Solid Modeling

Marc Rochel

April 2002

Table of contents

1 Introduction

There are a large number of papers released in the last few years in the area of solid modeling. The intent of the document is to give an overview of a selection of these papers. The contents are simplified and some notations are different to the papers. Please refer to the papers for special details.

The topics are not clearly separated from each other. Nevertheless, I have tried to sort the papers in three sections. Section two is about methods about volumetric modeling. Section three deals with algorithms which use topological information to construct or work with the models. Section four presents methods about modeling with polygonal meshes.

2 Volumetric Modeling methods

2.1 Fast Free-Form Volume Deformation Using Inverse-Ray-Deformation

Real-time rendering of volume data has become possible due to the computing power available today. This paper deals with the extending of the possibilities the known volume rendering techniques offer by introducing a new way to render the volume data enabling to deform the data by an attached Free-form-deformation (FFD) without the need to perform the time-consuming deformation on the object itself.

For the FFD-Grid, the uniform B-spline presentation is used. The deformation is never applied to the object itself. Instead, when the object is rendered, each ray that is cast to calculate a pixel of the picture to be produced is inverse deformed by the FFD.

This idea is implemented as follows. To calculate the deformed ray, some equidistant points are inverse deformed by the FFD. Afterwards, because of the deformation, the distances between deformed points must not be equal any more. To correct this, new points are calculated by linear interpolation between two points that belong together. The number of points created is adapted to the distance between the deformed points. In the end, most likely there will be some distances between newly calculated points, which's distances are not equal. This calculated ray is traversed like the rays in normal volume rendering methods using the low albedo model.

Well known methods to speed up normal volume data rendering like early-ray-termination, which means to stop traversing a ray when its density gets high enough to be accepted as opaque from the user, or space-leaping, to make several steps at once if the space in between is empty, can be adapted to be used in the method.

It is obvious that this only approximates the real inversion of the ray. It can be accepted that the ray is traversed in small steps, at least, if they get smaller or equal to 0.5 * voxel size, the error gets minimal. This is expressed in the Shannon Theorem. However, a problem exists due to the linear interpolation after the inverse FFD. The position of a point may be wrong, especially if the deformation defined by the FFD-Grid is big. No error calculation is given in the paper. If this error is acceptable or not depends on the application the method should be used for. In medical applications, small errors can result in big problems.

Another error of the described method lies in the implementation of the ideas explained above. To calculate the inverse deformed point of a point x by the FFD, the authors of the paper use the following formula (the notation slightly differs from the paper):

$$FFD^{-1}(x) = x - (FFD(x) - x) = 2x - FFD(x) \tag{1}$$

Furthermore, the following formula always holds:

$$FFD^{-1}(FFD(x)) = x \tag{2}$$

Replacing FFD^{-1} by the first formula results in:

$$2FFD(x) - FFD(FFD(x)) = x \tag{3}$$

It can be easily seen that this formula does not hold for any point x. As a result, the first formula is essentially wrong and can lead to strange rendering results. As a simple example of the errors possible to produce, you can think of rays that intersect each other. This is not allowed.

To correct this error you have to calculate the real inverse of the FFD. I do not know if this is easier than applying the deformation to the object itself. In this case, the rendering time will not be shorter than normal volume data rendering techniques. Additionally, normal rendering would allow adding other objects to the scene; these will not be rendered correctly if the deformation should not be applied to the other objects. Further on, this method does not use the available features of today's hardware accelerators. Finally, there is no speed comparison between normal deformation, rendering and their method. But the idea to use an FFD to deform volume data is a good idea. It can be used e.g. to manually animate medical data sets. However, there is nothing about that in the document.

2.2 Tetrahedron Based, Least Squares, Progressive Volume Models with Application to Freehand Ultrasound Data

This paper describes a method to reconstruct a three dimensional volumetric model from several two-dimensional ultrasonic pictures and their position in space. The model's resolution is adapted to the ultrasonic data available at a specific position.

To capture the ultrasonic data a standard ultrasonic probe is used in combination with a position tracker. The collected images are transformed and scaled into a three-dimensional unit cube, which is initially divided in 6 congruent tetrahedrons. (One way they describe to do this is to insert an edge from (0,0,0) to (1,1,1) and one edge on each side of the cube having one vertex (0,0,0) or (1,1,1).)

To calculate the vertex intensities a global least square error approximation is used. The function to minimize is (there is a small mistake in the paper, this is the correct version):

$$\sum_{j=0}^{M-1}((\sum_{i=0}^{N-1} I_i \Phi_i(d_j)) - F(d_j))^2 \tag{4}$$

Where M is the number of points acquired from the ultrasonic data, N the number of vertices in the mesh, I_i the intensity at vertex v_i, d_j the three dimensional position of point j of the ultrasonic data, $\Phi_i(v_j) = \delta_{ij}$ and F(p) the intensity of the ultrasonic data at position p.

In order to minimize this function, the Cartesian coordinates are replaced with barycentric coordinates by a function $B_v^T(p)$ that returns the barycentric coordinates of p corresponding to v if p is inside T, 0 otherwise.

The computation results in the problem of least square solving the following system of equations

$$A = IB$$

$$A_{i,j} = \sum_{k=0, T_l \in (S(v_i) \cap S(v_j))}^{M-1} B_{v_i}^{T_l}(d_k) B_{v_j}^{T_l}(d_k) \tag{5}$$

$$b_i = \sum_{k=0, T_l \in (S(v_i) \cap S(v_j))}^{M-1} B_{v_i}^{T_l}(d_k) F(d_k)$$

Because of the size of this system, it cannot be solved with direct matrix inverse methods. A better way to do this is to use a modification of the Gauss-Seidel iterative method. This is faster than the first method and can be optimized to require less memory because the matrix A is not changed during the optimization so the elements of A must not be kept in memory, instead they can be calculated on demand.

After this calculation for each tetrahedron the error inside it is calculated as follows: Every tetrahedron has a list of points d_j from the images that lie inside of it. The actual intensity of a point inside the tetrahedron is linear interpolated between the vertexes of it. Using barycentric coordinates p_i of p this interpolation is

$$I(p) = \sum_{i=0}^{3} p_i I_i \tag{6}$$

The error is defined as the normalized sum of the least square error between the interpolated intensity and the intensity acquired from the images over all points in the tetrahedron (there is a small mistake in the formula in the paper, I guess this is what they meant):

$$mse_{T_i} = \frac{\sum_{j=1}^{D_{T_i}} (I(d_j)) - F(d_j))^2}{D_{T_i}} \tag{7}$$

Where D_{Ti} is the number of data points inside the tetrahedron T_i and F(p) is the intensity from the captured data at p.

If the following sum is below a specified value, the construction process is stopped:

$$rms = \sqrt{\frac{\sum_{i=1}^{N_T} mse_{T_i} D_{T_i}}{M}} \tag{8}$$

Otherwise, the 5% of the tetrahedron with the biggest error are divided along their longest edge and the process is repeated, starting with global solving of (5) with the new vertexes. During the split process, the barycentric coordinates have to be recalculated and in the end, a refinement process has to be applied to remove all pairs of edges with different lengths that lie on each other. This may result in strange effects during rendering. For other special details, please refer to the original document.

The structure presented to store the volume data has several advantages. Its resolution is adaptive to the amount of data that should be stored in it and can recursively extended with new data. However, the reconstruction process does not correct small errors produced during capturing the data. The information acquired from the positional tracker might not be perfect so some calibration of the different images in space might be useful. Otherwise these small errors result in smoothing effects, can be seen in Figure 8 of the paper, or create displacements, see Figure 9 in the paper. An artery would not have such a wavy boundary as shown in this reconstruction. There is no use in medical practice for this system if small details get lost or get deformed like this. Additionally, during an ultrasonic examination, the patient is asked to move a little bit or the probe is pressed a bit inside the body to see more of a deeper organ or the patient breaths. This deforms the material and therefore the different images cannot be used to reconstruct the object with this method. This error gets even bigger, if a moving object is scanned, like a heart. Since the different images are not taken simultaneously, you will not see anything in the reconstruction.

3 Topological Modeling methods

3.1 Structural Operators for Modeling 3-Manifolds

This paper introduces the Morse Operators. They are able to build and unbuild any combinatorial orientable three-dimensional manifolds embedded in R^n ($n>2$). Additionally, a suitable data structure for the representation of three-manifolds is presented.

The main theory, the morse operators are based on, is:

"Every compact orientable 3-manifold with or without boundary has a handle decomposition, i.e., it can be obtained by attaching a handle to a sequence of compact orientable 3-manifolds with boundary starting with a 3-ball."

Where a handle describes the identification between two boundary faces, all vertices and edges that are not incident to both boundary faces become identified, too.

There are 5 types of identifications that can be applied, for each of them exists a Morse operator, which builds a new manifold with the operation applied.

1) "The faces are on different three-manifolds: OMV_1
2) The faces are on different boundary surface components of a three-manifold: OMV_2
3) The faces are on the same boundary surface component of a three-manifold and none of its edges are incident to both faces: OMV_3
4) The faces are on the same boundary surface component of a three-manifold and some of its edges are incident to both faces. OMV_4
5) The faces are on the same boundary surface component of a three-manifold and all of its edges are incident to both faces. OMV_5"

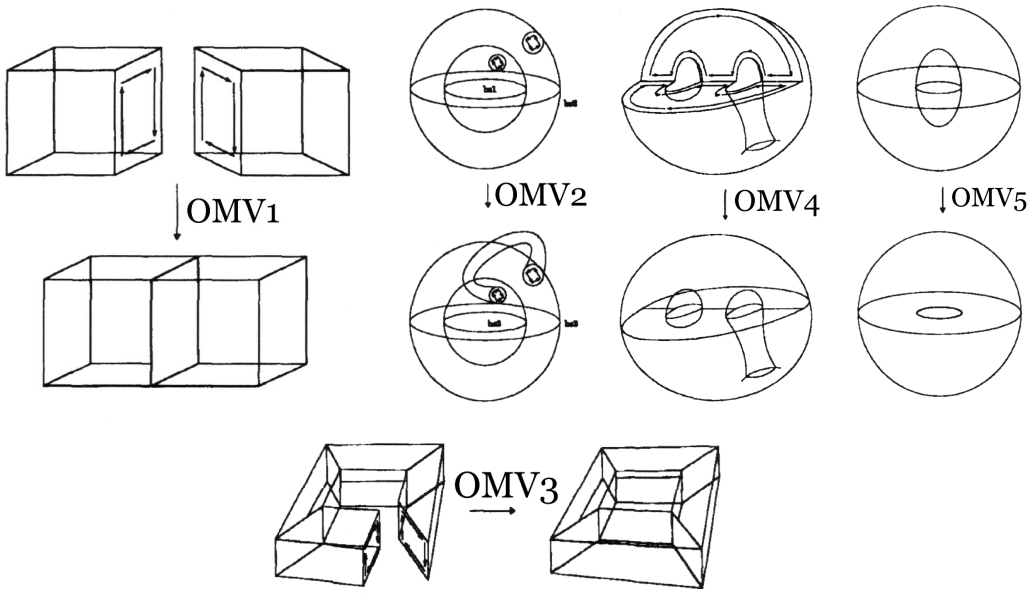

Figure 1: The five Morse operators

Additionally, for each operator exists an inverse operator $OIMV_x$.

The Handle-Face data structure described in the rest of the paper is a hierarchical representation of an object, that allows to apply these five operators easily.

3.2 Dealing with Topological Singularities in Volumetric Reconstruction

This paper introduces a new data structure, called "Handle-Strata", to represent three-dimensional stratified manifolds based on Morse theory, presented in the previous Paragraph. This data structure is used for volumetric data reconstruction from planar sections, avoiding topological singularities. The Handle Strata data structure is a replacement for the Handle-Face data structure, specialised for this application.

The Handle Strata structure is a hierarchical representation of the object to be described. Object representations are built and unbuilt on this data structure by Morse Operators, which correspond to connecting handles on manifolds with boundary.

The inputs for the following algorithm are different slices belonging to an object that should be reconstructed. First of all, the two-dimensional contours on the different slices are extracted. Therefore, equivalents to OMV_1, OMV_3, OMV_4 and OMV_5 are used. These were originally defined in the other paper for three-manifolds only and instead of a 3-ball a 2-ball (disc) is initially used. They are called local operators. Then a restricted Delauney triangulation is applied between the slices. This is the implementation of another OMV_1 equivalent, which connects two 2-manifolds to one 3-manifold:

1) "Build a 3-dimensional Delauney triangulation D using the vertices of all contours,
2) Mark the edges of the contours that are not contained on D,
3) Subdivide all marked edges, inserting new vertices on the contours,
4) Make local modifications on D to obtain a new Delauney triangulation that includes those new vertices,
5) Repeat these steps until the triangulation contains all contour edges."

Afterwards all produced tetrahedrons with at least one external edge, meaning edges, that lie in the plane of one of the two slices between which the tetrahedron is constructed and lie outside any contour on these slices, are removed. This step may produce singularities, which are removed as follows: If the singular edge lies in a contour, reinsert the tetrahedron and split it at the external edge. Move the newly created vertex in between the two slices. Otherwise, if the singular

edge lies on a contour, split the connected components. In the end, the produced meshes are merged together to one object. Here, the real OMV_1 is used. The last two operators are referred to as global operators in the paper.

The presented method is useful to avoid singularities in the reconstruction of volumetric data from strata, which otherwise could cause trouble in the later use of the reconstructed object. In comparison, the presented results seem intuitively better than the results created from the other methods from Nonato and Tavares or Nuages. Additionally the Handle-Strata data structure is flexible enough to handle these other methods too.

3.3 Topology Matching for Fully Automatic Similarity Estimation of 3D Shapes

The document presents a method to evaluate the similarity between two objects given as meshes. The calculated values are invariant to transformations of the objects.

Similarity is compared with respect to the topological attributes of the objects. Since many still existing objects only exist as meshes without any topological information, this information is derived from the mesh. To get this information, a multiresolutional Reeb Graph (MRG) is constructed. A Reeb Graph is defined as follows:

"Let $\mu : C \to R$ be a continuous function defined on an object C. The Reeb graph is the quotient space of the graph of μ in $C \times R$ by the equivalent relation $(X_1, \mu(X_1)) \sim (X_2, \mu(X_2))$ which holds if and only if $\mu(X_1) = \mu(X_2)$, and X_1 and X_2 are in the same connected component of $\mu^{-1}(\mu(X_1))$."

The main difference to the multiresolutional Reeb graph lies in the allocation of the scope of μ, R, in discrete intervals. This is necessarily to be able to calculate the multiresolutional Reeb graph for a specific object. The construction of the MRG begins with the construction of a Reeb graph having the finest resolution.

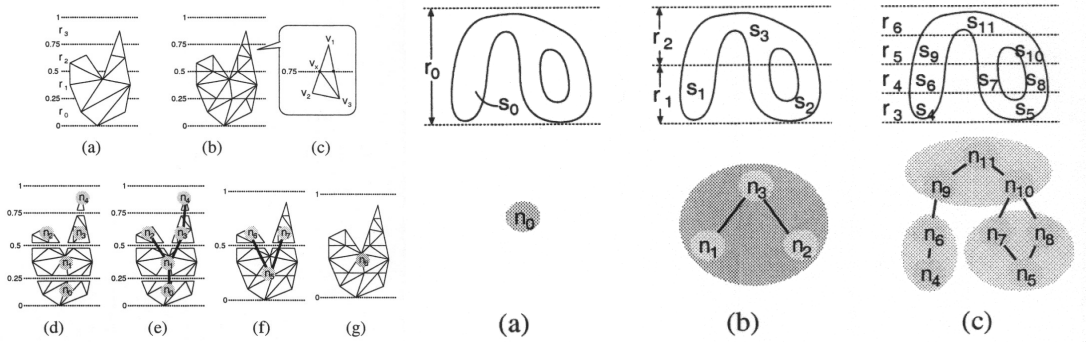

Figure 2: Construction of a MRG (left), MRG using a height function (right)

The function μ itself has to fulfil some attributes. It has to be invariant to transformations and uniform scaling. Let us have a look at the following function:

$$\mu(v) = \int_{p \in S} g(v, p) dS \tag{9}$$

Where g(v, p) returns the geodesic distance between v and p on the surface S. To achieve invariance to uniform scaling, this function is normalized to

$$\mu_n(v) = \frac{\mu(v) - \min_{p \in S} \mu(p)}{\max_{p \in S} \mu(p)} \tag{10}$$

The denominator is not $range(S) = \max_{p \in S} \mu(p) - \min_{p \in S} \mu(p)$, not to amplify errors when $range(S)$ is small. This function is actually calculated by discretisation of the surface triangles and approximating the calculation of the geodesic distance g(v, p) by adding the length of path along the edges or shortcuts between the two points of v and p.

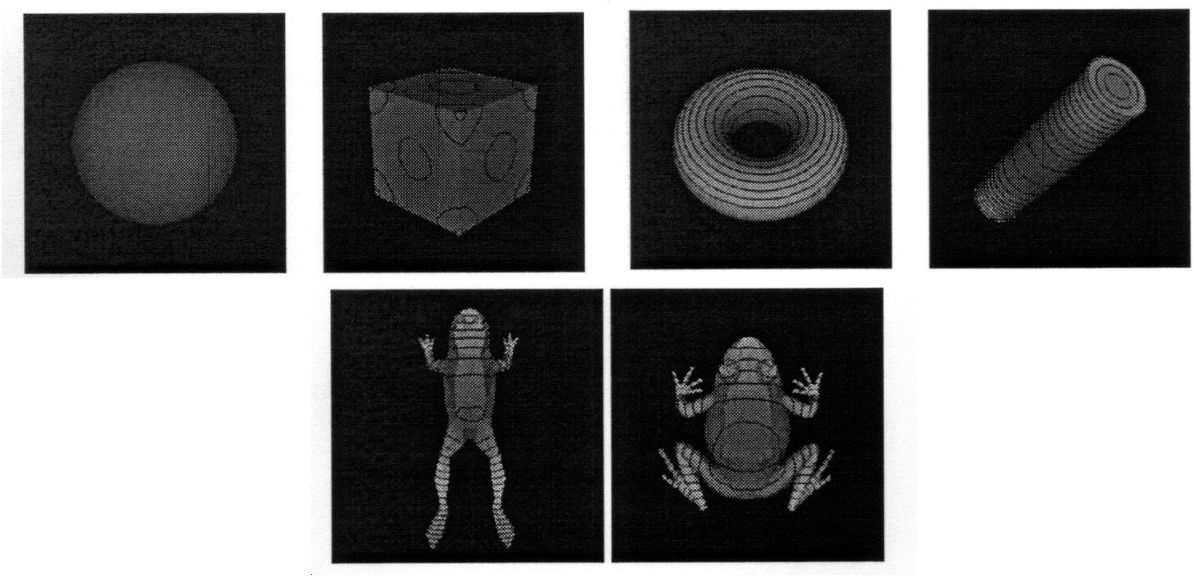

Figure 3: Examples of function μ

For both objects to be compared, such an MRG is build. First of all, beginning with the lowest resolution, the nodes of each MRG are matched and adaptively refined. Approximately the more nodes of the graphs are matched, the more similar the two objects are evaluated. A matching function which additionally considers different attributes, which can be defined for the nodes, e.g. material for example, can be specified. Please refer to the paper for more details.

The described method is useful for comparison between two objects with unknown topological information, only having the mesh. The application mentioned in the paper, i.e. the use as a key for searching for similar objects in a database is a good idea, but it is not developed enough. The algorithm presented in the paper has a run time complexity of O(n), where n is the number of objects in the database, because the objects have to be compared with every object in the database. E.g. the average time their implementation requires to search in a database with 230 objects is 12sec. But a real database would consist of perhaps thousand times more of objects. To use this method for database queries, it has to be extended by some kind of merged MRG, where all MRGs of the objects in the database are merged in one big MRG which is processed in the query. Early determination of wrong objects is required instead of calculation of the similarity for every unsimilar object. So their method could probably be extended to be used as an index over the objects in the database und a run time complexity of O(log n) might be possible. Further research is required.

3.4 Boundary Representation Models: Validity and Rectification

This paper describes a method to validate und rectify solid models, represented by the typical B-rep data structure. Topological information is used to rectify the object by reconstruction. Interval solids care about numerical robustness.

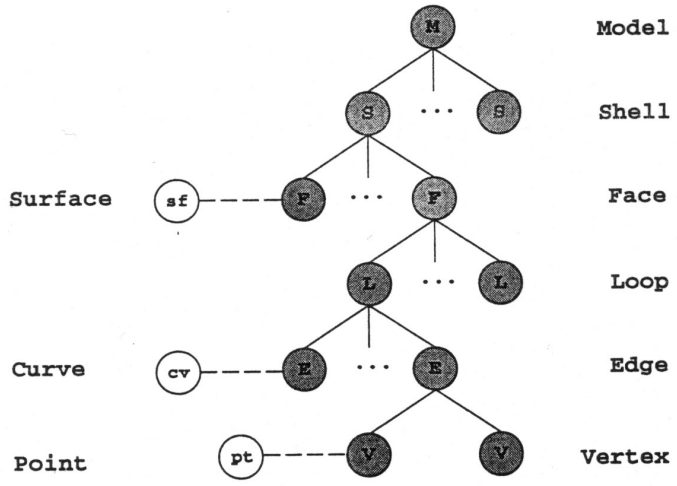

Figure 4: The B-rep data structure

A B-rep data structure is shown in Figure 4. The structure is built hierarchically. Due to numerical errors created by operations that were applied to the object during the modeling phase, the data in the structure can become inconsistent. E.g. an underlying edge of a face might not lie on the attached surface. Since the operations of today's modeling tools are not closed on scope of the used data types, these errors cannot be prevented and if problems occur, the model has to be rectified.

To formalize the given problem, first of all, definition of solids and their boundaries are given. This has to be done to actually know, what a correct object is and where rectification should lead to. The relation to the boundary is useful, because in the B-rep structure, the object itself, the 3-manifold with boundary, is represented by its boundary, the 2-manifold without boundary. Furthermore, sufficient conditions for validity are given for every node in the B-rep data structure. Please refer to the paper for the exact definitions.

The following definitions are needed to represent the reconstruction algorithm: Lower case characters represent the models and face nodes of the models. Upper case characters represent the solids. Let m_0 be a model with topological structure $G(m_0)$. Then there exists a nonempty set

$$M = \{ m \mid m \text{ is a valid model and has topological structure } G(m_0) \} \tag{11}$$

For simplicity, models have only one shell. Rectification only makes sense, if geometric information and topological information are inconsistent, $m_0 \notin M$. One step of the rectification process is to reconstruct a solid m_n. If $m_n \in M$, m_n is topologically equivalent to the model incorporating the design intent. If $m_n \notin M$, the topological equivalence between m_n and m_0 can be imposed by requiring that the genus of dM_n is equal to the genus of dM_i, where $M_i \in M$. This is denoted by $g(m_n) = g(m_0)$. Let $\phi(dA, dB)$ be a function that calculates the geometric difference between dA and dB where $A, B \in M$, ε a tolerance for the geometric change. Let $\Gamma(dA, dB)$ be a function that calculates the topological structure change needed to achieve $g(a) = g(b)$.

The actual algorithm works as follows (The steps are renumbered in comparison to the paper to clarify the 3 approaches):

1) Try to reconstruct the new model m_n, with $m_n \in M$ and $\phi(dM_0, dM_n) \leq \varepsilon$ and $\phi(dM_0, dM_n)$ minimal.
2) If no such model exists, try to reconstruct the new model m_n, with $g(m_n) = g(m_0)$ and $\phi(dM_0, dM_n) \leq \varepsilon$ and $\Gamma(dM_0, dM_n)$ minimal.
3) If no such model exists, try to reconstruct the new model m_n, with $\phi(dM_0, dM_n) \leq \varepsilon$ and minimal topological change.
4) Else, no Object is reconstructed.

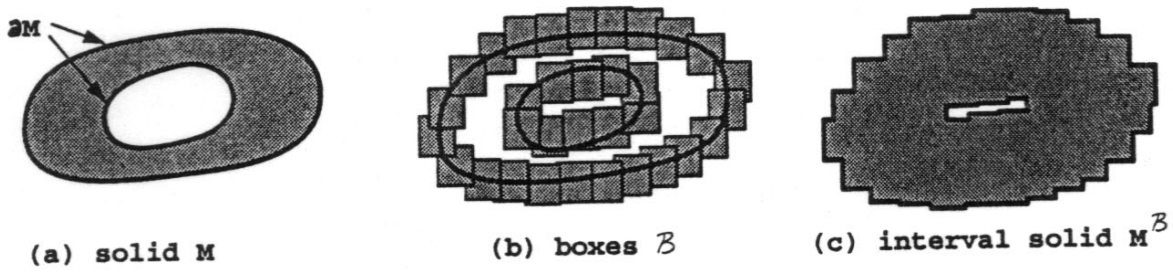

Figure 5: A solid (left) and its associated interval solid (right).

A complexity analysis results in Step 2) is NP-Hard.

For numerical robustness, the interval solid of a solid is defined. The basic idea is to represent the different shells of the object by many boxes covering the surface und filling the space in between to get a solid. Have a look at Figure 5. For more details, the reader is referred to the paper. This method is adapted to interval faces. These objects are used to evaluate if the underlying nodes in the B-rep structures are valid and, if not, the underlying nodes are modified, e.g. faces are grown a little bit. The size of the boxes approximately defines the resolution of the reconstruction and is related to ε.

The presented method is able to rectify B-rep models with errors produced by previous operations. To correct the inconsistent data, the topological structure is tried to be kept. As a result, the user can use the rectified model as before, e.g. no random triangulation is applied. Finally, the whole problem cannot be solved by the presented method at all. It is not transparent for the user; some constants have to be specified and perhaps have to be altered to achieve the desired result. I suggest expanding research on closed operations. This prevents these errors and no validation and rectification methods are needed.

4 Polygonal Modeling methods

4.1 Polyhedral modeling

The paper introduces a method for construction smooth surfaces from triangulated polyhedral mesh of arbitrary topology. The method is completely local and normal vectors can be interpolated. It can be used to beautify data acquired from three-dimensional reconstruction.

The basic idea of the smoothing process can be best explained in two dimensions. There, several connected edges build a B-Spline of degree 1. To smooth this line, the degree can be increased resulting in some degrees of freedom available to satisfy C^x continuity. This idea is adapted to three-dimensional meshes. In the paper G^1 continuity is used.

This basic idea is extended with some considerations. G^1 continuity normally stands for visual smoothness, but it does not guarantee a "nice shape". Waves and self-intersections may occur; they must be avoided.

The rest of the paper essentially consists of mathematical formulas which transfer these ideas to practice.

The presented method can be used to smooth triangulated polyhedral meshes. It depends on the application if this is useful. E.g. if the mesh to be smoothed is created by a set of slices got from a medical investigation, this smoothing might not only beautify the result, it may also get

inconsistent with the original slices, because of the applied transformations on the single points of the surface. Furthermore, if there is more than one object in the scene, intersections between the different objects may occur, because only self-intersection is prevented. So this method should be used with caution. A better way to smooth objects would be to apply the smoothing during the creation of the mesh. E.g. if G^1 continuity is demanded during the surface extraction from the slices, the correctness of the mesh can be preserved.

4.2 Modeling Murex cabritii Sea Shell with a Structured Implicit Surface Modeler

This paper does not describe a general method to construct a model, instead it is an example to implicit surface modeling. Constructive solid geometry (CSG), warping, 2D texture mapping and operations based on the BlobTree are used to construct a model of a sea shell.

The underlying data structure to hold the model is the BlobTree. In the BlobTree, an implicit surface model is defined using a tree data structure, which combines implicit surface primitives as leaf nodes, with arbitrary operations such as blending, warping, and Boolean operations as interior nodes. Available operations for these nodes are: n-ary operations: ∪ (union), ∩ (intersection), - (difference), + (blend), $\Diamond_n$ (super-elliptic blend), c (controlled blend), unary operations: w (warp), t (translate), s (scale), r (rotation), and m (2D texturemap). Due to the flexibility of the BlobTree, new operations can be added. See Figure 6 for an example of a BlobTree.

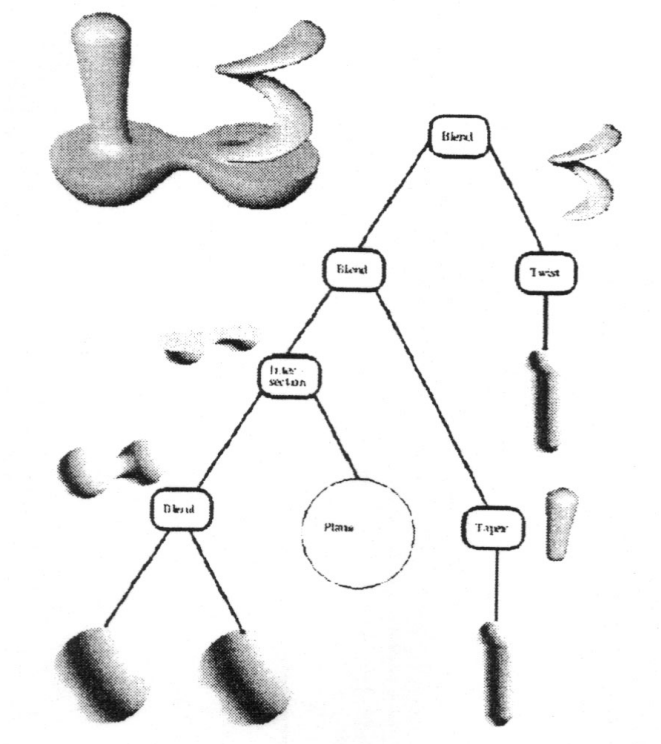

Figure 6: Example of a BlobTree

Primitives are defined as functions.
Super elliptic blending is defined as

$$f_{B_a \Diamond B_b} = \left({f_{B_a}}^n + {f_{B_b}}^n \right)^{\frac{1}{n}} \tag{12}$$

The standard blending operator + is a special case of this formula with n = 1. Furthermore:

$$\lim_{n\to+\infty}(f_{B_a}{}^n + f_{B_b}{}^n)^{\frac{1}{n}} = \max(f_{B_a}, f_{B_b}) \tag{13}$$

So the super elliptic blending operator interpolates between the union and the standard blend operator.

Controlled blending enables the option to control the blending between more than two BlobTrees. For example, a BlobTree can be blend with two others, but these two are not blended to each other. Controlled blending is defined as follows:

$$\begin{aligned} &c(b_1, b_2, ..., b_n)(B_1, B_2, ...B_m) \\ &b_i = (j_i, k_i) \\ &j_i, k_i \in \{1, ..., m\} \forall i \in \{1, ..., n\} \end{aligned} \tag{14}$$

Implicit modeling is a powerful tool to define models, which offer the possibility to be described with mathematical formulas. Otherwise such models might be hard to model with normal NURBS modelers, like the sea shell described in the paper. However, implicit modeling also makes higher demands against the designer. The user has to understand the mathematics that describes the model. Furthermore, it is not possible to simply move some vertexes to change some details of the object like in other modeling techniques.

4.3 Distance Computation between Non-convex Polyhedra at Short Range Based on Discrete Voronoi Regions

This paper describes a method to compute the distance between triangulated meshes at interactive speed rates.

The algorithm needs some precalculation I will describe at first. A three-dimensional voxel grid is constructed around the objects to be used. For every voxel in the grid, a list of triangles, the "Closest Feature List" (CFL), of the model, that have possibility to be the closest to the points in the voxel, is calculated. This is done as follows: For a voxel V and triangle A we can calculate the minimum and maximum distance between them (Figure 7).

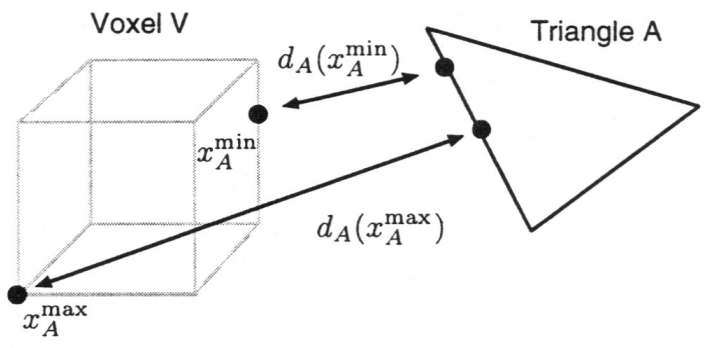

Figure 7: Minimum distance between triangle and point in voxel

For every point x in V it applies that

$$d_A(x_A^{\min}) \le d_A(x) \le d_A(x_A^{\max}), \forall x \in V \tag{15}$$

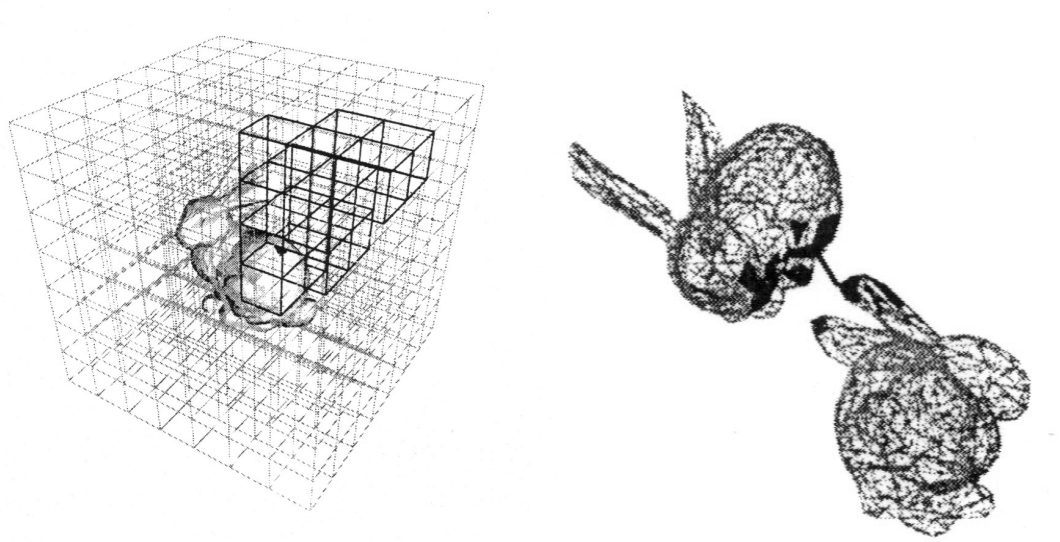

Figure 8: Voxels associated with a triangle (left), triangles (black) involved in distance calculation (right)

If, for two triangles A and B, $d_A(x_A^{\max}) < d_B(x_B^{\min})$ applies, we get together with the previous equation

$$d_A(x) < d_B(y), \forall x \in V, \forall y \in V \tag{16}$$

Therefore, B cannot be closest triangle for any point in V and is left out of the CFL. So by calculating all the distances between V and the triangles and sorting them and checking this condition, the CFL is computed for V. This precalculation takes some time. E.g. to calculate the CFLs of a 30^3 voxelgrid with an object of 1324 triangles took 40 minutes.

Let A and B be two objects which's distances we want to calculate. Then we check, which triangles of B intersect with the voxels of A. This can be done efficiently by defining a BSP tree on the voxels on A and an OBBTree on the triangles of B. Afterwards this is done again for the voxels of B and the triangles of A. But you only have to check the voxels of B where the triangles of A returned by the first step are inside of, and respectively, the triangles of A, that are inside of a returned voxel of B. This is sufficient because the voxel-triangle pair we finally want holds the following condition: Feature X intersects voxel V_Y, the CFL of V_Y includes feature Y and feature Y intersects voxel V_X, the CFL of V_X includes feature Y. Then, the minimum values of distances of the voxels are calculated and are sorted according to them. Finally, the real distances are computed between the triangles in the CFL and the triangle interfering the voxel. This calculation can be early terminated, if the minimal distance of the next voxel in the sorted list is bigger than the real distance recently calculated. If the objects do not intersect the voxels of each other, the distance is approximated by the distances between the convex hulls of the models. An extension to this approach is given in short by introducing a method to cull voxels by approximated Voronoi Regions.

The presented method enables to calculate the distances between complex triangulated models in real-time. A relatively long time for precalculation is needed. The method does not support non-triangulated models, e.g. models defined by NURBS. In comparision to another method implemented in the PQP library, the presented method is 3.4 times slower. Nevertheless, they say, this depends on their implementation and it could be as fast as the method used in the PQP library. They suggest using this for collision detection in dynamics simulation. However, in collision detection you do not need the actual distance between the objects, you are only interested if they collide or not. I propose using a more efficient algorithm to check for intersection than to compute the distance and check if it is zero.

4.4 Interior/Exterior Classification of Polygonal Models

Visualization of complex objects with interior structure is a complicated problem. It is useful to determine interior and exterior parts of the model, to render them with different attributes. E.g. the interior could be rendered opaque and the exterior transparent (Figure 9). This determination is done manually in existing methods. This paper introduces a method to automatically solve this problem.

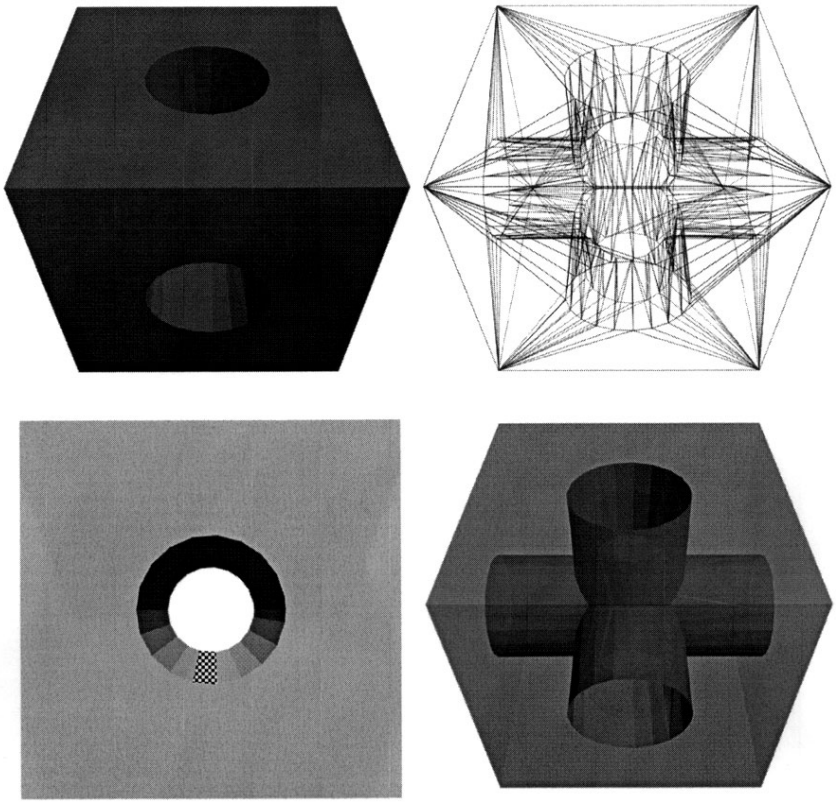

Figure 9: Cube with internal structure rendered in three different ways. The picture on the bottom right shows the side of the cube.

Let S be the surface of the model and L a line. Then the extremal points of S along L are defined as the points of $T = S \cap L$ with minimum and maximum value of t(p), where $p \in T$ and t(p) returns the solution for t for $p = u + t\,v$, where u is the supporting vector and v the direction vector of L. An exterior point is defined if it is extremal for some line L. If it is not extremal for any Line L it is defined interior. In the algorithm, this definition is approximated by defining a point as exterior, if it is extremal for a fixed set of lines which's directions are distributed over a hemisphere. Else, it is defined as interior. Thus, determination if a point is external or internal gets calculably. Practically this is calculated quite efficient by rendering the model with orthographic projection from all the specified directions into so-called "deep" buffers. This buffer works like a zbuffer except it also stores the layers behind the front layer. Thus, to determine if a point p is extremal at the line with a direction v_i, you only have to look at the appropriate "deep" buffer and there at the pixel containing p. If it is the first or the last point in the pixel, the point is extremal, else it is not. Thus, the extremal determination is done parallel for many points.

Based on it, extremal points are marked as visible and a polygon p is marked as visible if all its vertices are visible, else it is marked invisible. Then, when p is finally classified, it is marked as visible, if it is marked as visible by at least a before specified amount of directions. The visible polygons are specified as exterior, the invisible as interior. The intent to introduce a threshold value is to prevent polygons to be marked as visible that lie for example in tunnels and are visible

from a small number of directions but intuitively would be specified as being interior. However, this produces another problem with polygons that are visible from a small number of directions but intuitively would be specified as being exterior, like triangles on a complex surface. The value of this threshold depends on the model to be visualised. A method to solve this problem is described later on. Instead of only classifying interior and exterior polygons; it is also possible to directly assign different attributes depending on the amount of directions from which a polygon is visible. A completely invisible polygon can be marked as opaque and the more directions it is visible from, the more transparent it gets.

The method has problems with large triangles. These can have different visibilities at different positions. Thus, to prevent strange rendering of such polygons after classification, they must be subdivided until the length of the largest edge in the model is below a user specified threshold.

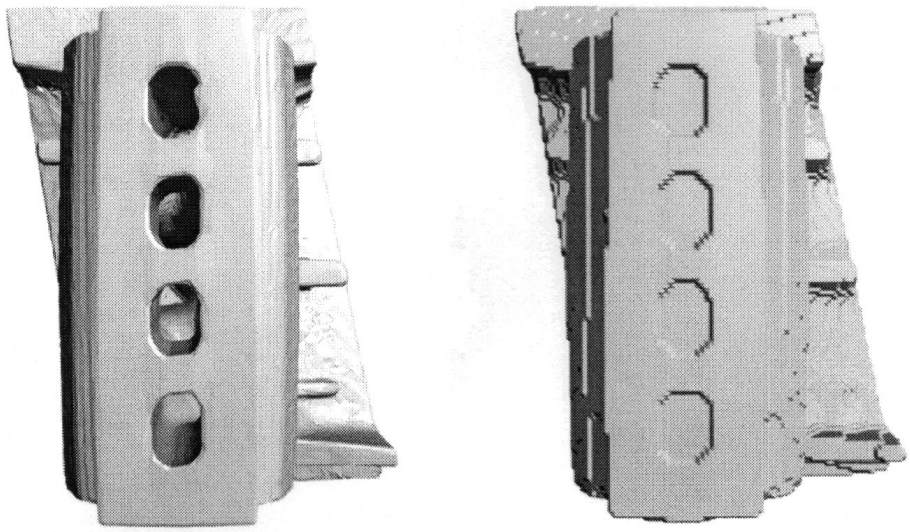

Figure 10: Example of a model before and after the applying of dilation and erosion. The holes get closed.

The basic idea presented to solve the problem previously announced is to close tunnels and holes in the model and to use it to determinate visibility of the original polygons. Therefore, the original model is converted to a volumetric representation. Then, the holes and tunnels are closed by applying a dilation followed by erosion (Figure 10). To obtain a polygonal model, the Marching Cubes algorithm is used. The classification of a polygon relative to a direction is modified as follows: First, it is checked if it is near the exterior surface of the original "deep" buffer. If this is not true, the polygon is classified as interior. Else, the polygon is projected in a new "deep" buffer, already containing the modified model with the closed holes. If the polygon does not lie near the surface of the modified model, it is classified as interior, exterior otherwise.

The presented method helps to visualize complex models without the need of manual editing of material attributes. Due to the use of "deep" buffers, modern hardware-rasterizers can be used to determine visibility. Nevertheless, the classification takes too long to do it on the fly before displaying the model. The examples took about 10 minutes. It depends on the desired application if this is worth to wait for it. The method can be adapted to use non-triangulated models like NURBS by triangulating them. This has to be done in any case if the model should be rendered in interactive environments to make use of the hardware-accelerations available. Though, this creates no additional expenditure.

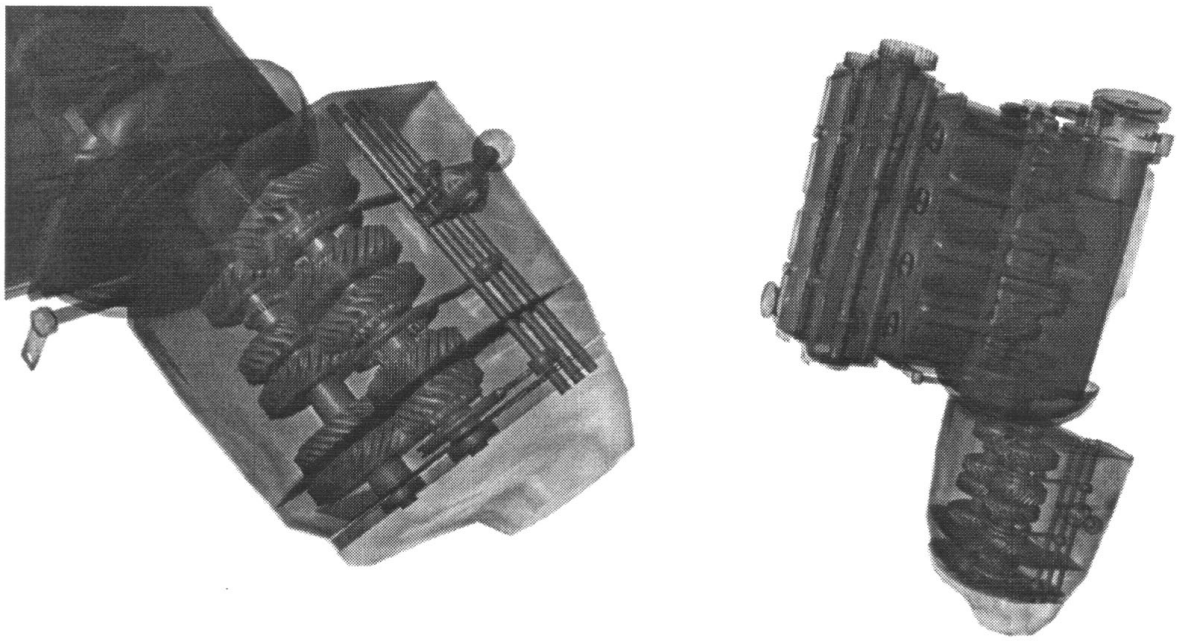

Figure 11: The model of a motor rendered with material attributes modified by the presented algorithm.

5 References

[1] Bonneau, Georges-Pierre and Stefanie Hahmann, Polyhedral modeling, University of Grenoble

[2] Chen, Haixin, Jürgen Hesser, Reinhard Männer, Fast Free-Form Volume Deformation Using Inverse-Ray-Deformation, Universität Mannheim

[3] Galbraith, Callum, Przemyslaw Prusinkiewicz, Brian Wyvill, Modeling Murex cabritii Sea Shell with a Structured Implicit Surface Modeler, Dept. of Computer Science, University of Calgary, T2N 1N4, 2000

[4] Hilaga, Masaki, Yoshihisa Shinagawa, Taku Kohmura, Tosiyasu L. Kunii, Topology Matching for Fully Automatic Similarity Estimation of 3D Shapes, Los Angeles, ACM SIGGRAPH, 2001

[5] Kawachi, Katsuaki and Hiromasa Suzuki, Distance Computation between Non-convex Polyhedra at Short Range Based Discrete Voronoi Regions, Department of Precision Machinery Engineering, The University of Tokyo, 2000

[6] Lopes, Hélio and Geovan Tavares, Structural Operators for Modeling 3-Manifolds, Department of Mathematics, Catholic University of Rio de Janeiro, 1997

[7] Lopes, H., L. G. Nonato, S. Pesco, G. Tavares, Dealing with Topological Singularities in Volumetric Reconstruction, Nashville, Vanderbilt University Press, 2000

[8] Nooruddin, F. S. and Greg Turk, Interior/Exterior Classification of Polyhedronal Models, GVU Center, College of Computing, Georgia Institute of Technology

[9] Patrikalakis, Nicholas M., Takis Sakkalis, Guoling Shen, Boundary Representation Models: Validity and Rectification, Cambrige, Massachusetts Institute of Technology

[10] Roxborough, Tom and Gregory M. Nielson, Tetrahedron Based, Least Squares, Progressive Volume Models with Application to Freehand Ultrasound Data, Arizona State University, Tempe AZ 85287-5406