Piecewise Linear Approximation of Signed Distance Fields

Jianhua Wu, Leif Kobbelt
Computer Graphics Group, RWTH Aachen, Germany

Abstract

The signed distance field of a surface can effectively support many geometry processing tasks such as decimation, smoothing, and Boolean operations since it provides efficient access to distance (error) estimates. In this paper we present an algorithm to compute a piecewise linear, not necessarily continuous approximation of the signed distance field for a given object. Our approach is based on an adaptive hierarchical space partition that stores a linear distance function in every leaf node. We provide positive and negative criteria for selecting the splitting planes. Consequently the algorithm adapts the leaf cells of the space partition to the geometric shape of the underlying model better than previous methods. This results in a hierarchical representation with comparably low memory consumption and which allows for fast evaluation of the distance field function.

1 Introduction

The design of efficient algorithms always has to be accompanied by the design of suitable data representations. In 3D computer graphics applications the data to be processed usually consists of geometric objects or scene descriptions. The two major categories for representing this data are *parametric* and *implicit*.

In the parametric case, a surface is given as the *range* of a function $\mathbf{f} : R^2 \to R^3$ and hence points on the surface can easily be generated by just evaluating $\mathbf{f}$. As a consequence, parametric representations are mostly used for computations *on* the surface since most 3D problems can be reduced to 2D problems in the parameter domain.

An implicit representation defines a surface as the *kernel* of a function $f : R^3 \to R$, i.e., as the set of points (x, y, z) such that $f(x, y, z) = 0$. This simplifies geometric queries such as inside/outside tests to mere function evaluations. Although there are many different choices for the function f to represent a given surface, the most common one is the signed distance function which assigns to every point in space its Euclidean distance to the surface.

In this paper we consider scenarios where a geometry processing algorithm is equipped with an additional representation of the signed distance function to obtain direct access to implicit shape information. Typical applications are, e.g., mesh decimation or mesh fairing where one frequently has to check the current deviation from the originally given surface. Here the error estimates can be found by evaluating the implicit representation while the actual processing is performed on an explicit representation. One eminent example [20] has been published recently. In this setup it is easy to identify the major requirements that an implicit representation has to satisfy.

- **Approximation power**: For freeform shapes the implicit representation usually provides only an approximation of the original surface. Hence an explicit to implicit conversion algorithm has to take the maximum error tolerance as an input parameter. The approximation power of a representation measures the rate by which the memory consumption increases if the tolerance is decreased.
- **Adaptivity**: To minimize the memory requirements one is also interested in a representation that adapts to the shape of the underlying object. In its simplest form, adaptivity of an implicit representation means that high accuracy is guaranteed only in the vicinity of the surface itself. Far away from the surface the approximation of the signed distance function does not have to be as precise. More subtle adaptivity further takes the fact into account that flat surface regions can be approximated much easier than highly curved regions.
- **Efficiency**: A critical property is the complexity of the evaluation algorithm. The ideal situation would be constant complexity but this usually contradicts the goal of adaptivity. By using hierarchical representations it is at least possible to bound the complexity by the logarithm of the precision tolerance.

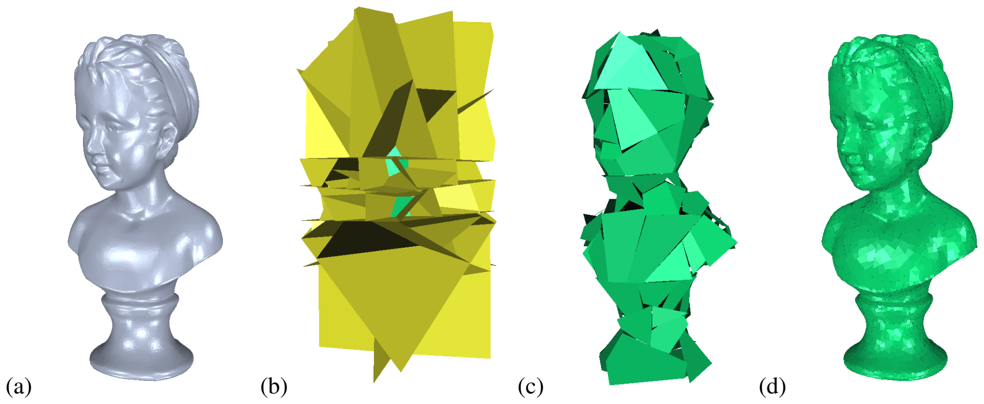

Figure 1: For a given polygon mesh surface (a) we compute a binary space partition (BSP) tree and (b) assign a linear distance function to every leaf cell. Figure (c) shows the discontinuous zero-set of the resulting piecewise linear scalar field and (d) shows another zero-set with a smaller approximation error and hence a finer space partition. Our goal in this paper is, for a given error tolerance, to choose the BSP splitting planes in an effective way.

In the following sections we are proposing an implicit representation that is able to approximate the signed distance function of a given surface up to any prescribed precision (cf. Figure 1). As input format, we are focussing on polygon meshes but in principle it would be easy to generalize the construction to arbitrary input data. The major features of our representation are that ...

- ... we use a piecewise linear instead of a piecewise tri-linear representation since both have the same approximation power but linear functions need less coefficients and are faster to evaluate.
- ... we are not requiring the approximate distance field to be a continuous function. Again, this does not have any effect on the approximation power but it decouples neighboring cells since no C^0 boundary conditions have to be taken into account. In particular we do not need any balancing mechanism that propagates local refinement operations into neighboring cells to bound the level jump in a hierarchical representation.
- ... we use a general BSP tree which means that the splitting planes that define the partition cells do not have to be axis aligned. As a consequence the partition can adapt even better to the surface since, from a certain refinement level on, most splits will cut the surface in normal direction rather than in tangential direction. We derive geometric heuristics that promote good splits and prevent bad ones.

2 Related works

Over the last years many volumetric geometry representations have been proposed in the field of implicit modeling and volume graphics and many algorithms have been developed to convert parametric representations into implicit ones [15, 24, 20, 26] and vice versa [18, 16, 14]. Interested readers can find detailed overviews in [5, 8, 21].

In this paper, we are focusing on representations that approximate the signed distance field by a trivariate piecewise polynomial function defined over a space partition. The most basic data structure till now is a uniform 3D grid $[s_{ijk}]$ that stores a scalar value at each grid node (i, j, k). This discrete data is interpolated into the voxel cells by tri-linear functions [18].

Since for signed distance fields the accuracy in the vicinity of the (zero-) surface is usually more important than further away from the surface, the uniform grid is not memory efficient because the spatial scalar field is sampled with the same rate everywhere and this makes it difficult to deal with high-resolution/precision objects. This redundancy can be avoided by adapting the sampling density to the distance from the surface [25]. The three colors “white”, “black”, and “gray” are assigned to cells that are completely inside, completely outside, or intersect the surface, respectively. Then starting from some initial cell decomposition, the gray cells are recursively refined until a prescribed error tolerance is met. If we stick to cubical voxel cells then this approach leads to adaptively refined octrees, the so-called 3-color-octrees [22].

To further reduce the redundancy, we can restrict the local refinement to those cells for which the tri-linear interplant deviates more than the prescribed tolerance from the actual distance field. This restriction leads to even better adaptivity since extreme refinement is only necessary in the vicinity of the highly curved surface regions [11].

Besides these variations of voxel-based piecewise tri-linear distance field representations there are alternatives like the permission grid [27] where piecewise constant functions are used or wavelet-based representations with higher order basis functions [19]. While the approximation power of piecewise constant functions is not sufficient for practical precision requirements, higher order functions provide good approximation. However, since the corresponding basis functions span over several neighboring voxels, the inter-dependencies between these voxels make it necessary that a whole neighborhood of cells has to be refined to guarantee consistency. This is called balancing and in practice it usually causes the generation of many redundant cells.

3 Overview

We compute an approximation of the signed distance function by taking the major requirements mentioned in the introduction into account.

First of all we want to achieve sufficient approximation power. In practice, piecewise tri-linear functions have been used successfully. We however prefer to use piecewise linear functions instead. These have the same approximation order but need less coefficients and consequently they are faster to evaluate and need less memory. In 2D, *continuous* piecewise linear functions have already been applied successfully to approximate the distance functions of curves [23, 17]. A linear function in 3D is given by a normal vector $\mathbf{n}$ and an offset value d (vs. 8 scalar values for a tri-linear function). In our implementation we quantize the possible orientations of the normal vectors to 16 bit by using the refined-octahedron-indexing scheme of [7] and the offset d as another 2 Bytes.

In order to avoid the balancing effect that causes a local refinement operation to spread over neighboring cells, we do not enforce any continuity of the distance function between neighboring cells which also differs from the previous 2D applications [23, 17]. Using C^{-1} continuous functions instead of C^0 does not affect the asymptotic approximation power but the additional degrees of freedom usually lead to a space partition with significantly fewer cells for the same approximation tolerance.

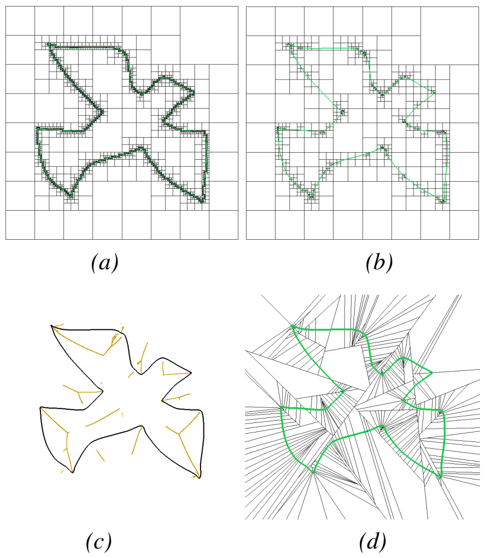

Figure 2: The piecewise approximation of the signed distance field to the contour shown in (c). The 3-color quadtree in (a) uses 102040 cells for a given approximation tolerance ε and produces a C^0 piecewise bi-linear distance field. In (b) the same approximation error can be obtained by using a piecewise linear C^{-1} approximation. In this case the approximation can adapt to the local curvature such that 895 cells are sufficient. In (d) we show a BSP-tree decomposition. By using a linear distance function for every cell we obtain the same approximation error with only 254 cells. For the selection of the splitting planes we used the medial axis information as brown segments shown in (c).

Trying to most flexibly adapt the size, shape and orientation of the cells in the space partition, we are using a general BSP-tree data structure. This allows us to use both, the location and the orientation of the splitting planes for the adaptation. In Figure 2 we show a 2D example and demonstrate the effects of the various improvements.

4 Generation steps

Given a polygonal mesh $\mathcal{M}$ as input surface plus an error tolerance ε, our goal is to generate an *nearly* optimized piecewise linear approximation of the corresponding signed distance field. For this we have to choose the BSP splitting planes in an effective way such that the approximation errors within each cell stays under the prescribed error bound while using as few cells as possible. The recursive algorithm is given by the pseudo-code in Fig. 3. Initially we call the procedure with an empty root node T and the complete mesh $\mathcal{M}$ as input parameters.

```
CREATE ( $tree\,T,\ mesh\,\mathcal{M}$ ) :
Find a linear function $f$ that approximates the distance
function to $\mathcal{M}$ and estimate the approximation error $\delta$
if ( $\delta > \varepsilon$ )
    select a splitting plane $P$;
    split $\mathcal{M}$ into $\mathcal{M}_{left}$ and $\mathcal{M}_{right}$ by $P$;
    $T.plane = P$;
    CREATE ( $T.left\_child,\ \mathcal{M}_{left}$ );
    CREATE ( $T.right\_child,\ \mathcal{M}_{right}$ );
```

Figure 3: Pseudo-code for the generation algorithm.

In the generation algorithm, there are two major sub-procedures: (1) how to approximate the distance function to a mesh $\mathcal{M}$ by a linear function f with minimum error and (2) how to select a proper BSP splitting plane that generates an optimal space/shape partition. These issues will be addressed in the following sections.

4.1 Least squares linear approximation

For the mesh $\mathcal{M}$ that lies within some cell of the BSP tree, we need to compute a linear function f that approximates $\mathcal{M}$'s distance function. Here we use the standard least squares approach to the point set $\{P_i,\ i = 1 \ldots n\}$ formed by all vertices in $\mathcal{M}$. The center of gravity is $\bar{P}$ and the point variance set is $\{Q_i = (x_i, y_i, z_i)^T = P_i - \bar{P},\ i = 1 \ldots n\}$. Then from the covariance matrix of $\{Q_i\}$ we compute three real eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3$ and we denote the corresponding eigenvectors by $V_1,\ V_2,\ V_3$ respectively. The normal direction N of the optimal approximation plane in the least squares sense is then given by V_3 (direction of minimal variance). To derive the optimal offset value d we compute $d_i = N^T P_i$ for all mesh vertices and set $d = (\max\{d_i\} + \min\{d_i\})/2$. Now the function $f(X) = N^T X - d$ approximates the signed distance function up to a factor of ± 1. This factor can be determined by comparing N to the average normal direction across the mesh patch $\mathcal{M}$.

Moreover we immediately find an estimate for the approximation error which is $\delta = (\max\{d_i\} - \min\{d_i\})/2$ where we exploit the fact that triangles lie within the convex hull of their corner vertices. Notice, however, that this estimate is only valid over that part of the plane $N^T X = d$ that is covered by the orthogonal projection of the mesh $\mathcal{M}$. In most of the practical configurations this region will cover the whole cell for which the distance field approximation is computed. In some exceptional cases it can happen that the cell extends beyond this region which makes further refinement necessary. This is explained in detail in section 4.3.

4.2 BSP splitting plane selection

We present three different strategies for the selection of the splitting planes. The first strategy uses a simple and fast heuristic. In the second strategy we use the medial axis of the given surface to find splitting planes that lead to an improved segmentation of the surface and in the third strategy we additionally disallow splitting planes that lead to problematic configurations where the mesh falls into several components.

Strategy A:

A natural candidate for the splitting plane can be derived from the above least square approximation. The eigenvector V_1 indicates the direction of maximum variance and yields the (statistically most efficient) splitting plane's normal vector N_P. The offset value d_P is adjusted such that some balancing criterion is met, e.g., the resulting sub-meshes $\mathcal{M}_{left}$ and $\mathcal{M}_{right}$ have the same number of vertices or the same surface area, or we set d_P such that the splitting plane contains the center point $\bar{P}$.

Strategy B:

The least squares approach computes the statistical distribution of the mesh vertices in R^3 but it does not take the geometric shape of the mesh into account. As in Fig. 4 it is a good heuristic to segment the mesh along curvature maxima since this produces subpatches $\mathcal{M}_{left}$ and $\mathcal{M}_{right}$ with maximum expected flatness. It can be detected by looking at the medial axis (MA) [6, 1] of the surface: for each curvature maximum there is a branch of the medial axis that points to it.

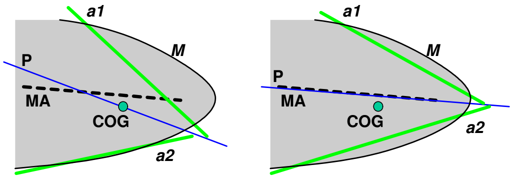

Figure 4: Comparing strategy A (left) and B (right) shows that the piecewise linear approximation after the split is tighter if the medial axis (MA) is used to determine the split direction.

Computing the exact medial axis is difficult and unstable [13, 2], hence we use the sphere growing method [4] to compute discrete sample points on the medial axis. Before recursive BSP generation, we store for each mesh vertex V_i the tuple (C_i, w_i, b_i, j), where C_i, representing a MA sample, is the center of the smaller sphere among two possible maximum spheres touching V_i; j is the index of another vertex touching that sphere; w_i is the sum the triangle areas adjacent to V_i to compensate for varying vertex density in the input mesh $\mathcal{M}$; b_i is a flag indicating if this MA sample lies inside or outside the object.

During the recursive procedure, the new splitting plane candidate is the least squares plane fitting to the MA samples which is most likely to split the mesh in a highly curved region. To reduce the disturbing effects that occur when the MA falls into several components (inside and outside the surface) we classify the samples by flags b_i and only use those ones that belong to the majority. Even if it may happen that the medial axis has several inside (outside) components this heuristic leads to sufficiently reliable results once the partition is fine enough such that only one major curvature maximum lies within a cell.

A *weighted* least squares fitting is used to compute the splitting plane by MA samples, i.e., we start by computing the weighted average:

$$\bar{C} \ = \ \sum_i w_i \, C_i \ / \ \sum_i w_i$$

and then define the weighted variance set as $\{Q_i = w_i\,(C_i - \bar{C})\}$ to form a covariance matrix. The resulting least squares plane puts more emphasis on MA samples which belong to mesh vertices in sparsely sampled regions and less emphasis on samples from denser regions. The weight coefficient in fact guarantee a constant impact per surface area. The improvement of the approximation due to the MA based splitting plane selection strategy is demonstrated in Fig. 8. If the ratio of largest eigenvalue of the weighted covariance matrix to the smallest one falls below some threshold (25 in our implementation), we consider the medial axis information not as reliable and simply fall back to selection strategy A.

Once the splitting plane is found, we separate the given mesh $\mathcal{M}$ into two sub-mesh parts together with the MA samples. In order to guarantee that they are not influencing each other after the split, we remove the subset of MA samples (C_i, w_i, b_i, j) where the two vertices V_i and V_j belong different sub-meshes by setting the weight coefficients w_i of the corresponding samples to zero.

Strategy C:

Selection strategy B promotes good splits that separate the mesh into sub-meshes with maximum expected flatness. In the third strategy we add another criterion that prevents bad splits. Here a bad split is characterized by the property that one of the sub-meshes $\mathcal{M}_{left}$ and $\mathcal{M}_{right}$ falls into several components. Those splits are disadvantageous since the least squares fitting described in section 4.1 assumes that the mesh has just one connected component. In fact the resulting linear approximation of the signed distance function is only valid over the regions that are covered by the projection of the corresponding mesh. If the mesh has several components, the distance estimate is invalid in between the projections of these components (cf. Fig. 5). In order to find a simple criterion to detect bad splits, we observe that such splits typically cut off a *cap* from an apical point of the surface. Hence we have to identify all potentially bad splits and disallow them in the recursive splitting procedure.

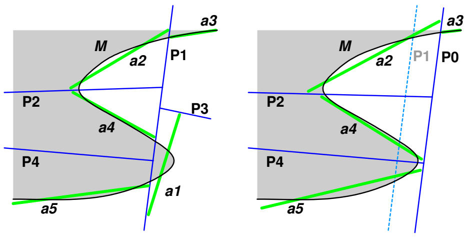

Figure 5: Left: the splitting plane (thin line P1) chops off a 'cap' in the low-right subspace leaf cell enclosed by P1 and P3 whose linear approximation (thick line a1) is not proper although its approximation error is within the tolerance; Right: After the translation of that splitting plane to the parallel one (thin line P0), the 'cap' can be avoided.

We extend the elegant 2D Hough transform [3] into 3D. Given a plane $P(x, y, z) = x\,\cos(\theta)\,\cos(\phi) + y\,\sin(\theta)\,\cos(\phi) + z\,\sin(\phi) - c = 0$ in Euclidean space, we can transform it into Hough space where it is represented by a single point with coordinates (θ, ϕ, c). Hence, a collection of planes in Euclidean space corresponds to a collection of points in Hough space.

Let P be a vertex of the mesh surface M and N its normal vector. According to the above discussion we want to disallow all splits that cut through the ε-vicinity of P in *almost tangential* direction where ε is the prescribed error tolerance. First we compute the points P^+ and P^- by shifting P in positive and negative normal direction by ε. All almost tangential planes are characterized by the fact that their first two Hough coordinates (θ, ϕ) are within some interval

$$[\theta_i - \nu, \theta_i + \nu] \times [\phi_i - \nu, \phi_i + \nu] \qquad (1)$$

where (θ_i, ϕ_i) are defined by the normal vector N and ν is some directional tolerance set by the user. In our implementation we usually set this tolerance to $\nu = \pi/12$. If we fix a certain normal vector and consider all parallel planes while we are moving from P^- to P^+ then the third coordinate c varies from $c^-(\theta, \phi)$ to $c^+(\theta, \phi)$. As a consequence we can describe the set of potentially bad splitting planes in the vicinity of the vertex P by the Hough volume enclosed between the surfaces $c^-(\theta, \phi)$ and $c^+(\theta, \phi)$ as (θ, ϕ) varies over the interval (1). The union of all these volumes for every vertex P of the mesh M defines the set of bad splitting planes that should be avoided in the recursive procedure (cf. Fig. 6).

We discretize the Hough space and then scan convert the above volume. For the discretization we do not need a very high resolution in θ and ϕ direction. Since θ runs over the interval $[0, 2\pi]$ while ϕ only runs over the interval $[0, \pi/2]$ we usually set the corresponding resolution to 200×50. The resolution in c direction is more critical since if it is chosen too low then the rasterization of the Hough volumes will leave hardly any empty cells. Empty cells, however, are used later to indicate that a certain splitting plane is not forbidden. In our implementation we use a resolution of 500.

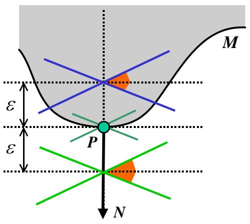

Figure 6: Computing the Hough map (2D).

During the recursive procedure, given a certain splitting plane, we first check if the corresponding raster cell in the discretized Hough map is empty. If not we cannot use the selected plane, we have to look for an alternative that is as close as possible. We do this by searching in the Hough map in the c direction for nearest empty cell. In Euclidean space this corresponds to parallely moving the splitting plane in normal direction. In case we cannot find an empty cell for the current (θ, ϕ) coordinates, we simply fall back to strategy B.

Note that for every raster cell of the Hough map, we only need 1 bit to indicate if it is occupied or not. Given the above resolution for the 3D Hough map, we only need 610KB which does not cause significant memory overhead for the algorithm.

4.3 Final linear approximation correction

In Section 4.1 we found that the linear approximation in each cell is only valid over the projection of the corresponding mesh patch. In some rare cases, it may happen that a cell is unnecessarily extended such that this validity condition is not satisfied (cf. Fig. 7). In this case some extra cell refinement is necessary to prune the linear approximation properly. This is done as a post process after the recursive BSP generation.

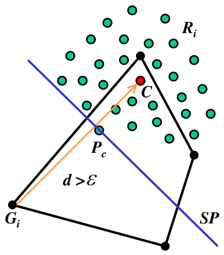

Figure 7: The mesh vertices are projected to the approximation plane. If some corner G_i is too far away from projected points, the plane SP is used to split the cell.

For every leaf node we check the validity of the linear approximation by projecting the vertices of the corresponding mesh patch into the zero-plane. We denote these projected points by $\{R_i\}$ and let C be their center of gravity. We also compute the zero-plane polygon by intersecting the zero-plane with all splitting planes that define the current cell. Let $\{G_i\}$ be the corners of this polygon. For every corner G_i we find the nearest point P_c in $\{R_i\}$ in the direction of $N_i = C - G_i$ and compute their distance d. If it is larger than the prescribed tolerance ε, we split the current cell along a plane that is defined by P_c and the normal vector N_i such that all mesh vertices are lying on one side with a minimum distance of ε. This generates an empty cell but this cell is guaranteed to lie outside the error bound ε as well as a non-empty cell(cf. Fig. 7). This procedure will be repeated in that non-empty cell until the distance validity is guaranteed.

4.4 Discussions

The availability of several splitting plane selection strategies allows the user to adjust the trade-off between processing speed and approximation quality. Figure 8, and Table 1 show examples of how effective the medial axis (MA) heuristics is in reducing the number of cells in the space partition and how much computing time is spent to achieve this. Also these initial computation effort will be amortized since once generated, the distance fields can be used in many applications.

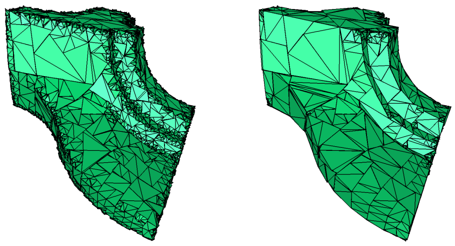

Figure 8: The zero-sets of the piecewise linear distance fields to the Fan model generated without (left) and with (right) MA information (strategy B).

strategy	time	BSP tree		average
	(s)	inner node	linear func.	error
A	8	1922	1853	0.01
A+B	240	641	605	0.009

Table 1: The performance of two piecewise linear distance fields with same maximum error tolerance 0.03. The original model has 101,136 triangles and a bounding box diagonal length 7.6.

5 Results and applications

All the experiments have been performed on a commodity PC with P4 2.8 GHz CPU. If not specified otherwise, error tolerances are given as a percentage of the model's bounding box main diagonal length.

The performance of our algorithm is summarized in Table 2. The major parts are used to compute the sample points on the medial axis and to create the discretized Hough maps. Some typical zero-sets of the linear distance field approximations are shown in Fig. 1, 8 and 10.

Table 3 and Fig. 10 show various levels of piecewise linear distance fields to the *Max* model which are computed for different error tolerances. The storage of each node has been calculated as follows: inner BSP nodes need 14 Bytes which stores a splitting plane in 4 Bytes (normals and offsets quantized both in 2 Bytes), two children pointers in 8 Bytes and another 2 Bytes tagging the leaf status of its two children; A leaf node with linear function needs 4 Bytes to store that approximation plane. The average depth indicates the average distance evaluation time and it is computed for all leaf cells that are not empty. For a leaf cell i with a depth d_i, we clip the linear function against the cell boundary and compute the area A_i of the resulting convex polygon. Then the average depth of a BSP tree is depicted as $\Sigma(A_i d_i)/\Sigma A_i$ and this depth value corresponds to the expected path length that a distance query traverses to find the proper leaf cell.

model	faces	ε (%)	time (min.)	
	(input)		A+B	A+B+C
Bunny	69,656	0.1	1.6	11.1
Bust	61,388	0.06	1.4	11.7
Fan	101,136	0.4	4.0	7.3
Horse	96,962	0.2	3.0	8.3
Max	99,999	0.5	3.2	6.5

Table 2: The running-times of the algorithm on diverse models with different splitting selection strategies.

ε (%)	averg. ε (%)	max. depth	averg. depth	inner cells	linear func.	storage (KB)
1	0.6	14	8.7	263	228	4.5
0.5	0.3	16	9.9	601	507	10.2
0.25	0.15	24	11.3	1545	1222	25.9
0.06	0.04	24	13.6	6468	6171	112.5
0.03	0.02	24	14.7	13908	13617	243.3

Table 3: The statistics of piecewise linear distance fields to the *Max* model when decreasing the error tolerance.

The above results show that our piecewise linear distance field approximation is compact and adapts very well to the shape of the underlying surface. We also have exploited this representation in various applications by providing fast access to the distance estimates for a given model: **Error bounded** mesh processing can be easily and efficiently implemented. For decimation (cf. Fig 9), we have similar results as the latest error-bounded method, Permission Grids [27], while their grids need over 100 times more space than our compact representation. Global error control is achieved by checking every atomic operator in the pre-computed approximation. It is natural to extend this to error bounded mesh smoothing. **Surface extraction** using [18, 16] can be applied to our representation by evaluating the distance function on a uniform grid (cf. Fig 10). **Level of detail** models is a straightforward extension of our hierarchical representation (cf. Fig 10) and volumetric **CSG operators** can also be applied.

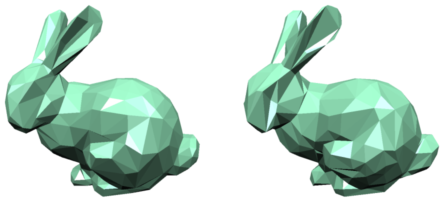

Figure 9: The Bunny decimated by a QEM [12] based algorithm with our distance field approximation to guarantee a global error bound of 1% (left, 700 triangles, 7.1 sec.) and the standard QEM algorithm without error control (right, 700 triangles, 2 sec.). The corresponding Hausdorff errors are 0.93% and 1.5% respectively. [27] simplified to 687 triangles with similar running time.

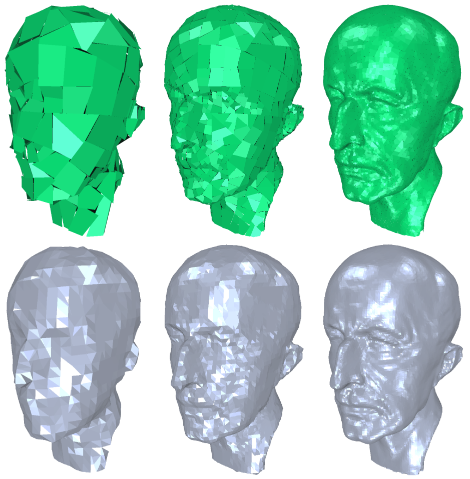

Figure 10: In the top row, from left to right are the zero-sets of the piecewise linear distance fields to *Max* model with error tolerance 1%, 0.25% and 0.03%. The bottom row are meshes extracted from the corresponding distance fields with 4004, 22324, 90092 triangles respectively (cf. Table 3).

6 Conclusions and future work

We proposed a new construction for the approximation of a signed distance function by a piecewise linear function over a binary space partition. We explained several splitting plane selection strategies that promote good splits and prevent bad ones. We show the efficiency of this generic geometry representation with various results and some beneficial geometry processing applications.

There are many directions for the development of new algorithms based on this representation. So far we used a uniform resampling step to extract surfaces or to apply CSG operations. In principle it should be possible to avoid this resampling and extract explicit surface information directly from the BSP. The major challenge for this is to to extract the neighborhood relation between the BSP cells.

References

[1] N. Amenta, M. Bern, M. Kamvysselis. A New Voronoi-Based Surface Reconstruction Algorithm. *Proceedings of ACM SIGGRAPH 1998*, 425–421, 1998.

[2] N. Amenta, S. Choi, R. Kolluri. The Power Crust, Unions of Balls, and the Medial Axis Transform. *Computational Geometry: Theory and Application*, 19(2-3):127–153, 2001.

[3] D. Ballard. Generalized Hough Transforms to Detect Arbitrary Patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(2):111–122, 1981.

[4] S. Bischoff, L. Kobbelt. Ellipsoid Decomposition of 3D-models. *3DPVT Proceedings*, 480–488, 2002.

[5] J. Bloomenthal, etc. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers Inc., 1997.

[6] H. Blum. A Transformation for Extracting New Descriptor of Shape. *Models for the Perception of Speech and Visual Form*, 362–380, MIT Press, 1967.

[7] M. Botsch, A. Wiratanaya, L. Kobbelt. Efficient High Quality Rendering of Point Sampled Geometry. *Eurographics Workshop on Rendering*, 2002.

[8] M. Chen, A. Kaufman, R. Yagel. *Volume Graphics*. Springer-Verlag, 2000.

[9] J. Cohen, A. Varshney, D. Manocha, G. Turk, etc. Simplification Envelopes. *Proceedings of ACM SIGGRAPH 1996*, 119–128, 1996.

[10] B. Curless, M. Levoy. A Volumetric Method for Building Complex Models from Range Images. *Proceedings of ACM SIGGRAPH 1996*, 303–312, 1996.

[11] S. Frisken, R. Perry, A. Rockwood, T. Jones. Adaptively Sampled Distance Fields: A general representation of shape for computer graphics. *Proceedings of ACM SIGGRAPH 2000*, 249–254, 2000.

[12] M. Garland, P. Heckbert. Surface Simplification Using Quadric Error Metrics. *Proceedings of ACM SIGGRAPH 1997*, 209–216, 1997.

[13] P. Giblin, B. Kimia. A Formal Classification of 3D Medial Axis Points and Their Local Geometry. *Proceedings of Computer Vision and Pattern Recognition(CVPR)*, 2000.

[14] T. Ju, F. Lasasso, S. Schaefer, J. Warren. Dual Contouring of Hermite Data. *ACM Transaction on Graphics (SIGGRAPH 2002 Proc.)*, 21(3):339–346, 2002.

[15] A. Kaufman. Efficient Algorithms for 3D Scan-conversion of Parametric Curves, Surfaces, and Volumes. *Computer Graphics (Proceedings of ACM SIGGRAPH 87)*, 21(4):171–179, July 1987.

[16] L. Kobbelt, M. Botsch, U. Schwanecke, H. Seidel. Feature Sensitive Surface Extraction from Volume Data. *Proceedings of ACM SIGGRAPH 2001*, 57–66, 2001.

[17] D. Laney, M. Duchaineau, N. Max. A Selective Refinement Approach for Computing the Distance Functions of Curves. *Eurographics - IEEE TCVG Symposium on Visualization Proceedings*, May 2001.

[18] W. Lorensen, H. Cline. Marching Cubes: A High Resolution 3d Surface Construction Algorithm. *Computer Graphics (Proceedings of ACM SIGGRAPH 87)*, 21(4):163–169, July 1987.

[19] K. Nguyen, D. Saupe. Rapid High Quality Compression of Volume Data for Visualization. *Computer Graphics Forum (Eurographics 2001 Proc.)*, 20(3), 2001.

[20] F. Nooruddin, G. Turk. Simplification and Repair of Polygonal Models Using Volumetric Techniques. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):191–205, 2003.

[21] S. Osher, R. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer-Verlag, 2003.

[22] H. Samet. *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, 1994.

[23] G. Taubin, R. Ronfard. Implicit Simplicial Models for Adaptive Curve Reconstruction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(3):311–325, March 1996.

[24] L. Velho, J. Gomez. Approximate Conversion of Parametric to Implicit Surfaces. *Computer Graphics Forum (Eurographics'96 Proc.)*, 15(3):327–337, 1996.

[25] J. Wilhelms, A. Gelder. Octrees for Fast Isosurface Generation. *Computer Graphics*, 24(5):57–62, 1990.

[26] G. Yngve, G. Turk. Robust Creation of Implicit Surfaces from Polygonal Meshes. *IEEE Transactions on Visualization and Computer Graphics*, 8(4):346–359, 2002.

[27] S. Zelinka, M. Garland. Permission Grids: Practical, Error-Bounded Simplification. *ACM Transactions on Graphics*, 21(2), April 2002.