A Survey and Analysis of Common Polygonization Methods & Optimization Techniques

Burkhard Wünsche
Department of Computer Science
University of Auckland
email: bwue001@cs.auckland.ac.nz

August 1997

Abstract

Implicitly defined surfaces of scalar fields (isosurfaces) are a common entity in scientific and engineering science. Polygonizing an isosurface allows storing it conventionally and permits hardware assisted rendering, an essential condition to achieve real-time display. In addition the polygonal approximation of an isosurface is used for simplified geometric operations such as collision detection and surface analysis. Optimization Techniques are frequently employed to speed up the polygonization algorithm or to reduce the size of the resulting polygon mesh.

1 Introduction

In this paper we review popular polygonization methods for implicitly defined surfaces (isosurfaces), which are important in scientific and engineering science. Isosurfaces are used to visualize scalar fields [Bli82, DK91] and, by reduction to scalar quantities, vector fields and tensor fields [DH93, PvW94]. Brill et al. use isosurfaces to define streamballs for flow visualization [BHR$^+$94].

Isosurfaces are also common in geometric modeling. Several blending methods use implicitly defined surfaces [HH87, War89, HH91, HL92]. In addition implicitly defined surfaces prove useful to define "blobby models". These models use a scalar field composed of many field generating primitives [Bli82, WMW86b, Mur91, BW90, BS91]. Colburn defines a smoothed object as isosurface in a scalar field obtained by convolving the object's characteristic function with a spherical filter [Col90]. Lobb presents an efficient approximation to the convolutional smoothing process [Lob96].

The polygonization of implicitly defined surfaces is essential for hardware assisted real-time rendering. Additionally the polygonized approximation facilitates geometric operations such as collision detection and surface analysis.

A practical introduction to polygonization methods for implicitly defined surfaces is provided by Bloomenthal who also gives working C code [Blo94]. Ning and Bloomenthal give a good evaluation of polygonization algorithm [NB93]. Kalvin [Kal92] presents an extensive survey of algorithms for constructing surfaces from 3D volume data. He assumes as input a regular grid of sample points and that no resampling is possible. Allgower and Gnutzman [AG87] give a more theoretical approach for a polygonization method and yield error bounds based on the mesh size. Dobkin et al. [DLTW90] gives a contouring algorithm for general dimensions. Finally Gelder and Wilhelms [vGW94] give a thorough discussion of design-objectives of isosurface algorithms, isosurface generation and solving of ambiguities.

This paper gives an overview and analysis of popular polygonization methods for implicitly defined surfaces and possible optimization methods.

To get a basis for discussion we first introduce some notations. We then explain four specific methods, which demonstrate useful principles in more detail. We discover a common framework for a general polygonization method for implicitly defined surfaces. Next we list some general quality criteria and review how the presented methods relate to them. We conclude with a review of optimization methods, both to improve speed and to reduce the size of a polygonization.

2 Notations & Definitions

An implicit surface is given as all points $x \in \mathbb{R}^3$ such that $\rho(x) = c$ for a scalar field $\rho : \mathbb{R}^3 \to \mathbb{R}$ and a constant $c \in \mathbb{R}$. The resulting surface is called a *c-isosurface*. Note that the c-isosurface of a function $\rho(x)$ is equal to the 0-isosurface of the function $\rho(x) - c$. Without loss of generality we use $c = 0$ and don't mention the c value explicitly.

A polygonization method approximates an implicit surface with a mesh of polygons. The implicit surface is either given as a function or as a set of sample values. In the following we assume that the isosurface is a 2-manifold, i.e., that it is locally homeomorphic to $\mathbb{R}^2$. Bloomenthal and Ferguson published recently a polygonization method for non-manifold isosurfaces and explain the inherent problems [BF95].

All reviewed polygonization methods take data samples in the volume of interest and compute or approximate from them points on the isosurface. Those isosurface points are connected to form a polygon mesh. To avoid confusion we introduce here a set of notations that we use throughout this paper.

A data sample is referred to as a *voxel*. A convex polyhedral region bounded by voxels is called a computational *cell*, and voxels at the cell's corners are called *vertices*. Generation of the isosurface involves sampling of the scalar field and defining computational cells. For each cell determine whether the underlying function takes on the threshold value within the cell, and if so, approximate where the isosurface lies. We shall call a vertex value *positive* if its value is greater than or equal to the threshold, and *negative* if not.

An *intersection point* is the point at which the isosurface is estimated to cross the edge connecting two adjacent cell vertices that have different sign. Such intersection points become vertices of one or more *topological polygons*. These polygons specify the topology of the approximated surface but are usually not planar.

We define the *center* of a set S_n of n points $p_1 \dots p_n$ as

$$center(S_n) = \frac{\sum_{i=1}^{n} p_i}{n}$$

The *central estimate* of a scalar field ρ at the center of a set S_n of n points $p_1 \dots p_n$ is defined as

$$central\_estimate(S_n) = \frac{\sum_{i=1}^{n} \rho(p_i)}{n}$$

The center of a face is the center of the face's vertices, the center of a cell is the center of the cell's vertices. Similarly for the central estimate.

We use the word *cell edge* for an edge of a polyhedral cell, and *polygon edge* for an edge of a polygonal approximation of the isosurface.

3 Four Common Polygonization Methods

Many published methods exist for finding a polygonal approximation to an implicitly defined surface. Though often written with a specific application in mind all methods that we review in this paper can be used to approximate a general isosurface.

In the following subsections we present four selected algorithms in more detail. First we choose the *Marching Cubes* method because it is popular and fast. Next we analyze the *Soft Object method* from Wyvill et al. [WMW86b]. This method is fast and eliminates the ambiguities of the original Marching Cubes method. Hall's and Warren's algorithm [HW90] and Bloomenthal's method [Blo88] are good examples for adaptive solutions. The former algorithm performs a tetrahedral subdivision of space, whereas the latter one is interesting because it uses an octree representation.

3.1 Marching Cubes: A High Resolution 3D Surface Construction Algorithm

The Marching Cubes algorithm combines simplicity with high speed. In the original implementation [LC87, CLL$^+$88] the Marching Cubes algorithm assumes discrete input data such as results from computed tomography (CT), magnetic resonance (MR), and single-photon emission computed tomography (SPECT). The scalar field ρ is unknown. The algorithm processes the 3D data in scan-line order and builds a logical array of cubes. Each cube is created from eight voxels; four each from two adjacent slices. The algorithm determines how the surface intersects this cube, then moves to the next cube.

The isosurface intersection is determined by the sign of the scalar field at the cube's vertices. Each edge with vertex values of different sign is assumed to intersect the isosurface once. The intersection point is approximated by linearly interpolating the scalar field values between the vertices.

Since there are eight vertices in each cube and two values, *positive* and *negative*, there are $2^8 = 256$ ways the surface can intersect the cube. Lorensen and Cline use symmetries to reduce the number of patterns to 15 which are shown in figure 1[1].

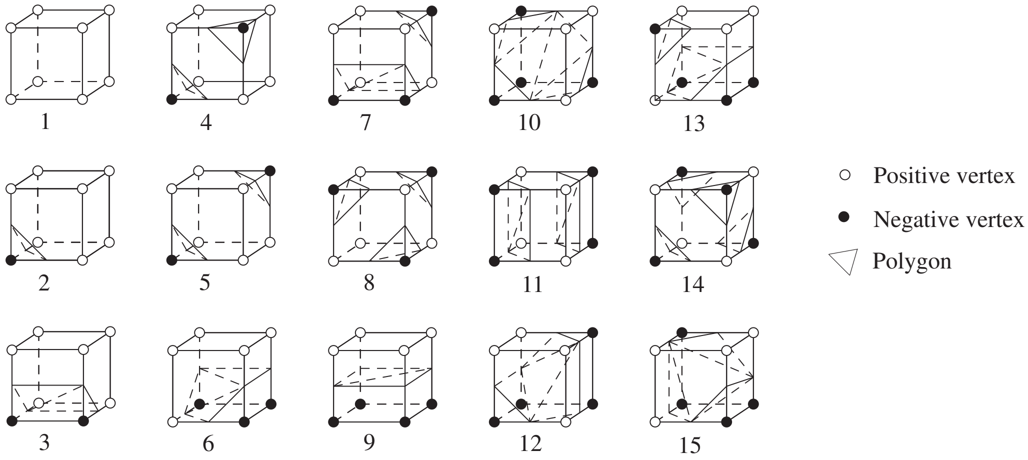

Figure 1: Triangulated Cubes.

The algorithm can be summarized as

- Scan two adjacent slices and create cubical cells between them.

[1] The cases 12 and 15 are reflective with respect to the xy-plane. This leaves 14 topologically distinct patterns (22 without inversed patterns) [LVG80].

- Calculate an 8-bit index for the cube from the sign of the eight scalar field values at the cube vertices.

- Using the index, look up the list of edges forming triangles from a precalculated table.

- Using the scalar field values at each edge vertex linearly interpolate the isosurface intersection.

The main disadvantage of the algorithm is that some patterns in figure 1 are topologically ambiguous as noted by van Gelder and Wilhelms [vGW94, pages 343 – 344]. This may produce a surface with a hole as pointed out by Düurst [Düu88] (see figure 2).

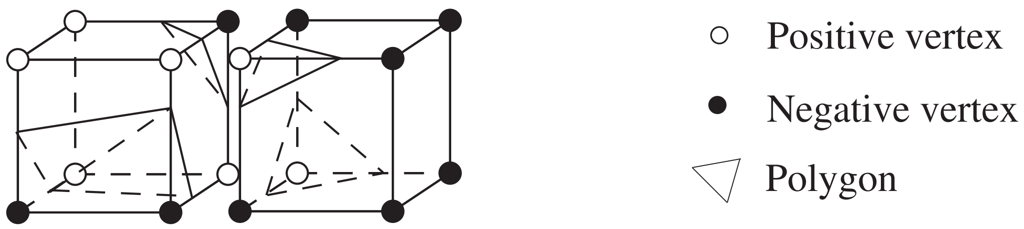

Figure 2: A hole in the polygonization because of a face ambiguity.

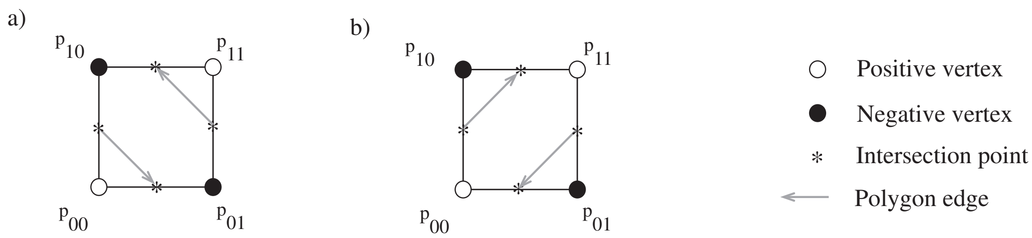

Figure 3: An ambiguous face.

Nielson and Hamann [NH91a] present a solution to the problem. They recognize that the ambiguities in the cube patterns are caused by ambiguous faces of the cube. Figure 3 shows a face with an ambiguous connection of its edge intersections. The authors achieve a disambiguation by bilinearly interpolating the scalar field ρ over the face:

$$B(s,t) = (1-s, s) \begin{pmatrix} \rho(p_{00}) & \rho(p_{01}) \\ \rho(p_{10}) & \rho(p_{11}) \end{pmatrix} \begin{pmatrix} 1-t \\ t \end{pmatrix} \tag{1}$$

The topology of the isosurface of the bilinear interpolant is determined by interpolant's value at the intersection of its asymptotes $\frac{dB}{ds}|_{t=T_\alpha} \equiv 0$ and $\frac{dB}{dt}|_{s=S_\alpha} \equiv 0$:

$$B(S_\alpha, T_\alpha) = \frac{\rho(p_{00})\rho(p_{11}) - \rho(p_{10})\rho(p_{01})}{\rho(p_{00}) + \rho(p_{11}) - \rho(p_{01}) - \rho(p_{10})} \tag{2}$$

Figure 4 shows the bilinear interpolant for a face with the scalar field values $\rho(p_{00}) = \rho(p_{11}) = 0.75$ and $\rho(p_{01}) = \rho(p_{10}) = -1.25$. The bilinear interpolant at the intersection of its asymptotes is $B(S_\alpha, T_\alpha) = -0.25$. Since the value has the same sign as $\rho(p_{01})$ and $\rho(p_{10})$ the isosurface of the bilinear interpolant (the bold curves in the figure) can not cross the diagonal $\overline{p_{01}p_{10}}$.

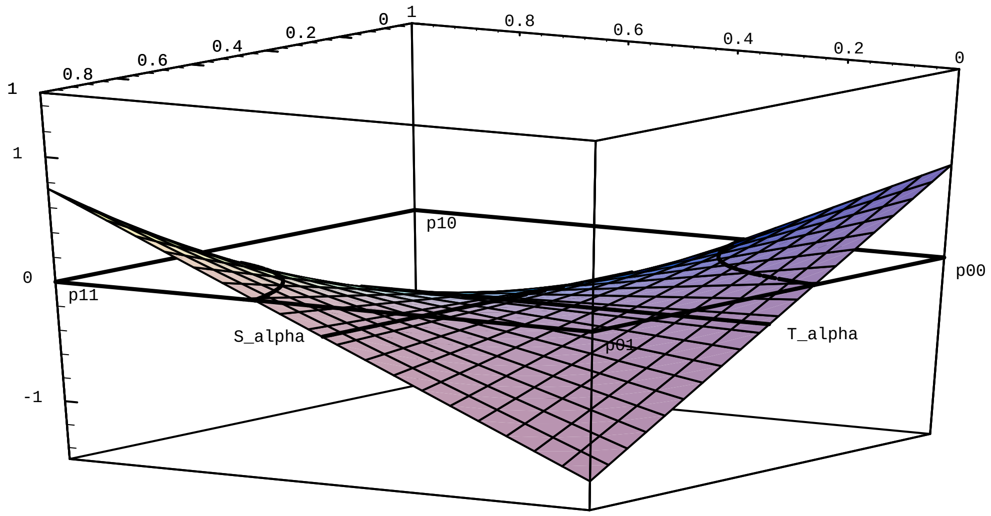

Figure 4: The bilinear interpolant over a face with its isosurface (face intersection) and its asymptotes $\frac{dB}{ds}|_{t=T_\alpha} \equiv 0$ and $\frac{dB}{dt}|_{s=S_\alpha} \equiv 0$.

The authors take the isosurface of the bilinear interpolant as an approximation to the isosurface of the scalar field ρ. If the value $B(S_\alpha, T_\alpha)$ is negative the topology of figure 3 (a) is correct, otherwise the topology of (b) is correct.

Mackerras shows that the test with equation 2 can be replaced by sorting the four intersection points along one coordinate [Mac92]. The first pair and the last pair of the sorted points are connected.

The number of possible topologically different triangulations for a pattern from figure 1 depends on its number of ambiguous faces. As an example consider the pattern 4, where only the front face is ambiguous. The two possible triangulations are shown in figure 5.

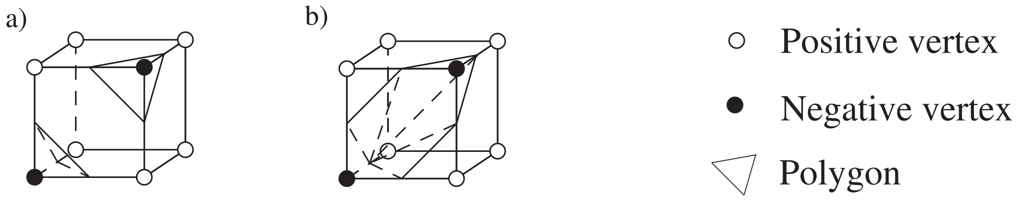

Figure 5: Two topologically different triangulations for an ambiguous pattern.

The above presented method of Nielson and Hamann is in the spirit of the original Marching Cubes algorithm (i.e., no additional sample points are necessary) but considerably more complicated. The pattern 14 in figure 1, e.g., has 6 ambiguous faces and therefore $2^6 = 64$ topologically different triangulizations, which however, similar to the Marching Cubes table can be reduced to 10 distinct cases. Another disadvantage is that some of the patterns can only be triangulized by inserting an extra point.

The Marching Cubes algorithm can be simplified by keeping all 255 pattern in memory and accesing them directly [HH92, Mac92]. This saves the time for computing the correct pattern in figure 1. The resulting memory overhead is on insignificant on modern machines.

Several interesting modifications of the Marching Cubes algorithm exist. Montani et al. [MSS94] discretize the Marching Cubes algorithm to facilitate a mesh reduction postprocessing step (see subsection 5.2). Gallagher and Nagtegaal [GN89] generalize the Marching Cubes algorithm for irregular grids of sample points as often occur in finite-element analysis. They use bicubic polynomials to approximate the isosurface and polygonize the bicubic patches only for rendering. Howie and Blake [HB94] provide the same generalization, but use a cell propagation method to approximate the isosurface with triangle strips. They do not solve the ambiguity problem but instead fill the holes resulting from discontinuities between adjacent cells.

3.2 Data Structure for Soft Objects

Wyvill, McPheeters and Wyvill report a polygonization method designed for soft objects [WMW86b, WMW86a, WWM87, BW90] but which can be readily applied to polygonize general isosurfaces.

The authors construct a polygon mesh in two distinct stages. In a first step they partition the space occupied by the isosurface with a three dimensional cubic grid. In the original application the authors start with a set of seed cubes, at least one for every disconnected component. Starting at the seed cubes, they track the surface by cell propagation: if a cube is intersected by the isosurface the process continues for each cube neighboring an intersected face. A hash table is used to prevent cells being revisited during recursion.

In general for a given scalar field and an isovalue a set of seed cubes is not known and therefore the whole grid must be scanned.

In the second stage the authors deal only with cubes that are intersected by the surface. They construct a local polygonal approximation to the isosurface by linearly interpolating the intersection points of the isosurface with the edges of the cube. The intersection points on a face are connected to give polygon edges. Ambiguities are resolved by considering the center point of a face. The scalar field value at the center is estimated as the average of the vertex values of a face. Figure 6 illustrates the seven possible cases.

This calculation is consistent across adjacent cubes with shared edges. By tracing the natural successors of each polygon edge the authors construct topological polygons. Since the resulting topological polygons are in general not planar the authors divide them into triangles by connecting each polygon vertex to the central average of the topological polygon.

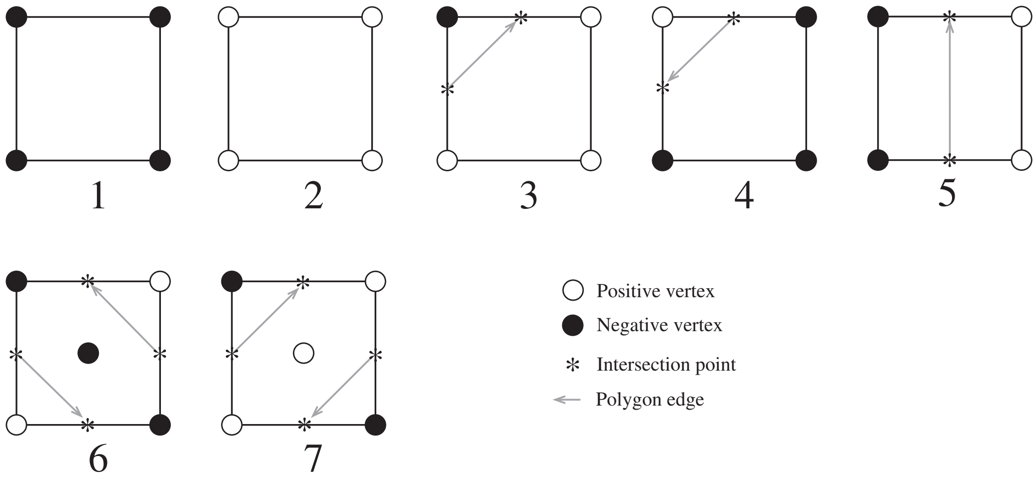

Figure 6: Seven different cases for connecting intersection points.

3.3 Adaptive Polygonization of Implicitly Defined Surfaces

Hall and Warren [HW90] report an adaptive polygonization method that performs a tetrahedral subdivision of space. The authors maintain a tetrahedral honeycomb of space at all time. A polyhedral subdivision of space forms a honeycomb if every face is shared by at most two polyhedra. Because the recursive subdivision of a single tetrahedron might cause the honeycomb property to be lost, the method partially subdivides the neighbors of that tetrahedron to maintain the property. The adaptive subdivision algorithm decides independently for each tetrahedron in the honeycomb whether it should be recursively subdivided. The algorithm defers processing of tetrahedra not recursively subdivided until it has considered all tetrahedra. It then makes a second pass through the list of unsubdivided tetrahedra. For each tetrahedron, the algorithm checks each edge to see if it must be subdivided. The faces and polyhedra are then split according to the number of subdivided edges of each face.

In the second stage of the algorithm for each polyhedron the isosurface inside it is approximated by polygons. The authors determine first for each edge of a polyhedron the intersection point with the isosurface by successive linear interpolation. The resulting intersection points are connected by one or two triangles. To ensure continuity the authors compute the edge intersections once and store them into a hash table.

3.4 Polygonization of Implicit Surfaces

Bloomenthal [Blo88] detects an isosurface by partitioning the domain of the implicit function with an octree, which may either converge to the surface or track it. The polygonal surface approximation is derived from the octree. Bloomenthal reports three steps:

- Spatial partitioning: Bloomenthal considers two methods for sampling the implicit surface. The first method represents the implicit surface as an octree, which is a hierarchical partitioning of space formed by subdivision of cubes, beginning with a cube that bounds the surface. The octree converges to the surface by subdivision of those cubes that intersect the surface. A disadvantage is that small surface details may be missed by a large cube, resulting in a premature termination in the subdivision of the cube. This drawback is overcome by the second method which tracks the surface by cell propagation. This is the same technique as used by Wyvill et al. (see subsection 3.2).

  In both cases, the author evaluates the scalar field ρ at each of the cell's vertices. Only those cells that intersect the surface are retained in the partitioning. Bloomenthal determines the intersection points of the cell's edges with the isosurface by root search. To ensure continuity between polygons Bloomenthal refers to Wyvill's method of storing the intersection points in a hash table. Alternatively he suggests keeping for each cell eight pointers to its vertices. As new cells are created, they must point correctly to shared vertices.

- Adaptive refinement of the octree: Bloomenthal improves the estimation of the surface by subdividing those cubes containing elements of high curvature.

- Polygonization of the octree nodes: The final surface approximation is obtained by polygonizing the octree nodes (final subdivision cells).

For each cube to be processed, the intersection points are ordered, forming a convex polygon whose sides are each embedded in a cube face [WMW86b]; the process is local to each cube. Bloomenthal introduces a simple algorithm, illustrated in figure 7 to perform the three-dimensional ordering of intersection points. The ordering begins with any intersection point on the cube and proceeds towards the positive vertex and then clockwise about the face to the right until another intersection point is reached.

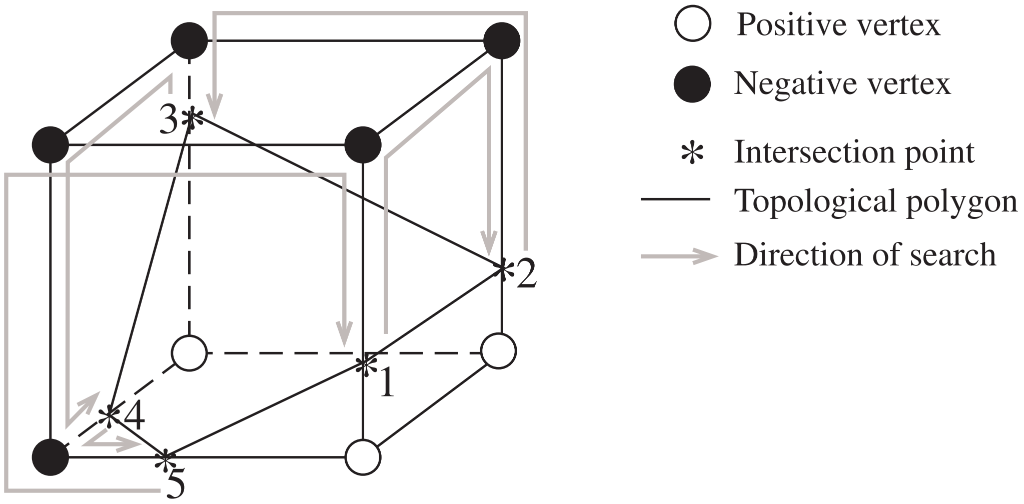

Figure 7: Algorithm to order vertices.

The (topological) polygons resulting from this method are decomposed into triangles. Note that the adaptive subdivision may destroy the honeycomb property of the spatial partition. Bloomenthal ensures continuity between subspace polygons, by tracking the edges of the topological polygon along the more highly divided face (the light grey vertices in figure 8). He resolves ambiguities by taking the central average as an additional sample.

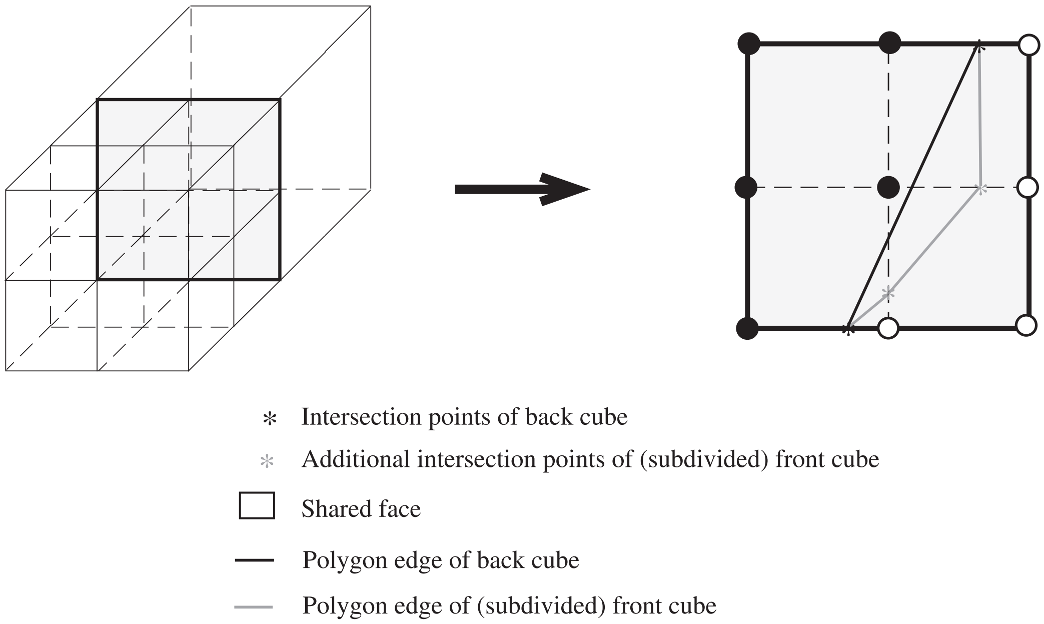

Figure 8: To guarantee continuity Bloomenthal forms polygon edges by always tracking along the more highly divided face.

4 Analysis of Polygonization Algorithms

In this section we analyze the methods presented above and extract a common framework. Three aspects are found:

1. Polyhedral subdivision of space
2. Subspace polygonization (approximating the isosurface inside a cell)
3. Ensuring continuity

4.1 Space Subdivision

During subdivision of space most methods maintain a honeycomb, the 3D analog of a tessellation. The honeycomb guarantees that linear functions defined over a polyhedron form a continuous surface.

The simplest honeycomb used is an array of cubes (e.g., [LC87, WMW86b]). Bloomenthal [Blo88, BW90] and Ning and Hesselink [NH91b] use an octree to achieve an adaptive subdivision based on cubes.

As noted by Bloomenthal [Blo88] vertex locations and face planes are computed more simply if the cells are identical and similarly oriented. In three dimensions, the only such cell that fills space is the cube. Also it enjoys a number of rotational symmetries, and divides into eight similarly oriented cubes. Note though, that a honeycomb can be maintained only by dividing all cubes of the array. Bloomenthal [Blo88] avoids this problem by tracking the isosurface approximation along the more highly divided face.

An additional disadvantage of a cubical cell is that its positive and negative vertices can not be separated by a single plane. This may lead to ambiguities during the second stage of the corresponding polygonization algorithm and may ultimately result in discontinuities for the polygonized surface (see subsection 3.1).

In contrast, the vertices of a tetrahedron can always be separated by a single plane, thereby avoiding ambiguities during the polygonization. Also a tetrahedron can be subdivided into tetrahedra without subdividing its faces. This allows for a local subdivision of a tetrahedral honeycomb. The use of tetrahedral subdivision for an adaptive polygonization is discussed in [HW90].

An easy tetrahedral subdivision is achieved by using a grid of cubes and dividing each cube into 5 [PT90, HW90, DK91, NFHL91, GH95] or 6 tetrahedra [KDK86, NFHL91]. André Guéziec and Robert Humme [GH95] suggest a compact data structure and efficient look-up tables for tetrahedra resulting in an fast and topologically correct polygonization method. If the five-tetrahedral decomposition is chosen it must be mirrored between face adjacent cubes. A resulting problem is, that if linear interpolation is used a spiky surface approximation may result as figure 9 shows.

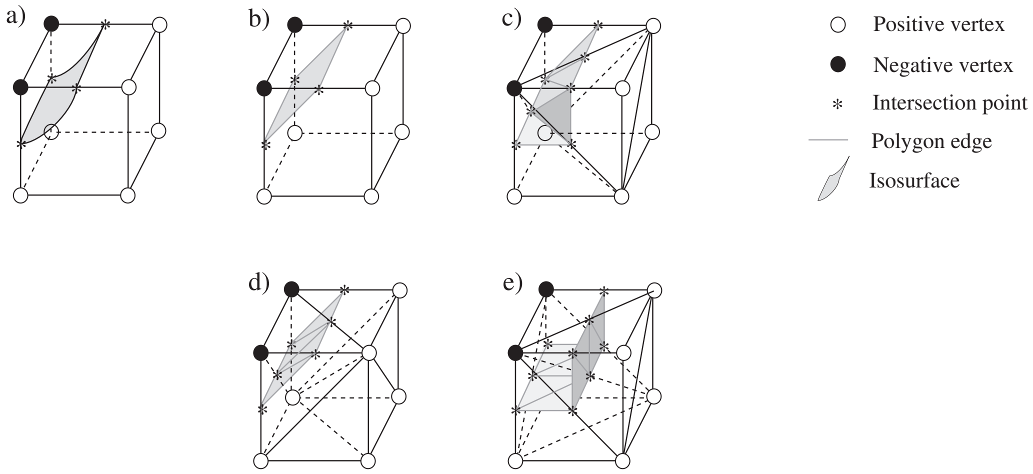

Figure 9: An isosurface intersection with a cubic cell (a) is approximated using linear interpolation (b). Decomposing the cell into five tetrahedra leads to a spiky approximation. Part (c) of the pictures shows half a spike. The full spike becomes visible if mirroring the cube at its front face. The two possible decompositions into six tetrahedra (d) and (e) lead to a smoother approximation.

The six-tetrahedral decomposition does not share this problem and can be consistently applied to all cells, which will be an advantage for parallel execution. Bloomenthal [Blo88] subdivides a cube into 12 tetrahedra by taking the cell's center as an additional sample.

A general tetrahedral subdivision in n-space is commonly called Delaunay triangulation. Bowyer [Bow81] gives an efficient solution which is used by Petersen et al. [PPW87]. A comprehensive bibliography of Voronoi diagrams, the dual of Delaunay triangulation, and related structures is given by Aurenhammer [Aur91].

It is interesting that the existing literature hardly mentions the use of a space subdivision with general polyhedra. Wünsche achieves such a subdivision in a specialized case with BSP trees [Wün96]. General polyhedra have the advantage that every polyhedral partition of space can easily be transformed into a honeycomb. The transformation is done by subdividing each face that faces more than one polyhedral face. After the subdivision of the face the resulting object is still a polyhedron, but has several coplanar faces. Figure 10 gives an example.

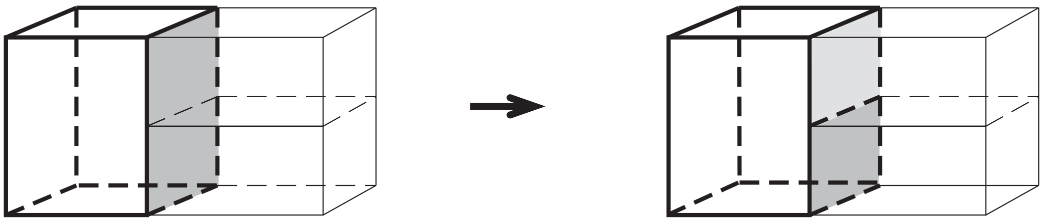

Figure 10: Transforming a polyhedral subdivision into a honeycomb.

4.2 Subspace Polygonization

The second step of a polygonization methods approximates the isosurface inside a polyhedral cell by polygons Two different methods of this subspace polygonization step are used by the reviewed algorithms.

Lorensen and Cline [LC87] use a binary classification of the cells' vertex values to index a precomputed table yielding a set of polygons for the cell. The polygon vertices are given by the intersection points of the cell's edges with the isosurface. The other methods first precompute these intersection points and use them to determine the isosurface intersection with each face. By tracing the resulting edges topological polygons are formed. In a final step the topological polygons are divided into planar polygons.

The subspace polygonization can be simplified and disambiguated by decomposing a cube into tetrahedra. Several authors [NB93, Nie95, GH95] report that this decomposition produces 150% – 250% more triangles then the original Marching Cubes method. Bloomenthal [Blo94] shows this with a graphical example.

4.2.1 Computing intersection points

All reviewed polygonization methods compute intersection points of the cells' edges with the isosurface. An intersection point exists if the scalar field has opposite sign at the end points of the edge.

If binary vertex values are used or exactness is not important the intersection point is most easily approximated as the midpoint of the edge (e.g., [MSS94]). A more exact method is to use linear interpolation (e.g., [LC87, WMW86b]). Bloomenthal [Blo94] shows that this can produce ragged unnatural approximations.

If the sample values are computed from a known underlying function, an alternative to simple interpolation is to employ a root search. This is computationally more expensive but leads to a better polygonization for smooth functions. Here a potential problem exists if an edge has multiple isosurface crossings. A unique edge isosurface intersection is achieved by computing each intersection point once and then referring to it by pointers or by a hash table [Blo88].

If the dataset is available only as a discrete grid of sample values the underlying function must be approximated by using some form of interpolation. Marschner and Lobb [ML94] discuss a wide range of possible interpolation methods ("reconstruction filters"). This technique can be understood as generalization of the above mentioned linear interpolation of the intersection point.

Linear interpolation (or using the midpoint) has the advantage that it is symmetric, which means neighboring cells, sharing the same edge, share the same intersection point. However, even in this case computing an intersection point only once is advisable for efficiency reasons [WMW86b].

Note that linear interpolation does not work if two adjacent cells have collinear edges of different lengths. However, polyhedral subdivisions with this property are not common since they are prone to various discontinuity problems [Wün96].

4.2.2 Forming a topological polygon

Topological polygons are most easily formed by tracing the natural successor of an edge. For an efficient implementation Wyvill [WMW86b] gives a polygon edge an orientation and stores it in an array indexed by its start point. The end point gives than the index for the start point of the next edge.

Bloomenthal connects intersection points by tracing along the cell boundaries [Blo88]. This method was explained in subsection 3.4 (see figure 7).

In both cases the orientation of a polygon edge is such that its start point lies between a negative and positive vertex if traversing the face vertices in anticyclic order (see also [Wün96]).

Savchenko and Pasko [SP95] use the edge intersections to form a connection graph. The cycles in this graph give the topological polygons.

If unoriented polygon edges are used the resulting topological polygon might be falsely oriented. However, it is easy to check the sign of an outside vertex and change the orientation of the polygon if necessary. Doi and Koide use a determinant test for tetrahedral cells [DK91], which Guéziec and Hummel replace by a look-up table [GH95].

4.2.3 Subdividing a topological polygon

The subdivision of a topological polygon is usually not unique. The easiest solution is to triangulate the topological polygon with its center [WMW86b, Wün96]. Wallin [Wal91] connects two consecutive edges of the polygon to form triangles, removes them from the polygon, and applies the procedure recursively to the remaining polygon until he encounters a triangle on the cell boundary. Then he connects the remaining vertices to their center. Ning and Bloomenthal present a short discussion of triangulation of topological polygons [NB93]. Two good triangulation criteria to obtain a smooth surface are given in [CSYL88] and [Mat94].

Finally for the Marching Cubes algorithm the triangulation of a topological polygon is defined implicitly by the triangulated cubes shown in figure 1.

4.3 Ensuring Continuity

The third aspect of a polygonization algorithm is to guarantee surface continuity. We identify three places at which continuity is an issue: on shared edges, on shared faces and inside a cell.

All reviewed algorithms proceed by ensuring the following sufficient conditions in this sequence:

1. Cells that meet at a common edge share a common intersection point.

2. Cells that meet along a common face share common polygon edges.

3. The subspace polygonization inside a cell is continuous, i.e., every polygon edge inside a cell is shared by a neighbored polygon.

The first condition can always be fulfilled (for a honeycomb) by computing the edge intersections by linear interpolation (subsection 3.1). Another approach, taken by the other three presented algorithms, is to precompute edge intersections only once and use them for all cells sharing that edge.

Given condition one, the second condition is trivially fulfilled for a tetrahedral honeycomb (subsection 3.3). For a non-tetrahedral honeycomb there may be more than two intersection points, leading to ambiguities. Wyvill et al. (subsection 3.2) solve the ambiguities explicitly by taking the central estimate of a face (see figure 6 case 6 and 7). Wünsche shows that this method can also be used for general convex polyhedra. However, in this case the center of all intersection points with the face edges must be used for disambiguation [Wün96].

Lorensen and Cline overlook ambiguities in their original implementation, which results in possible discontinuities. Howie and Blake [HB94] fill the resulting hole (see figure 2) with two triangles. Since ambiguities are not resolved the resulting surface is not unique. Also for every ambiguous face the neighboring cell must be checked for a hole.

Nielson and Hamann [NH91a] present a better solution which is based on bilinear variation of ρ on an ambiguous face. Their disambiguation criterion was further simplified by Mackerras [Mac92] (see subsection 3.1). Matveyev gives a topologically correct approximation to the isosurface obtained by trilinear interpolating over the cell. Natarajan also uses a trilinear interpolant and reports a 20 % slower execution time than the original Marching Cubes algorithm [Nat91].

A simple solution to solve ambiguities on a face is to use the central estimate of the face [WMW86b, Wal91]. Baker [Bak88, Bak89] and Kalvin [Kal91] assume 6 adjacency for the positive vertices, and therefore always connect negative vertices. The same method is chosen by Lorensen [Lor96] and Oh and Park [OP96]. Their approach gives 6 additional configurations to the 14 original Marching Cubes configurations.

Kalvin reports that this method is best applied to binary and segmented volumes and is usually faster than methods based on resampling and interpolation [Kal92].

Van Gelder and Wilhelms show that disambiguation based on linear interpolation can fail for a quadratic scalar field. They suggests a computationally rather expensive tricubic interpolation or two different gradient heuristics [vGW94].

Wünsche solves ambiguities for arbitrary convex faces by resampling at the center of the intersection points with the face edges [Wün96].

Bloomenthal (subsection 3.4) offers a different solution. His subdivision does not have the honeycomb property. However, he knows that two faces facing each other are either the same or one is the subdivision of the other. By always computing the isosurface intersection for the more highly divided face he gains continuity.

Polygonization methods guaranteeing a surface without artifacts such as holes are called *topologically consistent*. Polygonization methods using a disambiguation that matches some assumed interpolant are called *topologically correct*.

At this point the four reviewed algorithms result in a set of contours lying on the cell faces. Condition one guarantees that they form closed topological polygons. All of the above algorithms conclude by dividing the topological polygons into planar polygons (triangles), maintaining the third continuity condition.

The properties of the reviewed algorithms are summarized in table 1. The original and modified Marching Cubes algorithm are listed here separately.

4.4 Quality Criteria

Quality criteria for polygonization algorithms are usually dependent on the application. However, van Gelder and Wilhelms [vGW94] suggest a set of

		Lorensen & Cline (subsection 3.1)	Nielson et al. (subsection 3.1)	Wyvill et al. (subsection 3.2)	Hall & Warren - (subsection 3.3)	Bloomenthal (subsection 3.4)
	Type of cells	Cubes	Cubes	Cubes	Tetrahedra	Cubes
1	Honeycomb	Yes	Yes	Yes	Yes	No
	Adaptive subdivision	No	No	No	Yes	Yes
	Discretized input	Yes	Yes	Yes	No	No
	Ambiguities	Yes	No	No	No	No
2	Continuous surface	No	Yes	Yes	Yes	Yes
	Computation of intersection points	linear interpolation	linear interpolation	linear interpolation	root search (regula falsi)	root search
	Continuity at shared edge	Interpolates shared edge linearly	Interpolates shared edge linearly	Compute edge intersections only once	Compute edge intersections only once	Compute edge intersections only once
3	Continuity at shared face	(no continuity)	Has honeycomb and resolves ambiguities	Has honeycomb and resolves ambiguities	Has tetrahedral honeycomb	Computes face intersections only once
	Disambiguation	(not resolved)	bilinear variation	central estimate	(no ambiguities)	form tetrahedra with cell center

Table 1: Comparison of the reviewed polygonization algorithms.

desirable features of a general-purpose polygonization method. We will repeat them here because it is interesting to see how our analyzed algorithms fulfill them:

1. The algorithm should yield a continuous surface. Each polygon edge should be shared by exactly two polygons or lie in an external face of the entire volume.

2. The isosurface should be topologically correct when the underlying function is “smooth enough”.

3. The isosurface produced should be neutral with respect to positive and negative sample data values (relative to threshold). Multiplying the samples (and threshold) by -1 should not alter the surface.

4. The algorithm should not create artifacts not implied by the data, such as bums and holes.

5. The algorithm should be fast.

6’. The isosurface should be a continuous function of the input data. A small change in the threshold value or some data value should produce a small change in the isosurface.

We think the last point is difficult to quantify, since it can not be fulfilled if the underlying scalar field undergoes a topological change. Instead in an attempt to capture the notion of algorithmic “elegance” we take the feature

6. The algorithm should be easy to understand (and therefore easy to implement).

Table 2 shows which quality criteria the presented algorithms fulfill. The classification of the implementation difficulty should be understood as a relative measure and is just our rough estimation based on our own experience.

	Quality criteria	1. Lorensen & Cline (subsection 3.1)	2. Nielson et al. (subsection 3.1)	3. Wyvill et al. (subsection 3.2)	4. Hall & Warren (subsection 3.3)	5. Bloomenthal (subsection 3.4)
1.	Continuous surface	No	Yes	Yes	Yes	Yes
2.	Topologically correct[a]	Yes	Yes	Yes	Yes	Yes
3.	Neutral to sample values	No[b]	No	Yes	(unknown)[c]	
4.	Free from artifacts	No	Yes	Yes	Yes	Yes
5.	Speed	Van Gelder and Wilhelms [vGW94] report similar speed for 1. and 3.			(unknown)	(unknown)
6.	Implementation	easy	medium	medium	hard	hard

[a] But what is "smooth enough"?
[b] Case 12 in figure 1.
[c] With some extra effort this property can be achieved.

Table 2: Quality criteria of the reviewed polygonization algorithms.

We conclude this section with some remarks regarding adaptive subdivision. Adaptive subdivision results from the desire to approximate the isosurface both accurately and efficiently. This means the polygonization method must sample the function closely. In the process the algorithm may sample heavily in areas where the function is nearly linear. The solution is to sample adaptively, i.e., sampling more closely near highly curved portions of the surface. This is achieved by recursively subdividing a cell if the isosurface within the cell is more than some user-defined tolerance away from being planar.

5 Optimization Techniques

All above described polygonization methods subdivide a scalar field into cells. The resulting grid can consist of millions of cells and the resulting polygonization can consists of hundreds of thousands of polygons. Three optimization objectives exists: first it is desirable accelerate the polygonization method. Subsection 5.1 introduces three classes of speed-up methods which follow different goals. Secondly, for a real-time interactive display of the polygonization, it is necessary to reduce the number of polygons. This can be achieved by a mesh optimization technique as a post-processing. Subsection 5.2 describes several techniques classified by their design objective. Finally parallelization is an increasingly important optimization goal since modern supercomputers are often highly parallel. Some recent results are summarized in subsection 5.3.

5.1 Speed-up Techniques

Conventional polygonization methods subdivide a scalar field ρ into $O(n)$ cells and visit them all. For large volumes these methods become increasingly inefficient, since the number of intersected cells for a 2-manifold isosurface is $O(n^{\frac{2}{3}})$. Van Gelder and Wilhelms [vGW94] report that between 30% and 70% of the time spent in isosurface generation is spent examining empty cells.

Several methods have been proposed to remedy this situation. We subdivide them into three classes:

1. **Information based methods** use information about the scalar field to avoid traversing cells that are not intersected by the isosurface.

   The most common information based method is cell propagation (also called contour tracing) [AFH80, WMW86b, KDK86, Blo88, DLTW90, Blo94, IK94, HB94]. This algorithm needs seed cells intersecting the isosurface and follows the isovalues over the cell boundaries. In order to extract the complete isosurface one seed cell must be known for every disconnected component of the isosurface.

   Artzy et al. [AFH80] obtain seed cubes by user input. Bloomenthal [Blo94] assumes a connected isosurface and finds a seed cube by random search. Wyvill et al. [WMW86b] assume a scalar field defined by a special geometric model. Each component of their model is enclosed by the isosurface. The authors find a seed cube for every disconnected component of the isosurface by casting a ray from every component of the underlying model. Howie and Blake [HB94] optimize the cell propagation technique further by producing triangle strips whenever possible during the propagation. The authors report that triangle strips are more compact in storage and are rendered 2.1 to 2.2 times faster than the corresponding discrete triangles.

Wünsche [Wün96] polygonizes a scalar field defining a quasi-convolutionally smoothed object. He uses the information about the underlying geometric object to define a BSP tree to subdivide the scalar field and to extract all cells intersected by the isosurface.

2. **Adaptive methods** use a coarse and fast initial subdivision and refine it only where the surface is interesting. Fine surface details might be missed by the initial subdivision step.

   Bloomenthal [Blo88, BW90] constructs an initial cubic mesh by octree subdivision or surface tracking (see subsection 3.4). He subdivides a cube based upon object characteristics, such as tangency and curvature.

   Hall and Warren [HW90] construct an initial tetrahedral grid which on subdivision remains a honeycomb property at all times. The subdivision criteria is an estimate of the surface curvature.

   Beier [Bei90] uses a Marching Cubes style algorithm with a rather coarse cubic grid. The resulting triangles are subdivided by dividing its edges. For edges that are relatively flat, i.e., the angle between the normals in its end point is small, the edge midpoint is chosen. Otherwise a new point on the isosurface is calculated.

3. **Preprocessing methods** examine all cells in a preprocessing step and store information about the subdivision in a suitable data structure. The information is used in the isosurface extraction step to avoid visiting non-intersected cells. The isosurface extraction step is therefore usually an information based method. Preprocessing methods often have a large memory overhead but prove useful if several isosurfaces must be found in a scalar field.

   Itoh and Koyamada [IK94, IK95] construct an extrema graph for the scalar field ρ. For each isosurface value the extrema graph is used to find a number of seed cells. The isosurface is then be extracted by cell propagation. The authors report a speed up of 2 – 10 compared with Doi and Koide [DK91].

Giles and Haimes [GH90] form two ordered cell lists in a preprocess by sorting the cells' maximum and minimum values. For each isosurface value they determine an active cell list containing intersected cells. If a new isosurface value is specified only the active cell lists must be updated. The algorithm exploits space coherence and is efficient if the isosurface value changes smoothly.

Gallagher [Gal91] groups the cells according to their range of scalar values and subdivides each group according to the cell's minimum value. For each isosurface value only subgroups of the resulting data structure are visited. The algorithm is sensitive to clustering and performs best if the scalar field values are evenly distributed.

Shen and Johnson [SJ95] order the cells by minimum and maximum values and identify for a given isovalue the minimum and maximum index of the intersected cells in the ordered lists. If a new isosurface is chosen only a part of the cell list must be visited depending on the new isovalue and the current minimum and maximum index. The minimum and maximum index are updated for each new isovalue. This part of the algorithm is called the *sweep algorithm* and performs best if the isosurface value changes smoothly. To improve worst case performance the authors recursively subdivide the list of cells into subgroups at different levels according to their range of vertex values. For each level the sweep algorithm is then only applied to the subgroup containing the isovalue.

Wilhelms and van Gelder [WvG92] use an octree for faster isosurface extraction. In a preprocessing step they store at each node the minimum and maximum vertex values found in that node's subtree. For isosurface extraction only branches are explored with the isosurface value between the minimum and maximum value of the branch.

Livnat, Shen, and Johnson [LSJ96] give an excellent overview of preprocessing methods. They classify the methods into algorithms that decompose the geometric space, i.e., the cells, and algorithms that decompose the value space, i.e., the range of voxel values. (see table 3 for examples).
Algorithms decomposing the value space have the advantage that the underlying geometric structure is of no importance and therefore the

approach also works well for unstructured grids. Livnat et al. recognize that the search in value space is equivalent to a search in two dimensions. To find cells intersected by the isosurface it is sufficient to know the minimum and maximum vertex value of a cell. These values define a point in a 2-D space, the so-called *span space*. Figure 11 shows the span space for an isovalue c in grey shade and the cells as black dots according to their minimum and maximum vertex values.

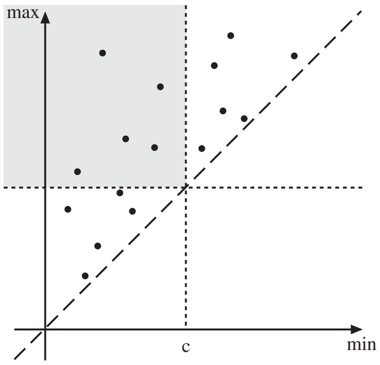

Figure 11: Search over a span space.

The authors now find a nearly optimal speed up method by subdividing the span space with a Kd-tree (see [Ben75]). If n is the number of cells in a given subdivision of space, and k is the number of intersected cells, then the preprocessing step, i.e., subdivision with the Kd-tree, takes $O(n \log n)$ time and for any given isovalue the intersected cells are found in at most $O(\sqrt{n}+k)$ time. Note that the latter time complexity is for most cases optimal, since a 2-dimensional surface in 3-D space usually intersects $O(n^{\frac{2}{3}})$ cells and $O(\sqrt{n} + n^{\frac{2}{3}}) = O(n^{\frac{2}{3}})$

5.1.1 Comparison

We conclude this section with a comparison of preprocessing methods. Most of our results are taken directly from Livnat et al. [LSJ96]. Again n gives the number of cells, and k the number of cells intersected by the isosurface. Table 3 differentiates the algorithms introduced above by the following criteria:

1. Type of the search space. As explained above a preprocessing method decomposes and searches either the geometric space or the value space.

2. Is the method suitable for an unstructured grid?

3. Time complexity of the preprocessing step.

4. Time complexity of a search operation to identify all cells intersected by an isosurface. For methods which only find seed cells (e.g., [IK94]) this also includes a $O(k)$ term reflecting the minimal time needed for cell propagation.

5. Space complexity of the preprocessing step.

6. Can coherence between isosurfaces be exploited?

	Search space	Unstruc. grid	$t_{preprocessing}$	t_{search}	Space	Coherence exploited
[IK94]	Geometric	Yes	$O(n)$ best case	$O(n^{\frac{2}{3}})$ best case $O(n)$ worst case	$O(n)$	No
[GH90]	Value	Yes	$O(n \log n)$	$O(n)$ worst case	$O(n)$	Yes
[Gal91]	Value	Yes	$O(n)$ best case	$O(n)$ worst Case	$O(n)$	No
[SJ95]	Value	Yes	$O(n \log n)$	$O(n)$	$O(n)$	Yes
[WvG92]	Geometric	No	$O(n \log n)$	$O(k \log \frac{n}{k})$ worst case	$O(n \log n)$	No
[LSJ96]	Value	Yes	$O(n \log n)$	$O(\sqrt{n} + k)$	$O(n)$	No[a]

[a]The authors offer a neighborhood search as an extension to their method, which exploits coherence between isosurfaces, but mention several disadvantages of this approach.

Table 3: Classification of preprocessing methods.

The above results indicate that subdividing the value search space might give better results than subdividing the geometric search space. Many previous solutions in the former category, however, were difficult to understand

and to implement. The solution from Livnat et al. [LSJ96] seems to represent the most efficient solution without being too complex.

5.2 Mesh Reduction Techniques

Large meshes are common in computer graphics, for example when using devices such as CT, MRI, range cameras, or satellite data. Since large meshes put a strain on storage capacity, communication, and rendering hardware a mesh reduction algorithm must often be applied. The chosen technique depends on the application. Whereas some techniques aim only to eliminate small and badly shaped polygons, other technique try to achieve a maximum reduction of polygons. Often the user wants to get a fast preview of the mesh with the choice of increasing resolution (*multiresolution surface meshing*) or the user wants to see only selected parts of the model in a higher resolution (*local level-of-detail control*). The latter technique is used in digital terrain modeling where far away objects can be approximated less accurately to improve rendering speed. An introduction into mesh reduction techniques is given by Schroeder [Sch95] and a short overview and classification is contained in [Red96]. We identify five classes of mesh reduction techniques:

I. Polygon merging combines coplanar or nearly coplanar polygons into bigger ones. The original topology of the mesh is not changed.

Hinker and Hansen [HH93] merge nearly coplanar polygons and retrace their boundary to eliminate collinear edges. The resulting polygons are triangulated. The so=called *Geometric Optimization* algorithm is best suited for objects with gradually changing gradient. The authors mention that the algorithm is parallelizable and that its runtime is dominated by a $O(n \log n)$ function.

Kalvin and Taylor [KT96] employ a similar technique but additionally limit the approximation error. The authors start with an initial seed face and employ a greedy strategy to merge it with bordering faces. The borders of the resulting *superfaces* are then straightened and the resulting faces are triangulated. Kalvin and Taylor claim that their algorithm is more efficient than Hinker and Hansen's because they employ a $O(n)$ greedy strategy to merge n faces in contrast to Hinker and Hansen's $O(n \log n)$ merge step.

Reddy [Red96] suggests a perceptually-driven polygon reduction. The algorithm collapses certain vertices so that they become co-linear with two neighboring vertices and then merges nearly coplanar polygons. The resulting color of a new polygon is found by an area-weighted averaging of the RGB colors of each component polygon. The error measure for the vertex collapse is an approximation to the human *Contrast Sensitivity Function* which considers both size of a polygon and surface curvature. The author segments the model into subvolumes such that only small details are eliminated.

II. Modified isosurface extraction modifies the polygonization method such that a simplified or optimized mesh reduction technique can be employed as a postprocessing step. We present here three algorithms that produce a surface topologically identical to the surface produced by the (modified) Marching Cubes algorithm with a distance error smaller than grid size. A fourth method employs a specialized data structure for a simplified mesh optimization step.

Montani et al. [MSS94] achieve a simplified merging of coplanar faces by changing the isosurface extraction of the Marching Cubes algorithm step in a way that it produces coplanar polygons for nearly planar surface areas. The resulting algorithm is called *Discretized Marching Cubes*. The authors extract the isosurface with a Marching Cubes algorithm but allow only a finite number of positions for the edge isosurface intersections. For example, if a binary discretization is chosen the isosurface can intersect an edge only at its midpoint. The subspace polygonization therefore generates faces which lie on a finite set of planes (13 for the binary approach), thus allowing simple merging of the output faces into large coplanar polygons. The authors report that the merge step takes about 85% of computation time and that a reduction of polygons of up to 90% is achieved.

Oh and Park [OP96] also apply a modified Marching Cubes algorithm. They classify the configurations of the Marching Cubes approach into types according to the orientation of the produced faces. Surface patches in neighboring cubes of the same type are merged to produce fewer and larger triangles. However, the authors mention that not all possible faces are merged because the processing order is sometimes inconsistent with the merging direction of the cubes. The authors report execution time that is about 25% slower than the original Marching Cubes algorithm and a triangle reduction of about 50%, with comparable image quality.

Müller and Stark [MS93] present a Marching Cubes algorithm with inherent mesh reduction. They take as input a cuboidal regular grid and recursively subdivide the cuboid until a grid cell is reached. At each step they compute an isosurface approximation for the current cuboid by taking its eight vertex points and indexing the Marching Cubes table. If the isosurface approximation for a subsequent subdivision is topologically different inside a cuboid from the current isosurface approximation the new approximation replaces the current one. Cracks between boxes of different resolution are prevented by sharing polygon edges between adjacent cells. It is interesting to note that the authors choose here the polygon edge on the least subdivided face if that is topologically correct and only otherwise take the polygon edges on the most subdivided face (as in figure 8). Note that this method is not an adaptive speed-up method, like for example the octree partitioning of Bloomenthal (see subsection 3.4), since the recursion always proceeds to cell level.

Kalvin et al. [KCHN91] use a *winged-edge* data structure to encode the polygonized surface. The polygonization algorithm grows the isosurface slice by slice, in what seems to be a rather complicated process. In a second step the authors merge coplanar faces meeting at a common vertex. The authors report that the winged edge data structure not only guarantees fast and topologically consistent face merging but it also allows efficient manipulation and measuring of the surface.

Cignoni et al. [CFM$^+$94] achieve a polygonal approximation of an isosurface by computing a multiresolution tetrahedralization of the underlying scalar field ρ. The local resolution of the tetrahedralization depends on the gradient of the scalar field and a distance measure. Each level of the tetrahedralization is used to define a polygonal approximation of the isosurface with a Marching Cubes like algorithm e.g., [DK91, GH95] thereby defining a multiresolution mesh.

III. Polygon elimination deletes polygons by collapsing their vertices to single points.

Moore and Warren [MW91, MW92] provide a mesh reduction algorithm for polygonization methods using a polyhedral subdivision. Their method is only valid for triangle meshes but can be extended to general polygon meshes. The motivation of the authors is that badly shaped (thin) polygons often cause undesirable shading artifacts in lightning models and degrade further processing steps (e.g., as input to a finite element method). The aim is to improve the quality of the polygonal mesh without changing its approximation error. The authors record during the subspace polygonization for each triangle vertex the closest cell vertex, called a *satellite*. A postprocessing step deletes all triangles that have two or three vertices with the same satellite (thin and small triangles, respectively). Mesh vertices with an identical satellite are replaced by the average position of the corresponding satellites. Moore and Warren report a polygon reduction of 40% – 60%.

Bernd Hamann [Ham94] orders the set of triangles of a mesh by considering the curvature at the mesh vertices. He then iteratively replaces a triangle with a point given as the origin of a bivariate interpolant. The definition of the bivariate function depends on the local geometry of the mesh, such that, for example, a triangle on the boundary of the mesh is replaced by a point on the boundary. The hole resulting from the triangle removal is retriangulated with the new point.

Reddy [Red96] reports two commercial systems in this class. The GENIE system removes polygons which projection on the screen lies under a given area threshold [Kem93]. The Viper system displays large triangles first and eliminates small triangles if the system becomes overloaded [Hol91].

IV. Vertex or edge elimination deletes vertices or edges and retriangulates the resulting holes.

Schroeder et al. [SZL92] removes vertices that pass a minimum distance (planarity) criterion. The resulting holes are triangulated with a recursive polygon splitting algorithm, which aims for triangles with maximum aspect ratio. The authors also ensure that the topology of the mesh is preserved and they identify sharp edges and corners that must be retained such that the resulting geometry closely resembles the original. Note that the minimum distance criterion measures only the deviation of the new mesh from the old mesh.

Cohen et al. [CVM+96] remove vertices from the mesh and attempt to fill the resulting hole by retriangulation. The authors bound the maximum error of the approximation by restricting it to a *Simplification Envelope* (two modified offset surfaces). They additionally suggest a global algorithm, which finds all three tuples of vertices (candidate triangles) that lie between the offset surfaces. The algorithm orders the candidate triangles in decreasing order and builds a triangulation with a greedy method (see [Var94] for details). The authors claim to achieve a much improved solution for the same error bound if compared with [LDW94, EDD+95].

Hoppe et al. [HDD+93a] collapse, swap, and split edges in order to optimize the mesh. They use an energy function over a mesh to minimize both the distance of the approximating mesh from the original, as well as the number of approximating vertices. In [HDD+93b] the authors prove that their implementation does not change the topological type of a mesh. To achieve a mesh reduction Hoppe et al. first randomly sample the original mesh with additional sample points at boundary edges and then add these points to the original vertex set.

Hoppe [Hop96] presents *Progressive Meshes*. A mesh is simplified with an edge collapse operation similar to [HDD+93a]. A multiresolution mesh is then stored as a base mesh with a number of detail records inverting the edge collapse operations. A soft transition between approximations is achieved by a special morphing operation. The goal of the optimization procedure is not only to preserve the geometry of the original mesh but also the overall appearance defined by its attributes such as color, normals, and texture coordinates. Hoppe achieves this goal by using an energy metric such that an edge collapse for an edge with different attributes has a low priority. A modification of this algorithm [Hop97] allows view-dependent refinement based on view frustum, surface orientation, and screen-space geometric error. Popović and Hoppe [PH97] generalize the Progressive Meshes to *Progressive Simplicial Complexes* which use a more general base model and allow topological changes in the refinement resulting in a better fidelity.

Ronfard and Rossignac [RR96] propose a fast multiresolution scheme for triangulated polyhedra. The authors merge vertices by moving one end point of an edge onto the other. Edges are ordered according to a local error function based on the maximum distance of one end point of the edge to the planes of the polygons containing the other end point of the edge. In order to collapse edges in the right order they are stored in a heap data structure. The authors make some adjustments to merge vertices without changing the shape of the object, but intentionally allow the object topology to change. Ronfard and Rossignac estimate the complexity of their method is $N_0 \log^2 \frac{N_0}{N_L}$ for bringing the number of vertices down from N_0 to N_L.

Algorri and Schmitt [AS96] reduce the number of vertices in a mesh by collapsing edges: each edge is replaced by a vertex at its center. The authors use a planarity-threshold for the dihedral angle between two adjacent triangles to identify geometric features such as corners and regions of high curvature. The identified features are represented by characteristic lines and define clusters in which the mesh is subsequently locally reduced. In a last step the edges of the characteristic lines are reduced by using a collinearity-threshold for the angle between two edges. The advantage of using clusters is that retriangulation operations are locally bounded and also that the geometry and topology of the mesh is preserved. The authors mention that for many object a decimation limit of 80% – 90 % applies because otherwise dense mesh objects with little surface characteristics such as a dense sphere lose their global shape characteristic.

V. Mesh Approximation approximates the polygonization with a coarser one using some error criteria. The original polygonization is used only as an error measure.

DeHaemer and Zyda [DZ91] assume a mesh obtained from a regular grid of sample points. The authors fit trial polygons through a subset of the corners of the mesh and recursively subdivide them into two or four subpolygons until they lie within a given tolerance level to the original mesh. Edges of neighboring polygons are likely to not coincide. The resulting gap is either filled with a polygon (being nearly orthogonal to the original polygons!) or extra vertices are inserted resulting in non-planar polygons. Both methods lead to shading artifacts. The authors suggest as a criterion for subdividing polygons a hybrid technique by either subdividing at the location of the maximum error or at the location of the maximum curvature.

Turk [Tur91] distributes a set of points on a mesh by point repulsion, with density weighted by estimates of local curvature. The old vertices and new points are triangulated such that each polygon of the original model is tiled with the new points lying on it. He then removes the old vertices and uses a greedy triangulation to fill the resulting holes. Several constraints guarantee that the resulting triangulation does not change the surface topology. If a topology preserving triangulation is not possible the old vertex is retained. The method is best suited for models with curved surfaces and is less suited for models with sharp corners.

VI. Multiresolution Wavelet Analysis is used to decompose a simple function into a low resolution part and so-called wavelet coefficients necessary to recover the original function. To apply multiresolution analysis to mesh reduction the mesh is expressed as parametric function. The low resolution part of the function gives then a reduced mesh in which the new vertices are weighted averages of the original vertices. This technique is popular for multiresolution surface meshing. An introduction to wavelets for computer graphics is given by [SDS95a, SDS95b, SDS96].

Multiresolution wavelet analysis for mesh reduction was original introduced by Lounsbery et al. [Lou94, LDW94]. The authors present a new class of wavelets based on subdivision surfaces which can be applied to functions on arbitrary topological domains. The input meshes, however, must have subdivision connectivity, i.e., they are obtained from a base mesh by recursive 4-to-1 splitting. Eck et al. (see below) overcomes this shortcoming. Lounsbery et al. provide also an algorithm to switch smoothly between models of different resolution by treating the wavelet coefficients as continuous function of the viewing distance.

Eck et al. [EDD+95] describe how multiresolution analysis can be applied to approximate an arbitrary mesh. The authors first approximate an arbitrary mesh M by a mesh M^J that has subdivision connectivity and then convert M^J to a multiresolution representation using the methods of Lounsbery et al. . The method provides a guaranteed maximum error to the original mesh.

Gross, Gatti, and Staadt [GSG96, GGS95] present a method for regular surface grids. They transform the initial surface data grid into a quadtree structure and then use a wavelet transform to decide which vertices to remove. The resulting quadtree cells are retriangulated using a table look up. The authors achieve a local level-of-detail control by filtering the wavelet space with a Gaussian ellipse.

Certain et al. [CPD+96] deal with complex colored meshes and use separate multiresolution representations for geometry and color that are combined only at display time. The authors use the wavelet transform of Lounsbery [Lou94, LDW94] to decompose the shape and color functions into a low resolution base mesh and correction terms at increasing resolution (wavelet coefficients). An efficient algorithm and data structure is used to construct higher resolution approximations from the base mesh at interactive rates.

5.2.1 Comparison

We have now introduced a variety of mesh reduction techniques. Unfortunately the literature gives only a few direct comparisons of above methods. In order to convey better understanding of the presented methods we first classify the algorithms according to nine criteria and then summarize some evaluations and comparisons which we have found in the existing literature.

Our criteria for a classification are:

1. Does the algorithm achieve the approximation with a given number of vertices or polygons (*bounded number approximation*)?

2. Does the algorithm achieve the approximation with a given maximum error of ϵ (*bounded ϵ approximation*)? Note that in this case the error is usually measured according to the removal criterion (point 9.).

3. Does the algorithm provide *multiresolution surface meshing* to obtain meshes of different resolution?

4. Does the algorithm provide a *geomorph* to interpolate smoothly between models?

5. Does the algorithm use a subset of the original mesh vertices (vs. resampling)?

6. Does the algorithm preserve the mesh topology? Note that in general topological changes are not desired. However, in order to achieve a coarse approximation a controlled change of the topology can be desirable. Consider for example a metal plate with many small holes. If viewed from a far distance an approximation of the plate without holes is sufficient. Methods which allow a controlled change of the topology are marked with an asterix.

7. Does the algorithm allow a *local level-of-detail control*? This is useful if only selected parts of a model must be approximated in detail.

8. Does the algorithm work for arbitrary meshes (i.e., for a triangular 2-manifold mesh)?

9. Which error measure/removal criteria is used: 1. Distance measure 2. Polygon size 3. Surface curvature (estimated by normal angle) 4. Wavelet coefficient.

The results of classifying the presented algorithms with these criteria are shown in table 4.

The mesh optimization algorithm from Hoppe et al. [HDD$^+$93a] and the mesh decimation method from Schroeder et al. [SZL92] currently appear the most popular. This is probably due to the fact that they are relative simple and also have had time to become established. A lot of recent research has gone into multiresolution wavelet analysis (group VI. in our classification) and the results look very promising.

Schroeder [Sch95] suggests that "Re-tiling" [Tur92] is best for curved, round objects and that "Mesh Optimization" [HDD$^+$93a] appears to give best results, but is much too slow for large meshes. The author mentions that "Mesh Decimation" [SZL92] and "Geometric Optimization" [HH93] appear to be fastest and that the best approach is probably to mix a high-speed algorithm with the optimization approach.

		Criteria								
		1.	2.	3.	4.	5.	6.	7.	8.	9.
I.	[HH93]	No	No	No	No	Yes	Yes	No	Yes	3
	[KT96]	Yes	No	No	No	Yes	Yes	No	Yes	1+3
	[Red96]	No	No	No	No	Yes	Yes	No	Yes	3+2
II.	[MSS94]	No	No	No	No	Yes	Yes	No	No	3
	[OP96]	No	No	No	No	Yes	Yes	No	No	3
	[MS93]	No	No	No	No	No	Yes	No	No	1
	[Kal91]	No	No	No	No	Yes	Yes	No	No	3
	[CFM$^+$94]	Yes	No	Yes	No	Yes	No	No	No	1
III.	[MW91]	No	No	No	No	No	Yes	No	No	2
	[Ham94]	Yes	Yes	No	No	No	Yes	No	Yes	3
IV.	[SZL92]	Yes	No	No	No	Yes	Yes	No	Yes	1
	[CVM$^+$96]	No	Yes	No	No	Yes	Yes	Yes	Yes	1
	[HDD$^+$93a]	Yes	No	No	No	No	Yes	No	Yes	1
	[Hop96]	Yes	No	Yes	Yes	No	No*	Yes	Yes	1
	[RR96]	Yes	Yes	Yes	No	Yes	No*	No	Yes	1
	[AS96]	No	No	No	No	No	Yes	No	Yes	3
V.	[DZ91]	No	Yes	No	No	No	No	No	No	1+3
	[Tur91]	Yes	No	No	Yes	No	Yes	No	Yes	-
VI.	[LDW94]	Yes	Yes	Yes	Yes	No	Yes	No	No	4
	[EDD$^+$95]	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	4
	[GSG96]	Yes	Yes	Yes	No	Yes	Yes	Yes	No	4
	[CPD$^+$96]	Yes	Yes	Yes	Yes	No	Yes	No	Yes	4

Table 4: Comparison of mesh reduction techniques.

Kalvin gives a listing of execution times of different algorithms for different meshes on different machines [KT96]. As the author suggests a direct comparison of the results is not possible, but the indications are that "Superfaces" [KT96], "Mesh Decimation" [SZL92], and "Geometric Optimization" [HH93] are relatively fast and that "Mesh Optimization" [HDD$^+$93a] is relatively slow.

It is worth noting that in order to achieve an extreme reduction of the mesh size a change of the topology might be necessary. A typical example is a plate with many small holes. Table 4 shows that only two algorithm allow a controlled change of the mesh topology.

Note that any incremental mesh reduction algorithm (e.g., [HDD$^+$93a] or [SZL92]) can be transformed to generate a multiresolution mesh with local level-of-detail control. De Floriani et al. [FMPB97] achieve this by encoding a set of mesh fragments and a partial order defined on such fragments. The partial order is described as a directed acyclic graph. Two special nodes, a source with no incoming arcs and a drain with no outgoing arcs, correspond to the highest and lowest resolution mesh, respectively. A mesh of the desired resolution is defined by cutting the graph in a suitable way.

David Luebke and Carl Erikson [LE97] suggest a framework for any mesh optimization algorithm based on vertex collapse operations to obtain a robust view-dependent simplification of polygonal scenes. The authors cluster vertices together in a tree structure and extract from this only those polygons important for the current view point. Subtrees defining a volume occupying a screen-space smaller than a user defined tolerance are collapsed and degenerated polygons are filtered out.

5.3 Parallel Algorithms

Nowadays many high performance computers have multiple CPUs. In order to fully use the available computing power of such a machine a polygonization algorithm must be parallelized. We give here a short collection of available literature describing parallelized polygonization algorithms.

Hansen and Hinker [HH92] present a SIMD implementation of the Marching Cubes algorithm. To avoid complications with communication the algorithm processes each voxel independently by assigning it a virtual processor. The authors report superlinear speed-up if the ratio of virtual to physical processors increases, which is due to improved efficiency. They also mention that the time spent in each virtual processor is constant regardless of the number of polygons in a cell.

Mackerras [Mac92] uses an MIMD implementation of the Marching Cubes algorithm, which is based on an efficient serial implementation. The volume is partitioned into contiguous blocks. Every processor is allocated one or more blocks and runs a serial Marching Cubes code on it. The author reports a speed up larger than the number of processors and suggests that this is due to cache effects.

Guéziec and Humme [GH95] describe a parallel polygonization algorithm based on the decomposition of a cubic cell into 5 tetrahedra. The authors perform the subspace polygonization by table indexing and present a coding scheme which allows to store vertex information local to the tetrahedra without duplications.

Savchenko and Pasko [SP95] present an algorithm for a transputer network with a toroidal architecture, which has the advantage that the maximal distance between two processors is of order $O(\sqrt{n})$. The authors use a sample grid of cubic cells and divide the cells between processors. For each cell the isosurface intersection with the edges is computed by linear interpolation. The intersection points are used to form a connection graph. Face ambiguities are resolved with the bilinear interpolant over the face. The cycles in the connection graph give the polygonal patches used to approximate the isosurface.

6 Conclusion

We have reviewed four polygonization methods in detail and extracted three common aspects: polyhedral subdivision of space, subspace polygonization and achievement of continuity. These aspects were discussed using the reviewed polygonization methods and numerous alternative algorithms as examples.

This part of our work might form a good basis for the development of new both specialized and general polygonization algorithms.

We then gave a set of quality criteria and classified the reviewed polygonization methods accordingly.

Not surprisingly there is no optimal polygonization method, but the method of choice usually depends on the application. In general a compromise must be made between speed, accuracy and correctness and implementation complexity. The Marching Cubes algorithm and its variants seem to be the most popular methods. Some ambiguities of the original Marching Cubes algorithm can be remedied by introducing additional complexity.

Using a tetrahedral decomposition might simplify the implementation but leads to fragmentation and in certain cases to a spiky approximation. If seed cubes for the isosurface are known a surface tracking method such as the algorithm from Wyvill et al. is recommended for improved speed.

In the second part of our work we showed briefly the need for improvements of both the speed and the result of a polygonization algorithm. We discussed several techniques to improve the speed of finding an isosurface approximation, classified them into three classes and compared them. We then looked into the problem of optimizing a polygonal mesh. Several methods were introduced, classified into six classes, and compared.

As a whole our work provides an overview of polygonization methods and optimization techniques used in the scientific community. We hope that our work forms a good basis for the decision making about which method to implement and we hope that our results will prove helpful for the development of new both specialized and generalized polygonization algorithms.

References

[AFH80] Ehud Artzy, Gideon Frieder, and Gabor T. Herman. The theory, design, implementation, and evaluation of a three-dimensional surface detection algorithm. In *Computer Graphics (SIGGRAPH '80 Proceedings)*, pages 2–9, July 1980.

[AG87] Eugene L. Allgower and Stefan Gnutzmann. An algorithm for piecewise linear approximation of implicitly defined two-dimensional surfaces. *SIAM Journal of Numerical Analysis*, 24(2):452 – 469, April 1987.

[AS96] María-Elena Algorri and Francis Schmitt. Mesh simplification. In Jarek Rossignac and François Sillion, editors, *Com-*

puter Graphics Forum (Eurographics '96), volume 15, pages C77 – C86. Eurographics Association, University Press, Cambridge, UK, 1996. Futoroscope - Poitiers, France August 26 – 30, 1996, ISSN 0167-7055.

[Aur91] Franz Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345 – 405, September 1991.

[Bak88] H. Harlyn Baker. Building, visualizing, and computing on surfaces of evolution. *IEEE Computer Graphics and Applications*, 8(4):31 – 41, July 1988.

[Bak89] H. Harlyn Baker. Building surfaces of evolution: The weaving wall. *International Journal of Computer Vision*, 3(1):51 – 71, May 1989.

[Bei90] Thaddeus Beier. Practicel uses for implicit surfaces in animation. In *SIGGRAPH '90, course notes #23 - Modelling and Animating with Implicit Surfaces*, pages 20–1 – 20–11. ACM SIGGRAPH, August 1990. Held in Dallas, Texas, 6 – 10 August.

[Ben75] Jon Louis Bentley. Multidimensional binary search trees used for associative search. *Communications of ACM*, 18(9):509 – 516, 1975.

[BF95] Jules Bloomenthal and Keith Ferguson. Polygonization of Non-Manifold implicit surfaces. In Robert Cook, editor, *SIGGRAPH '95 Conference Proceedings*, Annual Conference Series, pages 309–316. ACM SIGGRAPH, Addison Wesley, August 1995.

[BHR+94] Manfred Brill, Hans Hagen, Hans-Christian Rodrian, Wladimir Djatschin, and Stanislav V. Klimenko. Streamball techniques for flow visualization. In D. Bergeron and A. Kaufman, editors, *Proceedings of Visualization '94*, pages 225 – 231. IEEE, Computer Society Press, 1994.

[Bli82] James F. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):235 – 256, July 1982.

[Blo88] Jules Bloomenthal. Polygonization of implicit surfaces. *Computer-Aided Geometric Design*, 5(4):341 – 355, November 1988.

[Blo94] Jules Bloomenthal. An implicit surface polygonizer. In Paul S. Heckbert, editor, *Graphic Gems*, volume IV, chapter IV.8. Academic Press, Cambridge, MA 02139, 1994.

[Bow81] A. Bowyer. Computing dirichlet tesselations. *The Computer Journal*, 24(2):162 – 166, May 1981.

[BS91] Jules Bloomenthal and Ken Shoemaker. Convolution surfaces. *Computer Graphics*, 25(4):251 – 256, July 1991.

[BW90] Jules Bloomenthal and Brian Wyvill. Interactive techniques for implicit modeling. *Computer Graphics*, 24(2):109 – 116, March 1990. Special Issue on 1990 Symposium on Interactive 3D Graphics.

[CFM+94] P. Cignoni, L. De Floriani, C. Montani, E. Puppo, and R. Scopigno. Multiresolution modeling and visualization of volume data based on simplicial complexes. In *Proceedings ACM Symposium on Volume Visualization'94*. ACM, October 1994. URL: `http://disi.unige.it/ftp/person/PuppoE/PS/ACM-VV94.ps.gz`.

[CLL+88] H. E. Cline, W. E. Lorensen, S. Ludke, C. R. Crawford, and B. C. Teeter. Two algorithms for the three-dimensional reconstruction of tomograms. *Medical Physics*, 15(3):320 – 327, 1988.

[Col90] Steve Colburn. Solid modeling with global blending for machining dies and patterns. SAE technical paper series, SAE International, 400 Commonwealth Drive, Warrendale, PA 15096-0001 U.S.A., April 1990. 41st Annual Earthmoving Industry Conference.

[CPD+96] Andrew Certain, Jovan Popović, Tony DeRose, Tom Duchamp, David Salesin, and Werner Stuetzle. Interactive multiresolution surface viewing. In Holly Rushmeier, editor, *SIGGRAPH 96*

Conference Proceedings, Annual Conference Series, pages 91–98. ACM SIGGRAPH, Addison-Wesley Publication Company Inc, August 1996. Held in New Orleans, Louisiana, 04-09 August 1996.

[CSYL88] B. K. Choi, H. Y. Shin, Y. I. Yoon, and J. W. Lee. Triangulation of scattered data in 3D space. *Computer-Aided Design*, 20(5):239 – 248, 1988.

[CVM+96] Jonathan Cohen, Amitabh Varshney, Dinesh Manocha, Greg Turk, Hans Weber, Pankaj Agarwal, Frederick P. Brooks, Jr., and William Wright. Simplification envelopes. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 119–128. ACM SIGGRAPH, Addison Wesley, August 1996. Held in New Orleans, Louisiana, 04-09 August 1996, URL: `http://www.cs.unc.edu/~geom/envelope.html`.

[DH93] Thierry Delmarcelle and Lambertus Hesselink. Visualizing second order tensor fields with hyperstreamlines. *IEEE Computer Graphics and Applications*, 13(4):25 – 33, 1993. URL: `http://www.nas.nasa.gov/NAS/TechReports/RelatedPapers /StanfordTensorFieldVis/CGA93/abstract.html`.

[DK91] A. Doi and A. Koide. An efficient method of triangulating equivalued surfaces by using tetrahedral cells. *IEICE Trans. Commun. Elec. Inf. Syst.*, E-74(1):214 – 224, January 1991.

[DLTW90] David P. Dobkin, Silvio V. F. Levy, William P Thurston, and Allan R Wilks. Contour tracing by piecewise linear approximations. *ACM Transactions on Graphics*, 9(4):389 – 423, October 1990.

[Düu88] Martin J. Düurst. Additional reference to "marching cubes". *Computer Graphics*, 22(2):72, April 1988. Letter.

[DZ91] Michael De Haemer, Jr. and Michael J. Zyda. Simplification of objects rendered by polygonal approximations. *Computers and Graphics*, 15(2):175–184, 1991.

[EDD+95] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution analysis of arbitrary meshes. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 173–182. ACM SIGGRAPH, Addison Wesley, August 1995. Held in Los Angeles, California, 06-11 August 1995, URL: `ftp://ftp.cs.washington.edu/tr/1995/01/UW-CSE-95-01-02.d`.

[FMPB97] L. De Floriani, P. Magillo, E. Puppo, and M. Bertolotto. Multiresolution representation and reconstruction of triangulated surfaces. In *Proceedings 3rd International Workshop on Visual Form*, May 1997. URL: `http://disi.unige.it/ftp/person/MagilloP/PS/iwvf-97.ps.gz` Also appeared in a longer version as Technical Report N. PDISI-96-26.

[Gal91] Richard S. Gallagher. Span filtering: An optimization scheme for volume visualization of large finite element models. In Gregory M. Nielson and Larry Rosenblum, editors, *Proceedings of Visualization '91*, pages 68 – 75, Los Alamitos, California, October 1991. IEEE, Computer Society Press.

[GGS95] Markus H. Gross, Roger Gatti, and Oliver G. Staadt. Fast multiresolution surface meshing. In Gregory M. Nielson and Deborah Silver, editors, *Proceedings of Visualization '95*, pages 135 – 142, Los Alamitos, California, 1995. IEEE, Computer Society Press.

[GH90] M. Giles and R. Haimes. Advanced interactive visualization for CFD. *Computer Systems in Engineering*, 1(1):51 – 62, 1990.

[GH95] André Guéziec and Robert Hummel. Exploiting triangulated surface extraction using tetrahedral decomposition. *IEEE Transactions on Visualization and Computer Graphics*, 1(4):328 – 342, December 1995. ISSN 1077-2626.

[GN89] Richard S. Gallagher and Joop C. Nagtegaal. An efficient 3-D visualization technique for finite element models. *Computer Graphics*, 23(3):185 – 194, July 1989.

[GSG96] Markus H. Gross, Oliver G. Staadt, and Roger Gatti. Efficient triangular surface approximations using wavelets and quadtree data structures. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):130 – 143, June 1996.

[Ham94] Bernd Hamann. A data reduction scheme for triangulated surfaces. *Computer-Aided Geometric Design*, 11(2):197 – 214, 1994.

[HB94] C. T. Howie and E. H. Blake. The mesh propagation algorithm for isosurface extraction. *Computer Graphics Forum (Eurographics '94)*, 13(3):C65 – C74, 1994.

[HDD+93a] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzel. Mesh optimization. In *Computer Graphics Proceedings '93*, pages 19 – 26. ACM SIGGRAPH, 1993. URL: `ftp://ftp.cs.washington.edu/tr/1993/01/UW-CSE-93-01-01.PS.Z`.

[HDD+93b] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzel. Mesh optimization. TR 93-01-01, University of Washington, Department of Computer Science and Engineering, January 1993. URL: `http://cs-tr.cs.washington.edu:80/Dienst/UI/2.0/Describe/ncstrl.uwash_cse%2fTR-93-01-01` .

[HH87] Christoph Hoffmann and John Hopcroft. The potential method for blending surfaces and corners. In Gerald E. Farin, editor, *Geometric Modelling: Algorithms and New Trends*, pages 347 – 365, Philadelphia, 1987. Society for Industrial and Applied Mathematics.

[HH91] Josef Hoschek and Erich Hartmann. G^{n-1}-functional splines for modeling. In Hans Hagen and D. Roller, editors, *Geometric Modeling - Methods and Applications*, pages 185 – 211. Springer Verlag, 1991. Based on lectures presented at an international workshop held in Böblingen, Germany in June 1990.

[HH92] Charles D. Hansen and Paul Hinker. Massively parallel isosurface extraction. In *Proceedings of Visualization '92*, pages 77

– 83, Los Alamitos, California, October 1992. IEEE, Computer Society Press.

[HH93] P. Hinker and C. Hansen. Geometric optimization. In G. M. Nielson and D. Bergeron, editors, *Proceedings of Visualization '93*, pages 189 – 195, Los Alamitos, California, 1993. IEEE, Computer Society Press.

[HL92] Josef Hoschek and Dieter Lasser. *Fundamentals of Computer Aided Geometric Design*, chapter 14, pages 572 – 601. AK Peters Ltd., Wellesley, MA 02181, second edition, 1992.

[Hol91] Richard L. Holloway. Viper: A quasi-real-time virtual-worlds application. UNC technical report no. tr-92-004, Department of Computer Science, University of North Carolina, Chapel Hill, NC, December 1991. URL: `ftp://ftp.cs.unc.edu/pub/publications/techreports/92-004.tar`.

[Hop96] Hugues Hoppe. Progressive meshes. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 99–108. ACM SIGGRAPH, Addison-Wesley Publication Company Inc, August 1996. Held in New Orleans, Louisiana, 04-09 August 1996, URL: `http://www.research.microsoft.com/research/graphics/hoppe/pm_sig96_bw.ps.gz`.

[Hop97] Hugues Hoppe. View-dependent refinement of progressive meshes. In *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 189–198. ACM SIGGRAPH, Addison-Wesley Publication Company Inc, August 1997. Held in Los Angeles, California, August 03-08, 1997, URL: `http://www.research.microsoft.com/research/graphics/hoppe`.

[HW90] Mark Hall and Joe Warren. Adaptive polygonization of implicitly defined surfaces. *IEEE Computer Graphics and Applications*, 10(5):33 – 42, November 1990.

[IK94] Takayuki Itoh and Koji Koyamada. Isosurface generation by using extrama graphs. In R. Daniel Bergeron and Arie E. Kaufman, editors, *Proceedings of Visualization '94*, pages 77 – 83, Los Alamitos, California, October 1994. IEEE, Computer Society Press.

[IK95] Takayuki Itoh and Koji Koyamada. Automatic isosurface propagation using an extrema graph and sorted boundary cell lists. *IEEE Transactions on Visualization and Computer Graphics*, 1(4):319 – 327, December 1995.

[Kal91] Alan D. Kalvin. *Segmentation and surface-based modeling of objects in three-dimensional biomedical images*. PhD thesis, New York University, New York, 1991.

[Kal92] Alan D. Kalvin. A survey of algorithms for constructing surfaces from 3D volume data. Research report rc 17600 (#77606) 1/16/92, IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY 10598, January 1992.

[KCHN91] Alan D. Kalvin, C. B. Cutting, B. Haddad, and M. E. Noz. Constructing topologically connected surfaces for the comprehensive analysis of 3D medical structures. *SPIE*, 1445 Image Processing:247 – 259, 1991.

[KDK86] A. Koide, A. Doi, and K. Kajioka. Polyhedral approximation approach to molecular orbit graphics. *J. Molec. Graph.*, 4:149 – 156, 1986.

[Kem93] A. Kemeny. A cooperative driving simulator. In *Proceedings of the International Training Equipment Conference and Exhibition*, pages 67 – 71, 1993. London, UK, 4–6 May 1993.

[KT96] Alan D. Kalvin and Russell H. Taylor. Superfaces: Polygonal mesh simplification with bounded error. *IEEE Computer Graphics and Applications*, 16(3):64–77, May 1996. ISSN 0272-1716.

[LC87] W. Lorensen and H. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics*, 21(4):163 – 169, July 1987. Proceedings of SIGGRAPH.

[LDW94] Michael Lounsbery, Tony DeRose, and Joe Warren. Multiresolution analysis for surfaces of arbitrary topological type. Technical report 93-10-05b, Department of Computer Science and Engineering, University of Washington, January 1994. URL: `ftp://cs.washington.edu/pub/graphics/TR931005.ps.Z`.

[LE97] David Luebke and Carl Erikson. View-dependent simplification of arbitrary polygonal environments. In *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 199–208. ACM SIGGRAPH, Addison-Wesley Publication Company Inc, August 1997. Held in Los Angeles, California, August 03-08, 1997.

[Lob96] Richard Lobb. Quasiconvolutional smoothing of polyhedra. *The Visual Computer*, 12(8):373 – 389, 1996.

[Lor96] Bill Lorensen. Extracting surfaces from medical volumes. In *Volume Visualization: Principles and Practice*. ACM SIGGRAPH, 1996. SIGGRAPH '96, Course Notes #34.

[Lou94] Michael Lounsbery. *Multiresolution Analysis for Surfaces of Arbitrary Topological Type*. PhD thesis, Department of Computer Science and Engineering, University of Washington, September 1994. URL: `ftp://cs.washington.edu/pub/graphics/LounsPhd.ps.Z`.

[LSJ96] Yarden Livnat, Han-Wei Shen, and Christopher R. Johnson. A near optimal isosurface extraction algorithm using the span space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):73 – 84, March 1996.

[LVG80] S. Lobregt, P. W. Verbeek, and F. C. A. Groen. Three-dimensional skeletonization: Principle and algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-2(1):75 – 77, January 1980.

[Mac92] Paul Mackerras. A fast parallel marching-cubes implementation on the Fujitsu AP1000. Technical report, Department of Computer

Science, Australien National University, August 1992. URL: `http://cs.anu.edu.au/techreport/1992/TR-CS-10.ps.gz`.

[Mat94] Sergey V. Matveyev. Approximation of isosurface in the marching cube: Ambiguity problem. In R. Daniel Bergeron and Arie E. Kaufman, editors, *Proceedings of Visualization '94*, pages 288 – 292, Los Alamitos, California, October 1994. IEEE, Computer Society Press.

[ML94] S. R. Marschner and R. J. Lobb. An evaluation of reconstruction filters for volume rendering. In R. Daniel Bergeron and Arie E. Kaufman, editors, *Proceedings of Visualization '94*, pages 100–107, Los Alamitos, California, October 1994. IEEE, Computer Society Press.

[MS93] Heinrich Müller and Michael Stark. Adaptive generation of surfaces in volume data. *The Visual Computer*, 9(4):182 – 199, January 1993.

[MSS94] C. Montani, R. Scateni, and R. Scopigno. Discretized marching cubes. In D. Bergeron and A. Kaufman, editors, *Proceedings of Visualization '94*, pages 281 – 286. IEEE, Computer Society Press, 1994.

[Mur91] Shigeru Muraki. Volumetric shape description of range data using "blobby model". *Computer Graphics*, 25(4):227 –235, July 1991. Proceedings of SIGGRAPH '91.

[MW91] Doug Moore and Joe Warren. Mesh displacement: An improved contouring method for trivariate data. Technical report tr91-166, Rice University, Houston, Texas, September 1991.

[MW92] D. Moore and J. Warren. Compact isocontours from sampled data. In D. Kirk, editor, *Graphic Gems III*, pages 23 – 28. Academic Press, 1992.

[Nat91] Balar K. Natarajan. On generating topologically correct isosurfaces from uniform samples. Tech. Rep. HPL-91-76, Software and Systems Laboratory, Hewlett-Packard Co., Page Mill Road, Palo Alto, Ca., June 1991.

[NB93] Paul Ning and Jules Bloomenthal. An evaluation of implict surface tilers. *IEEE Computer Graphics and Applications*, 13(6):33 – 41, November 1993.

[NFHL91] G. M. Nielson, T. A. Foley, B. Hamann, and D. A. Lane. Visualization and modelling of scattered multivariate data. *IEEE Computer Graphics and Applications*, 11(3):47 – 55, May 1991.

[NH91a] Gregory M. Nielson and Bernd Hamann. The asymptotic decider: Resolving the ambiguity in marching cubes. In Gregory M. Nielson and Larry Rosenblum, editors, *Proceedings of Visualization '91*, pages 83 – 91, Los Alamitos, California, October 1991. IEEE, Computer Society Press.

[NH91b] Paul Ning and Lambertus Hesselink. Adaptive isosurface generation in a distortion-rate framework. In *SPIE Proceedings Conference 1459*, pages 11 – 21, February 1991.

[Nie95] Gregory M. Nielson. Modeling and visualizing volumetric and surface-on-surface data. In *SIGGRAPH '95 - course notes # 30*, pages 35 – 101. ACM SIGGRAPH, 6 – 11 August 1995, Los Angeles, California, August 1995. An updated version has appeared in "Focus on Visualization", H. Hagen, H. Mueller, G. M. Nielson, editors, Springer, ISBN 3-540-54940-4, pp. 191-242.

[OP96] Kwang-Man Oh and Kyu Ho Park. A type-merging algorithm for extracting an isosurface from volumetric data. *The Visual Computer*, 12(8):406 – 419, 1996.

[PH97] Jovan Popović and Hugues Hoppe. Progressive simplicial complexes. In *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 217–224. ACM SIGGRAPH, Addison-Wesley Publication Company Inc, August 1997. Held in Los Angeles, California, August 03-08, 1997, URL: `http://www.research.microsoft.com/research/graphics/hoppe`.

[PPW87] Carl S. Petersen, Bruce R. Piper, and Andrew J. Worsey. Adaptive contouring of a trivariate interpolant. In Gerald E. Farin, editor, *Geometric Modelling: Algorithms and New Trends*, pages 385 – 395, Philadelphia, 1987. Society for Industrial and Applied Mathematics.

[PT90] Bradley A. Payne and Arthur W. Toga. Surface mapping brain functions on 3D models. *IEEE Computer Graphics and Applications*, 10(2):41 – 53, February 1990.

[PvW94] F. H. Post and J. J. van Wijk. Visual representation of vector fields: recent developments and research directions. In L. J. Rosenblum et. al., editor, *Scientific Visualization: Advances and Challenges*, pages 367 – 390. Academic Press, 1994.

[Red96] M. Reddy. SCROOGE: Perceptually-driven polygon reduction. *Computer Graphics Forum*, 15(4):191 – 203, October 1996. ISSN 0167-7055.

[RR96] Rémi Ronfard and Jarek Rossignac. Full-range approximation of triangulated polyhedra. In Jarek Rossignac and François Sillion, editors, *Computer Graphics Forum (Eurographics '96)*, volume 15, pages C67 – C76. Eurographics Association, University Press, Cambridge, UK, 1996. Futoroscope - Poitiers, France August 26 – 30, 1996, ISSN 0167-7055.

[Sch95] William J. Schroeder. Polygon reduction techniques. Course notes # 30, ACM SIGGRAPH, 1995.

[SDS95a] Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin. Wavelets for computer graphics: A primer, part 1. *IEEE Computer Graphics and Applications*, 15(3):76–84, May 1995.

[SDS95b] Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin. Wavelets for computer graphics: A primer, part 2. *IEEE Computer Graphics and Applications*, 15(4):75–85, July 1995.

[SDS96] Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin. *Wavelets for Computer Graphics - Theory and Applications.* The Morgan Kaufmann Series in Computer Graphics and Geometric Modelling. Morgan Kaufmann Publishers, Inc., San Francisco, California, 1996.

[SJ95] Han-Wei Shen and Christopher R. Johnson. Sweeping simplices: A fast iso-surface extraction algorithm for unstructured grids. In Gregory M. Nielson and Deborah Silver, editors, *Proceedings of Visualization '95*, pages 143 – 150, Los Alamitos, California, October 1995. IEEE, Computer Society Press.

[SP95] V. V. Savchenko and A. A. Pasko. Parallel polygonization of implicit surfaces on transputers: algorithm, time performance evaluation and rendering results. In H. Arabnia, editor, *Transputer Research and Applications 7 (NATUG-7, Proceedings of the Seventh Conference of the North American Transputer Users Group)*, pages 22–30. IOS Press, 1995.

[SZL92] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 65 – 70, July 1992.

[Tur91] Greg Turk. Generating textures on arbitrary surfaces using reaction-diffusion. *Computer Graphics*, 25(4):289 – 298, 1991. (SIGGRAPH '91).

[Tur92] Greg Turk. Re-tiling polygonal surfaces. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 55–64, July 1992.

[Var94] Amitabh Varshney. Hierarchical geometric approximations. Ph.d. thesis tr-050-1994, University of North Carolina, Chapel Hill, NC 27599-3175, 1994. URL: `ftp://ftp.cs.unc.edu/pub/publications/techreports/94-050.ps.Z`.

[vGW94] Allen van Gelder and Jane Wilhelms. Topological considerations in isosurface generation. *ACM Transactions on Graphics*, 13(4):337 – 375, October 1994.

[Wal91] A. Wallin. Constructing isosurfaces from CT data. *IEEE Computer Graphics and Applications*, 11(6):28 – 33, November 1991.

[War89] Joe Warren. Blending algebraic surfaces. *ACM Transactions on Graphics*, 8(4):263 – 278, October 1989.

[WMW86a] Geoff Wyvill, Craig McPheeters, and Brian Wyvill. Animating soft objects. *The Visual Computer*, 2(4):235 – 242, August 1986.

[WMW86b] Geoff Wyvill, Craig McPheeters, and Brian Wyvill. A data structure for soft objects. *The Visual Computer*, 2(4):227 – 234, August 1986.

[Wün96] Burkhard C. Wünsche. A fast polygonization method for quasi-convolutionally smoothed polyhedra. Master's thesis, University of Auckland, August 1996. URL: `http://www.cs.auckland.ac.nz/~bwue001/Thesis/thesis.pdf`.

[WvG92] Jane Wilhelms and Allan van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201 – 227, July 1992.

[WWM87] Geoff Wyvill, Brian Wyvill, and Craig McPheeters. Solid texturing of soft objects. *IEEE Computer Graphics and Applications*, 7(12):20 – 26, December 1987.