Reconstruction and simplification of surfaces from contours

Reinhard Klein, Andreas Schilling and Wolfgang Straßer
Wilhelm-Schickard-Institut, GRIS
Universität Tübingen
Auf der Morgenstelle 10 / C9
72076 Tübingen, Germany
E-mail: {andreas|reinhard|strasser}@gris.uni-tuebingen.de
http://www.gris.uni-tuebingen.de

Abstract

In this paper we consider the problem of reconstructing triangular surfaces from given contours. An algorithm solving this problem has to decide which contours of two successive slices should be connected by the surface (*branching problem*), and, given that, which vertices of the assigned contours should be connected for the triangular mesh (*correspondence problem*).

We present a new approach that solves both tasks in an elegant way. The main idea is to employ discrete distance fields enhanced with correspondence information. This allows us not only to connect vertices from successive slices in a reasonable way but also to solve the branching problem by creating intermediate contours where adjacent contours differ too much. Last but not least we show how the 2D-distance fields used in the reconstruction step can be converted to a 3D-distance field that can be advantageously exploited for distance calculations during a subsequent simplification step.

Keywords: distance field, reconstruction from contours, mesh simplification.

1 Introduction and previous work

A large number of publications treat the problem of connecting contours, see [15, 8, 3, 22, 2, 24, 1, 27] and [23, 26, 30] for overviews. All these algorithms have to decide which vertices in the neighboring contour must be connected with a given vertex to form triangles (*correspondence problem*). Unfortunately, this problem cannot be answered uniquely (especially in cases with large differences between neighboring contours) and suffers from missing information caused by the undersampling of the original data. Furthermore, if two successive slices are given where M contours correspond to N contours ($N \neq M, N, M > 0$), how should they be connected (*branching problem*)?

One class of algorithms tries to overcome these problems by introducing intermediate contours [22, 27]. The idea introduced by Levin [22] is to calculate in each slice a signed distance field that assigns each point its closest distance to the contour. With this distance field the contours can be regarded as isocurves with isovalue zero. These distance fields are interpolated in $z-$ direction in order to obtain new fields in between the given ones. In this way it is possible to get intermediate contours at arbitrary positions between the original ones again defined as isocurves with isovalue zero in the interpolated fields. For example, a distance field for a new slice located halfway between two neighboring slices with given distance fields is obtained by adding the two distance fields pixelwise and dividing by two.

Later distance fields were used by several authors in a similar context for different purposes [28, 10]. The principle of Levin's distance field interpolation can be

used to directly construct the isosurface by simply running a marching cubes algorithm on the distance fields [14, 20, 6].

In a recent paper [7] Cong and Parvin use distance fields for the reconstruction of surfaces from dense contours with the "Equal Importance Criterion". Their solution is exactly identical to distance field interpolation as in [14, 20]. They also present a method for achieving smooth surfaces with a variational approach.

The use of the marching cubes algorithm leads to very large numbers of triangles. This is one of the reasons for developing algorithms for the direct triangulation of the contours.

Oliva et al. [27] calculate a so-called Angular Bisector Network (ABN) between the corresponding contours of two successive slices. The ABN is an approximation of the Voronoi Diagram between the polygon segments of the original contours. To each edge of the contours there is one corresponding cell of the ABN that (approximately) contains the points closest to this edge. The proximity information is used to identify the cells that can be triangulated in a straightforward way. Where such a straightforward triangulation is impossible, an intermediate contour consisting of the border between cells belonging to contours in different slices is inserted. If needed this procedure is continued recursively.

The intermediate contours of the first recursion step in the algorithm of Oliva et al. [27] can be interpreted as an approximation of Levin's implicit contours in a distance field located in the middle between two successive slices.

This interpretation leads to the idea that combines the two approaches. Our algorithm is based on this idea. Instead of the complicated calculation of the angular bisector network, simple discrete distance fields are used to get intermediate contours and the needed correspondences. For a fast computation of the needed distance fields and correspondences we use the z-buffer of standard graphics hardware.

In addition to their use in the reconstruction algorithm the distance fields can be exploited to efficiently measure a 3D distance in a subsequent simplification algorithm.

A large number of simplification algorithms for triangle meshes have been developed, but only a part of them can guarantee a certain geometric approximation error between simplified and original mesh [19, 5]. However, this error must be known to guarantee a certain quality of the rendered images of the simplified model. Unfortunately, the simplification algorithms with this feature are very slow (see [4] for a comparison) or produce over-estimations of the error [29, 9] that make the results not suitable for multiresolution models aiming for view-dependant refinement. By using a 3D distance field the time critical step of calculating the distance between approximating and original mesh can be accelerated.

In the next section we give an overview of the new algorithm. Section 3 contains our new method to calculate the distance field and the correspondences efficiently. Section 4 contains the reconstruction part of the algorithm and sections 5 and 6 the simplification part.

2 Overview of the algorithm

In principle the algorithm consists of two main steps: the reconstruction of the surface and the simplification of this surface. However both steps are closely related since the distance field is used for the reconstruction as well as for the simplification step. The outline of the algorithm is as follows:

2.1 Reconstruction

2.1.1 Simplification of contours

Each contour is simplified up to a certain user defined approximation error. In this way for each edge of a simplified contour the maximum geometric approximation error between the edge itself and the corresponding part of the original contour is known.

2.1.2 Computation of the 2D-distance fields including correspondences

The 2D-distance field is calculated on a rectangular grid. In the following the grid cells are considered as the pixels of an image. Using a standard distance transform to compute the distance field in image space is not suitable for two reasons: first we need the exact distance values to the original contours and not to a contour that is defined in the pixel grid. Furthermore, in our algorithm we need not only the distance but also the point on the contour to which this distance is measured. Therefore, we developed

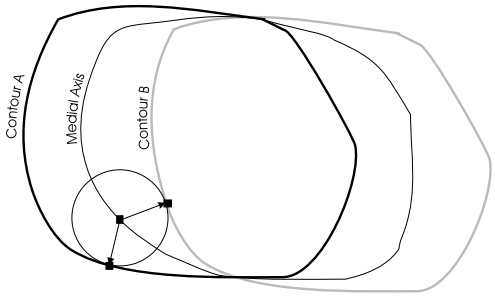

Figure 1: The arrows from a pixel on the medial axis to the two contours A and B indicate the correspondences to the closest pixels on the contours.

a new algorithm that exploits standard graphics hardware to compute the 2D-distance field very fast.

2.1.3 The medial axes

After this step the medial axis is available as the zero-set of the sum of distance fields in neighboring slices. It is easy to track the medial axis in a discrete way as it is on the border between pixels with negative and positive sum. In addition for each pixel on the medial axis, we get the closest pixels on both contours, see Fig. 1.

2.1.4 Surface triangulation

After the computation of the distance fields and the correspondences above, a triangulation of the resulting surface is computed. Here the main idea is to trace the medial axes and use the correspondences to pixels on the two neighboring contours to connect the vertices by edges.

2.2 Simplification

2.2.1 Edge collapse

In the simplification algorithm simple edge collapse operations are performed [12]. The order of the edge collapse operations is determined by a priority queue based on the error that would be introduced if the edge was collapsed.

2.2.2 The 3D-distance field

To allow fast error measurement during the simplification algorithm a 3D distance field is computed from the existing 2D-distance fields.

2.2.3 Error measurement

To measure the error introduced by an edge collapse, the 3D distance field is exploited. The newly created triangles are appropriately scan converted into the voxel-cube which is constituted by the slices. The resulting samples in the distance field contain are read out. They deliver the deviation between the triangle and the original surface.

2.3 Simplification of contours

To simplify the contours a modified version of the Douglas-Peucker-algorithm is used [11]. This algorithm starts with one arbitrary vertex of the original contour. In each subsequent step a further point of the original contour with greatest distance to the current polygon (or at the beginning to the initial vertex) is inserted. To find this point with greatest distance a convex hull technique is used that reduces the complexity of the algorithm from $O(n^2)$ to $O(n \log n)$ [11].

3 The distance field

Since our approach is based on the discrete distance field we briefly review this method.

Let $z_0, \ldots, z_n$ be the coordinates of the levels under consideration, Ω be the 3D object and

$$\Omega_i = \{(x, y) | (x, y, z_i) \in \Omega\}$$

the corresponding cross sections. Then the distance fields at the levels $z_0, \ldots, z_n$ are defined by

$$D_i(x, y) = \begin{cases} -dist((x, y), \partial\Omega_i) & \text{if} (x, y) \in \Omega_i \\ dist((x, y), \partial\Omega_i) & \text{otherwise} \end{cases}$$

where $\partial\Omega_i$ denotes the boundary of Ω_i which is described by the contours and $dist$ denotes the Euclidean distance within each slice. Now an interpolation of the distance values in z-direction is used to find intermediate contours (where the interpolated distance is zero).

3.1 The computation of the distance field

For our purpose the application of a simple distance transform to compute the distance field and the medial axis is not sufficient, since we also want to know for each pixel which is (are) the closest pixels on the border and since we need the accurate distance to the actual and not to a rasterized border. Graphics hardware can be used for this purpose while rendering the distance function as polygons. The distance field is then recorded in the z-buffer. The idea to use graphics hardware for distance calculations was introduced by Müller in the context of collision detection [25]. Recently, Hoff et al. presented an algorithm based on the same idea to calculate voronoi diagrams on the Siggraph conference [13]. However, the problem using graphics hardware is that only polygonal approximations of the necessary cones are available and used. Unfortunately, without knowing apriori the minimal distance between all vertices and lines, the needed approximation error cannot be calculated. We noticed that even using very small bounds on the approximation error (slowing down the algorithm) still produced wrong results. To overcome this problem, our new algorithm uses the sweep surfaces described below.

Let $p_1, \ldots, p_n$ be the polygon describing the contour. Note that the p_i's are given in floating point precision and normally do not necessarily lie on pixel centers of our image. For the line-segments of the polygon we define two quadrilaterals in such a way that the z-value on the quadrilaterals define the Euclidean distance to the line-segment. The corresponding surface for points, where again the z-value represents the Euclidean distance from the point is a cone with its apex at the point itself. We approximate this cone with several triangles. Note that, depending on the angle betwenn two consecutive line-segments, only a small part of the cone needs to be approximated. One or two triangles would be sufficient. The whole process is shown in Figure 3. Now, all objects defined in this way are rendered into the z-buffer. After rendering the z-buffer contains the correct distance values and is read out.

One problem that must be solved in this context is that the exact cones are approximated by triangles. In this way the z-values generated by these triangles are not the exact Euclidean distances. Therefore, it can happen that "voronoi regions" which are computed in this way are not connected. This is illustrated in Figure 2. This means that the z-buffer may establish correspondences to wrong contours. One way to overcome these difficulties is to calculate apriori the needed accuracies for the approximation of the cones [13]. However, in our application we must

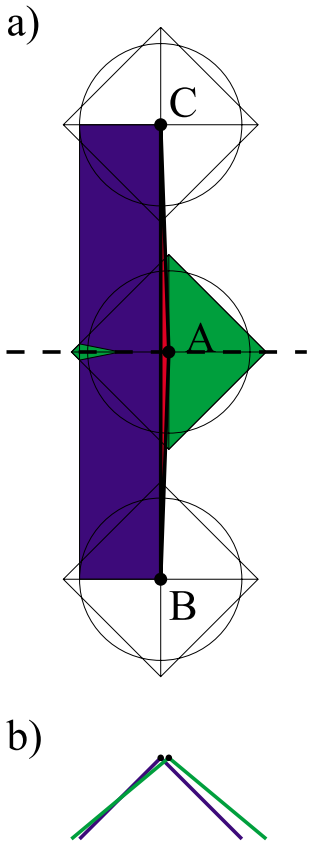

Figure 2: Disconnected "voronoi region": Consider the red triangle Δ(ABC): (a) The distances around vertices A, B and C are approximated using pyramids (the one around A is shown in green, only partly colored). The pyramid is steeper than 45° in the middle of the sides, but not as steep as 45° at the edges. The distances to the line segment (BC) are calculated using the roof with an angle of exactly 45° in z-direction (shown in blue, only colored on the left side). As a result, points in the green area on the left side wrongly seem to be closer to A than to (BC). (b) shows the sideview of the above configuration (intersection at the dashed line). As can be seen, the roof is intersected by the pyramid.

guarantee that at least within the region of the screen the voronoi regions will not be seperated into unconnected regions. The needed accuracy depends on the smallest distance between a vertex and edges not incident to this vertex, see again Fig. 2. Unfortunately the computation of this distance is so expensive that the advantage of the hardware support would be lost.

Therefore, in our solution we sweep the pyramids along the polygon edge and use the resulting surface as the distance function. Fortunately, this sweep surface is simply a transformed rectangle, see Figure 3. Note that our approximation of the Euclidean distance only deforms the exact voronoi regions, but does not change their topology, even if we use a very crude approximation of the cones.

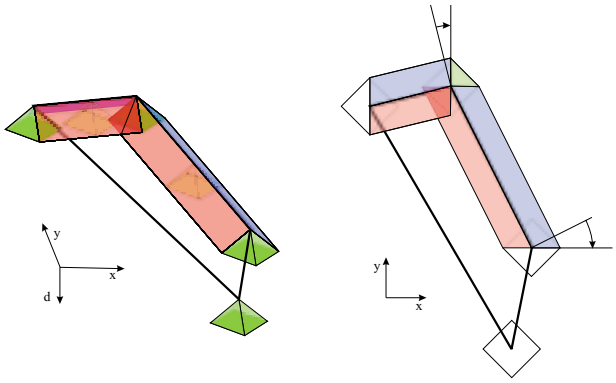

Figure 3: A crude approximation of a cone by a pyramid (L_1 distance [21] instead of Euclidean distance) is swept along a line segment. On the right, the top view is shown. The sweep surface between the pyramids on the edges consists of two deformed rectangles. The deformation is determined by the angle α. The rectangles inside the polygon are drawn in red, the ones outside in blue. For demonstration the pyramids are colored green. In the algorithm the triangles of the pyramid must also be drawn in red or blue dependent on their location in the inner or outer of the contour. The z-values of the drawing primitives at vertices on the contour are set to zero, at all other vertices to d. Inaccuracies caused by the approximation of the cone by triangles are smaller than 1% if the inner angle of the triangles approximating the cones is less than $16.2°$. This means that a cone must be approximated by at least 24 triangles (of which only a few must actually be drawn). To avoid confusion, in the figure the drawing primitives are clipped at a small distance.

3.2 The correspondences

To get correspondences, for each line-segment and each vertex the color of the drawn primitive is incremented. In this way the color represents the number of the closest line-segment or vertex. In order to get the sign of the distance, one bit of the color is used to code the side of the primitive with respect to the oriented polygon, see Figure 4.

3.3 The medial axis

After the computation of the distance field it is easy to find the medial axes between contours of neighboring slices. For this purpose the two distance fields are added. The resulting image contains connected regions of positive (including zero) or negative values, respectively, see Figure 4. Those points where both distance values are zero indicate intersection points of the two contours in their projections. The border between these areas constitutes the medial axis. This border can be extracted with an algorithm which scans the image for a change of the sign of the distance values and then immediately traces and marks the pixels of the encountered medial axes. For each new medial axis a pointer to one of its pixels is stored.

4 Triangulation of the surface

Using the medial axis and the known correspondences to the closest points on the contours the triangulation of the resulting surface is straightforward.

The basic step is to trace the medial axes, which are closed polygons. The tracing starts at an arbitrary pixel of the medial axis. Let M be the medial axis and $M_i, i = 1, \ldots, m$ its pixels. Let P and Q denote the contours corresponding to M and let P_i, Q_k be the vertices of the polygons P and Q, respectively.

When the pixels of the medial axis are traced in a sequential way, for each of its pixels the numbers i, k of the two corresponding contour edges P_iP_{i+1} and Q_kQ_{k+1} are recorded. If, while tracing the medial axis the corresponding contour segment changes, e.g. from edge i to

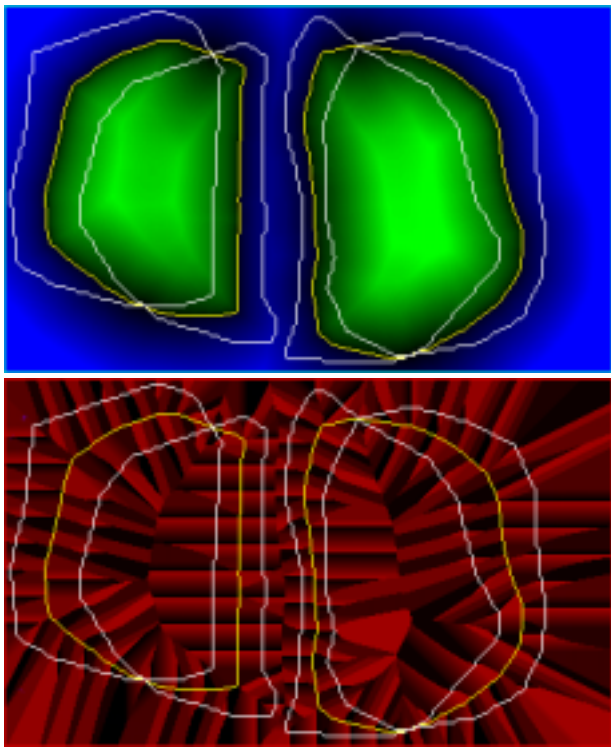

Figure 4: Adding the two distance fields of consecutive slices delivers the medial axes at pixels where the sign changes. Each image contains two contours. The color in the left image encodes the distance to the contours. The sign is indicated by the green and blue coloring. The red-values in the right picture indicate the correspondences to the different contour pixels (to make the correspondences better visible, only the lower bits of the segment numbers are shown). The medial axis itself is shown in yellow.

edge $i+1$, the vertex P_{i+1} is put onto a stack S[1]. At the beginning we trace the medial axis until in the sequence of recorded vertices, a vertex on contour P is followed by a vertex on contour Q or vice versa. For simplicity we assume for the following that we had a change from P to Q. Then these two vertices are connected by an edge and all other vertices are removed from the stack. Now

[1]The handling of the special (simpler) case, where P_i and Q_k correspond to the same pixel on the medial axis is not described here, but is straightforward.

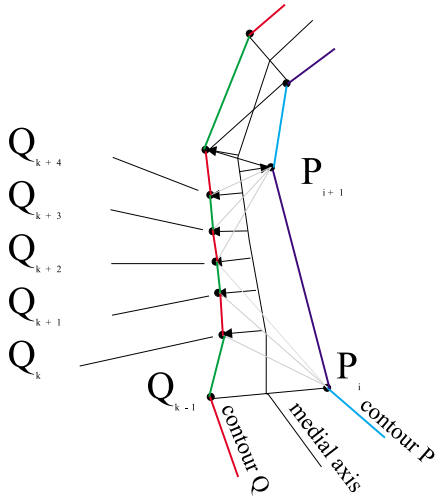

Figure 5: Vertices 'Q_k through Q_{k+4} are connected to P_i or P_{i+1}, whichever is closer.

the tracing proceeds until again a change in the recorded vertices, now from contour Q to P occurs. Again, these two vertices are immediately connected. The remaining vertices on the stack form a sequence of the form $P_iQ_k, \ldots, Q_{k+n}, P_{i+1}$, with one or more vertices from contour Q between two vertices of contour P. The polygon defined by these vertices is triangulated as follows: We connect all vertices on Q that are closer to P_i than to P_{i+1} with P_i and the others with P_{i+1}, see Fig. 5.

Until now we have assumed that we always found consecutive edges while tracing the contours, but sometimes this may not happen, see Fig. 6. In these cases new vertices are inserted into the contours. Let us assume that the correspondence changes as shown in Fig. 6. Then we insert the vertices $P_{i,\alpha}, P_{m,\beta}, Q_{j\delta}$ into the contours P and Q, respectively and we introduce one of the two pixels on the medial axis M_i or M_{i+1} (we choose M_i). The corresponding stripe is triangulated as shown in Fig. 6 and the vertices $P_{m,\beta}$ and $Q_{j,\delta}$ are put on the stack. Now the algorithm can proceed as described above until all medial axes have been processed.

After processing all medial axes there remain parts of contours that had no correspondence to a medial axis and therefore are not fully integrated in the triangulation. These areas could be triangulated, resulting in flat areas

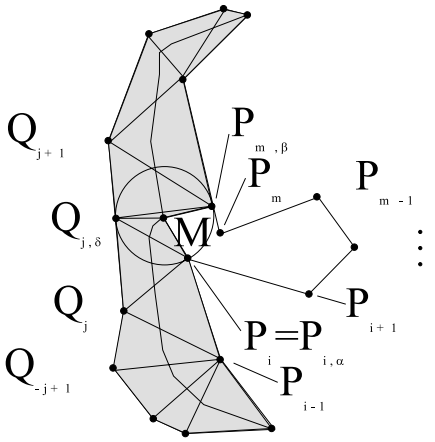

Figure 6: When edges without correspondence on the medial axis are present, additional vertices are introduced. Then the dark gray area can be triangulated. The white area remains to be handled later.

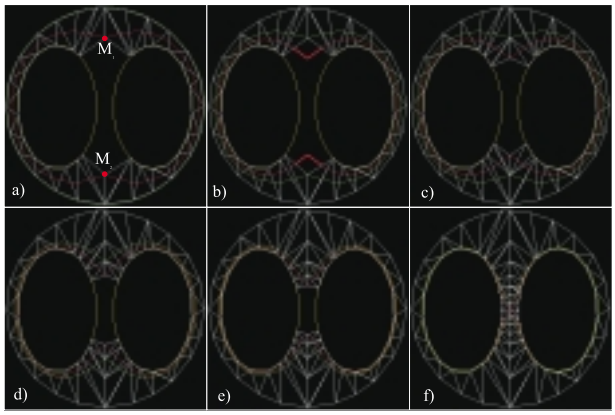

Figure 7: Recursive triangulation algorithm applied to example slices. a) In the first step of the algorithm the red medial axis is traced. Where correspondences from the white as well as from the yellow contour to midline exist, the triangulation is performed as described in the text. As the correspondence on the yellow contour jumps more than one segment a new vertex M on the medial axis is inserted into the triangulation. b) To triangulate the remainig area a new medial axis between the old one (now shown in green) and the yellow contour is computed using the distance field algorithm. Now the part of the medial axis that is contained in the area not yet triangulated is traced. c) The triangulation algorithm is applied again. Note that only the newly introduced vertex M on the original medial axis is included into the triangulation. d)-f) The process described in b) and c) is applied recursively. After reaching a certain recursion depth the polygon containing the remain area is triangulated arbitrarily.

parallel to the slices (with the exception of the vertex on the medial axis that is situated in the middle of the two slices) [1]. However, to improve the results the medial axis can be considered as a new contour (in between the original contours) and the described algorithm (including a new distance field and a new medial axis) is applied recursively, see Fig. 7.

Note that the results achieved with our simple and fast triangulation algorithm are similar to the ones achieved with the algorithm of Oliva et al. [27]. One of the most important differences with respect to the resulting surface is that without applying the recursion in our algorithm the medial axis is not included into the resulting triangulation and thus the number of triangles is reduced.

5 Review of multiresolution models

5.1 Generating the multiresolution model

The generation of a MRM of an object generally involves a sequence of local simplification operations like vertex removal, edge collapse, triangle collapse or vertex clustering. The sequence of local simplification operations defines a sequence of coarser and coarser approximations of the original model, the MRM. How this sequence is generated depends on the various simplification algorithms. In general a mesh simplification algorithm starts with the finest triangulation in 3D space approximating the original model. Then it simplifies the starting triangulation by clustering vertices, by collapsing edges or triangles or by removing vertices from the current triangulation and retriangulating the resulting holes. This is done until no further simplification step can be performed. In many algorithms the order in which the simplification steps are performed is determined by a priority queue. A cost function is evaluated for each possible simplification operation and the one with the lowest cost is performed. In general

the cost function represents the error (geometric distance) between original and simplified mesh.

5.2 Selective refinement of multiresolution models

If the inverse local simplification operations are known (e.g. vertex split as the inverse of edge collapse operation), we are able to refine a coarse approximation of the model by reversing the whole simplification process. However, if we want to perform only selective refinement we have to find a way to skip parts of the inverse simplification process and thereby change the sequence of refinement operations. Of course this is not arbitrarily possible (e.g. we cannot split a vertex which is not present in the current mesh). The relationship among different simplification steps define a hierarchy that can be described by a directed acyclic graph of modification operations or the associated triangles. Therefore, a general selective refinement algorithm starts with a crude approximation of the model and checks for each triangle if refinement is needed. If yes, the algorithm has to take care that all predecessor operations of the needed refinement operation have already been performed. The next section describes the measure that can be used to decide about the need of further refinement of a certain triangle.

6 Simplification

The most expensive part of the simplification algorithm is the evaluation of the cost function. In our case we want to guarantee a geometric distance between the original contours and the simplified model. In this section we describe, how the 2D-distance fields that we already computed can be exploited to get an upper bound for this geometric distance. The idea is, to generate a 3D-distance field from the 2D-field. This 3D-distance field enables us, to simply look up the distance from an arbitrary 3d-location to the original model. This mechanism is then used to determine the distance between simplified and original model.

According to our experience the quality of the resulting triangulation does mainly depend on the order of the simplification steps and not on the special topological operations like vertex removal, edge- or triangle collapse [18]. Therefore, we use a simple edge collapse technique, where no new vertices are introduced.

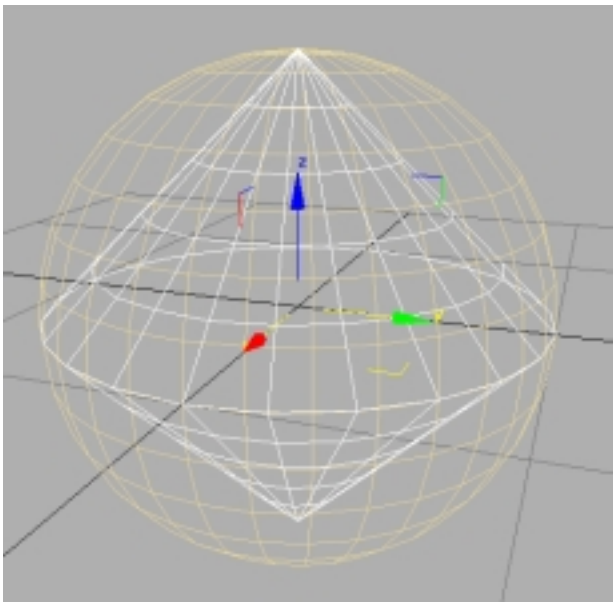

Figure 8: The unitsphere of our 3d-distance is a double cone. It is fully contained in the unit sphere of the Euclidean distance.

6.1 Calculating a 3D-distance field

The calculation of the 3D-distance field is computationally expensive, but the already available distance fields in the 2D slices can be used to greatly reduce the needed efforts. If a slight modification of the 3D distance is used we get an especially simple algorithm. Instead of using in 3D the Euclidean distance (L_2-norm, $||\cdot||_2$), we compose our new distance of the Euclidean distance in the 2D-slices and the distance in z-direction. Given two points $p_1 = (x_1, y_1, z_1)$ and $p_2 = (x_2, y_2, z_2)$ our distance is defined as follows

$$D_{3D}(p_1, p_2) = sqrt((x_1 - x_2)^2 + (y_1 - y_2)^2) + |z|.$$

The unit sphere of our distance function is shown in Fig. 8. Since the unit sphere of our distance function is fully contained in the unit sphere of the Euclidean distance this distance is a conservative estimate for the Euclidean 3D

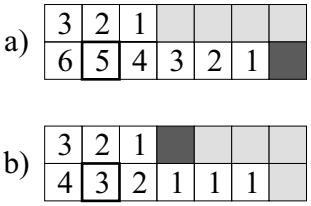

Figure 9: a) 2D-distance field. Each voxel contains the Euclidean distance to the grey object computed within the slice. b) 3D-Distance. E.g., the second voxel in the bottom row that originally had a 2d-distance of 5 gets a 3d-distance of 3 that is computed from the 2d-distance of the voxel above it (2) plus the z-distance of 1.

distance. An important point is that the isosurface defined by this 3D distance field remains the same as the isosurface defined by the 2D distance fields. Different distance measures, unit spheres and corresponding Voronoi diagrams are discussed in [16, 17].

As the 2D-distance is already present in the slices the only thing that remains to be computed is the z-component. If the 3D-distance of a point to the object is smaller than the already present 2D-distance the distance value has to be replaced by the new 3D-distance. An example is shown in Fig. 9.

The algorithm to compute the 3D-distance from the 2D-distances traverses the slices two times, the first time in positive z-direction and the second time in negative z-direction. The distance values of each slice are propagated to the next slice by adding (or subtracting) the distance s between two slices, measured in units of pixels. If the propagated distance is lower than the already present value, this value is overwritten with the propagated value.

If the sign of the distance changes between two neighboring voxels changes the surface of the object crosses between the two voxels. If the surface crosses between two slices our algorithm calculates the distance of the voxels in both slices to the surface by weighting the 2D-distances in an appropriate way: Consider two voxels in two consecutive slices separated by the surface. Let a be the positive 2D-distance value of voxel A and $-b$ the negative 2D-distance of voxel B. Then the 3D-distance of voxel A is given by $min(a, \frac{a}{a+b} * s)$ and the 3D-distance of voxel B by $-min(b, \frac{b}{a+b} * s)$, see Figure 10.

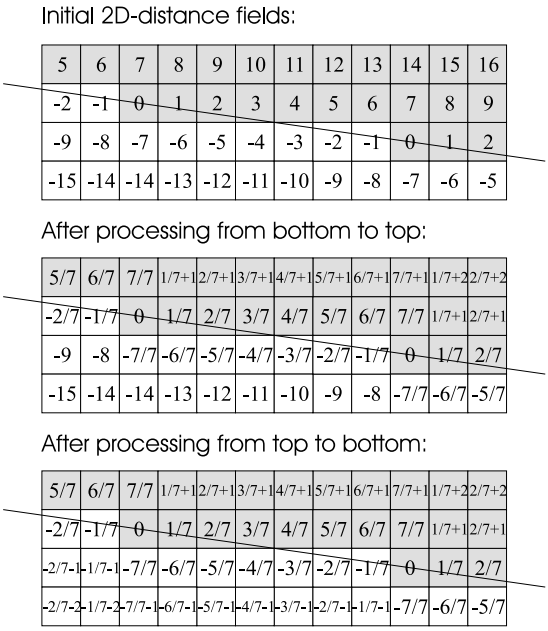

Figure 10: Calculating the 3D-distance field. The distance between neighboring slices is assumed to be 1 Pixel. The upper two pixels in the first column show the special situation described in the text. As a sign change occurs, the two pixel values are adjusted in such a way that the location of the interpolated zero remains on the contour (drawn from the upper left to the lower right). This is done applying the equation in the text, with $a = 5$ and $b = -2$.

The definition of our 3D-distance ensures that the distance value of a voxel has to be propagated only to the two neighboring voxels with the same x, y-coordinates. This makes the algorithm simple and fast.

6.2 Measuring the error

To measure the error between original and simplified surface model we use the fact that the 3D-distance field delivers automatically a set of envelopes of the original surface. Based on this observation the measurement of an error that would be introduced if an edge was collapsed can easily be performed in the following way: Let $\Delta = \Delta(p_j, p_k, p_l)$ be a triangle generated if the edge was

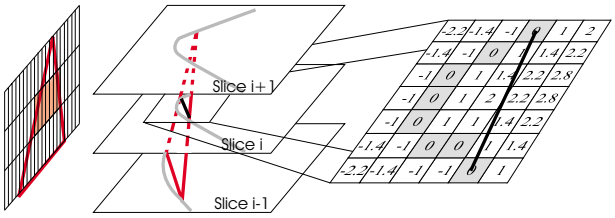

Figure 11: Measuring the approximation error by rendering the triangle on one face of the distance voxel cube and looking up the distances in the 3D-distance-field.

collapsed and let $n = (n_x, n_y, n_z)$ be its normal. The maximum $n_\xi = \max(n_x, n_y, n_z)$ defines the principal direction ξ for the triangle. Now, the idea is to scan convert the triangle into the 3D-distance field and to read out the corresponding distance values. This is done by rendering the triangle projected along its principal direction ξ, which is one of the axes x, y or z. The interpolated depth values of the triangle are then used to address the corresponding voxels in the 3D-distance field, see Figure 11.

If the z-value is decomposed into an integer part and a fractional part and the integer part is taken to determine the two closest distance values we can interpolate between them using the fractional part of the z-value. In this way the distance from the zero set of the distance field (the original surface) is determined with higher accuracy.

Using this way of distance calculation the drawback of the simplification envelopes algorithm [5] of having a fixed envelop and therefore not being able to build up a reasonable multiresolution model is avoided.

6.3 Acceleration of distance computation

For errors larger than one pixel in the plane defined by the principal direction of the triangle the read out of the distance values can be accelerated by skipping $\lfloor \epsilon - d - 1 \rfloor$ pixels, where d is the distance readout at the current position and ϵ is the already reached error between original and simplified model, since from pixel to pixel in image space the distance can grow at most by 1. Note that this technique can easily be adapted to unisotropically sampled data sets.

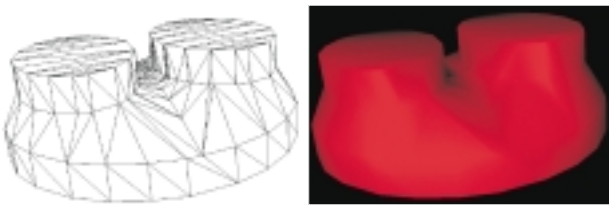

Figure 12: The reconstructed surface of Figure 7.

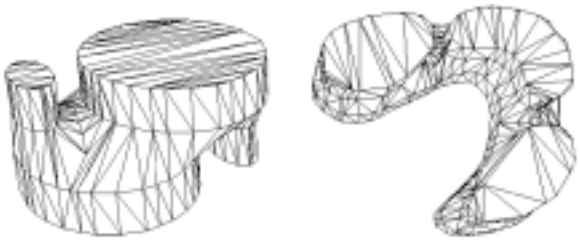

Figure 13: Two examples for branching, both triangulated with a recursion depth of four. Therefore between the branches at most four intermediate levels were introduced.

Figure 14: Reconstruction of a BMW from about 100 contour slices. The slices were generated by a laser range scanner. In the first step the scanner data was converted into polygons. In the second step, the polygons were simplified and then the actual reconstruction took place. In this example picture after the reconstruction step no further simplification was performed. Data by BMW AG München.

7 Conclusion and future work

The contribution to the problem of reconstruction from contours presented in this paper is twofold: On one hand

a distance field is used for a robust reconstruction algorithm. The medial axis defined by this distance field is used to solve the correspondence problem in a numerically stable way and delivers excellent triangulations even in otherwise problematic cases where the contours from consecutive slices have significantly different shapes. For the computation of the distance fields we have proposed an accurate algorithm that exploits widespread standard graphics hardware to be efficient. This algorithm can also advantageously be applied to other problems where discrete distance fields are needed.

On the other hand, the second big problem of current reconstruction algorithms, the huge number of resulting triangles, is solved with a new, very fast simplification algorithm that exploits the already calculated distance fields to guarantee a certain approximation error between the simplified surface models and the original contours. To achieve this we have proposed a new efficient method to measure 3D and not only 2D distances during simplification.

This guarantees that in each level of detail, the surface is approximated with a certain approximation error, since the simplified surfaces are within a certain envelope.

Our simplification algorithm is also an obvious choice for the simplification of the huge amount of data produced by the marching cubes algorithm. We investigate the possibilities of this technique in the context of a generally applicable mesh simplification algorithm. We suppose that in this way it will be possible to speed up known algorithms.

8 Acknowledgement

We would like to thank Holger Müller for the programming efforts which were necessary to gain the described results. Special thanks go to Myung-Soo Kim for his advice for improving the paper.

References

[1] G. Barequet and M. Sharir. Piecewise-linear interpolation between polygonal slices. *Computer Vision and Image Understanding: CVIU*, 63(2):251–272, Mar. 1996.

[2] J.-D. Boissonnat. Shape reconstruction from planar cross sections. *Computer Vision, Graphics, and Image Processing*, 44(1):1–29, Oct. 1988.

[3] H. N. Christiansen and T. W. Sederberg. Conversion of complex contour line definitions into polygonal element mosaics. *Computer Graphics*, 12(3):187–192, Aug. 1978.

[4] P. Cignoni, C. Montani, and R. Scopigno. A comparison of mesh simplification algorithms. *Computers & Graphics*, 22, 1998.

[5] J. Cohen, A. Varshney, D. Manocha, and G. Turk. Simplification envelopes. *Computer Graphics*, 30(Annual Conference Series):119–128, 1996.

[6] D. Cohen-Or, D. Levin, and A. Solomovici. Contour blending using warp-guided distance field interpolation. In *IEEE Visualization '96*. IEEE, Oct. 1996. ISBN 0-89791-864-9.

[7] G. Cong and B. Parvin. An algebraic solution to surface recovery from cross-sectional contours. *Graphical Models and Image Processing*, 61(4):222–243, July 1999.

[8] H. Fuchs, Z. Kedmen, and S. Uselton. Optimal surface resconstruction from planar contours. *Communications of the ACM*, 20(10):693–702, 1977.

[9] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In T. Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 209–216. ACM SIGGRAPH, Addison Wesley, Aug. 1997. ISBN 0-89791-896-7.

[10] G. T. Herman, J. Zheng, and C. A. Bucholtz. Shape-based interpolation. *IEEE Computer Graphics and Applications*, 12(3):69–79, May 1992.

[11] J. Hershberger and J. Snoeyink. Speeding up the Douglas-Peucker line-simplification algorithm. In P. Bresnahan et al., editors, *Proc. 5th Intl. Symp. on Spatial Data Handling*, volume 1, pages 134–143, Charleston, SC, Aug. 1992.

[12] H. Hoppe. Progressive meshes. In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 99–108. ACM SIGGRAPH, Addison Wesley, Aug. 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[13] K. H. III, T. Culver, J. Keyser, M. Lin, and D. Manocha. Fast computation of generalized voronoi diagrams using graphics hardware. *Proceedings of SIGGRAPH 99*, pages 277–286, 1999. ISBN 0-20148-560-5. Held in Los Angeles, California.

[14] M. W. Jones and M. Chen. A new approach to the construction of surfaces from contour data. *Computer Graphics Forum*, 13(3):C/75–C/84, Sept. 1994.

[15] E. Keppel. Approximating complex surfaces by triangulation of contour lines. *IBM J. Res. Dev.*, 19:2–11, 1975.

[16] R. Klein. Abstract voronoi diagrams and their applications. In H. Noltemeier, editor, *Proceedings of the International*

Workshop on Computational Geometry and its Applications, volume 333 of *LNCS*, pages 148–157, Berlin, Mar. 1988. Springer.

[17] R. Klein. *Concrete and abstract Voronoi diagrams*, volume 400 of *Lecture Notes in Computer Science*. Springer-Verlag Inc., New York, NY, USA, 1989.

[18] R. Klein and J. Krämer. Multiresolution representations for surface meshes. In *Proceedings of the SCCG (Spring Conference on Computer Graphics), Budmerice, Slovakia*, pages 57–66, 1997.

[19] R. Klein, G. Liebich, and W. Straßer. Mesh reduction with error control. In *IEEE Visualization '96*. IEEE, Oct. 1996. ISBN 0-89791-864-9.

[20] R. Klein and A. Schilling. Fast distance field interpolation for reconstruction of surfaces from contours. In *Eurographics '99, Short Papers & Demos proceedings*, 1999. Conference held in Milano, Italy,.

[21] D. T. Lee and C. K. Wong. Voronoi diagrams in L_1 (L_∞) metrics with 2-dimensional storage applications. *SIAM Journal on Computing*, 9(1):200–211, Feb. 1980.

[22] D. Levin. Multidimensional reconstruction by set-valued approximation. *IMA J.Numerical Analysis*, (6):173–184, 1986.

[23] M. Lounsbery, C. Loop, S. Mann, D. Meyers, J. Painter, T. DeRose, and K. Sloan. Testbed for the comparison of parametric surface methods. In L. A. Ferrari and R. J. P. de Figueiredo, editors, *Curves and Surfaces in Computer Vision and Graphics (Proceedings of SPIE)*, volume 1251, pages 94–105, 1990.

[24] D. Meyers, S. Skinner, and K. Sloan. Surfaces from contours. *ACM Transactions on Graphics*, 11(3):228–258, July 1992.

[25] H. Müller. Using graphics algorithms as subroutines in collision detection. In W. Straßer and F. Wahl, editors, *Proceedings Graphics and Robotics*, Springer Verlag, May 1994.

[26] H. Müller and A. J. Klingert. Surface interpolation from cross sections. In H. Hagen, H. Mueller, and G. Nielsen, editors, *Focus on Scientific Visualization*, pages 139–190. Springer-Verlag, 1993.

[27] J. M. Oliva, M. Perrin, and S. Coquillart. 3D reconstruction of complex polyhedral shapes from contours using a simplified generalized Voronoi diagram. *Computer Graphics Forum*, 15(3):C397–C408, Sept. 1996.

[28] B. A. Payne and A. W. Toga. Distance field manipulation of surface models. *IEEE Computer Graphics and Applications*, 12(1):65–71, Jan. 1992.

[29] R. Ronfard and J. Rossignac. Full-range approximation of triangulated polyhedra. *Computer Graphics Forum*, 15(3):C67–C76, C462, Sept. 1996.

[30] L. Schumaker. Reconstructing 3d objects from cross-sections. In W. Dahmen, M. Gasca, and C. Micchelli, editors, *Computation of Curves and Surfaces*, pages 275–309. Kluwer Academic, Dordrecht/Norwell, MA, 1989.