Computer Sculpting of Polygonal Models using Virtual Tools

James R. Bill
Suresh K. Lodha

UCSC-CRL-94-27
22 July 1994

Baskin Center for
Computer Engineering & Information Sciences
University of California, Santa Cruz
Santa Cruz, CA 95064 USA

ABSTRACT

This report describes a system for interactive manipulation of polygon meshes using virtual sculpting tools and mesh refinement operations. The virtual tools are geometric objects with shapes described by superquadric equations. The user controls a virtual tool to sculpt a free-form polygonal mesh model. The vertices of the mesh respond to the tool in a semi-realistic manner, allowing the user to push, pull, and deform the mesh in a variety of ways. Mesh refinement operations allow the user to control refinement levels and create smooth objects. Interactive shadows and a virtual trackball considerably enhance the intuitive manipulation and sculpting of models. The strength of the system has been demonstrated by sculpting free-form surfaces such as a dog, flower, television set, and dinosaur, all with widely varying levels of detail.

Keywords: Computer graphics, interactive modeling, computer sculpting, virtual tools, polygon meshes, smoothing, subdivision, shadows

Contents

1. Introduction

1.1 What is Computer Sculpting?

We will define computer sculpting as interactive geometric modeling in which the goals and/or techniques of traditional sculpture are emulated. Specifically, the key goal of sculpting is the design of free-form three-dimensional objects, while the principal techniques are direct manipulation of a material by hand or indirect manipulation through sculpting tools. The term *free-form* is used to contrast sculpted shapes with more traditional computer graphics models, which tend toward symmetric, precise, and angular forms. To clarify our definition, we require computer sculpting systems to possess the following characteristics:

1. Concentration on the modeling of generally free-form shapes
2. Intuitive model behavior reminiscent of a traditional sculpting material
3. Interactive, real time performance
4. Manipulation and deformation of model using virtual tools

In this report, *sculpting* will be used to mean computer sculpting. Its central goal is to create complex models suitable for use in rendering or animation programs [CP76, PMTT91]. There is currently an active consumer market for such models [Lew93]. In the past, difficulty in rendering and modeling of irregular free-form shapes has commonly resulted in computer generated images populated with unrealistically simple objects. Sculpting addresses this problem by allowing designers to translate their understanding of form directly into a finished geometric model.

1.2 Contributions of this Report

The goal of this work is to create an intuitive computer sculpting system. This has been achieved by a novel unification of known techniques, some of which have been adapted to our purposes from use in different contexts. Our work builds upon the techniques of interactive polygon manipulation using decay functions initially proposed by Parent [Par77] and extended by Allan et al [AWW89]. We further enhance these capabilities by incorporating adaptive subdivision and smoothing techniques. More importantly, we have used virtual tools to identify regions of interest, which can then be pushed or pulled by the user in an intuitive way. In previous sculpting systems, virtual tools have been used only in conjunction with boolean operations. Finally, interactive shadows and use of a virtual trackball provide the user the intuition necessary to manipulate these objects with ease. For convenience, we will refer to our system as SAM-IAM [1], an acronym for Sculpture of Artistic Models using InterActive Methods.

To our knowledge, no interactive sculpting system has contained the following features, all found in SAM-IAM:

- Virtual tool representation using superquadric functions
- Combination of decay functions with virtual tools
- Adaptive subdivision and adaptive smoothing using virtual tools
- Interactive shadows via shadow volumes and stencil bitplanes
- Use of a virtual trackball for tool and mesh manipulation

The rest of this report is organized in the following manner. Chapter 2 discusses the previous work in polygonal mesh modeling and virtual tools. Chapter 3 describes the features of the computer sculpting system SAM-IAM, while Chapter 4 focuses on implementation issues. Chapter 5 describes some models created using SAM-IAM. Chapter 6 concludes with a discussion of the advantages and disadvantages of the system. Readers interested in a more thorough description of any of these topics should see the original thesis [Bil94].

[1] Pronounced Sam-I-Am. Inspired by Seuss [Seu60].

2. Related Work

While there has been a great deal of interactive modeling research that is marginally related to sculpting, in this chapter we concentrate on research in polygon mesh sculpting and virtual tools. We have chosen the polygon mesh representation over other representations of geometric models because its simplicity allows the user to focus on the manipulation of tools rather than the properties of the surface, over which the user may have less intuition.

2.1 Polygonal Mesh Sculpting

The polygon mesh is a simple and powerful means of representing the geometry of an object; in fact, it is termed *the* most popular type of geometric modeling representation by Paouri et al [PMTT91]. Polygon meshes are well suited to sculpting applications, provided that a platform is available which allows storage and speedy rendering of large numbers of polygons.

The four most significant polygon mesh based sculpting system are those of Parent, Allan et al, Leblanc et al, and Elson and Malone [Par77, AWW89, LPMTT91, Els90b, Els90a]. Parent initiated use of the basic vertex-movement + decay function technique used in all of these systems. His is the only polygon mesh system that uses virtual tools. Allan advanced the use of decay functions and defined "move-vertex" as the fundamental deformation technique; both concepts are reflected in our work. Leblanc chiefly concentrated on improving the interface to an Allan-like system. Elson's papers illustrate models sculpted with the S-Geometry polygonal modeler created by Malone. Unlike the others, this system relied heavily on subdivision and smoothing techniques. Its explanation of the winged edge representation's suitability to polygonal sculpting inspired us to use a winged edge scheme as the foundation for SAM-IAM. Figure 2.1 summarizes the relationships between these polygon-mesh based systems and compares them to our system.

System	Virtual Tools	Mesh Refinement	Decay Functions	User Interface
Parent (1977)	Boolean operations using user-fashioned solid polyhedra	N/A	Simple decay functions	Dials, buttons, keyboard
Allan, Wyvill, Witten (1989)	N/A	Smoothing as a decay function, Overall subdivision	Wide variety with many options	Basic: mouse, keyboard
Leblanc, Kalra, Magnenat-Thalman, Thalman (1991)	N/A	Like Allan's; also individual vertex addition & deletion	Apparently same as Allan's	6D Spaceball, more sophisticated vieiwing
Elson, Malone (1991)	N/A	Robust smoothing and subdivision	Simple decay functions	GUI (as part of animation system)
SAM-IAM (1994)	Simple user-scalable shapes; define-region, push, and pull tools	Variety of *adaptive* smoothing/subdivision methods	Simple functions incorporated with virtual tools	GUI, virtual trackball, interactive shadows

Figure 2.1: Summary of features of polygon mesh based sculpting systems.

2.2 Virtual Tools

While the basic operation of moving a point is central to interactive modeling, virtual tools offer a higher level of deformation control. They allow multiple primitive operations to be applied to a model simultaneously, in an intuitive fashion. Virtual tools have been used in a number of modeling systems, most typically as a method of defining boolean operations to be applied to solid and volumetric models. They are used in this way by Parent (whose polygonal models described solids), Naylor (1990), and Mingxian et al (1993), and in a related way by Galyean [Par77, Nay90, GH91, MFD93]. Mingxian uses CSG-based virtual modeling tools in a non-free-form CAD system. Pushing and pulling points as we propose to do has been done to some extent by Brewer (1977), Hsu (1992), and Szeliski (1992) [BA77, HHK92, ST92]. Brewer pushed parametric surfaces' control points with a planar tool. Szeliski used tools to apply and cancel forces in a particle-based modeler. Hsu used pushing and pulling tools to impart free-form deformation (FFD). The most relevant work is that of Naylor and Galyean; this will be described in more detail below.

Naylor's SCULPT system extended Parent's idea of using boolean operations and virtual tools to sculpt polyhedral representations of solids. At the time of the 1990 paper on SCULPT, the CSG operations making up its sculpting technique were limited to reliance on simple, convex, predefined sculpting tools. Since no semblance to direct vertex movement with decay functions is provided, smooth shape variations would seem a difficult task for SCULPT.

Galyean's volumetric sculpting system makes extensive use of both virtual tools and interesting input devices. Virtual tools are represented as volumetric objects existing in the same space as the model. The tools implemented include a subtractive "routing" tool that removes material, an additive tool or "toothpaste tube" that adds material, and a "heat gun" tool that "melts away" material over time. Smoothing and coloring tools are also implemented, and Galyean includes a shopping list of other tools that would be useful. The user positions the current tool by using a pen-shaped pointing device which returns a stream of position and orientation data to the program. While a number of difficulties arise in its use, it does functionally correlate the user's space and the object space of the program to provide a truly 3D virtual sculpting experience. While the models producible with the system show improvement over previous systems, they remain very limited by the system's spatial resolution and the difficulties in establishing precise, intuitive control with the 3D input devices tested.

The SAM-IAM system advances the work of these researchers in a number of ways. Though virtual tools and decay functions have been implemented previously, no system to our knowledge has used virtual tools in conjunction with decay functions to sculpt a polygonal mesh. SAM-IAM's internal representation of tools as superquadric functions and its methods of tool/mesh collision detection are also extensions of these researchers' work.

3. Features of the System

This chapter presents an overview of the most notable features of the system; polygonal mesh modeling, mesh refinement operations, virtual tools, mesh sculpting, and the user interface. Implementation details of these features are discussed in Chapter 4.

3.1 Polygonal Mesh Modeling

We have chosen to use a polygonal mesh to represent the model being sculpted. A polygonal mesh is completely specified using only points in space and the connectivity of these points. Meshes are effective in sculpting applications due to their simplicity and generality; their simplicity makes them easy to understand, manipulate, and implement, while their generality allows them to represent free-form objects to any desired degree of precision. Benefits include:

- Polygonal meshes are probably the easiest to understand of all representation schemes.
- Polygonal data can easily be converted to and from other representation schemes.
- Digitized data is easiest to represent as a polygon mesh.
- Resolution can be easily controlled using mesh subdivision.
- Many current graphics machines are optimized for the display of polygonal surfaces.

Some of the difficulties associated with the polygon mesh representation are:

- Storage requirements are quite large.
- Large numbers of polygons are usually required to represent smooth surfaces.
- Geometric continuity is much more difficult to control than with higher-level geometric models such as parametric surfaces.

In SAM-IAM, the user initially selects a starting mesh or model, which may be read from a file or may be chosen from a number of predefined initial configurations, including plane, sphere, tetrahedron, cylinder, and box shapes. The user also chooses an initial level of refinement. Default meshes will be triangular, although the systems supports N-faceted polygonal meshes as well. SAM-IAM's only output will be the data files describing the vertices, edges, faces, and normal vectors of the sculpted model.

3.2 Mesh Refinement Operations

Mesh refinement operations differ from the sculpting operations to be explained in Section 3.4 in that they may change the actual topology of a mesh. While the two types of operations are applied differently, the define-region action allows the virtual tool metaphor to be used consistently throughout. The supported mesh refinement operations include adaptive mesh subdivision, smoothing, hole-filling, and deletion of subparts of a mesh. The intersection of the tool and mesh defines the region to apply the operation to. *Subdivision* allows the region to be subdivided to whatever degree is desired. Furthermore, a region can be *smoothed*, which transforms the mesh subregion into one that appears smoother. Depending on the type of smoothing done, the number of vertices in the region may or may not increase. *Hole-filling* allows a hole in the mesh to be replaced with a face. *Deletion* removes all vertices and incident edges in a a region, and can be used to either introduce holes to the mesh or to replace mesh subregions with a single face. After any refinement operation, the resulting mesh can be further refined and/or sculpted.

3.3 Virtual Tools

Virtual tools are 3D geometric objects of different shapes and sizes. The user selects a virtual tool and uses it to sculpt a polygonal mesh into the desired shape. Virtual tools are a very attractive, intuitive means for users to modify a mesh in very different ways while maintaining the consistent interaction metaphor of sculpting a model with a tool. Our tools have two main attributes: shape and action. Currently, the supported basic tool shapes are box, cylinder, and sphere, all of which are rendered as polyhedra. Since the tool can be interactively scaled along any principal axis at any time, a variety of tool shapes such as ellipsoids, cubes, and disks can be generated from the three basic forms.

The three tool actions supported are termed define-region, push, and pull. *Define-region* tools simply define regions of the mesh for the mesh refinement operations to be applied to. *Push* tools act like a solid and push away vertices they come in contact with. *Pull* tools pull vertices in contact with the tool as it is moved. *Define-region* tools simply define regions of the mesh for the mesh refinement operations to be applied to.

The user must also define a currently active *decay function* for the push and pull actions. The decay function determines the way a vertex translation propagates to surrounding vertices. All decay functions possess a type and a range. Figure 3.1 illustrates the basic shapes of SAM-IAM's standard set of decay functions; these include Goo, Bell, Cusp, Cone, and Flat. Decay functions also have a range, which determines the number of surrounding vertices that can be affected. The chosen decay function has a great effect on the way the tool affects the model.

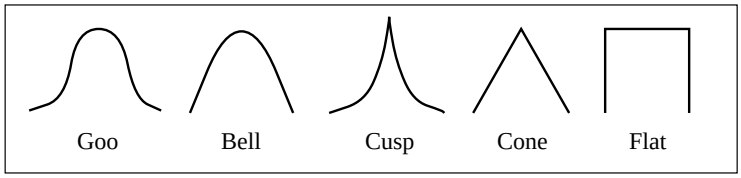

Figure 3.1: Shapes of decay functions in the basic set.

3.4 Collision Detection and Mesh Sculpting

Using the keyboard, a user can toggle between *deformation* and *non-deformation* modes. Non-deformation mode is simply for positioning and viewing of the tool and mesh. In this mode, no collision detection between the tool and mesh takes place.

It is in deformation mode that the virtual tools are actually used. Collisions are detected and result in vertices being translated based on the relative movement of the tool and model. Vertices which collide with the tool are translated directly, while surrounding vertices may also move based on their distance from collision points and the current decay function in use. The overall effect will be to create smooth or sharp bulges and stretches in the mesh. Mesh refinement operations require that SAM-IAM be in deformation mode and that the current tool be a define-region tool. An *undo* facility allows the single most recent deformation to be undone.

3.5 User Interface

Since sculpting systems are intended more for artists and designers than for graphics researchers, highly intuitive interfaces cannot merely be considered desirable– they are crucial. A central challenge of this work is to create as realistic a sculpting metaphor as possible using only the minimal input configuration of keyboard and mouse, and minimal feedback of a single object view. The two most interesting aspects of the interface are its geometric manipulation methods and its incorporation of shadows into the viewing of the tool and mesh.

3.5.1 Geometric Manipulation

The same geometric manipulation methods are used for the tools and the model. At any time, a single object is selected for manipulation. Through simple mouse movements and mouse buttons, the selected object can be translated in the xy plane parallel to the screen and the xz and yz planes perpendicular to the screen plane. Each object has either a local center of rotation (COR), which moves with it as the object is translated, or a global COR that remains stationary. Rotations around the COR can be in any direction and are controlled with a virtual trackball. The trackball also allows rotations to be constrained to a single axis. CORs can be freely edited, providing great flexibility in rotations; the user does this by defining a point in space using a 3D crosshair, and declaring the point to be a global or local COR to either object. Global scaling of the mesh and tools, using sliders, is also possible.

3.5.2 Viewing

The classical method of viewing an object in modeling systems, popular in CAD, is to simultaneously display several projections of the object from different viewpoints. This gives the viewer a clear notion of the object's complete structure, but is problematic in that the user's eyes must keep moving between different views instead of being able to concentrate on a single one. In this way the several-projection interface does not conform to a sculpting metaphor. In SAM-IAM, a single perspective view of the work space is maintained at all times. In order to better indicate the spatial relationship between the tool and model, the tool can cast a shadow on the model, and the positions of the two light sources can be freely modified.

4. Implementation Highlights

This chapter gives a very brief description of the algorithms and data structures used to implement the features described in the previous chapter.

4.1 Winged-Edge Object Representation

SAM-IAM uses the *winged-edge polyhedra* data structure to represent both the mesh and the tools. Winged-edge polyhedra were originally developed by Baumgart [Bau74] to represent solid geometric models. A solid geometric model is generally defined to be a representation of a physically realizable 3D object. As such, solid models have a well defined inside, outside, and surface (or boundary).

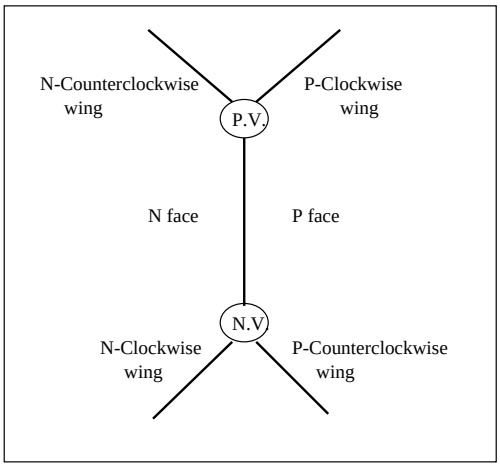

Figure 4.1: Winged edge configuration as viewed from exterior of object.

In a winged-edge representation, as seen in Figure 4.1, each edge is described by a structure containing pointers to its two incident faces, pointers to its two end vertices, and pointers to its four *wings*, which are defined to be the edges incident to the current edge and bordering the edge's two incident faces. Since edges in space have no inherent right, left, up or down, it is necessary to impose an artificial orientation on the edges in order to understand and maintain the spatial relationships between an edge and its vertices, faces, and wings. This is done by assuming that an edge is being viewed from the exterior of the object boundary at all times, and that it is oriented pointing upwards, from its $Nvertex$ to its $PVertex$. Then, the $PFace$ is said to be the face to the right of the edge, and the $NFace$ is said to be on the left. The upper left wing is the PCW, or P-ClockWise wing. Following the same convention, the other edges are termed the $PCCW, NCW$, and $NCCW$ wings. Figure 4.1 illustrates these relationships. Note that an edge may have other incident edges between its wings, but the four wings are the only incident edges it records.

The winged-edge data structure has been chosen to optimize rendering time and ease of object deformation. The implementation we use is essentially identical to that recommended by Hanrahan and Glassner and is described in detail in the original thesis [Han82, Gla91, Bil94].

4.2 Mesh Refinement Operations

The mesh refinement operations supported by our system include adaptive subdivision, adaptive smoothing, hole-filling, and deletion of subparts of a mesh. We make a clear distinction between smoothing and subdivision and support them independently.

Subdivision here refers to the splitting of a face into a number of smaller faces, while smoothing refers to perturbation of the vertices of a mesh in order to give the mesh a smoother appearance. While the two operations are especially powerful when combined, they are also useful independently. The term *adaptive* refers to application of an operation (here subdivision or smoothing) on a *subregion* of the mesh only, with surrounding regions of the mesh being altered only as is necessary to accommodate the changes in the smoothed or subdivided region.

4.2.1 Adaptive Subdivision

Adaptive subdivision allows mesh complexity to be increased only where detail is to be added. This "conservation of complexity" is invaluable in mesh sculpting, since the number of polygons in a region grows very quickly as that region is subdivided. SAM-IAM supports three types of subdivision, which we refer to as subdivision with propagation, subdivision without propagation, and triangulation. Figure 4.2 illustrates the difference between subdivision with propagation and subdivision without propagation. In subdivision with propagation, the faces adjoining the selected region are also subdivided such that no faces with more than 3 sides are created as a result of the subdivision. In subdivision without propagation, no subdivision occurs outside of the selected region, so faces outside the selected region may increase in number of vertices. *Triangulation* refers to subdividing only those selected faces that are not already triangular.

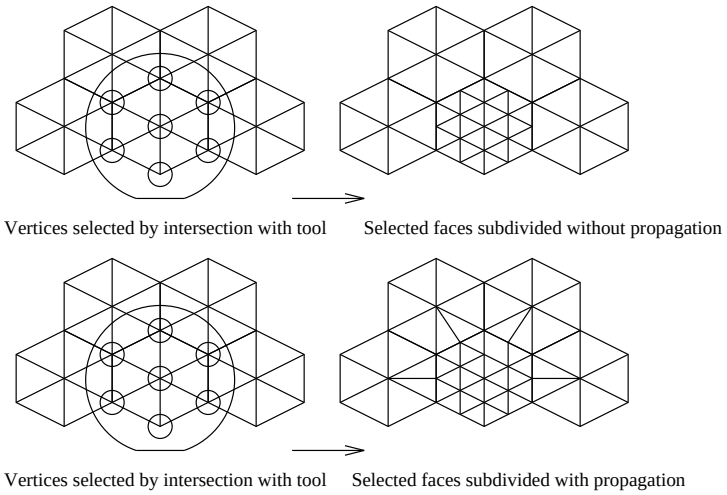

Figure 4.2: Triangular subdivision, with and without propagation.

A single function provides high-level control of all three types of subdivision using the following algorithm:

1. Set propagation flag to true if propagation requested
2. Clear data flags on all vertices and faces
3. Get the list of selected faces
4. Partition this list into lists of triangular and N-sided faces
5. Call the appropriate lower-level routine for each face in each list

The propagation flag indicates whether the non-selected faces adjacent to subdivided faces should be divided in order to enforce triangularity of the faces adjoining the subdivided region. When propagation is not enacted, no new faces are created outside of the selected region. In triangulation, all non-triangular faces in a region are split into triangular faces. Triangular faces in the selected

region are not modified. Since different lower-level subdivision schemes are used for triangular and N-sided faces, two separate lists of selected faces are created. The two distinct methods of subdividing individual faces are illustrated in Figure 4.3 and described separately below.

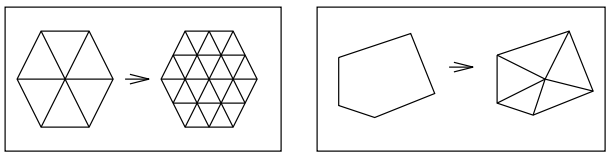

Figure 4.3: Quaternary triangular face subdivision and N-sided face subdivision.

Subdividing Triangular Faces

SAM-IAM subdivides single triangular faces using the *quaternary* method described by Samet [Sam90]. In this method, a single triangle is replaced with four triangles by connecting the midpoints of each edge of the triangle to form a new central face and three new faces adjacent to the central triangle and incident to the original triangle's corners (see fig. 4.3). This method maintains triangularity, limits vertex degree (new vertices all have five or six neighbors) and breaks up existing faces such that the new faces have a fairly regular shape.

While the subdivision of an individual face using this method is a trivial operation, subdivision of a group of connected triangular faces is nontrivial, since the new vertices generated for each original face must be properly incorporated into the subdivision of neighboring faces. Depending on how many neighboring triangular faces have been subdivided, a face to be subdivided may have anywhere from three to six vertices. Since vertices' data flags are set upon vertex creation and pre-existing vertices' data flags were cleared at the beginning of the subdivision process, the data flags can be used to distinguish between old and new corner vertices of the face to be subdivided. By traversing the perimeter of the face (a simple matter using edges' wing pointers), the relationship between old and new vertices is determined and the correct edges to split are found. If the propagation flag has been set, each adjoining non-selected face is split using the new vertex on its perimeter. Four-sided faces are converted to a pair of triangular faces through a split of the face from the new vertex to the vertex opposite it (see Figure 4.2). Faces with more than four sides are split using the function described below.

Subdividing N-Sided Faces

The previous method cannot be used for N-sided faces for several reasons: the central face created will not be triangular, the new faces may have widely varying areas, and the method is not well-defined for concave faces. Therefore, N-sided faces are divided by connecting each corner of the face to a single new central vertex. SAM-IAM currently averages the face's vertices to calculate the new vertex. The face's *centroid*, or center of area (a notion similar to a physical object's center of mass), would lead to a more regular subdivision. Since this method does not introduce additional vertices to adjacent faces, there is no notion of propagation of this type of subdivision; faces can be subdivided with no consideration of adjoining faces. The implementation of this method is therefore much simpler than that of the previous technique.

This method has two principal disadvantages: new faces tend to be long and thin (which can lead to rendering artifacts), and the number of edges incident to some vertices can become very large. The latter problem leads to a disparity in the influence of vertices upon their surrounding regions, since a vertex's influence is directly proportional to the number of other vertices it is adjacent to.

4.2.2 Adaptive Smoothing

Smoothing refers to alterations in vertex positions induced in order to make a region of the mesh appear more smooth. The method used by SAM-IAM calculates a new position for each vertex in the selected region by calculating a weighted average of the vertex's original position and the positions of its neighbor vertices, according to the formula:

$$v' = \frac{v\omega(n) + (v_1 + v_2 + \cdots + v_n)}{n + \omega(n)} \tag{4.1}$$

where

$$\omega(n) = \frac{n\beta(n)}{1 - \beta(n)} \tag{4.2}$$

and

$$\beta(n) = 2(\frac{3}{8} + \frac{\cos(2\pi/n)}{4})^2 - \frac{1}{4} \tag{4.3}$$

.

Here v is the original position of the vertex being smoothed, v' is its new position, n is the number or vertices surrounding v (its *valence*), and $v_1 + v_2 + \cdots + v_n$ is the vector sum of the vertices surrounding v. This smoothing algorithm, originally proposed and analyzed by Loop [Loo87], consists of a subdivision step in which each facet of a triangular mesh is subdivided into four faces as in SAM-IAM, and an averaging step in which the mask above ($\omega(n)$) is used to calculate the new position of a vertex. The best-known smoothing algorithm is perhaps Chaikin's corner cutting algorithm for biquadratic tensor-product B-spline surfaces and its generalization to bicubic tensor-product B-spline surfaces [Cha74]. Unfortunately, Chaikin's algorithm only works for rectangular meshes. The two other best-known smoothing techniques, those of Doo and Sabin [Doo78] and Catmull and Clark [CC78] are generalizations of Chaikin's algorithm to arbitrary meshes but still work best for rectangular and nearly rectangular meshes. Loop's subdivision algorithm seems to be the most judicious choice because it is a generalization of the box-spline smoothing algorithm from triangular meshes to arbitrary meshes, and the meshes used in our system are predominantly triangular. Previous experience in sculpting has shown that some sort of smoothing is highly desirable in any sculpting system. It is intriguing, however, that the inventors of sculpting systems are woefully silent about the details of the smoothing techniques that they have used. The models created using Elson and Malone's S-Geometry system suggest that they have used the Doo-Sabin algorithm, though this is not explicitly stated in their publications [Els90b, Els90a]. Allan et al also use a "smoothing" decay function which appears to do a sort of averaging with no subdivision step [AWW89], although no concrete details are provided.

SAM-IAM offers users the option of choosing whether or not to subdivide before applying the smoothing technique. Should the user decide on smoothing with subdivision, subdivision without propagation as described in the previous section is executed, and all selected vertices are then smoothed. Smoothing without subdivision applies the above perturbation technique directly to the selected vertices; the mesh undergoes no change in topology. Since the $\omega(n)$ are fixed for particular n, these values are stored in a lookup table to avoid unnecessary recalculation.

Since SAM-IAM allows adaptive smoothing to subregions of a mesh, models having smooth regions as well as rough, angular regions can be created.

4.2.3 Filling Holes and Deleting Mesh Regions

Users may refine meshes further by filling and creating holes and to replacing mesh regions with single faces. To fill holes, we define the average point of the vertices on the perimeter of the hole, and connect each vertex to this new vertex, replacing the hole as if it was an N-sided face.

A two-step algorithm is used both to replace regions with a single face and to create holes. First, it deletes all edges which border two selected faces, which eventually results in all selected regions being reduced to a single selected face. If the intent was to replace all selected mesh regions with single faces, we are finished at this point. If the intent was to create holes, we then delete the remaining singular selected faces and update the edges around them to point to outside world.

4.3 Tool Definition

Each of SAM-IAM's virtual tools is represented internally in two ways: as an analytical superquadric equation, and as a winged-edge structure of faces, vertices, and edges corresponding to the equation. Superquadrics are extensions to the standard *quadric* equations, which describe the familiar ellipsoid, torus, hyperboloid of one sheet, and hyperboloid of two sheets. The equation is used in collision detection calculations, while the winged-edge structure is used for the rendering of the tool. Both representations will be described in this section. It should be stressed that the two representations must be maintained separately by the program; neither is generated from the other.

4.3.1 Internal Analytical Description

Each of SAM-IAM's tool shapes is described internally using a *superquadric* equation. Superquadrics are a form of *implicit* equations, which have the general form $f(x, y, z) = 0$. Implicit equations are ideally suited for solid modeling and collision detection applications in which the "in/out/on" relationship of the surface to a point in space must be determined. Specifically, the function $f(x, y, z)$ is evaluated at the point of interest $p = (x_p, y_p, z_p)$. $f(x_p, y_p, z_p) < 0$ implies that p is inside the surface described by f, $f(x_p, y_p, z_p) = 0$ implies that the point is on the surface, and $f(x_p, y_p, z_p) > 0$ implies that p is outside the surface.

SAM-IAM's tools are all described by the *superellipsoid*, the superquadric version of the ellipsoid equation. Superellipsoids are described by the equation

$$f(x, y, z) = \left((x/a_1)^{2/\epsilon_1} + (y/a_2)^{2/\epsilon_2} \right)^{\epsilon_1/\epsilon_2} + (z/a_3)^{2/\epsilon_1} - r^2 = 0 \tag{4.4}$$

where
r is the radius of the (unscaled) shape,
a_1, a_2, and a_3 are scaling factors in the x, y, and z directions, and
ϵ_1 and ϵ_2 are squareness parameters in the longitudinal and latitudinal directions, respectively.

The squareness parameters allow the superellipsoid equation to describe angular shapes such as cylinders and boxes as well as basic ellipsoids and spheres. $\epsilon_1 = \epsilon_2 = 0.2$ is used for SAM-IAM's box-shaped tool, while $\epsilon_1 = 0.3, \epsilon_2 = 1$ is used for the cylindrical tool. Setting $\epsilon_1 = \epsilon_2 = 1$ gives the basic ellipsoid equation used to describe SAM-IAM's sphere and ellipsoid tools.

These equations provide a consistent, elegant means to represent tools internally and allow speedy collision detection using the equations' "in/out/on" capabilities. Furthermore, when a tool is scaled, changes to the a_1, a_2, and a_3 factors are all that is needed to reflect the tool's new shape. The use of superquadrics in interactive modeling is described in more detail by Barr and Franklin ([Bar81, FB81]) and Pentland et al ([PEF+91]).

4.3.2 Winged-Edge Representation

While the equation associated with a tool provides an effective means to detect collisions between the tool and mesh vertices, the tool must still be represented in terms of vertices, edges, and faces in order for it to be rendered. For each tool, this information is stored using a winged-edge scheme identical to that used for the mesh. Care is taken to ensure that the initial tool shapes described by the winged edge structure match the shape defined by the equation. Both representations must then be scaled consistently in order to maintain the correspondence between the tool shape the user

sees (its winged-edge structure) and the shape used internally in collision detection calculations (its equation).

4.3.3 Tool Actions

All tools have an *action* in addition to a shape. Each tool stores its own action; the current action refers to the action of the currently active tool. Pull, Push, and Define-Region actions are currently supported. As described earlier, SAM-IAM users toggle between deformation and non-deformation modes while sculpting, and collision detection only takes place in deformation mode.

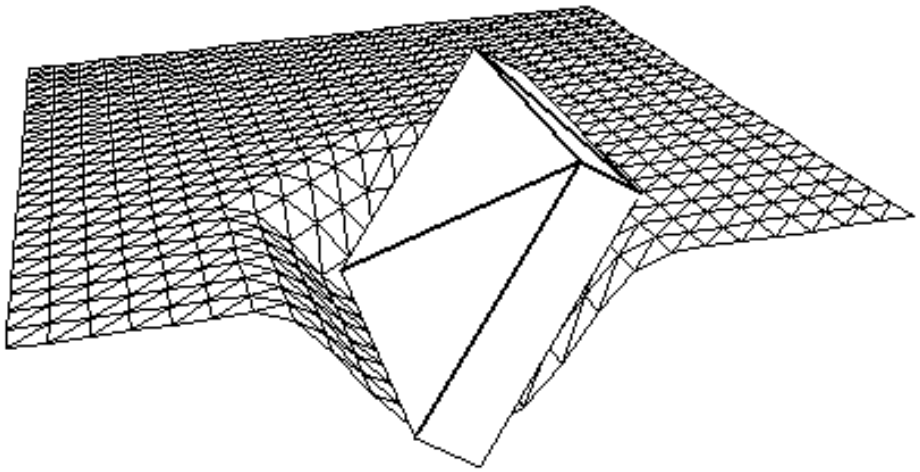

Figure 4.4: Box-shaped tool pushing downwards on flat mesh.

Push

When the current tool action is Push and the system is in deformation mode, a call to the collision detection function, described in section 4.4.1, must be made with each movement of the tool or the mesh. All vertices found to be within the tool then are translated away from it according to the incremental change in translation or rotation describing the movement. This simple current method produces action that is realistic as long as the tool moves into the mesh slowly. Quicker movements can cause vertices to jump away from the tool as it comes in contact with them. Figure 4.4 shows a box-shaped tool pushing down on a flat mesh while a *Goo* decay function with a range of four vertices is active. Decay functions are described in detail in Section 4.4.3.

Pull

When the current tool action is Pull and deformation mode is switched on, all vertices in contact with the tool are recorded via a single call to the collision detection function. During all subsequent tool or model movements until deformation mode is switched off, these vertices will be transformed such that their displacement in the tool's coordinate system remains constant. Figures 4.5 and 4.6 show the results of pulling a single vertex while various decay functions are in use.

Define-Region

Define-Region tools are used to used to define a region of the mesh to apply smoothing or subdivision to, or as a non-destructive means to view the intersection of the tool with the mesh. In deformation mode, the mesh vertices intersecting Define-Region tools are recorded and highlighted, but not moved. Decay functions are not considered when the current tool is of this type. As with

Push tools, collisions must be detected with every movement of an active Define-Region tool. Use of Define-Region and Push tools is therefore much slower than use of Pull tools.

4.4 Tool/Mesh Interaction

4.4.1 Collision Detection

In collision detection, a list of vertices in contact with the tool is generated and stored in a list referred to as *VertSet*. The collision detection function converts each vertex of the mesh into the tool's coordinate space and tests for collision with the tool through direct use of the tool's "in/out/on" equation. Selected vertices (those in `VertSet`) are subsequently marked by displaying a small red sphere at each's location. The current method of examing each vertex of the mesh is a very naive and slow method of collision detection. Our chapter on conclusions and future work explains how better performance could be obtained through spatial decomposition of the mesh vertices.

4.4.2 Decay Functions

Decay functions are a means by which movements of a vertex are propagated to the vertices surrounding it. They are an effective, though not physically correct means to simulate elasticity in real objects. They are also useful for generating deformations to aggregates of points through displacement of a single point.

At all times, SAM-IAM maintains a current decay function *range* and *type*. The range is the distance (in vertices) over which the decay function affects neighbors of a selected vertex (a vertex being moved by a tool). All decay functions DF have the following characteristics:

- $DF : \{0, 1, \ldots, R\} \rightarrow [0, 1]$, where R is the current range
- $DF(0) = 1$

SAM-IAM currently includes the following decay functions:

- **Goo:** $DF(n) = (1 + \sin(\frac{\pi}{2} - \frac{n}{R}\pi))/2$
- **Bell:** $DF(n) = 1 - \frac{n}{R}^2$
- **Cusp:** $DF(n) = (\frac{n}{R} - 1)^2$
- **Cone:** $DF(n) = 1 - \frac{n}{R}$
- **Flat:** $DF(n) = 1$

Figures 4.5 and 4.6 illustrate some of these functions applied to a mesh. Note that Goo and Cusp are the only ones that provide a continuous transition between the vertices in the decay range and the surrounding stationary mesh.

The main problem with this implementation of decay functions is that distances from the selected region are only measured in number of nodes. While this is often very reasonable, in certain situations using the true spatial distance would give better results. Therefore, SAM-IAM's decay functions produce the most intuitive results when applied in mesh regions with a relatively regular distribution of vertices.

4.5 User Interface

4.5.1 Rotation Using the Virtual Trackball

The user rotates objects through use of a *virtual trackball*, a popular interface mechanism through which a user can intuitively specify rotations around arbitrary axes in space using only a mouse. The trackball is represented to the user as a circular screen region corresponding to the forward-facing hemisphere of a simulated trackball. By clicking and dragging the cursor in this region, a rotation

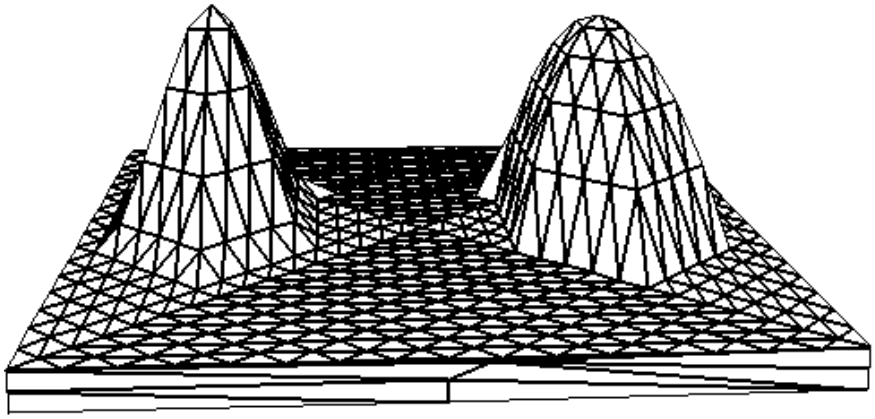

Figure 4.5: Goo and bell decay functions as applied to pulls of a single vertex.

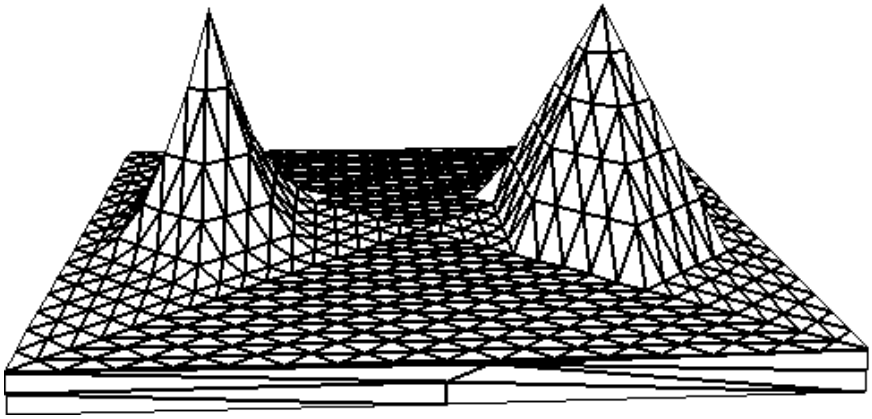

Figure 4.6: Cusp and cone decay functions as applied to pulls of a single vertex.

around some axis is described; the direction of the cursor movement determines the rotation axis, and the initial position of the mouse and the length of the movement vector determine the angle of rotation. Implementation specifics are given by Chen [CMS88].

Modifying the Center of Rotation

The center of rotation (COR) of an object is the point around which it may be rotated. A local COR follows an object as it is moved about; a global COR is unaffected by the transformations applied to an object. SAM-IAM increases the power of trackball-based rotation by allowing users to freely edit the COR of each object. At any time, users may choose to define a point in world space by using a 3D crosshair that is manipulated in the same manner as the mesh and tools. That point can than be declared to be the new COR for the tool or the mesh. Objects' CORs can be declared global or local at any time as well; they are local by default, as this is generally more intuitive and useful.

4.5.2 Shadows

In order to improve the spatial cues provided to users, SAM-IAM is able to display shadows cast by the sculpting tool onto the mesh. The algorithm used is based on the *shadow volume* technique initially proposed by Crow [Cro77] and updated by Hudson [Hud92]. Our algorithm improves upon these methods through use of the stencil bitplanes available on modern higher-end Silicon Graphics

workstations. *Stencils* are bitplanes which are modified by drawing routines according to various pixel-by-pixel *stencil functions* involving current values of the stencil, the zbuffer, and the color bitplanes [Sil90]. The shadows are drawn after the mesh, and before the tool. The process is illustrated in Figure 4.7. An overview of the algorithm is as follows:

1. Calculate the vector to the light source in the tool's coordinate system
2. Calculate the contour polygon of the tool
3. Define the shadow polygons making up the shadow volume
4. Switch off updates to the zbuffer and the color bitplanes
5. clear the stencil plane
6. Draw the shadow polygons into the stencil plane, incrementing stencil pixels wherever polygons would be drawn
7. Switch color bitplane updates back on
8. Using the stencil region with value 1 as a mask, draw the shadow polygons using screen-door transparency
9. Switch zbuffer updates back on

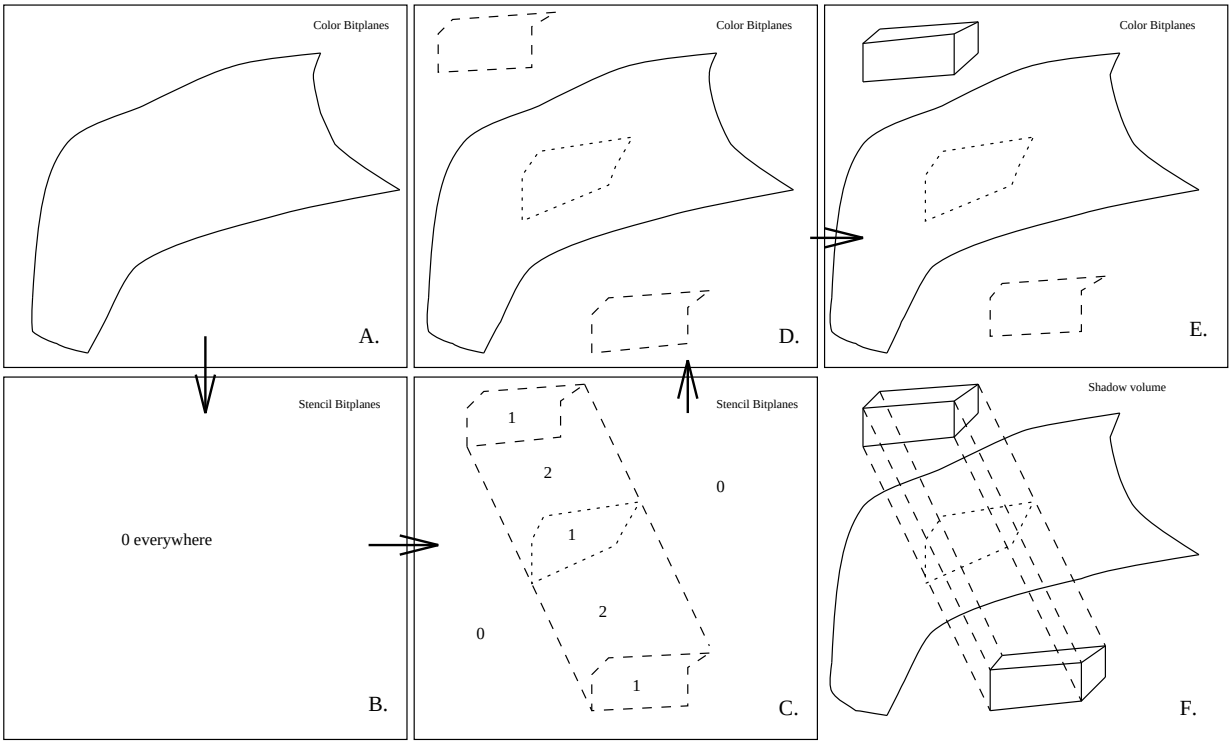

Figure 4.7: Shadow rendering: (A)-(D) are intermediate steps in the algorithm, (E) is the final view after the tool is drawn, and (F) is the final view with the shadow volume visible.

The contour polygon of the tool is the collection of edges which would make up its silhouette as viewed from the light source. Contour edges are identified by calculating the dot product of each of their incident faces with the vector to the light source. Should the signs of these dot products differ, the edge must be a contour edge; one of its faces faces the light source and the other faces away from it. The shadow volume is defined by forming a polygon for each contour edge by connecting the edge with an edge translated some large distance away from the light source. The collection

of these polygons encloses a volume extending away from the light source and having the contour polygon as its consistent cross section. This is the volume that the tool blocks from the light source (see Figure 4.7(F).) After the shadow polygon is created, stencil bitplanes are used to mask the shadow volume display to an area which will correspond to where the shadow of the tool would fall on other objects in its scene. The actual shadows are rendered using *screen-door transparency*, a common technique in which a fill pattern of alternating black and "clear" pixels is used to darken an area without completely obscuring it. The main advantage of this method over those of Crow and Hudson is that it is not necessary to partition shadow polygons into front-facing and back-facing sets, which can potentially be a very costly operation for complex volumes.

4.6 Loading and Saving Objects

The user may save the current mesh object to file at any time. The name of a file of this type may be given to SAM-IAM as an argument at startup, or may be specified interactively using the GUI. Several simple beginning mesh shapes can also be generated spontaneously via a menu. Currently, these include sheet, tetrahedron, box, sphere, and cylinder shapes. Since SAM-IAM allows only one mesh to be sculpted at any time, loading a mesh from file or replacing the current mesh with one of the default shapes entails freeing the memory required by the current mesh (if any).

The format of the files SAM-IAM produces and can read is as follows:

```
   VERTS
n
0 V0[x]   V0[y]   V0[z]   VN0[x]   VN0[y]   VN0[z]
...
n-1 Vn-1[x]   Vn-1[y]   Vn-1[z]   VNn-1[x]   VNn-1[y]   VNn-1[z]
FACES
m
0 FN0[x]   FN0[y]   FN0[z]
...
m-1 FNm-1[x]   FNm-1[y]   FNm-1[z]
EDGES
p
0   NV0   PV0   NF0   PF0   NCCW0   PCW0   NCW0   PCCW0
...
p-1   NVp-1   PVp-1   NFp-1   PFp-1   NCCWp-1   PCWp-1   NCWp-1   PCCWp-1
```

Here n,m, and p are the numbers of vertices, faces, and edges of the object; the first number of each data line is the number of that respective vertex, face, or edge. The V are the vertex coordinates, while the VN and FN are the vertex and face normal vectors. In the lines of edge information, all data fields are integers. Section 4.1 on implementation details regarding to the winged-edge data structure explains the meaning of these fields.

5. Models Created

This section illustrates a few models created with SAM-IAM to demonstrate its capabilities. We first give a detailed description of the creation of a dog model. A collection of other interesting models created using the system is then presented. Each model will be described by both an illustration and complexity statistics (number of vertices, edges, and faces).

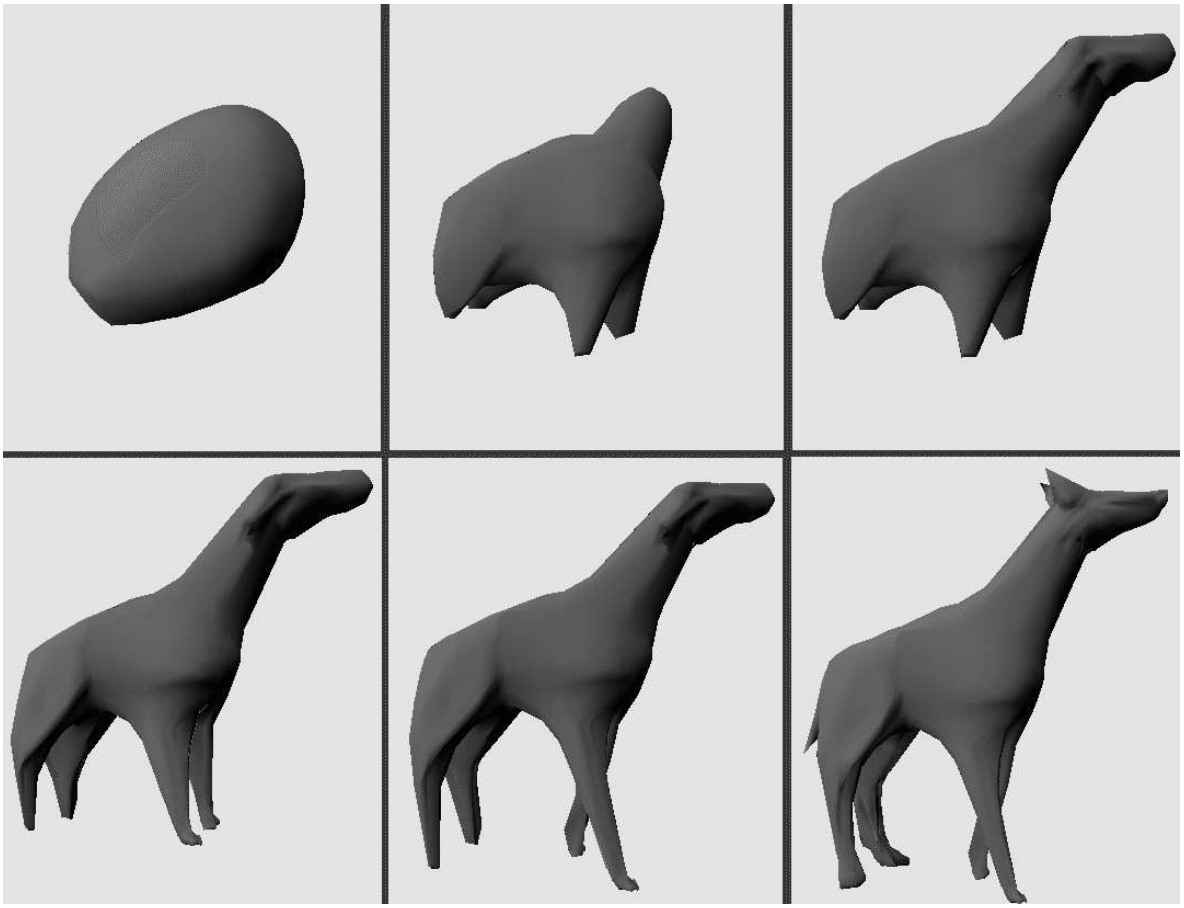

Figure 5.1: Major steps in the creation of a dog model.

5.1 Sculpting a Dog

Figure 5.1 shows six stages in the sculpting of a dog, supposed to be of the Doberman Pinscer variety. The initial ellipsoid-shape, shown in the upper left corner, was created through two applications of subdivision with smoothing to an initial box shape. In the top center image, the beginnings of the legs and neck have been pulled out of the torso. A Goo decay function was used for the rear legs, a Cusp function for the front legs, and a Bell function for the neck. In the third frame, the neck region was subdivided and a crude head shape formed. By the fourth frame, the tips of the legs have been pulled out and front paws added, and in the following frame the legs have been stretched further and pulled into a walking pose. The final (lower right) frame shows the resulting model after the head, paws, and tail regions, respectively, have been subdivided and detailed. The model in this image is made up of 2523 vertices, 4746 faces, and 7267 edges. Figure 5.2 shows a detail after the entire model was smoothed with subdivision one final time, resulting in a model with 9927 vertices,

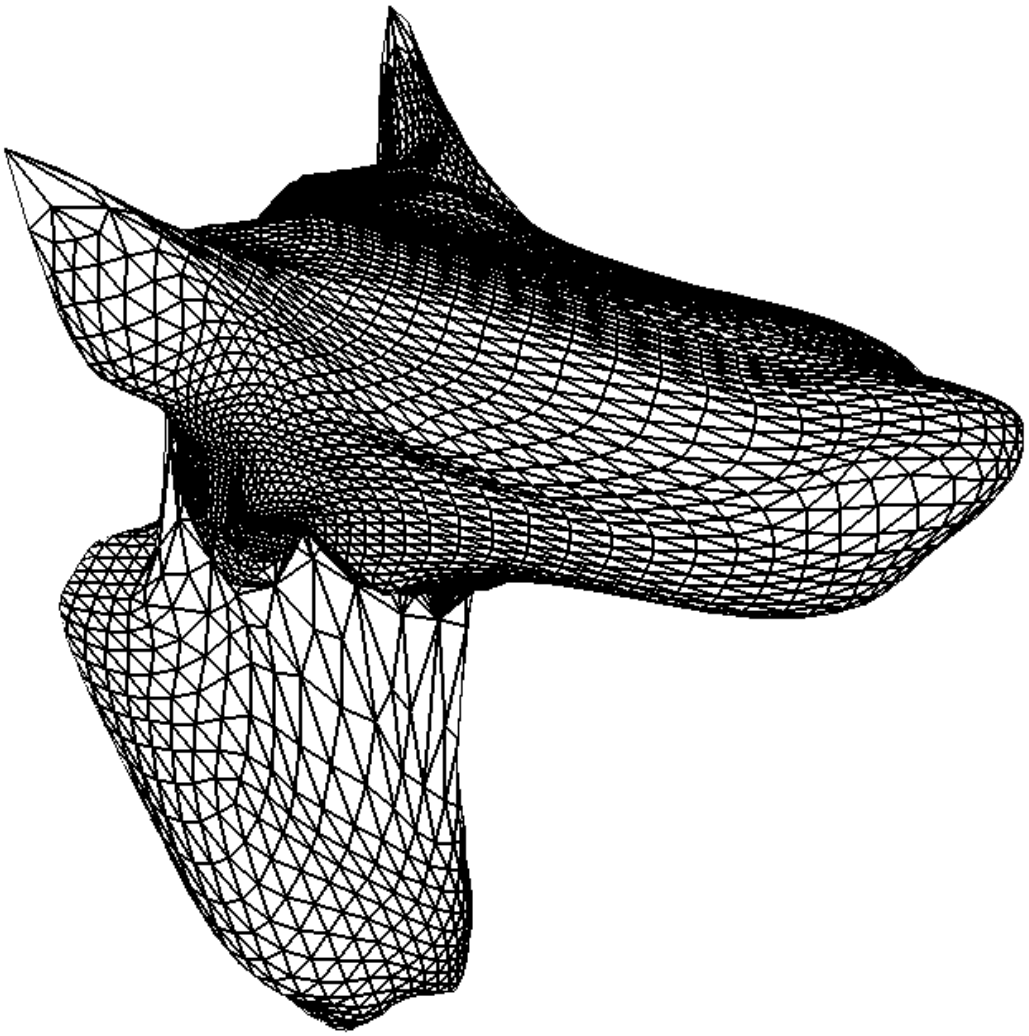

Figure 5.2: Detail of dog face after smoothing with subdivision.

19850 faces, and 29775 edges.

5.2 Other Models

1. **Fish** (Figure 5.3)
   Figure 5.3 presents a simple fish sculpted out of a sphere and rendered in hidden-line mode. Note the adaptive subdivision in regions where more detail is needed, such as the eye, fins, and tail. The fish is made up of 525 vertices, 1489 edges, and 966 faces, and originally was a sphere.

2. **Slug** (Figure 5.4)
   Here we see a 3D version of our campus mascot, the Fighting Banana Slug. This model was created during a live demo of the program and illustrates what may be done in a very short time using the system (the modeling time here was only about 5 minutes). The slug was created from a sphere and consists of 1036 vertices (a large proportion of which are grouped in the highly subdivided eyestalks region), 1966 faces, and 3000 edges.

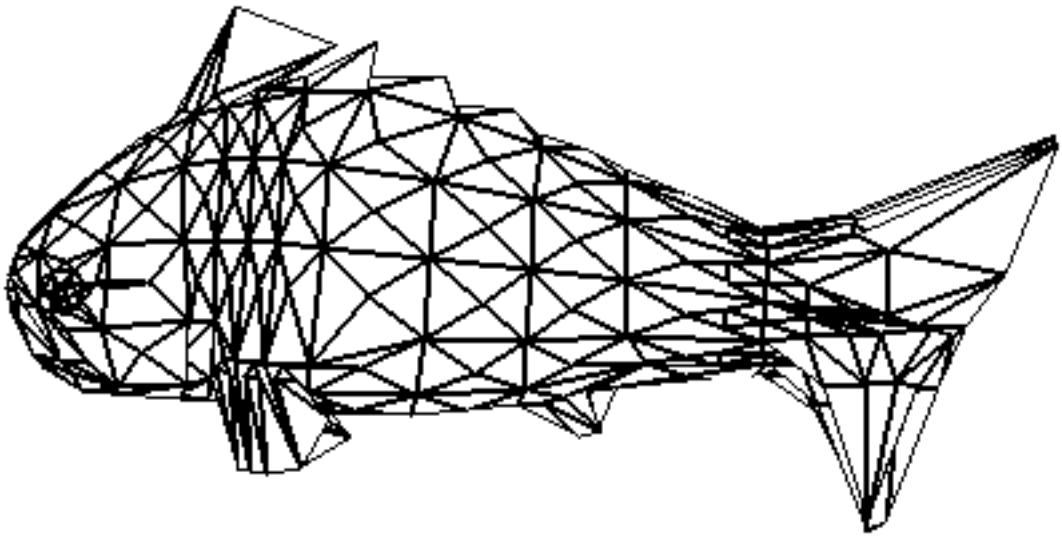

Figure 5.3: A fish.

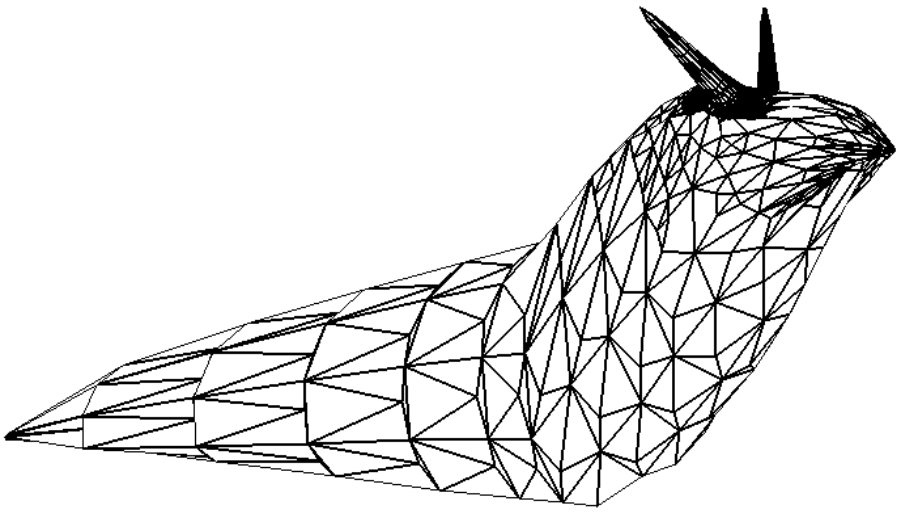

Figure 5.4: A banana slug.

3. **TV Set** (Figure 5.5)
   While the system is designed with the modeling of smooth, naturalistic shapes in mind, angular, non-natural shapes such as this TV set can also be produced. This model consists of 561 vertices, 985 faces, and 1544 edges.

4. **Triceratops** (Figure 5.6)
   A model of the famous dinosaur triceratops is seen in Figure 5.6. The triceratops was initially a cylinder; the final shape consists of 1700 vertices, 3181 faces, and 4879 edges.

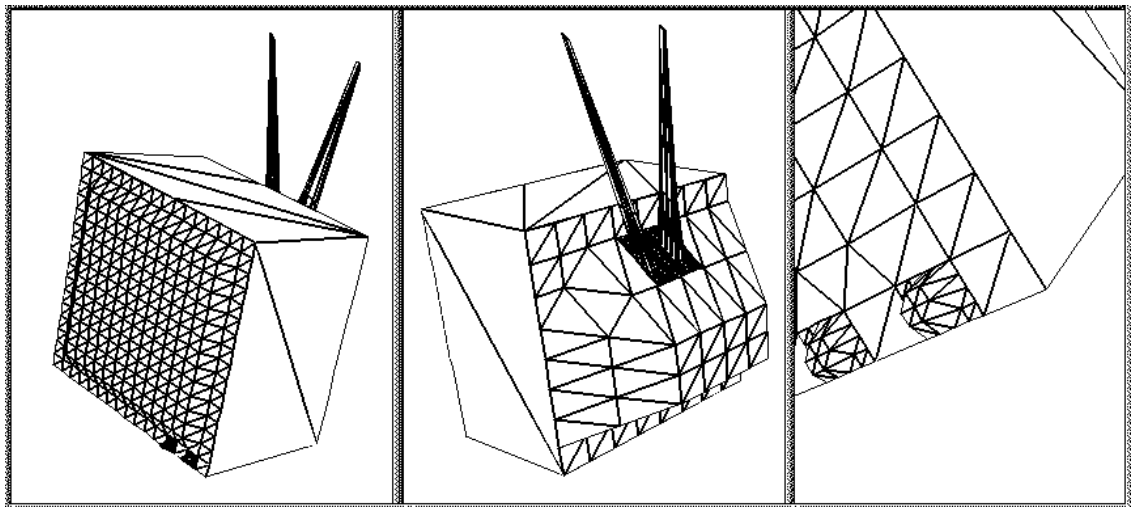

Figure 5.5: Three views of a TV set.

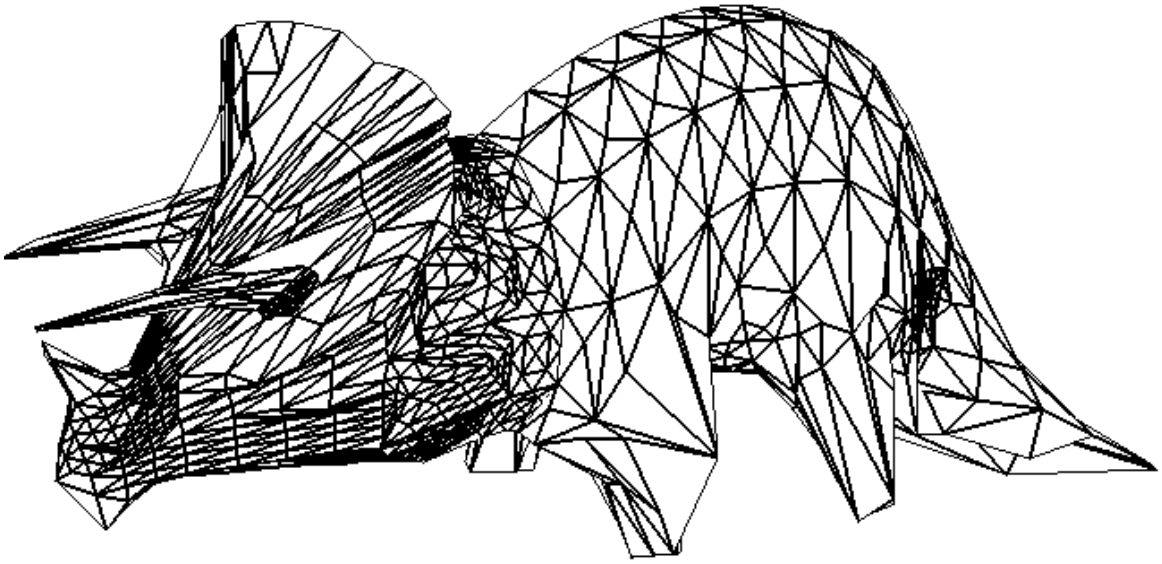

Figure 5.6: A dinosaur: triceratops.

5. **Flower** (Figure 5.7)
   This flower was sculpted during the first modeling session with the program. It was created by pulling up the corners of a flat sheet to create petals, and then pulling a stem and a pistil/stamen structure out of the mesh. The flower model consists of 894 vertices, 2456 edges, and 1563 faces.

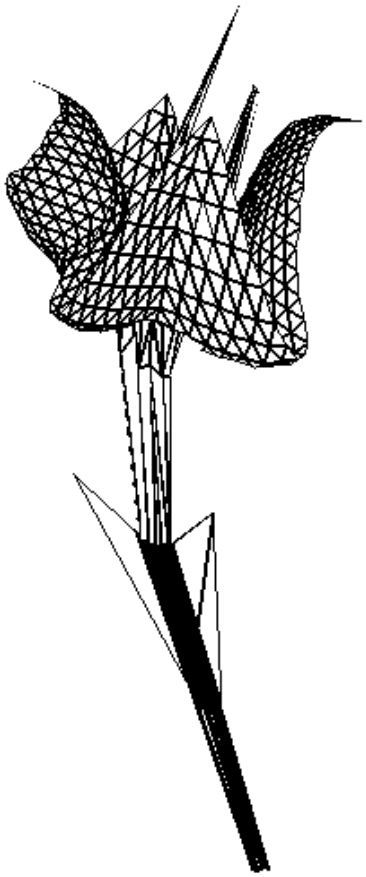

Figure 5.7: A flower.

6. Conclusions and Future Work

6.1 Analysis

In this chapter we evaluate the strengths and weaknesses of our system. Overall, SAM-IAM has proven to be a very easy to use, versatile, and effective system. Its main shortcomings are in speed (some operations are somewhat slow) and in its interface. Some of these problems are inherent in our design; others could be alleviated through the future modifications described in Section 6.2.

6.1.1 Successes

Virtual tools have shown themselves to be highly versatile, intuitive, and elegant means to specify operations to apply to a geometric model. In SAM-IAM, both sculpting operations and mesh refinement operations to be applied using virtual tools as a consistent interaction metaphor. Combining the tools with decay functions increases their usefulness greatly. An important goal of this project was to use tools to extend the power of the vertex manipulation operations implemented in many sculpting systems, and this goal has been achieved. The interactive shadows are immediately seen to be a great aid in understanding the spatial relationship of the tool and the mesh. This results in more accurate sculpting. The adaptive nature of subdivision and smoothing operations makes it very easy to vary detail levels and control smoothness in the mesh being sculpted. Particularly, adaptive subdivision and the ability to interactively replace mesh regions with single faces allows all unnecessary polygons to be eliminated from the mesh; thus, mesh complexity can be conserved.

6.1.2 Shortcomings

SAM-IAM can become bogged down when the number of vertices in the mesh being modeled grows into the low thousands. Since many more points than this can be required to represent complex objects, this is a significant problem. One possible way to reduce memory usage is presented in Section 6.2.

SAM-IAM uses a single view at all times, in an attempt to simulate actual sculpting. It also uses the basic input configuration of keyboard and mouse. Both of these approaches are inherently limited. While we have addressed their respective shortcomings using shadows and the virtual trackball, precise spatial understanding and object positioning can still be difficult. Again, some possibilities for improvement in this area are described in the next section.

Another inherent limitation of our system is its inability to detect and respond to collisions of the mesh with itself. This problem was also described by Allan et al as a shortcoming of their system [AWW89]. While allowing the mesh to intersect itself can sometimes be useful, care is usually needed to prevent the creation of meshes with unattractive crinkles and discontinuous shading due to unwelcome self-intersections. We have, however, found that judicious smoothing of these areas often alleviates the problem.

6.2 Future Work

We have demonstrated the effectiveness of our sculpting software through the example models presented in Chapter 5. This section describes a number of modifications likely to improve the system further. We also discuss the feasibility and potential inherent in combining our system with true 3D interfaces and parametric surface modeling.

SAM-IAM currently uses the naive method of checking all mesh vertices to determine collisions with the tool. Instead, a spatial decomposition technique or bounding-box tests could be used to generate a set of the vertices in the general neighborhood of the tool.

One major current problem with SAM-IAM is the excessive memory requirement due to the winged-edge implementation using the structure recommended by Glassner and Hanrahan ([Gla91, Han82], in which each face and vertex points its own ring of edge structures. In Baumgart's original implementation, each face and vertex stores only a single pointer to *any* adjacent edge, and doesn't have separate edge and edgedata structures. The other edges around the face or vertex are found by following the stored edge's wing pointers. While the ring method simplifies access to edges somewhat, the excessive redundancy in this structure seems to cripple memory unacceptably. In many cases, since the rings are not ordered, we must traverse wing edges anyway to determine their clockwise or counterclockwise ordering. It can be inferred that Baumgart's method uses approximately 1/3 to 1/2 of the memory of the other memory for typical meshes. Neither Glassner or Hanrahan gives an analysis of this organization with regards to space/time complexity. It would be relatively simple to convert to SAM-IAM to use Baumgart's original configuration; we suspect that this would allow larger meshes to be created without noticeable penalties to performance.

While the current push action is fairly realistic, the tool equations could be better utilized to correctly determine the distance from vertices to the nearest point on the tool. This would allow the vertex to be moved in a more realistic manner than is done currently. The vertex could be allowed to cling to the tool or slide against it with a variable degree of "friction". This touches on the very interesting question, explored by many researchers, of whether object interactions with a high degree of physical realism can be generated using only simple geometry, without differential equations [PW89, PEF+90, PEF+91, WGW90, TPBF87, TF88].

A wide variety of fairly low-level extensions could be made to the system that would make it even more useful. For instance, interactive coloring of objects could be done by defining a "paint" tool action in which a selected mesh region can be colored or textured by the user. The ability to manipulate and merge a number of meshes at once would be very useful. Additionally, the file I/O capabilities of our system are still quite primitive; only a single file format is readable. A method of converting from SAM-IAM's file format to those used by other popular graphics programs would obviously be of great benefit.

While the suggestions above are straightforward improvements to our system, many more advanced applications can be envisioned. One interesting enhancement would be to create a truly 3D virtual sculpting system through addition of 3D input devices such as force-feedback units and datagloves. Also, SAM-IAM could be adapted to manipulate control points of parametric surfaces or free-form deformation lattices. A system such as ours would also be very effective as a free-form subsystem to a traditional CAD or CSG-based modeler.

References

[AWW89] Jeff Allan, Brian Wyvill, and Ian H. Witten. A methodology for direct manipulation of polygon meshes. In R.A. Earnshaw and B. Wyvill, editors, *New Advances in Computer Graphics*, pages 451–469, Tokyo, Japan, June 1989. CG International, Springer-Verlag.

[BA77] J. A. Brewer and D. C. Anderson. Visual interaction with overhauser curves and surfaces. *Computer Graphics (ACM SIGGRAPH Proceedings)*, 11:132–1377, July 1977.

[Bar81] Alan H. Barr. Superquadrics and angle-preserving transformations. *IEEE Computer Graphics and Applications*, 1(1):11–23, January 1981.

[Bau74] Bruce G. Baumgart. Geometric modeling for computer vision. Technical Report CS-463, Dept of Computer Science, Stanford University, 1974.

[Bil94] James R. Bill. Computer sculpting of polygonal models using virtual tools. Master's thesis, Department of Computer and information Science, University of California – Santa Cruz, Santa Cruz, CA 95064, 1994.

[CC78] E. Catmull and J. Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer Aided Design*, 10(6):350–355, November 1978.

[Cha74] G. M. Chaikin. An algorithm for high-speed curve generation. *Computer Graphics and Image Processing*, 3:346–349, 1974.

[CMS88] Michael Chen, S. Joy Mountford, and Abigail Sellen. A study of interactive 3-d rotation using 2-d control devices. *Computer Graphics (ACM SIGGRAPH Proceedings)*, 22(4):121–129, August 1988.

[CP76] B. Chandrasekaran and Richard Parent. Moulding computer clay-steps toward a computer graphics sculptors' studio. In C.H. Chen, editor, *Pattern recognition and artificial intelligence : proceedings of the Joint Workshop on Pattern Recognition and Artificial Intelligence*, pages 86–107, New York, 1976. IEEE, Academic Press.

[Cro77] Frank C. Crow. Shadow algorithms for computer graphics. *Computer Graphics (ACM SIGGRAPH Proceedings)*, 11:242–247, July 1977.

[Doo78] E. Doo. A subdivision algorithm for smoothing down irregular shaped polyhedrons. In *Proceedings of the International Conference on Interactive Techniques in Computer Aided Design*, pages 157–165, New York, 1978. IEEE.

[Els90a] Matt Elson. Displacement animation: Development and application. In *Character Animation by Computer: SIGGRAPH 1990 Course Notes #10*, pages 14–36. 17th International Conference on Computer Graphics and Interactive Techniques, Dallas Convention Center, August 6th–10th 1990.

[Els90b] Matt Elson. Winged edge polyhedral models: Their use in the construction of characters, speech, emotion, and body language. In *State of the Art in Facial Animation: SIGGRAPH 1990 Course Notes #26*, pages 21–42. 17th International Conference on Computer Graphics and Interactive Techniques, Dallas Convention Center, August 6th–10th 1990.

[FB81] W. Randolph Franklin and Alan H. Barr. Faster calculation of superquadric shapes. *IEEE Computer Graphics and Applications*, 1(3):41–47, January 1981.

[GH91] Tinsley A. Galyean and John F. Hughes. Sculpting: An interactive volumetric modeling technique. *Computer Graphics (ACM SIGGRAPH Proceedings)*, 25(4):267–274, July 1991.

[Gla91] Andrew Glassner. Maintaining winged-edge models. In James Arvo, editor, *Graphics Gems II*. Academic Press, Boston, MA, 1991.

[Han82] Patrick M. Hanrahan. Creating volume models from edge-vertex graphs. *Computer Graphics (ACM SIGGRAPH Proceedings)*, 16(3):77–84, July 1982.

[HHK92] William M. Hsu, John F. Hughes, and Henry Kaufman. Direct manipulation of free-form deformations. *Computer Graphics (ACM SIGGRAPH Proceedings)*, 26(2):177–184, July 1992.

[Hud92] S.E. Hudson. Adding shadows to a 3d cursor. *ACM Transactions on Graphics*, 11(2):193–199, April 1992.

[Lew93] Renee Lewinter. The objects of your desire. (ready-to-use and custom 3d models for incorporation into computer renderings and animations). *Computer Graphics World*, 16(8):54–60, August 1993.

[Loo87] Charles T. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah, 1987.

[LPMTT91] A. LeBlanc, P.Kalra, Nadia Magnenat-Thalmann, and Daniel Thalmann. Sculpting with the 'ball and mouse' metaphor. *Proceedings of Graphics Interface 1991*, pages 152–159, June, 1991.

[MFD93] Fa Mingxian, T. Fernando, and P.M. Dew. Direct 3d manipulation techniques for interactive constraint-based solid modelling. *Computer Graphics Forum*, 12(3):C237–C248, 1993.

[Nay90] B. Naylor. Sculpt: an interactive solid modeling tool. *Proceedings of Graphics Interface 1990*, pages 138–148, May, 1990.

[Par77] Richard Parent. A system for sculpting 3d data. *Computer Graphics (ACM SIGGRAPH Proceedings)*, 11:138–147, July 1977.

[PEF+90] A. Pentland, I. Essa, M. Friedmann, B. Horowitz, and S. Sclaroff. The thingworld modeling system: Virtual sculpting by modal forces. *Computer Graphics (Special Issue on 1990 Symposium on Interactive 3D Graphics*, 24(2):143–144, March 1990.

[PEF+91] A. Pentland, I. Essa, M. Friedmann, B. Horowitz, S. Sclaroff, and T. Starner. The thingworld modeling system. In F. Deprettere and A. J. Van Der Veen, editors, *Algorithms and Parallel VLSI Architectures. Lectures and Tutorials Presented at the International Workshop.*, pages 425–434. Elsevier, Amsterdam, Netherlands, 1991.

[PMTT91] Arghyro Paouri, Nadia Magnenat-Thalmann, and Daniel Thalmann. Creating realistic three-dimensional human shape characters for computer-generated films. In Nadia Magnenat-Thalmann and Daniel Thalmann, editors, *New Trends in Animation and Visualization*, pages 89–99. Wiley, New York, 1991.

[PW89] A. Pentland and J. Williams. Good vibrations: modal dynamics for graphics and animation. *Computer Graphics (ACM SIGGRAPH Proceedings)*, 23(3):215–222, July 1989. See also art. in NSF Engineering Design Research Conference.

[Sam90] Hanan Samet. *Applications of spatial data structures : computer graphics, image processing, and GIS*. Addison–Wesley, Reading, Massachusetts, USA, 1990.

[Seu60] Dr. Seuss. *Green Eggs and Ham*. Beginner Books, New York, 1960.

[Sil90] Silicon Graphics, Incorporated, Mountain View, California, USA. *Graphics Library Programming Guide*, version 2.0, 1990.

[ST92] Richard Szelski and David Tonnesen. Surface modeling with oriented particle systems. *Computer Graphics (ACM SIGGRAPH Proceedings)*, 26(2):185–194, July 1992.

[TF88] Demetri Terzopoulos and Kurt Fleischer. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. *SIGGRAPH '88 Conference Proceedings*, 22(4):269–278, August, 1988.

[TPBF87] Demetri Terzopoulos, John Platt, Alan H. Barr, and Kurt Fleischer. Elastically deformable models. *SIGGRAPH '87 Conference Proceedings*, July, 1987.

[WGW90] A. Witkin, M. Gleicher, and W. Welch. Interactive dynamics. *Computer Graphics (Special Issue on 1990 Symposium on Interactive 3D Graphics*, 24(2):11–21, March 1990.