INCREMENTAL CONVERSION OF 3D WIRE FRAME MODELS TO POLYGONAL SURFACE MODELS

Jérôme Grosjean, Terii Stein, Sabine Coquillart

INRIA
Domaine de Voluceau
78153 Le Chesnay Cedex
France
Jerome.Grosjean / Terii.Stein / Sabine.Coquillart@inria.fr

ABSTRACT

Though a large variety of methods has been developed to model 3D objects, most of them are based on the construction of solid objects or curved surfaces. This work is part of a sketch-based project for modeling arbitrarily polygonal shapes. The method for modeling 3D objects is based on the user drawing or removing of edges. This paper addresses the automatic computation of the 3D polygonal surface model from the 3D wire frame. A straightforward incremental algorithm is proposed allowing an update of the surface model for each modification of the 3D wire frame. Objects of any genus, or topological type are handled by the method.

Keywords: geometric modeling, sketching, wireframe, surface reconstruction

1 INTRODUCTION

Many powerful modeling systems exist which provide the user with tools for modeling complex objects. One can distinguish two main classes of traditional systems. "CSG-based" [Foley90] systems define an object from a boolean combination of primitives such as cubes, spheres. "Surface-based" systems [Wavef] provide the user with tools like extrusion, loft, sweep and surface of revolution, for defining complex surfaces. In addition, deformation tools such as FFD [Seder86, Coqui90] are often proposed. These systems provide the user with powerful tools for accurately modeling complex objects or surfaces. However, most traditional systems are difficult and tedious to use, especially for inexperienced users. Finding effective interactive modeling techniques thus continues to be an active area of research. Recent work tends to prioritize the ease of use in return for less accuracy and less generality. Examples of techniques following this goal include the methods based on implicit surfaces [Bloom90, Marko99], deformation techniques like the "Wires" technique [Singh98] which makes use of wires for deforming objects and the direct manipulation technique [Hsu92], gesture-based approaches like "SKETCH" [Zelez96] and IDeS [Branc94] which introduce gesture-based interfaces for the rapid modeling of CSG-like models, sculpting methods [Galye91, Wang95], and sketching methods like "Teddy" [Igara99]: a sketching interface for the design of rotund objects.

Most of these recent techniques are more or less specific to some class of shapes which most of the time represent solid objects. The modeling of various complex shapes (in particular solids) has been addressed in the literature. However, few papers address the interactive modeling of arbitrarily polygonal surfaces which can't be defined with traditional sweep, extrusion, loft,... tools i.e. non predefined shapes. Examples of such shapes include surfaces like a paper shape or an open box, or solids like the ones presented on Figures 6, 7 and 8.

For these shapes a sketched-based approach which consists in interactively defining the edges of the surface is probably the best known solution from a user point of view. Sketch-based approaches include three stages: the drawing of the 2D edges, the transformation of the 2D edges from 2D to 3D, and the reconstruction of the faces from the 3D wire frame. We must note that the two first stages can be regrouped into one if a 3D input device is used. This paper addresses the third stage: the reconstruction of the faces from the 3D wire frame.

Most of the time, the reconstruction process is executed offline, once the whole 3D wire frame has been entered by the user (see related work section). In addition, most of the proposed methods are only valid for a subclass of closed solid objects. We believe that with modern interactive modeling systems, the reconstruction process has to take place online (i.e. for each edge entered or deleted by the user), the surface model has to be updated so that at each time, the user doesn't only see a wire frame representation of his model but the full surface model. Despite the large bibliography on offline reconstruction techniques, to the authors knowledge, no researches have been devoted to online reconstruction.

Most modeling systems avoid the problem by requiring the user to define a list of edges and the edge cycles representing each face. This solution doesn't fit our need because the user definition of each cycle is tedious. The order in which the edges are entered is an additional constraint and most edges have to be entered twice (one time for each adjacent face).

Our goal is to allow the user to define a 3D polygonal surface from the object edges entered in any order. For that purpose, this paper proposes a simple, interactive incremental online technique for reconstructing the object faces from the 3D edges. The next section reviews related work on 3D wire frame to face reconstruction. The proposed algorithm is then presented followed by a discussion and results section.

2 RELATED WORK

The automatic conversion from wireframes to surface models has been a research issue for some time now. Several topological or geometrical approaches can be found in the literature.

In 1980, Markowsky[Marko80] proposed a mixed geometrical and topological approach based on the planarity of faces. Markowsky's algorithm first generates a list of all the planes containing at least two intersecting edges. In a second time, it searches the so-called "virtual faces" lying inside each plane. A virtual face is a cycle of edges which can be either a true face, an internal face or a simple cycle due to a chance alignment of edges. The set of virtual faces defines virtual blocks (solid objects). Finally the blocks are discriminated so that the accepted ones define the final 3D solid model.

Dutton et al. [Dutto83] and Hanrahan [Hanra82] approach is strictly topological and uses the planar embedding technique [Tarja71]. Fundamentally, the wireframe is represented as a flat, two-dimensional edge-vertex graph whose faces represent the faces of the object. This method has the advantage of being able to handle objects with non-planar faces or edges but in return, it only handles valid models without holes. Ganter et al. [Gante83] and Courter et al. [Court86] proposed another topological approach also based on an edge-vertex graph. The graph is examined to find the fundamental cycles from which the faces are computed thanks to the so-called reduction process. This method also handles non planar faces but it requires valid 3-D connected solids without holes. The Shortest Path Approach by Bagali and Waggenspack [Bagal95] is another strictly topological approach. It uses a shortest path algorithm to produce a set of cycles from the graph and identifies in a second step which ones form a cycle basis for the cycle space of the graph, i.e. the set of facial cycles. This method again, works only for valid 3-D connected solids without holes. Agarwal and Waggenspack propose a different and original approach to the problem of conversion by decomposing 3D volumes into tetrahedras. Each tetrahedra yield a set of faces. The whole list of found faces is scanned and the faces appearing more than once are eliminated. The remaining faces are the true faces of the 3D model. This method doesn't apply to non-solid models.

3 ALGORITHM

All the previous methods have been developed for offline treatments while in our project, an online treatment is desired, i.e. an update of the surface model after each addition or removal of an edge. This constraint has three consequences: the conversion should be incremental, it has to work for surfaces of any topological type (not only solid models), and the computation needs to be efficient enough to be completed in real-time (or at least in a time compatible with the interaction). The purpose of the proposed algorithm is to find all the faces defined as planar cycles of edges. If some are not desired, they will be destroyed in a second time discussed in the "Discussion and Results" paragraph.

Using an incremental approach avoids the need to recompute the whole set of faces after each modification of the *edge-vertex graph* representing the wire frame model. Indeed when an edge is added or removed from the graph, only the faces adjacent to the edge need to be created, removed or altered. The edge-vertex graph is interactively constructed and updated by the user. The user operations may include: the addition or removal of edges, and the displacement of one or more vertices or edges. All these operations turn out to simplify into only two different cases for the incremental reconstruction algorithm: edge addition and edge removal. The displacement of edges can be treated by first removing edges and then possibly adding new ones. The addition of a new edge may create one or more new faces or subdivide existing faces. The removal of an edge, may result in the removal of one or more faces or in the fusion of faces. It is assumed that edges intersect only at endpoints. This is guaranteed by the wire frame construction method.

Each of the two cases: edge addition and edge removal will be studied in the two next sections. A third section will discuss the case of faces embedded into other faces.

3.1 Adding an edge

Two types of new edges have to be distinguished depending on whether their vertices are connected to other edges or not. A face being a cyclic set of edges, if at least one vertex of the new edge is not connected to the other edges, this new edge can't involve the creation of a face. The only case which needs to be considered is thus when both vertices of the new edge are connected to other edges.

The proposed method for updating the surface is inspired from the Markowski method but turns out to be significantly simpler thanks to the incremental and interactive approach. The proposed algorithm takes advantage of the planarity of the faces. It is a mixed method: both topological and geometrical. Geometrical information is used to speed up the search of new faces by restricting the search to a set of planes containing the added edge. Within these planes, a topological approach is proposed to derive the new faces. Therefore the method is divided into two stages: the first one to find a set of planes which could possibly embed the new faces and a second one to find the faces inside theses planes.

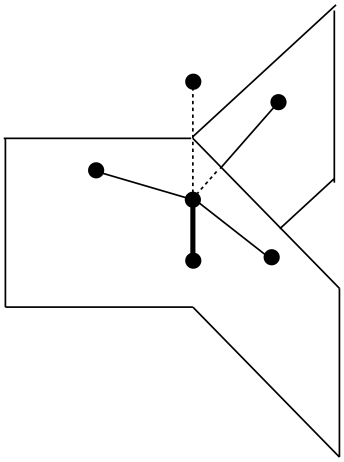

Figure 1: Finding the candidate planes for an edge

3.1.1 Finding the planes

Since all the planes have to include the new drawn edge, the two vertices of the new edge give the first clues to find these planes. Only one more point is needed to define a plane which can possibly include new faces. This third point is found by exploring the graph of connection, starting at one vertex of the new edge and using a Depth First Search algorithm. For each new path, the traversal stops as soon as a point not geometrically aligned with the two vertices of the new edge is found (see Figure 1). These points together with the two vertices of the new edge form the set of planes which possibly embed new faces. As some planes may be found more than once, a verification has to be made in order to remove the duplicated planes.

3.1.2 Finding the faces inside the planes

Each of the previously detected planes may include new faces. A straight forward topological algorithm is employed for testing, for each plane, whether it includes or not one or more faces. A plane can hold zero, one or two faces (one face each side of the new edge). An arbitrary orientation is given to the plane (by defining a canonical normal), in order to differentiate between the left and right side of an oriented edge. The next step consists in checking whether a "left (resp. right) face" has to be created. Both treatments being independent and similar, only the left face will be detailed. This problem reduces to a 2D problem. The 2D graph corresponding to a given plane is the sub-graph of the original one, and contains only the vertices and edges lying in that plane.

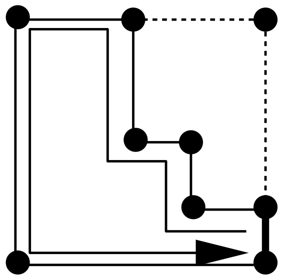

Figure 2: Example of a correct and not shortest cycle for a left face

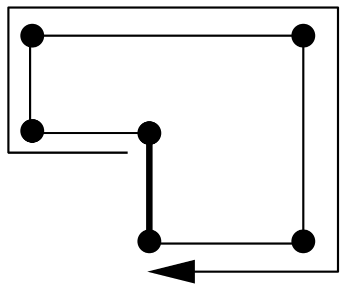

Figure 3: Search of a left face resulting in an infinite face

For simplicity we will only consider the 2D graph in the remainder of this paragraph. In the implementation, the 2D graph is not explicitly created. Instead, the whole graph is treated but only the vertices and edges lying into the plane are considered. We will also assume that for each vertex, the edges meeting this vertex are ordered with respect to the canonical normal of the plane. We are thus dealing with a planar graph (a graph is said to be planar if it can be drawn on the plane such that no two edges intersect geometrically except at a vertex at which they are both incident).

The problem thus reduces to finding, in the planar graph, a cycle of edges which includes the new edge and is located on its left side. Note that the correct cycles which represent true faces are not always the shortest ones. (see Figure 2).

The cycle (if it exists) is given by following a path which starts with the second vertex of the new edge and ends with the first one. It is defined so that for each traversed vertex, the outgoing edge follows the incoming one in clockwise order (counterclockwise order is used instead for right faces). The first traversed edge is the new edge and the edges cannot be traversed twice. The algorithm moves backward each time there is no available outgoing edge (dangling edge or path), and continues with the next edge of the previous vertex if any. Else the algorithm backtracks again. The edges along the backtrack path are forbidden for further traversal like in a depth first search algorithm. The search fails when no path can be found between the two vertices of the new edge, in which case no face is created.

The described algorithm detects two types of left faces: the valid ones and in some cases, some infinite ones when no valid one exist (see Figure 3). A simple way to differentiate an infinite face from a valid one is to sum for each traversed vertex the angle (clockwise) between the incoming and the outgoing edge (see Figure 4). For a valid face, the sum should be 360 degrees while it should be -360 degrees for an infinite face. A face is declared a valid left face if the sum is positive (the sum has to be negative for right faces).

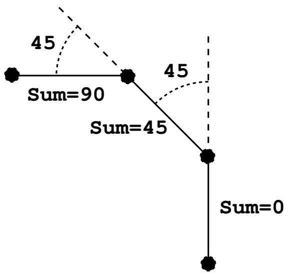

Figure 4: Summing up the edges' angular deviations

```
Adding an edge E:
if (one edge of E is not connected)
then nothing
else
{
  search_planes
  for each plane
  {
    search_left_path
    search_right_path
    if (merge(left_face,right_face)
        already exists)
    then delete old face
  }
}
```

3.1.3 Decomposition of a face

The addition of a new edge in the model can sometimes create faces which are the decomposition of an already existing face. When two faces (left and right) are found in one plane, the algorithm must check whether the two faces split an

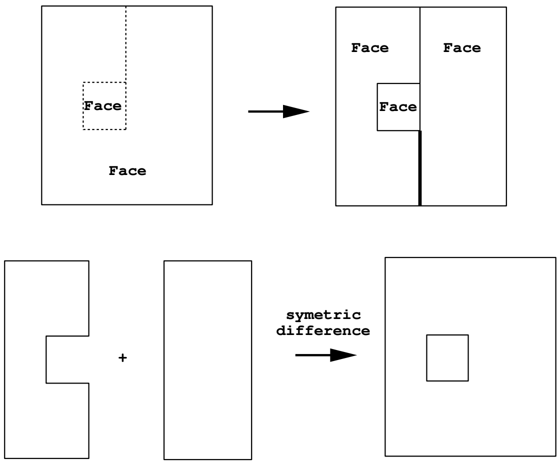

Figure 5: A simple symetric difference is not sufficient to merge correctly the new faces

existing face and if so delete the old face. The test is made by merging the two new found faces using the symetric difference [Court86] on the edges of each face. In most cases the symetric difference yields directly a valid face which can be compared with the "old" faces of our model. However in some tricky cases, where the frontier between the two new faces is not a simple path of edges (see Figure 5), the returned edges define instead a set of cycles. Therefore a search has to be made on the set of edges to identify each of these cycles in order to keep as the result of the merging the unique cycle which geometrically includes all the others. The cycle produced by the merging doesn't need to be compared with all the faces of the model. It is sufficient to compare it with the faces (at most two) lying in the plane associated with one of the edges of one of the two faces. We have seen that dangling edges are neglected during the graph traversal. Consequently some edges may have no associated face if they were dangling edges before the addition of the new edges.

3.2 Removing an edge

While removing an edge from the graph, some faces of the model may need to be deleted or merged. All the faces adjacent (in 3D) to the removed edge will be affected. If two of these faces lie in the same plane, then they are merged using the previously described merging method. All the faces which are alone in a plane are deleted.

```
Removing an edge E:
search faces including E
for each couple of faces in this list do
{
  if (faces have same normal vector)
  then merge the two faces
  else delete the two faces
}
```

3.3 Embedding a face into another face

A last case which has not yet been studied is when a face is embedded into another face. Note that it can be a filled face or an unfilled face (a hole). Both are treated similarly, a flag indicates whether it is filled or not and the only change concerns the display. We will call the embedded face the inside face and the face in which the inside face is embedded the outside face.

From a user point of view, there are three different approaches to create an inside face. The first one consists in subdividing the outside face so that no special treatment is needed. The obvious drawback of this method is that it requires the user to know when drawing the outside face that it contains an inside face. The second method, more natural for the user, consists in drawing the shape of the inside face like any other faces and linking the outside face and the inside face together so that they can be displayed correctly. The third solution consists in automatically determining whether a new face is embedded into (or embeds) an already existing face. This last solution is attractive but it is potentially time consuming because it requires considering all possible couples of faces. It may thus not be compatible with our interactive constraint. Consequently, we decided to adopt the second approach which fits well with the interactive status of our system. Note that if the information is not given by the user, the algorithm still works but two faces are superposed. The link(s) between the outside faces and the inside face(s) are stored in the data structure of the outside face.

If an edge is added which splits an existing inside face into two sub-faces, then the link between the outside face and the old inside one is destroyed and two links between the outside face and the two new faces are added. If an edge is added which cuts an outside face into two sub-faces, then the algorithm has to determine in which new face each of the inside faces lie and then add a link from the appropriate face to the inside face. If an inside face is destroyed by deleting one of its edges, then the link from the outside face to the inside face is destroyed too. If the edge of an outside face containing one or more inside faces

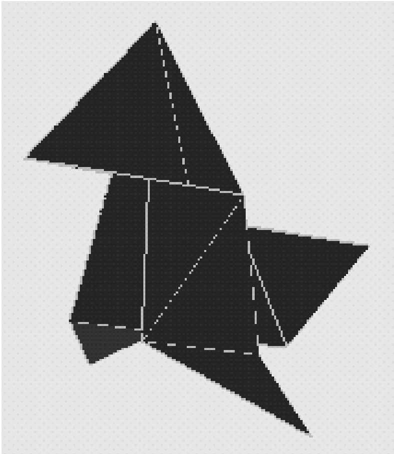

Figure 6: A paper shape

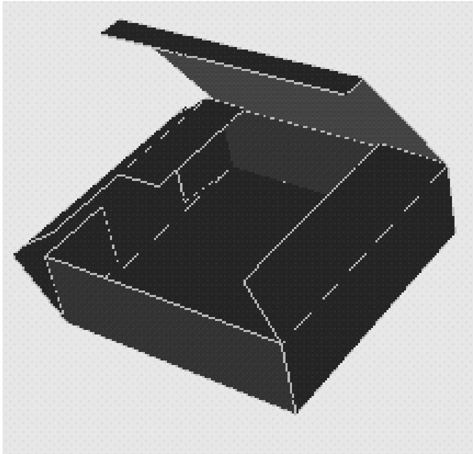

Figure 7: An open box

is deleted and the outside face is destroyed, the links to the inside faces are destroyed too. If an edge between two inside faces is deleted, the inside faces merge together and the link from the outside face to the previous inside faces is replaced by a link to the new resulting inside face. If the outside face merges with another face the link(s) to the inside face(s) is(are) transmitted to the resulting outside face.

4 DISCUSSION AND RESULTS

The proposed algorithm handles a large variety of 3D objects : objects which are solid or not, manifold or not, and of any genus. The reconstruction process finds all the possible correct faces (all the planar cycles of edges, convex or concave). However, in some specific cases of ambiguous wire frame models or internal faces, a user input may be necessary to remove the unwanted faces. For simplicity, it has been decided not to change the data structure but instead to use the "filled" "unfilled" information stored with each face (this information has been introduced in the previous paragraph for holes) in order to indicate whether a face is desired or not by the user. By choosing to automatically create all the faces and to let the user only destroy the few unwanted ones, we minimize as much as possible the user input (manually creating a face requires to click on each edge while removing a face only requires one click).

The average cost to find or destroy new faces after an addition or removal of an edge is O(1). The operation to find new faces is divided into two steps: the search of planes and the search of faces inside these planes. Both operations can be computed in an average time of O(1). In the worst case, all the vertices of the graph of connection have to be visited and these operations are linear. However such cases are rare (all vertices in the same plane or aligned on the same line). Destroying a face can be performed at the same very low cost. Figures 6, 7 and 8 present some objects modeled with the proposed method.

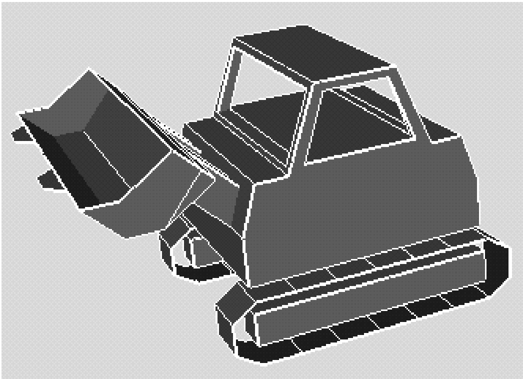

Figure 8: A bulldozer

5 CONCLUSION AND FUTURE WORK

We have presented an algorithm to convert a wire frame model into a polygonal surface model using an incremental process. Faces are constructed during the creation of the object. Modifications to the edge-vertex graph induce the incremental computation of the new set of faces. The algorithm is general and works for any valid polygonal surface. It is efficient enough to incrementally compute the wire frame model in real time during the interaction.

The main possible extension is the automatic computation of the inclusion of edges into faces. Trivial solutions suppose to check all the edge-face couples and are in $O(mn)$. An optimized method based on a partitioning of the space is under consideration. It is not our purpose but extending this technique to an offline treatment could also be of interest when arbitrary (solid or not, manifold or not, of any genus) objects have to be constructed.

6 ACKNOWLEDGMENT

The authors would like to thank David Tonnesen for proof reading the paper. This work was partly carried out with funding from the AFIRST association.

REFERENCES

[Bagal95] Siddarameshwar Bagali and Warren N. Waggenspack, Jr. A shortest path approach to wireframe to solid model conversion. In *SMA '95: Proceedings of the Third Symposium on Solid Modeling and Applications*, pages 339–350. ACM, May 1995. held May 17-19, 1995 in Salt Lake City, Utah.

[Bloom90] Jules Bloomenthal and Brian Wyvill. Interactive techniques for implicit modeling. In Rich Riesenfeld and Carlo Sequin, editors, *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, volume 24, pages 109–116, March 1990.

[Branc94] V. Branco, A. Costa, and F. Nunes Ferreira. Sketching 3D models with 2D interaction devices. In *Computer Graphics Forum,EUROGRAPHICS'94*, volume 13, pages 489–502. Eurographics, Basil Blackwell Ltd, 1994.

[Coqui90] Sabine Coquillart. Extended free-form deformation: A sculpturing tool for 3D geometric modeling. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 187–196, August 1990.

[Court86] S. M. Courter and J. A. Brewer. Automated conversion of curvilinear wire-frame models to surface boundary models; A topological approach. *Computer Graphics*, 20(4):171–178, August 1986.

[Dutto83] R. D. Dutton and R. C. Brigham. Efficiently identifying the faces of a solid (computer graphics). *Computers and Graphics*, 7(2):143–147, 1983.

[Foley90] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Fundamentals of Interactive Computer Graphics*. The Systems Programming Series. Addison-Wesley, Reading, MA, USA, second edition, 1990.

[Galye91] T. A. Galyean and J. F. Hughes. Sculpting: An interactive volumetric modeling technique. *Computer Graphics (Proc. ACM SIGGRAPH '91)*, 25(4):267–274, July 1991.

[Gante83] M. A. Ganter, Jr. and J. J. Uicker. From wire-frame to solid-geometric: Automated conversion of data representations. *Computers in Mechanical Engineering*, 2(2):40–45, September 1983.

[Hanra82] P. M. Hanrahan. Creating volume models from edge-vertex graphs. *Computer Graphics*, 16(3):77–84, July 1982.

[Hsu92] William M. Hsu, John F. Hughes, and Henry Kaufman. Direct manipulation of free-form deformations. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 177–184, July 1992.

[Igara99] Takeo Igarashi, Satoshi Matsuoka, and Hideko Tanaka. Teddy: A sketching interface for 3d freeform design. In *SIGGRAPH 99 Conference Proceedings*, Annual Conference Series, pages 409–416. ACM SIGGRAPH, August 1999.

[Marko80] George Markowsky and Michael A. Wesley. Fleshing out wire frames. *IBM Journal of Research and Development*, 24(5):582–597, September 1980.

[Marko99] Lee Markosian, Jonathan M. Cohen, Thomas Crulli, and John Hughes. Skin: A constructive approach to modeling free-form shapes. In *SIGGRAPH 99 Conference Proceedings*, Annual Conference Series. ACM SIGGRAPH, August 1999.

[Seder86] Thomas W. Sederberg and Scott R. Parry. Free-form deformation of solid geometric models. In David C. Evans and Russell J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 151–160, August 1986.

[Singh98] Karan Singh and Eugene Fiume. Wires: A geometric deformation technique. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 405–414. ACM SIGGRAPH, Addison Wesley, July 1998. ISBN 0-89791-999-8.

[Tarja71] R. E. Tarjan. An efficient planarity algorithm. Tech. Report 244, Computer Science Dept., Stanford Univ., 1971.

[Wang95] Sidney W. Wang and Arie E. Kaufman. Volume sculpting. In Pat Hanrahan and Jim Winget, editors, *1995 Symposium on Interactive 3D Graphics*, pages 151–156. ACM SIGGRAPH, April 1995. ISBN 0-89791-736-7.

[Wavef] Alias Wavefront. http://www.aw.sgi.com.

[Zelez96] Robert C. Zeleznik, Kenneth P. Herndon, and John F. Hughes. SKETCH: An interface for sketching 3D scenes. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 163–170. ACM SIGGRAPH, Addison Wesley, August 1996.