time consume of this conversion is in the same magnitude as performing an entire distance transform of the corresponding resolution.

6.3. Proximity Computations

To demonstrate the efficiency of our method, we implemented an application which performs real-time proximity/collision computations of two moving objects. For each frame, we compute a complete 3D distance field around one of the objects with a 128^3 voxel grid. While rendering the other object, the proximity to the first one is evaluated on the fly in a vertex shader. For each incoming vertex, we look up the respective discretized point in the computed distance field. Finally, we assign a color to the vertex which encodes the distances in a HSV-scale from red to blue in order to visualize the distances. Figure 8 shows a screenshot of this application, which runs with 125 frames per seconds.

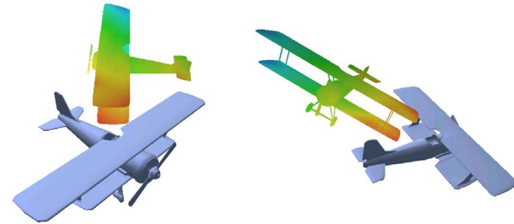

Figure 8: *A scene with two airplane objects shown from different perspectives. The proximity of one object to the other was evaluated for each vertex using our distance transform and is color-coded for visualization.*

A collision of the objects can be easily detected (or even predicted) using this framework. Also, the location of a collision can be obtained up to voxel precision.

7. Conclusion

We have presented an entire framework to compute a 3D distance field entirely on the GPU. This includes a robust and

© The Eurographics Association 2011.

efficient voxelization which provides the zero-set of the distance field and is also capable of solid voxelization in order to define signed distances. A big advantage is that the data type of the voxels is adapted to the required input data type for the distance transform. This provides a smooth and consistent pipeline from polygonal meshes to a discrete 3D distance field. The type of distance metric is restricted to integer metrics such as the Manhattan or the chessboard distance in order to allow for a fast propagation of discrete distance values. Our method is designed for applications which do not require exact Euclidean distances but need fast computation of proximities. The consistent combination of voxelization and distance transform also allows for a fast re-initialization of the distance field, which is important for real-time applications with high dynamic scenes.

References

[BC01] BAERENTZEN J. A., CHRISTENSEN N. J.: A technique for volumetric csg based on morphology. In *Proc. Conf. Volume Graphics '01* (2001), pp. 71–79.

[BC08] BASTOS T., CELES W.: GPU-accelerated adaptively sampled distance fields. *International Conference on Shape Modeling and Applications* (2008), 171–178.

[BM98] BUTT M. A., MARAGOS P.: Optimum design of chamfer distance transforms. *IEEE Transactions on Image Processing 7* (1998), 1477–1484.

[BMW00] BREEN D., MAUCH S., WHITAKER R.: 3D scan conversion of csg models into distance, closest-point and colour volumes. In *Proc. of Volume Graphics (2000* (2000), pp. 135–158.

[Bor96] BORGEFORS G.: On digital distance transforms in three dimensions. *Comput. Vis. Image Underst. 64* (1996), 368–376.

[BSB*00] BITTER I., SATO M., BENDER M., MCDONNELL K. T., KAUFMAN A., WAN M.: CEASAR: a smooth, accurate and robust centerline extraction algorithm. In *Proceedings of the conference on Visualization '00* (2000), pp. 45–52.

[CK07] CUNTZ N., KOLB A.: Fast hierarchical 3D distance transforms on the GPU. In *Proceedings of Eurographics '07 (short paper)* (2007), pp. 93–96.

[Dan80] DANIELSSON P.-E.: Euclidean distance mapping. *Computer Graphics and Image Processing 14* (1980), 227–248.

[DCB*04] DONG Z., CHEN W., BAO H., ZHANG H., PENG Q.: A smart voxelization algorithm. In *Proceedings of Pacific Graphics 2004* (2004), pp. 43–50.

[ED08] EISEMANN E., DÉCORET X.: Single-pass GPU solid voxelization for real-time applications. In *Proceedings of graphics interface 2008* (2008), pp. 73–80.

[FC00] FANG S., CHEN H.: Hardware accelerated voxelization. *Computers and Graphics 24* (2000), 200–0.

[FLM03] FOSKEY M., LIN M. C., MANOCHA D.: Efficient computation of a simplified medial axis. In *Proceedings of the eighth ACM symposium on Solid modeling and applications* (2003), pp. 96–107.

[FPRJ00] FRISKEN S. F., PERRY R. N., ROCKWOOD A. P., JONES T. R.: Adaptively sampled distance fields: a general representation of shape for computer graphics. In *SIGGRAPH '00* (2000), pp. 249–254.

[GBF03] GUENDELMAN E., BRIDSON R., FEDKIW R.: Nonconvex rigid bodies with stacking. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2003)* (2003), vol. 22, pp. 871–878.

[HKL*99] HOFF III K. E., KEYSER J., LIN M., MANOCHA D., CULVER T.: Fast computation of generalized voronoi diagrams using graphics hardware. In *SIGGRAPH '99* (1999), pp. 277–286.

[JBS06] JONES M. W., BAERENTZEN J. A., SRAMEK M.: 3D distance fields: A survey of techniques and applications. *IEEE Transactions on Visualization and Computer Graphics 12* (2006), 581–599.

[KBSS01] KOBBELT L. P., BOTSCH M., SCHWANECKE U., SEIDEL H.-P.: Feature sensitive surface extraction from volume data. In *SIGGRAPH '01* (2001), pp. 57–66.

[Mau03] MAUCH S. P.: *Efficient algorithms for solving static hamilton-jacobi equations*. PhD thesis, California Institute of Technology, Pasadena, CA, USA, 2003.

[Mon68] MONTANARI U.: A method for obtaining skeletons using a quasi-euclidean distance. *J. ACM 15* (1968), 600–624.

[MRS08] MORVAN T., REIMERS M., SAMSET E.: High performance GPU-based proximity queries using distance fields. *Computer Graphics Forum 27* (2008), 2040–2052.

[NDS04] NOVOTNY P., DIMITROV L. I., SRAMEK M.: CSG operations with voxelized solids. In *Proceedings of the Computer Graphics International* (2004), pp. 370–373.

[RP66] ROSENFELD A., PFALTZ J. L.: Sequential operations in digital picture processing. *J. ACM 13* (1966), 471–494.

[RT06] RONG G., TAN T.-S.: Jump flooding in GPU with applications to voronoi diagram and distance transform. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games* (2006), pp. 109–116.

[SB02] SVENSSON S., BORGEFORS G.: Digital distance transforms in 3D images using information from neighbourhoods up to 5x5x5. *Comput. Vis. Image Underst. 88* (2002), 24–53.

[SGGM06] SUD A., GOVINDARAJU N., GAYLE R., MANOCHA D.: Interactive 3d distance field computation using linear factorization. In *In Proc. ACM Symposium on Interactive 3D Graphics and Games* (2006), pp. 117–124.

[SKW09] SCHNEIDER J., KRAUS M., WESTERMANN R.: GPU-based real-time discrete euclidean distance transforms with precise error bounds. In *International Conference on Computer Vision Theory and Applications (VISAPP)* (2009), pp. 435–442.

[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.

[SPG03] SIGG C., PEIKERT R., GROSS M.: Signed distance transform using graphics hardware. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)* (2003), pp. 83–90.

[ZT99] ZHOU Y., TOGA A. W.: Efficient skeletonization of volumetric objects. *IEEE Transactions on Visualization and Computer Graphics 5* (1999), 196–209.

© The Eurographics Association 2011.