

THE UNIVERSITY OF CALGARY

Deformation Based Modelling

BY

Anja Haman

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

OCTOBER, 1991

© Anja Haman 1991



National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

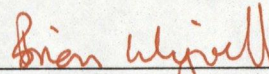
L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-75221-1

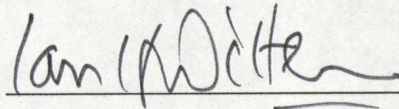
Canada

THE UNIVERSITY OF CALGARY
FACULTY OF GRADUATE STUDIES

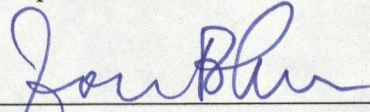
The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled, "Deformation Based Modelling," submitted by Anja Haman in partial fulfillment of the requirements for the degree of Master of Science.



Dr. Brian Wyvill, Supervisor
Computer Science



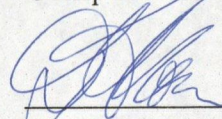
Dr. Ian Witten
Interim Supervisor
Computer Science



Dr. Jon Rokne
Head of Department
Computer Science



Dr. Bruce MacDonald
Computer Science



Dr. Doug Norrie
Mechanical Engineering

Date October 29, 1991

Abstract

Providing control over objects' shapes is a central problem—perhaps *the* central problem—in modelling for computer graphics. Many systems give users access to the underlying representation of surfaces to allow them to define and adjust the shapes they want. This has led to a wide variety of surface types, each requiring software specific to its representation. A different approach is to control shape by moulding predefined surfaces.

This thesis studies *deformation based modelling*, a surface-independent way of allowing users to deform objects to control their shape. Barr operators, free form deformation, and extended free form deformation are three different techniques that provide this style of shape control. Each is described, and the three are compared in terms of the flexibility they offer the user for creating new shapes. They have been implemented in a testbed system, named GROOK, which uses superquadric surfaces as primitives. A new method that combines Barr operators and free form deformation extends previous work by permitting surfaces to be twisted and tapered along curves.

To assess the viability of deformation based modelling, GROOK is compared to five existing systems that represent the state of the art in shape control. It is demonstrated that GROOK matches and extends most of the operations provided by these systems.

Acknowledgements

Firstly, I would like to thank the staff, students, and faculty with whom I have worked during my Masters. The friendly atmosphere at the UofC computer science department is quite remarkable, and I will take with me many happy memories.

I am grateful to Brian Wyvill for reading and commenting on the various versions of my thesis and for financial support.

Sincerest thanks to Ian Witten for his unwavering dedication in the final, crucial months of this thesis. He not only provided me with focus, but also instilled in me the confidence I needed to complete this research. His careful scrutiny of the work and prompt editorial feedback were motivating factors which that greatly improved the final thesis.

Bruce Macdonald and Graham Birtwistle read and commented on early drafts and their keen interest in my progress is very much appreciated. I thank Sue Stodart for her timely proof reading assistance and her scrumptious midnight muffins. Mark James and Eric Schenk spent many hours helping with latex and postscript problems.

Charles Herr's editorial feedback along with many useful discussions helped sustain my enthusiasm. His desk, with its layers of German lunches, drying hockey gear, year old pumpkins, and a sprinkling of texts is an ideal for which I strive. The unflappable Brian Schack spent many hours listening to my incessant worrying, and offered thoughtful advice with unfailing friendship. He is a gem.

Camille Sinanan is a woman I admire immensely, and whose caring and love have helped me grow in countless ways. Her friendship is one I treasure deeply.

My friends Brad, Michelle, Moira, Sherrie, and Thomas patiently awaited my

freedom and offered sunny breaks in the meantime. The Slind family took me into their home with love and kindness, providing warmth, acceptance, and many delicious meals. And they have a nice son.

Without my horse, kayaking, ultimate, and hockey, I might have finished sooner but not as happily. I thank the wonderous Lou for hanging in there, and Cheryl, Sue, and Suzanne for their help and encouragement. Marion Wooden's support over the years is greatly appreciated — her energy and enthusiasm is an inspiration!

Konrad Slind, my "supportive man of the nineties," provided me with love and warmth, and has shown me what patience and perseverance really mean. His continual stream of poetry for the months we spent apart brightened my days (and provided me with new insights about my forehead).

Lastly, I owe much to my family for they are a constant source of love and good humour. My parents, Doug and Wiebke, offer unquestioning support in my endeavors, whether I am writing a thesis or playing sports. To my mother, who is my greatest friend and confidante, I owe more than I can say. Her intelligence, compassion, honesty and good humour combine to make her an extraordinary person. If she could just mess up her house a little, she would be perfect.

Contents

Approval Page	ii
Abstract	iii
Acknowledgements	iv
List of Tables	viii
List of Figures	ix
1 Introduction	1
1.1 Thesis goals	3
1.2 Computer graphics methodology	3
1.3 Modelling	5
1.4 Thesis contributions	8
1.5 Thesis organisation	10
2 Shape control in modelling systems	11
2.1 Parent's polygon based modeller	12
2.2 Form Synth, a system for artists	13
2.3 Delta, a polygon mesh modeller	16
2.4 Cobb's B-spline based modeller	17
2.5 Forsey's hierarchical B-spline modeller	20
2.6 Graphicsland	21
2.7 Summary	21
3 Deformations	24
3.1 Superquadrics	24
3.1.1 Quadric definition as a spherical product	25
3.1.2 Superquadric definition	29
3.1.3 Tangents and normal vectors	29
3.2 An introduction to deformations	31
3.3 Barr operators	34
3.3.1 Tangent transformation rule	35
3.3.2 Normal transformation rule	36
3.3.3 Four deformation functions: taper, twist, bend, and shear	37
3.4 Free form deformation	42
3.5 Extended free form deformation	48
3.6 Summary	50

CONTENTS

4	Design of the GROOK system	51
4.1	GROOK	51
4.2	Superquadrics	54
4.2.1	Definition for superquadrics	54
4.2.2	Polygonising superquadrics	55
4.3	Barr operators	58
4.3.1	Taper	59
4.3.2	Twist	62
4.3.3	Bend	65
4.3.4	Nonlinear shear	66
4.4	Free form deformation	68
4.4.1	Interactive specifications	69
4.4.2	Normals and tangents	72
4.5	Extended free form deformation	73
4.5.1	Freezing the grid	73
4.5.2	Grid formulation	74
4.5.3	Normals and tangents	75
4.5.4	Problems with Newton's iteration	75
4.6	Combining deformation techniques	76
4.7	Summary	81
5	Evaluation	82
5.1	A metric for shape analysis	82
5.2	Global operations	83
5.2.1	Scale	83
5.2.2	Bending	86
5.2.3	Thicken	87
5.2.4	Twist and taper	87
5.2.5	Variable shear	88
5.3	Local deformations	89
5.3.1	Flatten	89
5.3.2	Warps	91
5.3.3	Freeze region	96
5.3.4	Duplicate operation	97
5.3.5	Refinement	98
5.4	Summary and comparisons	99
5.4.1	Comparison to existing systems	99
5.4.2	Comparison to metric and previous capabilities	99
	Conclusions	101
	Bibliography	106

List of Tables

2.1	Summary of Parent's system	13
2.2	Summary of Form Synth	15
2.3	Summary of Delta	17
2.4	Summary of Cobb's system	19
2.5	Summary of hierarchical B-splines	20
2.6	Summary of operations	22
3.1	Four surfaces defined with spherical product	28
3.2	Normals, tangents, and implicit equations for superquadrics	32
4.1	Explicit equations used for polygonising superquadrics	57
4.2	List of error checks for bend operator	66
5.1	Summary of operations and applicable deformation techniques	84

List of Figures

3.1	$h(\omega)$ curve for sphere	26
3.2	Sphere as defined by $\vec{h}(\omega)$ and $\vec{m}(\eta)$	27
3.3	Hyperboloid of one sheet	27
3.4	Torus	28
3.5	Superellipsoids: $E2 = 1.0$; $E1 = 0.2, 1.0, 2.0, 3.0$	30
3.6	Tangents with respect to η and ω	30
3.7	Bend along y in positive z direction	40
3.8	Parabolic and cubic shear functions	42
3.9	Initial grid as specified by Xo , S , T , and U	43
3.10	Example grid	44
3.11	Free form deformation process	46
3.12	Binomial weighting functions for $W_i^3(s)$	47
3.13	Example grid and the area it defines	48
3.14	Example grids used by extended free form deformation	49
3.15	Sampled point lying in convex hull of two grids	50
4.1	The windows in GROOK	52
4.2	Superellipsoid $E1 = E2 = 0.4$	55
4.3	Sampling with explicit equations	57
4.4	Chair seat	58
4.5	Deformations for washer blade	60
4.6	Washing machine agitator	61
4.7	Nonconstant twist applied to a tapered superellipsoid	64
4.8	Chair model	67
4.9	Ellipsoids sheared with superhyperbola	68
4.10	Free form deformation of superellipsoid	70
4.11	Grid volume does not closely fit grid	71
4.12	Grids for which Newton's iteration failed	76
4.13	Curved surface: sheared by $f(y) = 0.3y^2$	79
4.14	Twist along curved surface	79
4.15	Taper along curved surface	79
4.16	Nonconstant twist along curved surface	80
4.17	Taper and twist along curved surface	80
4.18	Twist using free form deformation	80
5.1	Torus scaled with free form deformation	86
5.2	Flattening a region using Barr's technique	90
5.3	Deformation with square base	92

5.4	Deformation with circular base	93
5.5	Grid used for warp with circular base	93
5.6	Warp with arbitrary base	94
5.7	Skeletal warp using free form deformation: C^0 continuity	94
5.8	Skeletal warp using free form deformation: C^1 continuity	95
5.9	Creating a sharp tip in the warp using free form deformation	96

Chapter 1

Introduction

Computer graphics is the science of creating images with a computer. Within that broad definition lie many sub-categories. The field has developed in response to a variety of demands for the pictorial display of information. Two-dimensional graphics is used for business applications to display charts and graphs, by animation studios to aid hand animators with traditional animation methods, and by designers in general—such as graphic artists, architects, and surveyors. Further applications can be found in computer vision, document preparation, and numerous other areas. Three-dimensional graphics opened the door for new techniques in fields such as medicine, physics, and engineering. For example, complex image synthesis is now used by physicians, fluid dynamics can be studied via computer animated simulations, and mathematical functions can be depicted to give mathematicians and engineers a better understanding of the complex functions with which they deal. The automotive and ship building industries have integrated computer aided design and manufacturing systems into their design processes.

Three dimensional graphics has been used as a tool for artists in graphic design, advertising, and the film industry. These applications differ from many of the others because the model being displayed does not need to represent physical reality. Since the image is the final product, many constraints needed for computer-aided design applications can be relaxed. This lack of constraints in the software usually places an extra burden on the designer, but allows a wider range of images to be displayed.

Modelling systems provide the tools for designers to create three dimensional scenes which are then converted to two dimensional images for display. Current modelling systems require:

- a mathematical representation to describe the objects being modelled;
- a set of *operations* for manipulating the objects so represented;
- a user interface which allows the designer to create the desired model.

The model often contains additional, application dependent, information about the object to be stored. Operations for manipulating the model generally consist of *transformations* which reposition and reorient it in three dimensions. *Primitives* are the surfaces that the model offers as its most basic form.

Most early systems used mathematically simple primitives with basic affine transformations such as translation, rotation, scaling, shearing, and reflection. As the need for more extensive shape definition became apparent, research focused on new primitives. Large systems integrating several different representations have been developed to provide designers with a variety of shapes; however, the complexities of data conversion between model formats can be formidable. The introduction of more complex primitives (as a result of faster computers and increased research in the area) allows a wider class of objects to be modelled. This expansion, however, is not without cost. As functions become more sophisticated, the software to process them becomes more intricate and system response time increases.

Another approach to modelling is to shift emphasis from the primitives to the manipulation techniques used to alter them. Altering the shape of a predefined primitive is referred to as *deformation*.

1.1 Thesis goals

The aim of this thesis is to demonstrate that deformation based modelling is a viable modelling technique. Specifically, it is shown that by using a simple set of primitives that can be deformed in a multitude of ways, the designer is able to describe a large number of shapes. The primitives used for this research are superquadric surfaces, which are parametric extensions to quadric surfaces [Bar81]. These are manipulated by three different deformation techniques: Barr operators [Bar84], free form deformation [SP86], and extended free form deformation [Coq90]. In the course of comparing these three techniques, it became evident that, in combination, they provide a very useful modelling system. Thus the techniques are discussed both in isolation and in combination.

1.2 Computer graphics methodology

The process used to create and display computer generated images depends on the application. In general, however, three stages can be identified: modelling, rendering, and animating.

Modelling is the creation and management of the underlying framework from which an image can be made. This framework is mathematically based, consisting of three-dimensional geometric objects plus any related information necessary to the application (such as colour specifications, normal vector information, object inter-connectivity).

Rendering is the display of the model as a two dimensional image. Several rendering techniques are available, providing images of different quality. Wire frame

representations are useful for displaying intermediate images of the model while it is being designed. Hidden line and surface calculations led the way to realistic colour images created with more complex rendering algorithms such as ray tracing [Gla89] and radiosity [FvDFH90]. Today, rendering methods can produce realistic images with various light attributes, shadows, haze, transparency, textures, and many other useful features.

Animating is creating the illusion of movement by displaying slightly different rendered images in rapid succession. Objects in the image can change their position, orientation, colour, shape, texture or any other attribute of the model. Lighting and camera changes can also be used to create the illusion of change over time [FvDFH90]. Although computer animation is used by the entertainment industry [Las87], it is also an important tool in other areas such as simulation [MTT91].

When creating a modelling system, the rendering and, perhaps, animating phases must be considered in the design. The rendering phase receives information about the image from the model. Different rendering algorithms use different information, but most use surface normals and tangents (for lighting calculations), and surface attributes such as colour and transparency. This information is needed for each object in the scene, and is calculated many times during the execution of the renderer. The modeller, then, must not only contain the information needed by the renderer, but also provide efficient access to it. The mathematical representation chosen for the model will greatly affect the information the modeller is able to provide, and how quickly it is able to retrieve it. The modelling information required for animation is discussed in [All88], but is not reviewed here because animation is not addressed by this thesis.

1.3 Modelling

There is a great abundance of modelling techniques. As different applications for computer graphics are discovered, new model descriptions arise. Allan [All88] provides a taxonomy of common techniques, and summarises many of them. This thesis addresses three deformation techniques and applies them to superquadric surfaces. In comparing this deformation based approach to others, various primitives and operations are discussed; they are briefly reviewed here.

Polygon models are one of the more common representations used in computer graphics. Their popularity is partially due to the fact that many other representations can be converted to polygons. Curved surfaces described with this method are approximated with planar polygons. *Polygon mesh modelling* provides the user with a way to change the geometry of the model, which consists of connected, planar polygons that share edges and vertices [All88].

Spline surfaces are based on approximation or interpolation of a set of *control points*, and are used extensively in computer aided design [Far88]. Many types of splines exist, and are described in [BBB87]. Spline curves are based on the piecewise connection of parametric polynomials. This approach permits complex curves to be represented without the use of high order polynomials, which can be unstable and inefficient [BF91]. Just as segments are joined to describe curves, spline *patches* can be joined to form surfaces.

Spline patches are defined by a set of *control points*, which form a *control mesh*. The user adjusts the position of the points in the mesh to alter the shape of the surface. The way in which the surface reacts to this change depends on the under-

lying polynomials, which are often based on either Bezier or B-spline curves. Bezier curves interpolate the first and last control points, and B-splines approximate all control points. This difference has many effects on the surface models, as outlined in [BBB87]. The most notable is that Bezier curves touch the first and last control points that define them, while B-spline curves do not touch any. Spline surfaces in general permit the user to define a surface's shape by adjusting the control points as desired, and in this way permit many shapes to be achieved.

Most modelling systems allow the models to be manipulated in some form. The standard operations are *translating*, *rotating*, and *scaling*. *Shear* and *reflect* operations are also used. Besides the three basic transformations, modelling systems usually allow designers to add, delete and save objects in the scene. Some other useful functions are:

- hierarchical ordering of objects (so that groups of objects can represent one item in a scene; such as a person made up of a head, a torso, two arms, and two legs);
- altering object attributes such as color, transparency, and texture;
- simple but fast rendering techniques to aid scene design.

The geometry of a surface can be altered by scaling or shearing it. These can be applied in any direction by rotating the surface in between manipulations. Other methods for altering an object's shape have recently attracted interest. Modelling with implicit surfaces [KAW91], for example, provides a method to model "soft" objects by blending primitives. This blending of surfaces is defined by the surface

representation itself. Methods to provide metamorphosis of primitives have proven useful for animation [KPC91], while physically based models have proven very effective for producing more realistic shape change in animation [TPBF87], [PB88].

One important technique for interactively changing a surface's shape is through *direct manipulation*. This permits the user to pick a surface point and interactively reposition it. The way in which the surrounding region reacts to this repositioning depends on the implementation, but can be specified by various pre-defined smoothing functions [All88]. A technique for directly manipulating B-spline surfaces has recently been developed, where the surface control points which affect the chosen region are moved in a manner consistent with the user's movement of the surface point [FB88].

Barr [Bar84] introduced deformations which bend, twist, and taper any primitives that can be point sampled. For example, they have been applied to B-spline surfaces [Cob84], and superquadric surfaces [Bar84]. Barr's work was followed by a new method, free form deformation, which permits interactive deformation of surfaces in a free form manner [SP86]. This work was extended by [Coq90]. This free form approach is based on deforming three-dimensional space, and can deform all point sampled surfaces. It has been used to deform polygonal models, implicit surfaces, spline patches, and surfaces of revolution [SP86].

Deformation based modelling refers to a modelling technique which relies on the deformation of surfaces for providing shape control, where the deformation does not rely on the underlying surface representation. In this sense, direct manipulation does not qualify as a deformation based technique. Deformations may be classified as *local* or *global*; the former manipulates only part of a surface, while the latter applies to

the entire surface.

This thesis examines deformations as applied to superquadric surfaces, where the deformations are described by Barr operators and free form deformation. Physically based models and blending paradigms are not addressed.

1.4 Thesis contributions

In order to analyse the effectiveness of deformation based modelling, a testbed was written. Named “GROOK,”¹ this allows experimentation with superquadric primitives, and with their deformation via Barr operators, free form deformation, and extended free form deformation. The contributions of this thesis to superquadrics, Barr operators, and free form deformation are:

Superquadrics

- review of Barr’s superquadrics: the superellipsoid, the superhyperboloids of one and two sheets, and the supertoroid;
- correction to the superquadric definition as defined by [Bar81].

Barr Operators

- review of twist, bend, and taper operators [Bar84];
- a new operator to shear objects;
- a design specification for the parametric implementation of twist, bend, taper, and shear operators;

¹A name first coined by Piet Hein for his poetry. Hein’s work with superellipses and superellipsoids has been integrated into various architectural designs [Gar77].

- a list of possible errors for each of the four operators.

Free form deformation

- a review, including normal and tangent calculations for deformed superquadrics;
- problems experienced while using the technique;
- a technique for twisting and tapering objects in the free form deformation environment.

Extended free form deformation

- a review of the method;
- problems experienced while using the technique;
- a method for twisting and tapering along curved surfaces.

Deformation based modelling

- the integration of the above techniques into a testbed system;
- an analysis of deformation based modelling using GROOK;
- a demonstration that neither Barr operators nor free form deformation techniques provide adequate shape control by themselves;
- a demonstration that free form deformation can be made more general by applying Barr operators to the grid's volume.

1.5 Thesis organisation

Chapter 1 provides background information for modelling techniques, with emphasis on deformation. Chapter 2 reviews five modelling systems and analyses the shape control they offer the designer. The types of shapes provided by the systems are listed, along with the techniques used to create the shapes. This list is used as the basis for the analysis of the GROOK modelling system.

Chapter 3 provides background needed to discuss the primitives and deformation techniques used in GROOK. Superquadric primitives are summarised. A detailed description of Barr's deformation operators is presented, as well as a new operator to variably shear surfaces. Free form deformation [SP86] and its extension [Coq90] are also described.

Chapter 4 describes the design and implementation of GROOK, including user interface issues. Chapter 5 contains an analysis of GROOK with respect to the operations listed in Chapter 2. Images created with GROOK are used to illustrate the versatility of deformation based modelling. It is shown that neither free form deformation nor Barr operators alone can adequately provide shape control, but together they do. New deformation techniques combining these methods are presented.

Chapter 6 concludes that deformation based modelling is a viable technique for achieving control over shape in modelling for computer graphics.

Chapter 2

Shape control in modelling systems

This chapter summarises five modelling systems in terms of the shape control they offer the user. A list of the operations they use for shape change is presented, along with the type of change the operation effects, so that the shape control offered by the GROOK modelling system can be analysed in Chapter 5 with these operations in mind.

Modelling systems must provide the user with a way to generate a desired model easily. What makes a modeller useful depends on how well it meets the needs of the application it is used for. There are several criteria by which such systems can be usefully judged, such as those outlined by [All88]. Unfortunately, there do not seem to be any generally held criteria that address the basic issue most fundamental to the designer's needs: shape control. If a designer cannot achieve the required model, the system will not be used. This thesis is concerned with shape control, with the understanding that the other criteria must also be considered for a complete modelling system analysis.

A modelling system developed to suit designers, artists and animators must permit flexible control over shape. How this is attained has not generally been addressed in the literature. Trying to classify the set of all shapes is beyond the scope of this thesis, however, achieving a high level of shape control requires some analysis of the problem. One way to better understand what shapes are useful for computer graphics applications is to analyse existing systems. Although many mod-

elling systems exist, most are not described in the literature, and are not available for public use. The modelling systems summarised in this chapter were chosen because they were available, and they offer shape control of various surface types. Here we outline modelling systems with respect to their primitives, the operations used to manipulate the primitives, and the motivation behind their design. This thesis addresses shape change as applied to individual surfaces, and therefore operations relating surfaces, such as blending [RO87], [KAW91] or metamorphosis [KPC91], are not considered. For each system, we summarise the types of manipulations and primitives it provides.

2.1 Parent's polygon based modeller

Parent [Par77] developed an early three-dimensional system that incorporated deformation based tools. It is based on the manipulation of objects as a sculptor would manipulate a piece of clay. Polygonal representations that can be manipulated with cutting, bending, and warping operations are used. Some three-dimensional primitives are predefined, but the user can create a new solid by providing a two-dimensional polygon which is then extruded to form a three-dimensional solid.

Table 2.1 summarises the functions available. *Cutting* involves calculating the difference of two overlapping polyhedra, implying the need for intersection calculations. New primitives can be created by taking the union, intersection, or difference of two intersecting polyhedra.

The *warp* command allows the user to push in or pull out a region of the surface. It allows the user to reposition one or more polygon vertices without affecting other

Primitives	Operations	Definable Shape
Polygonal	Cut	Slice through primitive
	Bend	2D Bend of 3D primitive
	Warp	Direct manipulation

Table 2.1: Summary of Parent's system

areas of the primitive. The user may select a point and move it to a new location, moving neighboring vertices as well if desired. The size of the region to be deformed is determined by a constant that represents the number of edges to be traversed from the chosen vertex to the outer boundary of the deformed region. The distance each vertex in the deformed region is to be moved is determined by a selection of weighting functions.

The *bend* operator allows the user to reposition a skeletal approximation of the surface rather than moving surface vertices directly, which can be a tedious task when deforming regions with many vertices. It is implemented by mapping a three-dimensional primitive to a two-dimensional skeleton of line segments. For each vertex in the primitive the nearest segment is calculated, and associated with that vertex. When a line segment is moved, all associated vertices in the primitive are moved in a similar manner.

2.2 Form Synth, a system for artists

Form Synth [Lat89] is a three-dimensional solid modelling system designed to offer artists a new art style. The emphasis in the design of the system is on allowing artists to *evolve* complex forms, rather than use the system to depict objects already

envisioned. The design is based on techniques already familiar to artists such as chiselling of stone, molding of clay, and carving of wood. The system simulates an artist's environment in the sense that objects can be manipulated directly through an interactive interface, but also defines a new art style by offering a set of rules specific to computer graphics.

Nine polygonal, three-dimensional, geometric primitives are offered: tetrahedron, cube, octahedron, dodecahedron, icosahedron, sphere, cone, torus, and cylinder. Only three-dimensional primitives are offered as it is argued that extruding two-dimensional curves into three-dimensional solids is not in keeping with the way artists think. The primitives used by Form Synth are based solely on one artist's requirements and are therefore not meant to be a comprehensive set of surfaces for all applications. This is justified by the fact that all artists are restricted by the rules which govern the art style they choose to work with, computer graphics being no exception. A general modeller permitting arbitrary shapes to be created was therefore not a design goal for Form Synth.

Each primitive has a set of *surface points* and a set of *internal points* associated with it to facilitate shape change operations. There are five such operations, summarised in Table 2.2. Beak, scoop, and bulge operate on surface points, which are scattered over the surface of each primitive, while stretch and slice operate on internal points, which are predefined locations near the center of each primitive. Local coordinate systems are also associated with all primitives. Complex objects composed of several primitives can be achieved through add and subtract commands, though these do not directly alter the shape of one primitive and are therefore not discussed here.

System	Operations	Definable Shape
Polygonal	Beak	Angular protrusion
	Scoop	Hollowing
	Bulge	Spherical extrusion
	Slice	Slice through primitive
	Stretch	Scale along axes

Table 2.2: Summary of Form Synth

Beak allows the user to choose a surface point and have an angular extrusion occur at that point on the primitive. This is much like pulling a polygon vertex out from the surface, and having all polygons containing that vertex move accordingly. This operation is a subset of Parent's warp operation.

Scoop is like the beak operation, except that the area around the chosen surface point is hollowed smoothly, much like the effect of chiselling stone.

Bulge creates a spherical bulge on the surface at the surface point chosen by the user.

Slice allows a primitive to be cut along one of its three predefined axes.

Stretch allows a primitive to be scaled along one of its three predefined axes.

These five operations simulate the basic strokes used by artists while working with clay (bulge, stretch), wood (beak, slice), or stone (scoop). However, they are restricted to the regions defined by the surface and internal points, as well as the directions defined by the local coordinate systems of the primitives. As a result, each operation limits the user by restricting the position and orientation of the desired manipulation.

2.3 Delta, a polygon mesh modeller

Allan's [All88] system Delta provides a variety of operations which alter the shape of a polygon mesh (see Table 2.3). The design was motivated for computer graphics applications (such as modelling for animation), but has been effectively used for residual limb modelling as well. The primary manipulation technique is to move a vertex in the mesh interactively. By providing options for this basic operation, Delta affords the user considerable shape control.

The move operation is extended by controlling three parameters: the *range* of neighboring vertices which are affected by the move, the *distance* these neighboring vertices move, and the *direction* in which they move. The range affected is determined by the Euclidean distance from the chosen vertex, or by the number of edges to be traversed from that vertex. The distance moved by the neighboring vertices is controlled by various predefined functions, called decay functions. Delta provides six of them: constant, cone, cusp, bell, wave, and random. The direction moved by the neighboring vertices can also be specified: rather than move in the same direction as the chosen vertex, neighboring points can move along their normal vectors, in a random direction, or in a way which makes the material appear elastic. A function which smooths discontinuities is also provided.

Two more options which control the shape of the mesh are given: one to *anchor* vertices and another to *bind* them. Anchoring vertices allows the user to specify regions which are not to be affected by a move operation, even if they fall within the affected region. Binding vertices allows the user to associate vertices of the mesh which are not necessarily neighbors, so that manipulating one vertex affects all others

Primitives	Operations	Definable Shape
Polygon mesh	Move vertex, plus options	Warp surface
	Anchor vertex	Freeze region
	Bind vertex	Duplicate operation
	Refine region	Change level of detail

Table 2.3: Summary of Delta

in the set the same way. This allows the same operation to be duplicated at different locations of the mesh simultaneously.

Finally, Delta allows regions of the mesh to be *refined* so that a finer level of detail can be achieved. Polygons within a given region are divided into smaller ones, or smaller polygons merged to give larger ones.

By offering extensions to one basic technique (the move vertex operation), Allan provides a coherent modeller which affords considerable shape control to the user.

2.4 Cobb's B-spline based modeller

Cobb's modelling system [Cob84] is based on B-Spline patches and was designed for sculpting surfaces in computer aided design applications. The patches are used to form solids using boundary representation. Previous systems based on boolean set operations did not allow the shape control Cobb deemed necessary, and therefore her system offers new techniques for editing B-splines.

Although Cobb's thesis describes several ways of creating graphical objects with B-splines, this analysis will concentrate on the surface manipulation techniques provided. There are seven such methods, summarised in Table 2.4.

Warping creates bumps in a surface. Cobb offers three types of warps, based on the shape of the region being deformed: circular, regional, and skeletal. Circular warp bases the warp region on the distance from the center point of the warp, thereby producing a circular boundary between the undeformed and the deformed surface. Region warp allows the user to specify polygonal shapes which form the boundary of the deformed region. Skeletal warps allow the surface to deform along a user defined polyline. All three warps are applied to the control points of the surface, and therefore rely on the distribution of the control points to give accurate results.

Flattening allows a region of a curved surface to be flattened. The control points of the B-Spline patch which fall into the region are mapped onto a user defined plane, thus flattening the surface and maintaining a fairly smooth transition to the flattened area because of the smoothing qualities inherent in B-splines.

Bending is based on bend operations discussed in [FW83] and Barr [Bar84]. A circular arc bend can be applied to the control points of a surface by having the designer specify three parameters:

- the arc of the bend in radians;
- the range over which the bend should occur;
- the fixed point in the bend (the center of the range if a symmetric bend is desired).

If the points in the control grid are too far apart, the bend may be shorter than desired since it is applied only to the grid and not directly to the surface.

Primitives	Operations	Definable Shape
B-splines	Warp: Circular Region Skeletal	Warp with circular base Warp with arbitrary base Warp surface along curve
	Flatten	Flatten region of surface
	Bend	Bend 3D primitive
	Variable offset and lift	Thicken surface
	Stretch	Scale along axes
	Taper	Variable scale in 2D
	Twist	Twist surface
	Refinement	Refine for local editing Add discontinuity

Table 2.4: Summary of Cobb's system

Variable offset and lifting operations allow a surface to be “thickened” by duplicating it, using the copy as the top of the surface, and joining both together to make a solid.

Stretching allows a surface to be scaled by moving the control points along one or more of the coordinate axes. A variable stretch is provided by allowing the distance moved by a control point to depend on its position in space. By applying this variable shear in two dimensions, a tapering effect can be achieved.

Twisting of a B-Spline surface is permitted by applying a twist to the control points of the surface.

Refinement of a B-Spline offers three operations: one to add control points to a region of the surface for increased flexibility, one to isolate a region so that local editing does not alter other parts of the surface, and one to add a discontinuity to a surface.

Primitives	Operations	Definable Shape
B-splines	Direct manipulation	Direct manipulation
	Create overlay	Change level of detail

Table 2.5: Summary of hierarchical B-splines

2.5 Forsey's hierarchical B-spline modeller

Forsey [FB88] describes a system for editing B-spline patches which incorporates two main advances: a facility for detailed editing of a region in a spline patch without affecting other areas of the patch, and a technique for directly manipulating the surface rather than the control points that define it. The method for local editing is based on hierarchical representation of the spline, where regions that need more control points to define their shape are represented as individual patches, but have their control points stored as relative offsets from the control points of the main patch. In this way local changes will only affect the independent surface patch, yet global changes to the entire patch will retain the local alterations made to the independent regions.

Direct manipulation of the surface allows the user to pick any point and move it, rather than having to choose a point in the control grid. This allows the grids defining the spline patches to become as complex as needed since the user no longer needs to see them, or try to manipulate them.

Table 2.5 summarises the operations Forsey offers to effect greater shape control. Deformations which act on the B-Spline control grid, such as those outlined by Cobb [Cob84], can also be applied to Forsey's models.

2.6 Graphicsland

Graphicsland is a research oriented graphics system developed at the University of Calgary. This section is not part of the survey because Graphicsland is not a modelling system, but rather a collection of modellers, renderers, motion control programs used for animation, and associated interfaces. It is used as a research testbed, and is discussed here briefly because GROOK is tied into it.

PG, the modeller central to Graphicsland, permits models to be read in, instanced, hierarchically organised into scenes, and piped to various renderers and viewing programs. Although it offers other facilities, these are the ones that help extend GROOK. Since GROOK only allows one surface model at a time, PG is used to combine the deformed surfaces created in GROOK into a scene. The figures in this thesis were created by reading a polygonal data file into PG, instancing the model if necessary, positioning the primitives as needed, and saving the file in printer format [WMG86].

2.7 Summary

This chapter examined five modelling systems with respect to the shape control they offer. The union of their shape control operations provides a standard by which to judge other modelling systems (see Table 2.6). In Chapter 5 we analyse the deformation based modeller of Chapter 4 in the light of this standard.

Many of the operations offered by the five systems are similar, but have been implemented in slightly different ways. The bend operator, for example, has been implemented by Parent to allow direct manipulation of the surface, whereas Cobb's

General Operation	System	System Operation
Scale	Most systems	Stretch
Bend	Parent Cobb	Bend along polyline Simulated bending with parameters
Thicken	Cobb	Variable offset and lift
Twist	Cobb	Twist surface
Taper	Cobb	Taper surface
Flatten	Cobb Form Synth	Flatten region Slice
Warp	Parent	Direct manipulation of vertices
	Form Synth	Beak (angular), bulge (spherical), scoop (warp inwards)
	Delta	Direct manipulation of vertices
	Cobb	Circular, regional, skeletal
	Forsey	Direct manipulation of surface
Freeze region	Delta	Anchor vertex
Duplicate operation	Delta	Bind vertex
Refinement	Delta	Refine region
	Cobb	Refine region/ add discontinuity
	Forsey	Hierarchical refinement
Direct manipulation	Most systems	Interactive manipulation of surface

Table 2.6: Summary of operations

technique relies on parametric information. Both, however, allow a surface to be bent. Table 2.6 lists the deformations that result when the operations of the five systems are merged.

Refinement and duplication do not directly address shape change, but have considerable effect on other operations. Refinement permits a better approximation of the surface, and thereby allows deformations of small regions to be depicted accurately. Duplicating an operation is an aid for the designer, who may want several similar deformations to be applied to one surface.

Since the goal of this work is to achieve maximal shape control, it is desirable to choose the most general implementation of a deformation. Chapter 5 introduces a yardstick by which shape change operations can be analysed, and shows that a deformation based modeller can offer most of the operations listed in Table 2.6, while extending some of them and providing new ones as well.

Chapter 3

Deformations

This chapter describes superquadric primitives and how deformation techniques can be applied to them. The first section explains the definition of superquadric surfaces. Then two deformation techniques are described in detail. The first is a functional method developed by Barr which includes three kinds of deformation: taper, twist, and bend. A new operation to variably shear a surface is described. The second technique is a recently developed deformation method, based on the sculptor's paradigm, which lends itself to interactive manipulation. This chapter contains a mathematical characterisation of the deformation techniques. Design and implementation issues are discussed in Chapter 4.

3.1 Superquadrics

Quadric surfaces are defined by the equation:

$$\begin{aligned} Ax^2 + 2Bxy + 2Cxz + 2Dx + \\ Ey^2 + 2Fyz + 2Gy + \\ Hz^2 + 2Iz + \\ J = 0 \end{aligned} \tag{3.1}$$

Examples include the sphere, paraboloid, and hyperboloid. This algebraic representation of quadric surfaces is often used for computer graphics modelling [Bli86], but geometric and parametric representations have also proven useful. Geometrically

defined quadrics are described by one point, two vectors, and three scalars, and are often used in constructive solid geometry to avoid the numerical instabilities associated with the other two representations [Gol83]. The parametric representation uses two parameters to trace the surface in three dimensions, and lends itself to display algorithms. This is the representation on which superquadrics are based, and it is explained below.

3.1.1 Quadric definition as a spherical product

The parametric definition of quadric surfaces is based on the spherical product of two curves [Bar81]. Given two parametrically defined curves $\vec{m}(\eta)$ and $\vec{h}(\omega)$

$$\vec{m}(\eta) = \begin{bmatrix} m_1(\eta) \\ m_2(\eta) \end{bmatrix}$$

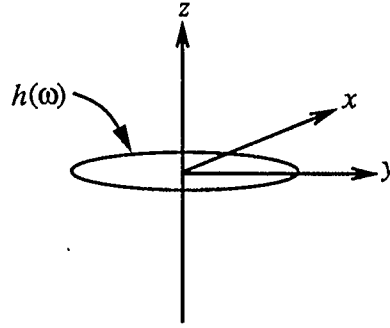
$$\vec{h}(\omega) = \begin{bmatrix} h_1(\omega) \\ h_2(\omega) \end{bmatrix}$$

their spherical product is defined as:

$$\vec{m} \otimes \vec{h} = \begin{bmatrix} m_1(\eta)h_1(\omega) \\ m_1(\eta)h_2(\omega) \\ m_2(\eta) \end{bmatrix}$$

$\vec{h}(\omega)$ defines a curve situated in the xy plane, which is modulated by $\vec{m}(\eta)$. $m_1(\eta)$ scales it in the xy plane, while $m_2(\eta)$ repositions it along the z axis.

To illustrate the geometric relationship between the two curves and the final surface, consider the parametric definition of the sphere:

Figure 3.1: $h(\omega)$ curve for sphere

$$\vec{m}(\eta) = \begin{bmatrix} \cos(\eta) \\ \sin(\eta) \end{bmatrix} \quad -\frac{\pi}{2} \leq \eta \leq \frac{\pi}{2}$$

$$\vec{h}(\omega) = \begin{bmatrix} \cos(\omega) \\ \sin(\omega) \end{bmatrix} \quad -\pi \leq \omega < \pi$$

$$\vec{m} \otimes \vec{h} = \begin{bmatrix} \cos(\eta) \cos(\omega) \\ \cos(\eta) \sin(\omega) \\ \sin(\eta) \end{bmatrix} \quad \begin{array}{l} -\frac{\pi}{2} \leq \eta \leq \frac{\pi}{2} \\ -\pi \leq \omega < \pi \end{array}$$

The curve defined by $\vec{h}(\omega)$ lies in the xy plane of the surface, and describes a full circle (see Figure 3.1). The curve defined by $\vec{m}(\eta)$ defines a half circle, which describes the surface's silhouette in the third dimension by scaling and repositioning $\vec{h}(\omega)$ along the z axis, as shown in Figure 3.2.

The parameters η and ω control the shape of the surface in different directions. Since ω traces the curve in the xy plane, it controls the surface's cross-sections parallel to that plane (this can be thought of as the east-west direction). η controls

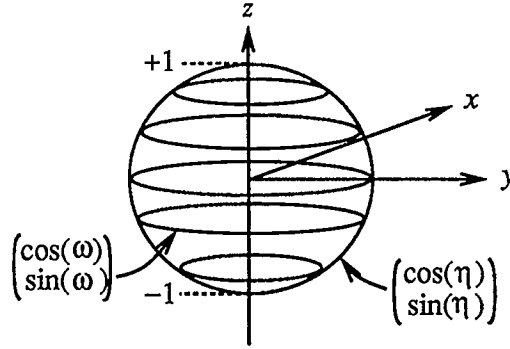


Figure 3.2: Sphere as defined by $\vec{h}(\omega)$ and $\vec{m}(\eta)$

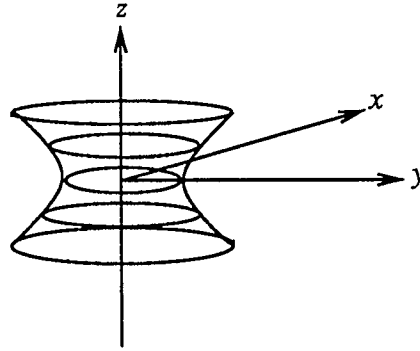


Figure 3.3: Hyperboloid of one sheet

the surface's silhouette when viewed with the z axis vertical (this can be thought of as the north-south direction).

Table 3.1 lists four quadrics based on the spherical product. The ellipsoid is a straightforward extension of the sphere defined above, where a_1 , a_2 , and a_3 are constants. The hyperboloid of one sheet is created by a circle in the xy plane that is modulated by a hyperbola along z . The circle is scaled by $\sec(\eta)$ and repositioned along the z axis by $\tan(\eta)$, to describe a hyperbolic silhouette in the north-south direction (see Figure 3.3).

The curve traced by ω in the hyperboloid of two sheets is a hyperbola. Since ω

Quadric	$\vec{m} \otimes \vec{h}$	Range
Ellipsoid	$\begin{bmatrix} a1 \cos(\eta) \cos(\omega) \\ a2 \cos(\eta) \sin(\omega) \\ a3 \sin(\eta) \end{bmatrix}$	$-\frac{\pi}{2} \leq \eta \leq \frac{\pi}{2}$ $-\pi \leq \omega < \pi$
Hyperboloid (1 sheet)	$\begin{bmatrix} \sec(\eta) \cos(\omega) \\ \sec(\eta) \sin(\omega) \\ \tan(\eta) \end{bmatrix}$	$-\frac{\pi}{2} < \eta < \frac{\pi}{2}$ $-\pi \leq \omega < \pi$
Hyperboloid (2 sheets)	$\begin{bmatrix} \sec(\eta) \sec(\omega) \\ \sec(\eta) \tan(\omega) \\ \tan(\eta) \end{bmatrix}$	$-\frac{\pi}{2} < \eta < \frac{\pi}{2}$ $-\frac{\pi}{2} < \omega < \frac{\pi}{2}$ <i>sheet 1</i> $\frac{\pi}{2} < \omega < \frac{3\pi}{2}$ <i>sheet 2</i>
Torus	$\begin{bmatrix} (a + \cos(\eta)) \cos(\omega) \\ (a + \cos(\eta)) \sin(\omega) \\ \sin(\eta) \end{bmatrix}$	$-\pi \leq \eta < \pi$ $-\pi \leq \omega < \pi$

Table 3.1: Four surfaces defined with spherical product

is extended to cover two ranges, two curves result (one in positive x and the other in negative x). The hyperbolas are scaled by $\sec(\eta)$ and repositioned along the z axis by $\tan(\eta)$, which together describe a hyperbolic silhouette edge.

The torus cannot be expressed by equation 3.1, and is therefore not a quadric surface. However, it can be defined as a spherical product since it is based on circular cross-sections in the xy plane. $\vec{m}(\eta)$ describes a full circle which is translated from

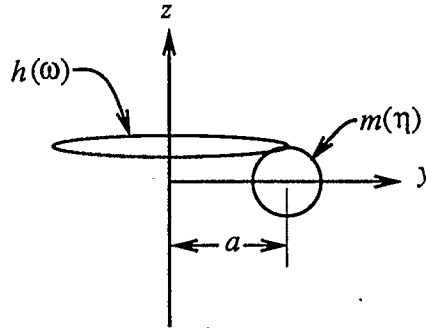


Figure 3.4: Torus

the origin by adding a . Figure 3.4 shows one of the $\vec{h}(\omega)$ curves tracing $\vec{m}(\eta)$. To avoid self-intersection, the constant a must be greater than one.

3.1.2 Superquadric definition

By raising the trigonometric functions of the surface definitions in Table 3.1 to exponents other than one, a superset of quadrics is defined [Bar81]:

$$\vec{m} \otimes \vec{h} = \begin{bmatrix} m_1^{E1}(\eta) h_1^{E2}(\omega) \\ m_1^{E1}(\eta) h_2^{E2}(\omega) \\ m_2^{E1}(\eta) \end{bmatrix}$$

The modification to the functions involving η alters the north-south shape of the object, while the modification to the functions involving ω alters the east-west shape. Exponents less than one push the surface outward, exponents of two create surfaces with squarish corners, and exponents greater than two pull the surface inward. Such superquadrics are illustrated in Figure 3.5.

The extra shape control offered by superquadrics combined with their relatively simple definition makes them an appealing set of primitives. Various systems are based on them [HE89], [Pen86], although little is published about implementation details.

3.1.3 Tangents and normal vectors

Barr defines tangent and normal vector formulations as well as implicit equations for each superquadric [Bar81]. Tangent vectors are needed to test surface continuity, while normal vectors are used in shading algorithms, and to determine surface orientation. With explicit representations for normal and tangent vectors for any point

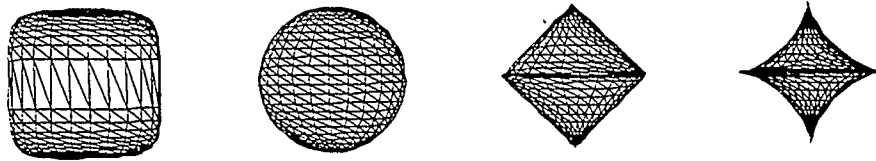
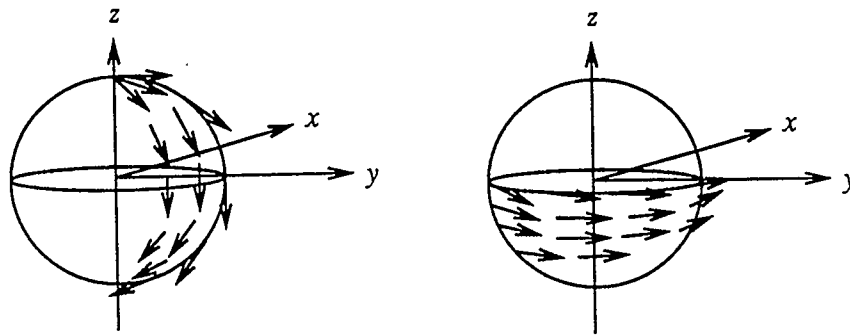


Figure 3.5: Superellipsoids: $E2 = 1.0$; $E1 = 0.2, 1.0, 2.0, 3.0$



(a) T_η tangents

(b) T_ω tangents

Figure 3.6: Tangents with respect to η and ω

on the surface, rendering programs can use exact information rather than approximations based on polygonal vertices. Implicit equations are used for testing point inclusion, and for calculating line-surface intersections. Knowing whether a point is inside, outside, or on a solid is very useful in solid modelling [Mor85], and line-intersection tests are required by ray tracing algorithms [Gla89]. Barr derives the implicit equations but they are not needed for this thesis.

Tangent vectors of parametric surfaces are defined by the partial derivatives of the surface function. Let x_η and x_ω be the partial derivatives with respect to η and ω respectively. Then x_η is the tangent vector in the north-south direction of the surface,

while x_w lies in the east-west direction (see Figure 3.6 (a) and (b) respectively).

Normal vectors can be calculated by taking the cross product of two linearly independent surface tangent vectors [Bar84] such as those discussed above. I tried to verify the normals as specified by Barr, but despite some effort was unable to do so. Maple [CGG⁺88a] was used to calculate the cross products of the tangents, but the results would not simplify to those specified by Barr. Implementation of Barr's normal vector calculations, however, has produced apparently correct normals.

Table 3.2 shows normal and tangent vector specifications, as well as implicit equations, for the four superquadrics. The normals are those derived by Barr. Superquadric surfaces may contain cusps, and at these points tangents and normals are not well-defined. This can cause shading irregularities around the cusp when rendered [Bar81].

3.2 An introduction to deformations

Deformation based modelling is a way of allowing shapes to be deformed by the user. Two recent techniques that address this problem are documented in [Bar84] and [SP86]. An extension to the second method is described in [Coq90].

Barr's deformations were originally introduced in [Bar81], where methods are given for deforming two dimensional curves, three dimensional curves, and three dimensional surfaces. The technique is described more fully, and with a stronger computer graphics emphasis, in [Bar84]. It is based on applying a function to a region in space to deform that space. Surfaces are deformed by sampling the surface and applying the deformation function to the sampled points. The function need

Surface	Normal vector and implicit equation	Tangent vectors T_η and T_ω
S-Ellipse	$\begin{bmatrix} \cos(\eta)^{2-E_1} \cos(\omega)^{2-E_2} \\ \cos(\eta)^{2-E_1} \sin(\omega)^{2-E_2} \\ \sin(\eta)^{2-E_1} \end{bmatrix}$ $(x^{\frac{2}{E_2}} + y^{\frac{2}{E_2}})^{\frac{E_2}{E_1}} + z^{\frac{2}{E_2}}$	$\begin{bmatrix} -E_1 \cos(\eta)^{E_1-1} \sin(\eta) \cos(\omega)^{E_2} \\ -E_1 \cos(\eta)^{E_1-1} \sin(\eta) \sin(\omega)^{E_2} \\ E_1 \sin(\eta)^{E_1-1} \cos(\eta) \end{bmatrix}$ $\begin{bmatrix} -E_2 \cos(\eta)^{E_1} \cos(\omega)^{E_2-1} \sin(\omega) \\ E_2 \cos(\eta)^{E_1} \sin(\omega)^{E_2-1} \cos(\omega) \\ 0 \end{bmatrix}$
S-Hyp1	$\begin{bmatrix} \sec(\eta)^{2-E_1} \cos(\omega)^{2-E_2} \\ \sec(\eta)^{2-E_1} \sin(\omega)^{2-E_2} \\ \tan(\eta)^{2-E_1} \end{bmatrix}$ $(x^{\frac{2}{E_2}} + y^{\frac{2}{E_2}})^{\frac{E_2}{E_1}} - z^{\frac{2}{E_2}}$	$\begin{bmatrix} E_1 \sec(\eta)^{E_1} \tan(\eta) \cos(\omega)^{E_2} \\ E_1 \sec(\eta)^{E_1} \tan(\eta) \sin(\omega)^{E_2} \\ E_1 \tan(\eta)^{E_1-1} (1 + \tan(\eta)^2) \end{bmatrix}$ $\begin{bmatrix} -E_2 \sec(\eta)^{E_1} \cos(\omega)^{E_2-1} \sin(\omega) \\ E_2 \sec(\eta)^{E_1} \sin(\omega)^{E_2-1} \cos(\omega) \\ 0 \end{bmatrix}$
S-Hyp2	$\begin{bmatrix} \sec(\eta)^{2-E_1} \sec(\omega)^{2-E_2} \\ \sec(\eta)^{2-E_1} \tan(\omega)^{2-E_2} \\ \tan(\eta)^{2-E_1} \end{bmatrix}$ $(x^{\frac{2}{E_2}} - y^{\frac{2}{E_2}})^{\frac{E_2}{E_1}} - z^{\frac{2}{E_2}}$	$\begin{bmatrix} E_1 \sec(\eta)^{E_1} \tan(\eta) \sec(\omega)^{E_2} \\ E_1 \sec(\eta)^{E_1} \tan(\eta) \tan(\omega)^{E_2} \\ E_1 \tan(\eta)^{E_1-1} (1 + \tan(\eta)^2) \end{bmatrix}$ $\begin{bmatrix} E_2 \sec(\eta)^{E_1} \sec(\omega)^{E_2} \tan(\omega) \\ E_2 \sec(\eta)^{E_1} \tan(\omega)^{E_2-1} (1 + \tan(\omega)^2) \\ 0 \end{bmatrix}$
S-Torus	$\begin{bmatrix} \cos(\eta)^{2-E_1} \cos(\omega)^{2-E_2} \\ \cos(\eta)^{2-E_1} \sin(\omega)^{2-E_2} \\ \cos(\eta)^{2-E_1} \end{bmatrix}$ $((x^{\frac{2}{E_2}} + y^{\frac{2}{E_2}})^{\frac{E_2}{2}} - a)^{\frac{2}{E_1}} + z^{\frac{2}{E_1}}$	$\begin{bmatrix} -E_1 \cos(\eta)^{E_1-1} \sin(\eta) \cos(\omega)^{E_2} \\ -E_1 \cos(\eta)^{E_1-1} \sin(\eta) \sin(\omega)^{E_2} \\ E_1 \sin(\eta)^{E_1-1} \cos(\eta) \end{bmatrix}$ $\begin{bmatrix} E_2(a + \cos(\eta)^{E_1} \cos(\omega)^{E_2-1} \sin(\omega)) \\ E_2(a + \cos(\eta)^{E_1} \sin(\omega)^{E_2-1} \cos(\omega)) \\ 0 \end{bmatrix}$

Table 3.2: Normals, tangents, and implicit equations for superquadrics

not be linear and therefore extends the traditional operations mentioned in Chapter 1. Barr operators have been integrated into various other systems (e.g. see [Cob84], [FB88]).

The second technique addressed in this thesis, free form deformation, is also based on deforming three dimensional space, but does so by describing a grid that encloses the volume to be manipulated. By interactively deforming this grid, the user can alter any surfaces in it. The initial grid is a parallelepiped, but the deformed one may contain curved surfaces; thus free form deformation permits nonlinear transformations.

Although developed independently, free form deformation is an elegant extension to a similar two-dimensional deformation system described by [BW76]. Their system is used for animation, and allows a character to be “deformed” into its next key position. The grids defining the space are quadrilaterals, and can be altered by moving any of the four vertices.

Although this thesis emphasizes the interactive aspects of free form deformation, it can be used in other frameworks as well. An excellent example is a modelling system [CHP89] that uses deformation to model the musculature of animated characters. The initial grid is associated with a skeletal bone structure, and is deformed according to kinematic and dynamic constraints that mimic the elasticity and contractility of musculature. Interactive manipulation of the grid is provided as a backup to the physical constraint formulation.

3.3 Barr operators

Barr operators are based on mathematical functions that alter space; in computer graphics this has the effect of deforming the objects embedded in this space. A deformation function maps points in object space to a new version of the space. A function to scale an object along the three coordinate axes simultaneously, for example, is described as:

$$F(x, y, z) = \begin{cases} ax \\ by \\ cz \end{cases}$$

This function is applied to surface points, thereby deforming the object. Barr refers to this as a *global* deformation specification.

The key to Barr’s technique, however, lies in what he calls *local* deformations. Barr’s use of the terms local and global differs from the way they are used in most computer graphics contexts. His “global” deformations apply to an entire three-dimensional space, mapping any points in that space to new positions. In contrast, the “local” deformations map the tangent and normal vectors of a surface to different orientations. In this sense, local does not imply that the deformation is restricted to a localized region of the surface, but rather that it applies only to the part of the space occupied by the surface itself—for it is only here that the tangents and normals are defined.

Barr provides two rules that allow tangent and normal vectors of the deformed surface to be calculated directly at any point on it, rather than having to approximate them [Blo88].

3.3.1 Tangent transformation rule

The tangent transformation rule states that the deformed tangent is equal to the Jacobian matrix of the deformation function multiplied by the original tangent vector. The Jacobian matrix is defined as:

$$J = \begin{pmatrix} \frac{\partial F_x}{\partial x} & \frac{\partial F_x}{\partial y} & \frac{\partial F_x}{\partial z} \\ \frac{\partial F_y}{\partial x} & \frac{\partial F_y}{\partial y} & \frac{\partial F_y}{\partial z} \\ \frac{\partial F_z}{\partial x} & \frac{\partial F_z}{\partial y} & \frac{\partial F_z}{\partial z} \end{pmatrix}$$

The example of scaling an object along the coordinate axes is used again to demonstrate Barr's technique. The global deformation function F , defined above, applies different scale factors along the three coordinate axes. Given a biparametric surface (which encompasses all superquadric surfaces):

$$\vec{S} = \begin{pmatrix} x(\eta, \omega) \\ y(\eta, \omega) \\ z(\eta, \omega) \end{pmatrix},$$

the deformed surface definition applies F to \vec{S} :

$$F(\vec{S}) = F(x(\eta, \omega), y(\eta, \omega), z(\eta, \omega)) = \begin{pmatrix} ax(\eta, \omega) \\ by(\eta, \omega) \\ cz(\eta, \omega) \end{pmatrix}$$

The partial derivatives of the deformed surface with respect to η and ω define tangent vectors to the new surface. Since the deformed surface is a composite function of F and S , the chain rule for multiple dimensions is used:

$$\frac{\partial F}{\partial \eta} = \frac{\partial F}{\partial x} \frac{\partial x}{\partial \eta} + \frac{\partial F}{\partial y} \frac{\partial y}{\partial \eta} + \frac{\partial F}{\partial z} \frac{\partial z}{\partial \eta}$$

(and similarly for $\frac{\partial F}{\partial \omega}$). Since the global deformation function F is a vector with separate functions for the x , y , and z components, the partial derivative of each component is taken:

$$\frac{\partial F}{\partial \eta} = \left(\frac{\partial F_x}{\partial \eta}, \frac{\partial F_y}{\partial \eta}, \frac{\partial F_z}{\partial \eta} \right)$$

(and similarly for $\frac{\partial F}{\partial \omega}$). The chain rule mathematics is simplified by the Jacobian matrix of the deformation function, which when multiplied by the original tangent vector gives the above partial derivative equations, and hence the new deformed tangent vector:

$$\begin{aligned} \vec{T}_{def} &= \frac{\partial F(S)}{\partial \eta} = \frac{\partial}{\partial \eta} F(x(\eta, \omega), y(\eta, \omega), z(\eta, \omega)) \\ &= \begin{pmatrix} \frac{\partial F_x}{\partial x} & \frac{\partial F_x}{\partial y} & \frac{\partial F_x}{\partial z} \\ \frac{\partial F_y}{\partial x} & \frac{\partial F_y}{\partial y} & \frac{\partial F_y}{\partial z} \\ \frac{\partial F_z}{\partial x} & \frac{\partial F_z}{\partial y} & \frac{\partial F_z}{\partial z} \end{pmatrix} \begin{pmatrix} \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \eta} \\ \frac{\partial z}{\partial \eta} \end{pmatrix} \end{aligned}$$

(and similarly for $\frac{\partial F}{\partial \omega}$).

3.3.2 Normal transformation rule

The normal transformation rule as given by Barr states that the deformed normal is equal to the inverse transpose Jacobian matrix multiplied by the original normal:

$$\vec{N}_{def} = J^{-1^t} \vec{N}$$

This is proved using the fact that normal vectors are the cross-product of two linearly independent tangent vectors [Bar84]. The determinant of the Jacobian is multiplied by the inverse Jacobian to attain the unit normal, though often only the normal's direction is required.

3.3.3 Four deformation functions: taper, twist, bend, and shear

Barr defines three operators, providing for each one the global deformation function F , the inverse of F , the Jacobian matrix of F , and the inverse Jacobian matrix. In this section we summarize the functions, and introduce a fourth which extends the traditional shear operation. Where formulations are given, the function is applied along the z axis for simplicity; the expressions can be rearranged to apply along the x or y axes instead.

Taper operator

The taper function changes an object's size along one of the coordinate axes. The following global function defines a taper along the z axis:

$$F(x, y, z) = \begin{cases} xf(z) \\ yf(z) \\ z \end{cases}$$

The function $f(z)$ defines the amount of tapering, and is subject to the following restrictions:

- if $f(z) = 0$ a singularity results;
- $f(z)$ must be piecewise differentiable and C^0 continuous.

The derivative of $f(z)$ defines the rate of change applied to the surface. If it is positive the surface will increase in size, while if it is negative it will decrease. If $f(z)$ gives negative values, the surface will be turned inside out.

The Jacobian matrix for the taper function is:

$$J = \begin{pmatrix} f(z) & 0 & x \frac{df}{dz} \\ 0 & f(z) & y \frac{df}{dz} \\ 0 & 0 & 1 \end{pmatrix}$$

The inverse Jacobian multiplied by its determinant yields the normal transformation matrix:

$$\det(J)J^{-1} = \begin{pmatrix} f(z) & 0 & 0 \\ 0 & f(z) & 0 \\ -x f(z) \frac{df}{dz} & -y f(z) \frac{df}{dz} & (f(z))^2 \end{pmatrix}$$

Twist operator

The twist operator twists space continuously along one of the coordinate axes. The global twist function involves rotating x and y around the z axis by θ radians, where θ depends on the position along z , say $\theta = f(z)$.

$$F(x, y, z) = \begin{cases} x \cos(\theta) - y \sin(\theta) \\ x \sin(\theta) + y \cos(\theta) \\ z \end{cases}$$

The derivative of $f(z)$ defines the rate of twisting around the z axis, in radians per unit z . The Jacobian matrix for the twist function is:

$$J = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & -x \sin(\theta) \frac{df}{dz} - y \cos(\theta) \frac{df}{dz} \\ \sin(\theta) & \cos(\theta) & -x \cos(\theta) \frac{df}{dz} + y \sin(\theta) \frac{df}{dz} \\ 0 & 0 & 1 \end{pmatrix}$$

The corresponding normal transformation matrix is:

$$\det(J)J^{-1t} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ y\frac{df}{dz} & x\frac{df}{dz} & 1 \end{pmatrix}$$

Bend operator

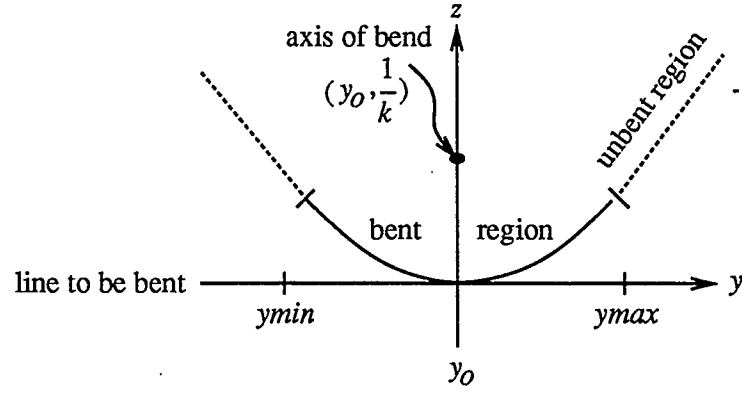
The bend operator bends a line parallel to one of the coordinate axes around an angle θ over a given range. The angle changes linearly in the bend region to map the straight line to a circular arc. Outside this region, the line is merely rotated and translated using rigid body transformations to match the endpoints of the bent region after deformation.

Assume that the line to be bent lies parallel to the y axis, is bent toward the positive z axis, and that the bending range is $(ymin..ymax)$ with the center of the bend at y_0 . Within this region, the bending angle θ is defined by k , the bending rate in radians per unit length, which is multiplied by the distance from the center of the bend to the current position along the y axis: $\theta = k(y - y_0)$. Outside the region θ remains constant since the line only needs to be rotated and translated to its new orientation. Figure 3.7 illustrates a bend.

The global bend function is divided into two functions: one for the bent region and the other for the unbent one. The bent region is:

$$F(x, y, z) = \begin{cases} x \\ -\sin(\theta)(z - \frac{1}{k}) + y_0 \\ \cos(\theta)(z - \frac{1}{k}) + \frac{1}{k} \end{cases}$$

The unbent region is:

Figure 3.7: Bend along y in positive z direction

$$F(x, y, z) = \begin{cases} x \\ -\sin(\theta)(z - \frac{1}{k}) + y_0 + \cos(\theta)(y_b) \\ \cos(\theta)(z - \frac{1}{k}) + \frac{1}{k} + \sin(\theta)(y_b) \end{cases}$$

Here,

- $y_b = y - y_{min}$ if $y < y_{min}$
- $y_b = y - y_{max}$ if $y > y_{max}$.

The Jacobian matrix for the bend function is:

$$J = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta)(1 - k^*z) & -\sin(\theta) \\ 0 & \sin(\theta)(1 - k^*z) & \cos(\theta) \end{pmatrix}$$

The corresponding normal transformation is:

$$\det(J)J^{-1t} = \begin{pmatrix} 1 - k^*z & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta)(1 - k^*z) \\ 0 & \sin(\theta) & \cos(\theta)(1 - k^*z) \end{pmatrix}$$

Here,

- $k^* = k$ in the bent region
- $k^* = 0$ in the unbent region

Shear operator

The above operators are based on basic affine transformations which are then made to vary in some way. The shear operation, which linearly shifts the space along one axis [FP79], can be extended similarly. This section describes a new operation that permits nonlinear shearing.

An example linear shear function is:

$$F(x, y, z) = \begin{cases} x + 2z \\ y \\ z \end{cases}$$

The surface is shifted in the x direction dependent on z . To extend this function, allow the shift value to include nonlinear functions:

$$F(x, y, z) = \begin{cases} x + f(z) \\ y \\ z \end{cases}$$

where $f(z)$ is a nonlinear function which must be single valued, C^0 continuous, and piecewise differentiable. The Jacobian matrix is:

$$J = \begin{pmatrix} 1 & 0 & \frac{df}{dz} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

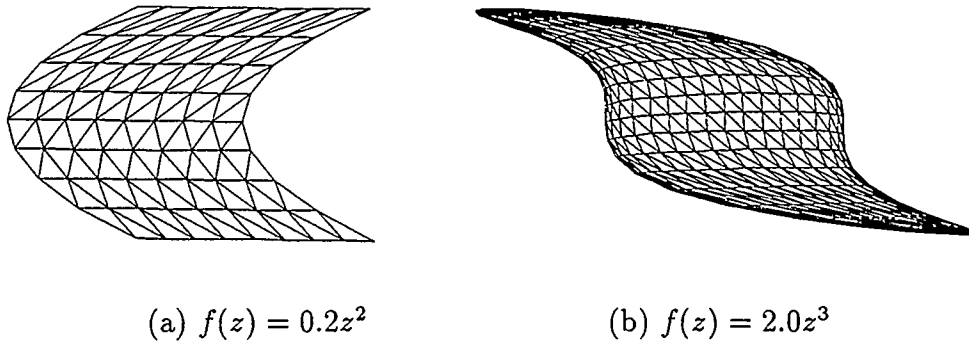


Figure 3.8: Parabolic and cubic shear functions

and its inverse is:

$$J = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -\frac{df}{dz} & 0 & 1 \end{pmatrix}$$

Figures 3.8 (a) and (b) illustrate the shear function applied to a plane and an ellipsoid respectively.

3.4 Free form deformation

Free form deformations are based on a three-dimensional mapping of one space to another, as in Barr's deformations. This involves placing a parallelepiped grid over the volume to be deformed, deforming the grid, and thereby deforming the space it defines. An analogy given in [SP86] describes the grid as a volume of clear plastic which embeds the objects to be deformed; when the plastic is deformed, so are the objects embedded in it. There are three steps:

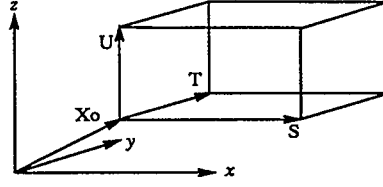


Figure 3.9: Initial grid as specified by X_0 , S , T , and U .

- an initial mapping of each surface point from object space to grid space;
- deforming the grid by moving its vertices;
- a final mapping of the surface points from the deformed grid space back to object space.

The two mappings required to calculate the deformation depend on the grid formulation. The undeformed grid specified by [SP86] is a parallelepiped, which permits the initial mapping from object space to grid space to be linear. Four vectors are used to describe the grid: three to represent the grid's local coordinate system (call them S, T, U), and a fourth to position the grid in object space (X_0). The S , T , and U vectors specify the length of the grid along each of the axes as illustrated in Figure 3.9.

A point in grid space (s, t, u) can be described in object space (x, y, z) by:

$$\vec{X} = \vec{X}_0 + s\vec{S} + t\vec{T} + u\vec{U} \quad 0 < s, t, u < 1 \quad (3.2)$$

To calculate the (s, t, u) coordinates for a point (x, y, z) , as needed for the initial mapping, vector algebra yields:

$$s = \frac{T \times U \cdot (X - X_0)}{T \times U \cdot S} \quad t = \frac{S \times U \cdot (X - X_0)}{S \times U \cdot T} \quad u = \frac{S \times T \cdot (X - X_0)}{S \times T \cdot U} \quad (3.3)$$

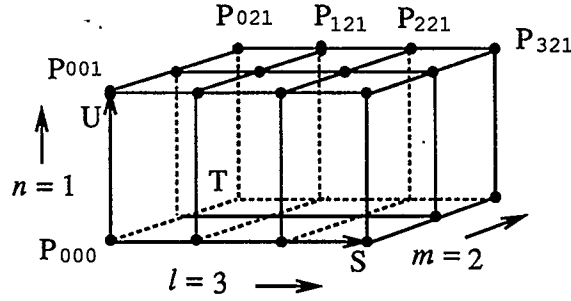


Figure 3.10: Example grid

The parallelepiped grid can be divided into sections by specifying a number of equal divisions along each axis. Let (l, m, n) indicate the number of sections along S , T , and U respectively. The corner points of each section are represented by grid vertices P_{ijk} , which are ordered according to their relative position along each axis (Figure 3.10):

$$P_{ijk} = X_0 + \frac{i}{l}S + \frac{j}{m}T + \frac{k}{n}U \quad (3.4)$$

The grid space coordinates range between zero and one, and represent the relative position of the point with respect to the grid boundaries. For example, $s = 0.5$ implies that the point is in the middle of the grid with respect to the S axis. To deform the grid, its vertices are moved to new locations in object space, which means that the P_{ijk} change values but the local coordinates of the surface point remain the same. It is when these local grid coordinates are mapped back to object space that the deformation becomes apparent: the relative position represented by (s, t, u) is now relative to vertices in new positions, and the surface point is therefore moved from its original location in object space to its deformed position.

If only linear transformations are permitted, the second mapping (from grid space to object space) can be calculated using equation 3.2. Maintaining only linear

transformations, however, is restrictive. Instead, [SP86] specifies the grid space for free form deformations with a tensor product trivariate Bernstein polynomial which defines a Bernstein-Bezier volume [Las85]. This is an extension of the well known Bezier curve and surface used in computer graphics, and allows the deformed space to be curved, thereby extending the types of deformation possible. The outer boundaries of the volume defined by the Bernstein polynomial are composed of Bezier surfaces which are described by the grid vertices of the outer six planes of the initial grid. In fact, each set of vertices with one of i , j , or k in common defines a Bezier surface.

The Bernstein polynomial uses the (s, t, u) grid position of the surface point and the deformed grid vertices to calculate the object space position of the deformed surface point. The polynomial is given in equation 3.5, and the process used to deform a point is illustrated in Figure 3.11.

$$B(s, t, u) = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \sum_{i=0}^l \sum_{j=0}^m \sum_{k=0}^n W_i^l(s) W_j^m(t) W_k^n(u) \begin{bmatrix} P_{ijk}.x \\ P_{ijk}.y \\ P_{ijk}.z \end{bmatrix} \quad (3.5)$$

where

- $W_i^l(s) = \binom{l}{i} (1-s)^{l-i} s^i$ is a binomial weighting function;
- the calculated vector is the deformed point in object space;
- (s, t, u) is the surface point in grid coordinates;
- l, m, n represent the number of sections the grid is divided into along each coordinate axis;

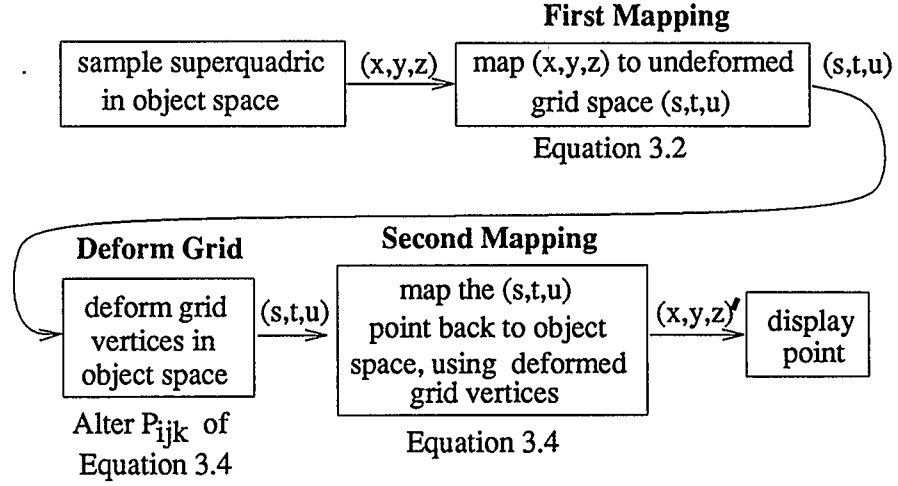


Figure 3.11: Free form deformation process

- P_{ijk} are the coordinates of the grid vertices in object space.

The deformed position of a surface point is calculated by multiplying each grid vertex by a weight which is based on three binomial weighting functions. The weighting takes account of the distance from the point to the grid vertex being considered: vertices that are close to the surface point have higher weights, and therefore receive priority.

As an example, consider the weighting functions applied to the S axis only, where the grid is divided into three sections ($i = 3$):

$$W_i^3(s) = \binom{3}{i} (1-s)^{3-i} s^i \quad 0 \leq i \leq 3, \quad 0 \leq s \leq 1$$

$W_i^3(s)$ describes four curves, one for each set of vertices with the same i index. The value of s defines the weighting that the grid vertex is multiplied by, and as can be seen from Figure 3.12, those vertices closest to s receive the highest weight values. Since the Bernstein polynomial is trivariate, each vertex in the grid is multiplied by

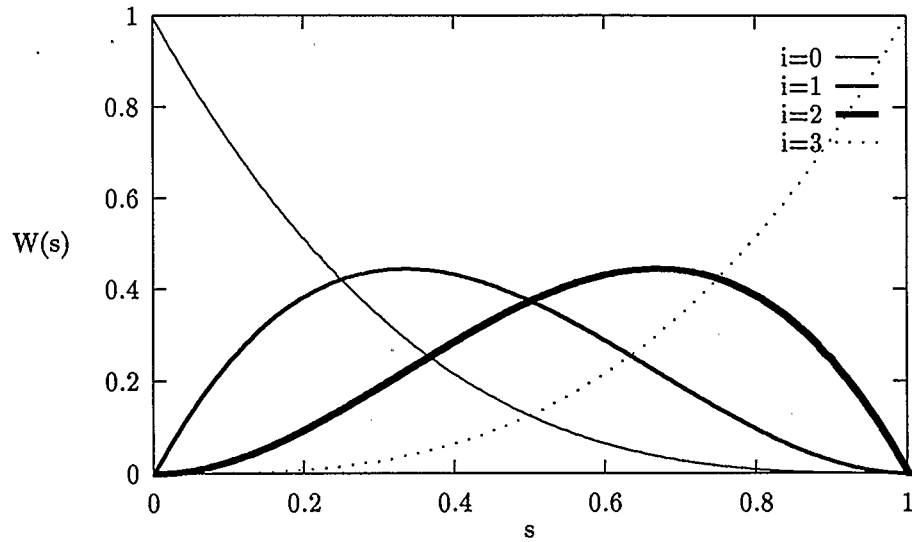
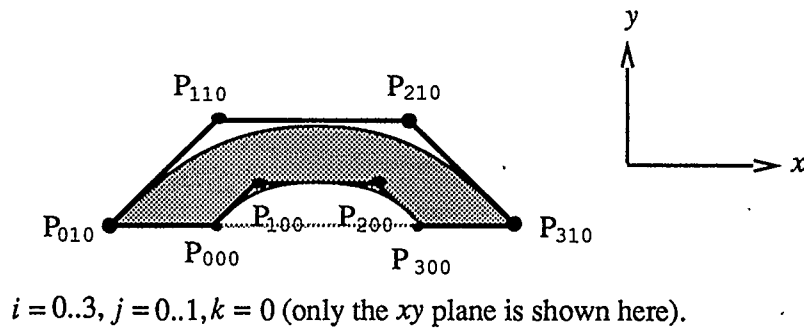


Figure 3.12: Binomial weighting functions for $W_i^3(s)$

three weights, one for each axis.

The deformed object space coordinates of a surface point are based on weighted sums of *all* grid vertices, which indicates that deformations of the grid are global with respect to the grid volume. That is, moving a vertex, even if it is far from the surface point and within a different section of the grid, will affect the surface point. Although the movement of a vertex far from the surface point may only alter the point slightly, it is impossible to deform the volume within just one section of the grid—a separate grid must be used for this.

It is important to note that the grid itself does not define the volume directly, but rather defines the control points which describe the Bezier volume. Figure 3.13 illustrates this relationship in two dimensions: the shaded region represents the area affected by the grid vertices.



The shaded region represents the grid space specified by the grid control points P_{ijk} .

Figure 3.13: Example grid and the area it defines

3.5 Extended free form deformation

Free form deformation is a versatile modelling tool. However, forcing the initial grid to be a parallelepiped restricts the deformations that can be created. Coquillart [Coq90] describes an extended free form deformation that permits initial grids to be arbitrarily shaped. Allowing arbitrary initial grids implies that the initial mapping from object space to grid space must permit nonlinear calculations. Coquillart uses the multidimensional Newton-Raphson technique to calculate the relative grid coordinates (s, t, u) of a surface point. Unfortunately, this does not guarantee a solution. A poor initial estimate may cause nonconvergence, indicating, but not proving, that the root does not exist [PFTV88]. Although Coquillart states that she did not encounter this problem, the results discussed in Chapter 4 indicate that some limitations exist.

Coquillart also introduces a slightly different formulation of the grid volume through the Bernstein polynomial. Her definition separates the grid into “chunks”

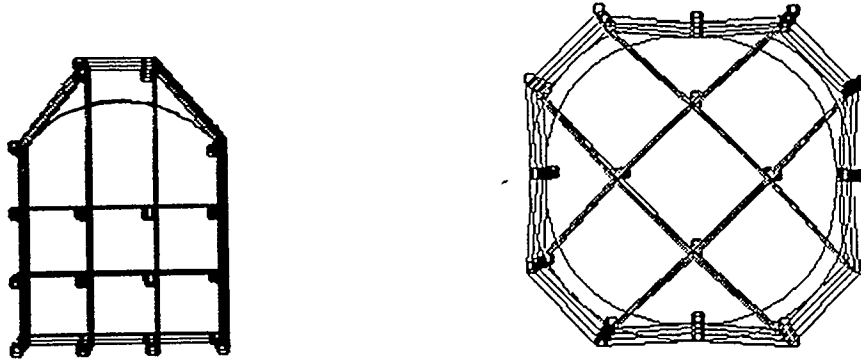


Figure 3.14: Example grids used by extended free form deformation

which are composed of 27 sections (the grid is a $3 \times 3 \times 3$ volume of sections). Each chunk is defined by a separate Bernstein polynomial of degree three, and is therefore independent of all other chunks. This allows deformations to be performed locally, because deformations to grid vertices *inside* a chunk will not affect other chunks. Figure 3.14 shows some example grid chunks.

To maintain separate chunks within a grid, one must determine which chunk a point lies in before using the Newton-Raphson algorithm to calculate its relative grid coordinates. Coquillart uses the convex hull property of Bernstein-Bezier volumes to determine this. Although this provides a reasonably accurate indication of point inclusion, it does not guarantee it. Figure 3.15 shows the problem: the point lies in the convex hulls of both chunks, but is only in the volume defined by the upper one. Similarly, the point may be in the convex hull of a chunk, but not in the grid volume at all. As a result, if the Newton-Raphson method does not converge for one chunk, the convex hull test must be used for neighboring chunks in the grid until all have been tested, or convergence is achieved.

As will be seen in Chapter 4, nonconvergence can occur even if the point lies in

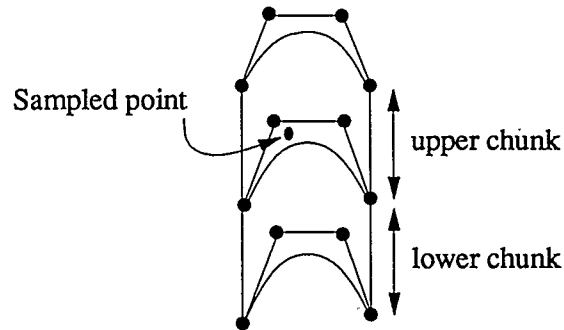


Figure 3.15: Sampled point lying in convex hull of two grids

the chunk being tested. This complicates the process. Nonconvergence implies one of three conditions: the point is in the current chunk, but a better initial estimate is required; the point is in a neighboring chunk; or the point is outside the Bezier volume.

3.6 Summary

This chapter presented the mathematics of superquadric surfaces and three deformation techniques: Barr operators, free form deformation, and extended free form deformation. A new operator for shearing surfaces was defined. The next chapter provides design and implementation details of a modelling system based on deforming superquadrics with these operations.

Chapter 4

Design of the GROOK system

This chapter describes GROOK, a system that implements the three deformation techniques outlined in Chapter 3, namely Barr operators, free form deformation, and extended free form deformation. The goal of the system is to provide a testbed for deforming superquadric primitives with Barr operators and free form deformation and its extension. The system is ultimately intended to be fully interactive; however, due to time constraints, not all features have been implemented.¹ Free form deformation and its extension are fully implemented to allow interactive manipulation since interactive deformation of surfaces is the key to the technique. Barr operators have been designed for interactive input, but in its present state the system requires parametric input to be hard coded.

4.1 GROOK

GROOK has four windows, as shown in Figure 4.1: three orthogonal views of the scene (one along each coordinate axis) and one general three-dimensional viewing window. The general viewing window allows the user to rotate the scene using a mouse. All windows can be zoomed and panned, and by clicking on a window bar they grow to fill the whole window space allocated to GROOK. Clicking the window bar again brings back the four viewing windows.

¹The skeleton for such a system exists, with menus defined but not all options available.

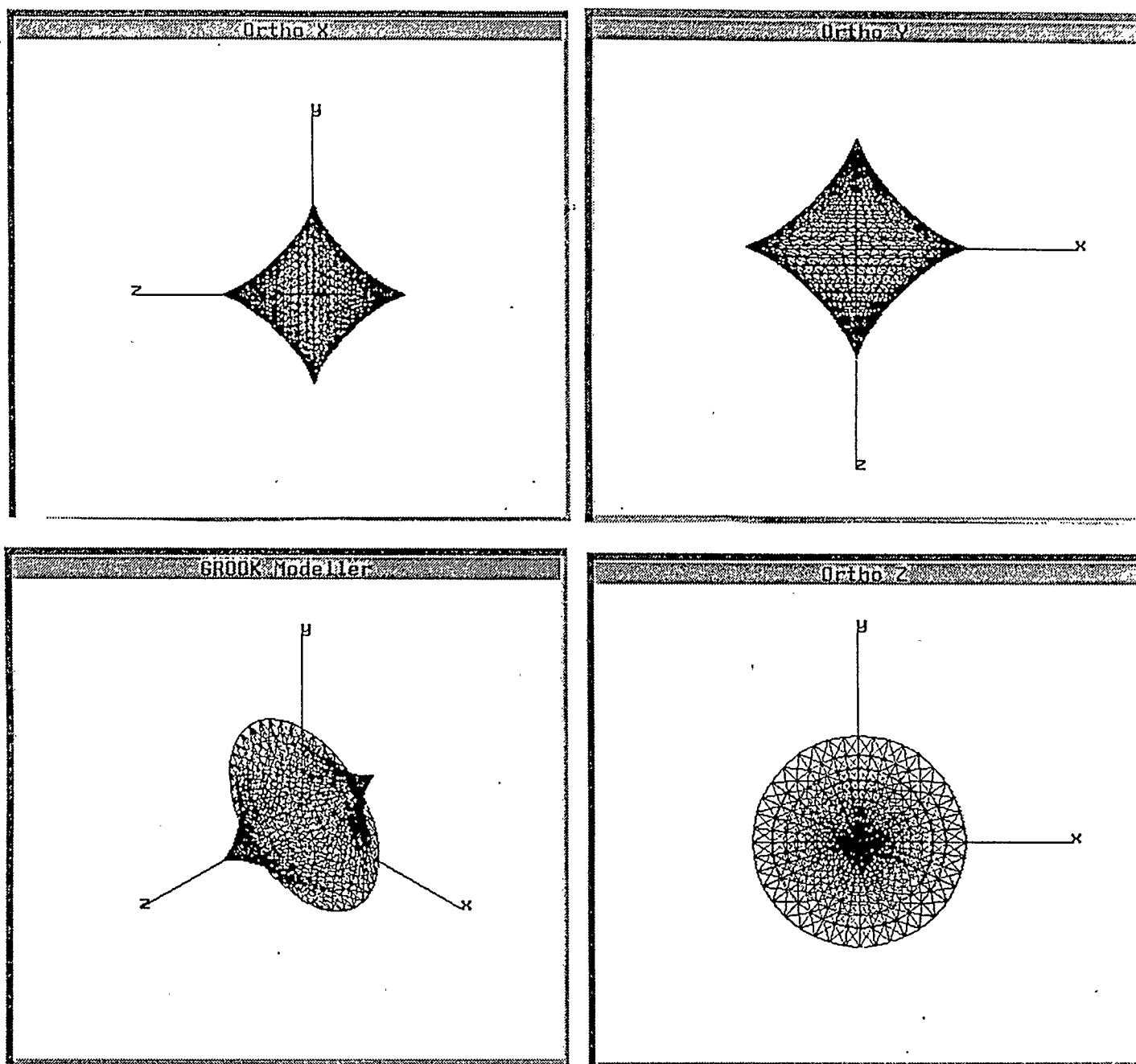


Figure 4.1: The windows in GROOK

There are four menus with which the user can create deformed objects, one to choose a superquadric and three to deform the chosen primitive, one menu for each deformation technique. To deform a superquadric, the user:

- chooses a superquadric primitive (which displays a wire frame image);
- specifies the parameters for the deformation via the deformation menus;
- returns to the superquadric menu and redisplay the primitive with deformation mode on.

GROOK does not allow more than one primitive to be deformed at a time, but does permit multiple deformations of the same primitive. Although only one deformation using the free form technique is permitted, a sequence of Barr operators can be applied, before or after the free form deformation is applied.

GROOK links to the Graphicsland research environment by writing polygonised versions of the object to a file that is read by the polygon modeller. This can read multiple files and group them hierarchically to create a scene. Various pictures in this chapter have combined surfaces in this way.

The code for GROOK is written in C, and runs on a Personal Iris workstation. There are approximately 8000 lines of code, two thirds of which deals with deforming and rendering primitives, while the remaining third constitutes window and menu handling routines. The windowing and menu routines are based on Joy [McP90], a local graphics package that extends the Iris windowing tools.

4.2 Superquadrics

The menu for superquadrics allows one of five primitives to be chosen: the superellipsoid, superhyperboloids of one and two sheets, the supertorus, and a plane. The plane is a degenerate superquadric, but was deemed useful for testing deformations on simple surfaces.

Superquadric exponent values can be increased or decreased interactively. Two types of polygonisation are implemented (see Section 4.2.2), and the user may select either one by toggling a button. The sampling rate can be altered to allow a fine or coarse level of polygonisation, and the result can be saved to a file for later display within Graphicsland.

4.2.1 Definition for superquadrics

The definition provided by Barr [Bar81] does not accurately characterise superquadrics as described in Chapter 3. When the exponents $E1$ and $E2$ are set to values less than one, they require n th roots of trigonometric functions to be taken. Since some of the trigonometric functions evaluate to negative numbers, run-time errors occur. The sign of the trigonometric function places the surface point in the correct quadrant, and the exponentiation calculates the correct magnitude. The solution is to store the trigonometric function's sign, apply the exponent to its absolute value, and multiply the magnitude by the stored sign.

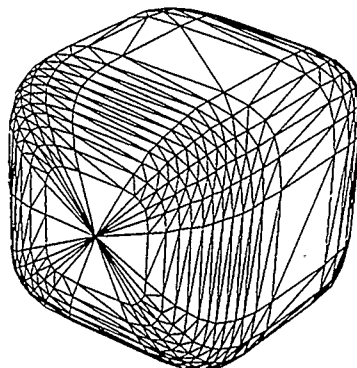


Figure 4.2: Superellipsoid $E1 = E2 = 0.4$

4.2.2 Polygonising superquadrics

Various schemes to polygonise superquadrics are discussed in [FB81]. The most basic method uses the parametric representation of superquadrics introduced in Chapter 3, where η and ω are used to trace the surface. Each superquadric is defined with two two-dimensional curves, one based on η , the other on ω . By sampling each curve and taking the spherical product of the two values, the three-dimensional surface point is calculated. If for every point sampled on the η curve, the entire ω curve is sampled, the whole surface is covered. GROOK creates triangular polygons as it increments along the ω curve.

The ease with which the surface is traced makes parametric representations suitable for display. In the case of superquadrics, the natural spacing that results is also desirable as more points are plotted where the curvature is high (Figure 4.2). Barr [Bar81] shows how a surface can be adaptively subdivided, but his technique depends on the particular superquadric being rendered, and requires complex cal-

culations involving first and second derivatives. Since the comparatively fast parametric sampling is easy to implement and approximates the curvature sampling to some degree, it is suitable for most applications. Increments used for η and ω as they trace the surface should evenly divide the regions between cusps so that the cusps are polygonised accurately. The superellipsoid, for example, may have cusps at $-\pi/2, 0$ and $\pi/2$ for η , and $-\pi, -\pi/2, 0$, and $\pi/2$ for ω .

Although the parametric method works well for undeformed superquadrics, when primitives are deformed using Barr operators or free form deformation a more evenly spaced sampling technique is required. Since deformation may be applied to relatively flat areas of a superquadric, the parametric polygonisation method does not provide enough sampling for the deformed surface. To correct this, the sampling rate has to be increased so much that the rest of the superquadric is oversampled. Figure 4.4 shows a superellipsoid that has been scaled and bent to represent a chair seat. The versions in Figure 4.4 (a) and (b) were sampled using the parametric method, the latter with a higher sampling rate. As can be seen, the bend in the seat needs more sampling to achieve a smooth curve, but even with the higher rate the deformed area is poorly approximated.

The explicit polygonisation discussed in [FB81] is more suitable for deformed surfaces since it samples the surface evenly. The algorithm resembles the parametric one, but uses explicit equations based on (x, y, z) instead of the trigonometric functions based on (η, ω) . The two forms are equivalent, but the explicit representation gives a sampling that is uniform in object space instead of parametric space. Table 4.1 gives equations that are equivalent to the parametric curves of Section 3.1.1.

There are two problems that affect the implementation of the algorithm: the

Curve	Explicit Function
superellipse	$x = (1 - z ^{\frac{2}{E1}})^{\frac{E1}{2}}$
superhyperbola	$x = (1 + z ^{\frac{2}{E1}})^{\frac{E1}{2}}$
curve for torus	$x = (1 - z + a ^{\frac{2}{E1}})^{\frac{E1}{2}}$

Table 4.1: Explicit equations used for polygonising superquadrics

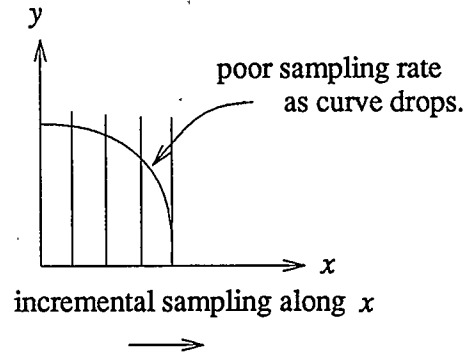


Figure 4.3: Sampling with explicit equations

explicit equations are multivalued, and sampling along an axis does not provide uniform sampling along a curve. Consider, for instance, a superellipsoid. The first problem is solved by calculating the positive quadrant and reflecting it seven times to complete the surface. The second problem is illustrated in Figure 4.3, where a quarter circle is sampled along the x axis: as the curve drops, the distance between sampled points on the curve increases. To overcome this problem the surface is sampled to the midpoint of the curve and then reflected. Hyperbolas (which are used for hyperboloids) are not affected by this sampling problem as much as ellipses, and are therefore sampled over the entire positive quadrant and then reflected to complete the curve.

The seat in Figure 4.4(c) was polygonised with the explicit method. Besides

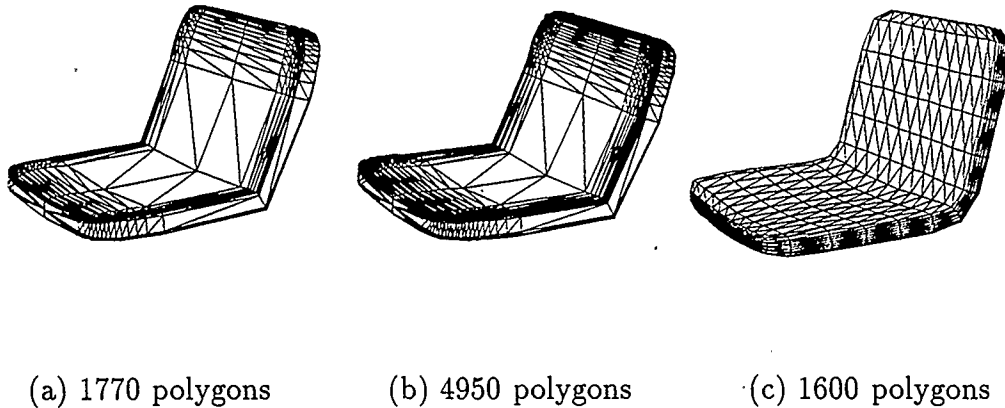


Figure 4.4: Chair seat

providing a better sampling of the deformed area, this has fewer polygons than the seats in Figure 4.4 (a) and (b).

4.3 Barr operators

Barr operators deform points in space by applying piecewise differentiable functions to them. To allow greatest flexibility in shape control, the user should specify the deformation function (F), and also has to provide:

- the range over which F is applied;
- the Jacobian matrix of F ;
- the inverse Jacobian of F .

Since the function is not restricted (except that it be C^0 continuous and piecewise differentiable), automatic calculation of the Jacobian requires advanced techniques

from symbolic algebra. This method of specification clearly requires users to have a good understanding of mathematics, and is prone to input errors.

To overcome these difficulties, the range of deformations can be restricted in return for easier control of input. GROOK provides specific functions that allow the user to control shape by specifying parameters. This technique is well suited to interactive input, and demands little mathematical knowledge of the user. A modelling system may provide both methods by offering predefined functions for interactive use and also allowing user-defined functions to be compiled into the code.

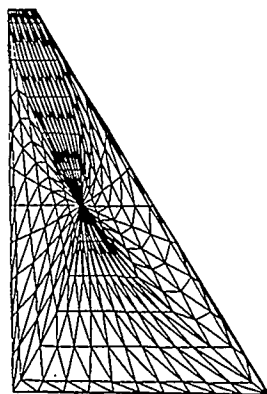
The following sections outline the operators offered by GROOK. Three are based on Barr's work and one is a new operator that extends the linear shear function. Although GROOK requires hard coded input, the design for interactive specification of input is discussed.

4.3.1 Taper

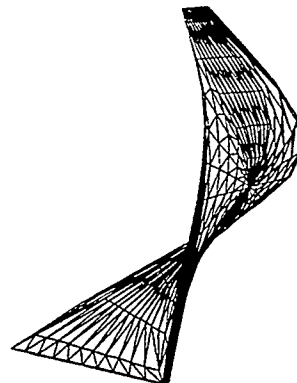
GROOK implements the linear taper function

$$F(x, y, z) = \begin{cases} xf(z) \\ yf(z) \\ z \end{cases} \quad \text{where } f(z) = mz + b$$

described in Section 3.3.3. The taper is specified by the four parameters described in Figure 4.5(c). Given these, m and b of $f(z)$ need to be determined. The slope of the taper, m , is easily calculated from the scale values at the beginning and end of the range, *init_scale* and *final_scale*. To allow tapers to be applied off the origin, b must be calculated by substituting *init_scale*, *zstart*, and m into $f(z)$. The four parameters allow the user to create a function intuitively by giving general dimensions



4.5(a) Taper



4.5(b) Twist

Parameters	Name	Example Values
Range over which taper function has effect	zstart	-0.1087
	zstop	-4.3513
Scale factor at the beginning of the range	init_scale	0.2
Scale factor at the end of the range	final_scale	2.0

4.5(c) Parametric input for linear taper: washer blade

Parameters	Name	Example Values
Range over which twist function has effect	zstart	-0.1087
	zstop	-4.3513
The number of twists per range	total_twists	0.5

4.5(d) Parametric input for twist: washer blade

Figure 4.5: Deformations for washer blade

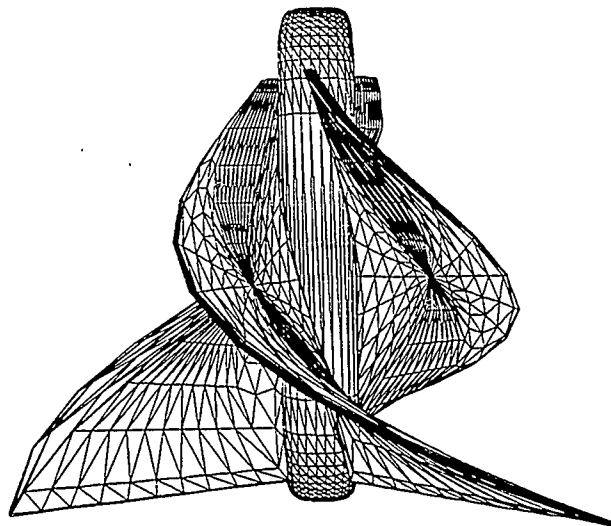


Figure 4.6: Washing machine agitator

relating to the object being tapered. The washing machine agitator blade shown in Figure 4.6 uses this operator to control the amount of tapering; the twist is described below. Figures 4.5(a) and (b) show the blade before and after the tapering.

It is necessary to check that the taper function $f(z)$ is always greater than zero to avoid singularities or everted² objects. A simple check that $f(z_{start})$ and $f(z_{stop})$ are greater than zero ensures positive values throughout the range. The taper function must also be applied off the origin, implying that the user-defined range must not include zero.

²to evert means to turn inside out

4.3.2 Twist

The twist deformation function is:

$$F(x, y, z) = \begin{cases} x \cos(\theta) + y \sin(\theta) \\ x \sin(\theta) - y \cos(\theta) \\ z \end{cases} \quad \text{where } \theta = f(z)$$

GROOK implements both a constant and a nonconstant twist function.

Constant twist

The function $f(z)$ controls the angle of rotation, while its derivative specifies the rate of twisting. For a constant twist the derivative must be constant:

$$\begin{aligned} f(z) &= \text{RadiansPerUnit} \times z + b \\ \frac{\partial f}{\partial z} &= \text{RadiansPerUnit} \end{aligned}$$

Specifying the the number of twists per unit z is desirable when the surface is tightly twisted; however, the user may prefer to specify the total number of twists over a specified range. This can be converted to radians per unit z :

$$\text{RadiansPerUnit} = \frac{\text{total_twists}}{(z_{\text{stop}} - z_{\text{start}})} 2\pi$$

The twist may be applied to a section of the surface, but continuity at its beginning and end must be ensured. This can be achieved by forcing the rotation angle at the beginning of the twist to be zero, and calculating the constant b accordingly:

$$\begin{aligned} f(z_{\text{start}}) &= 0 \\ 0 &= \text{RadiansPerUnit} \times z_{\text{start}} + b \\ b &= -(\text{RadiansPerUnit} \times z_{\text{start}}) \end{aligned}$$

Continuity at the end of the twist can be ensured if the total number of twists is an integer. However, it does not have to be an integer for the cases where the twist is applied to the end of the object. The washing machine agitator blades were twisted by specifying the number of twists over a range; Table 4.5(d) lists the parametric values used.

Nonconstant twist

Since the derivative of $f(z)$ defines the rate of twisting in radians per unit, a nonconstant derivative will give a nonconstant twist. The most basic form is a linearly increasing or decreasing function:

$$\text{Let } \frac{\partial f}{\partial z} = az + b$$

$$\text{Then } f(z) = \frac{a}{2}z^2 + bz + c$$

The user specifies the amount of twisting per unit z at the beginning and end of the desired range. This information is used to calculate a :

$$a = \frac{final_twists - init_twists}{zstop - zstart}$$

As before, b is used to permit the rate of twisting to be specified at locations other than $z = 0$:

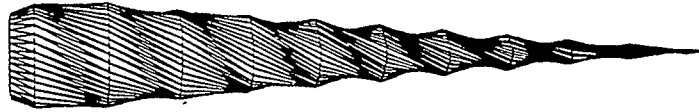
$$\frac{\partial f}{\partial z}(zstart) = a(zstart) + b$$

$$b = init_twists(2\pi) - a(zstart)$$

The fact that, as before, $f(zstart)$ must equal zero, is used to calculate c :

$$c = -\left(\frac{a}{2}(zstart)^2 - b(zstart)\right).$$

Figure 4.7 illustrates a nonconstant twist as applied to a tapered superellipsoid:



(a) The resulting picture

Parameters for Nonconstant Twist	Name	Example Values
Range over which the twist is applied	zstart	4.0
	zstop	8.0
Number of twists per unit length at zstart	init_twists	0.25
Number of twists per unit length at zstop	final_twists	1.0

(b) The parameters

Figure 4.7: Nonconstant twist applied to a tapered superellipsoid

4.3.3 Bend

The bend operator is specified by two functions, one for the bent region:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x \\ -\sin(\theta) \\ \cos(\theta) \end{pmatrix} \left(z - \frac{1}{k} \right) + \begin{pmatrix} 1 \\ y_0 \\ \frac{1}{k} \end{pmatrix} \quad (4.1)$$

and the other for the unbent region:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x & 0 & 0 \\ 0 & -\sin(\theta) & \cos(\theta) \\ 0 & \cos(\theta) & \sin(\theta) \end{pmatrix} \begin{pmatrix} 1 \\ z - \frac{1}{k} \\ y_b \end{pmatrix} + \begin{pmatrix} 0 \\ y_0 \\ \frac{1}{k} \end{pmatrix} \quad (4.2)$$

Where:

- $(y_b = y - y_{min})$ if $y < y_{min}$
- $(y_b = y - y_{max})$ if $y > y_{max}$.

These are for bends along the y axis in the yz plane. The parameters required are y_0, y_{min}, y_{max} , and k , as shown in Figure 3.7. To implement the bend operator, an intuitive manner for specifying the parameters is needed. Since the centerline is the y axis, the surface being bent should be aligned accordingly. Most of the parameters are then easily understood, except perhaps the bending rate k . In case the user does not wish to think in terms of radians per unit, the same information could be specified interactively by picking a point around which the line is to be bent. If this point is (y_c, z_c) , then $y_c = y_0$ and $z_c = \frac{1}{k}$.

Several different bend types can be achieved by varying the parameters.

Error Condition	Resulting Error
$ymin \geq ymax$	incorrect results
$k = 0$	divide by zero
k is positive, $z > k$	crimp in bend
k is negative, $z < k$	crimp in bend
y_0 not in $ymin..ymax$	unexpected rotations

Table 4.2: List of error checks for bend operator

- If y_0 is not halfway between $ymin$ and $ymax$, the bend is asymmetric. The torus used for the legs of the chair model in Figure 4.8 illustrates this feature.
- To bend toward the negative axis, k is negative.
- The y axis is the centerline of the bend, and is the only line to remain the same length after bending.

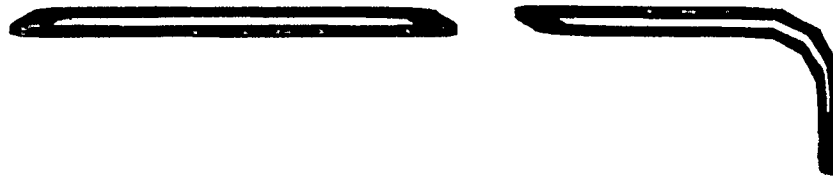
Many checks must be made for the bend operator to work reliably, as shown in Table 4.2. The chair depicted in Figure 4.8(d) uses four bend operations: two for the seat, and two for the legs. The parameters are listed in Figures 4.8(e) and (f).

4.3.4 Nonlinear shear

For the shear operator discussed in Chapter 3, the user must specify the shear function:

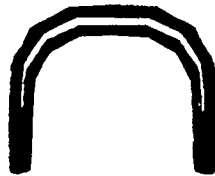
$$F(x, y, z) = \begin{cases} x + f(z) \\ y \\ z \end{cases}$$

GROOK allows the user to shear surfaces with superconics, the two-dimensional versions of superquadrics. The explicit representation of the curves is used instead of

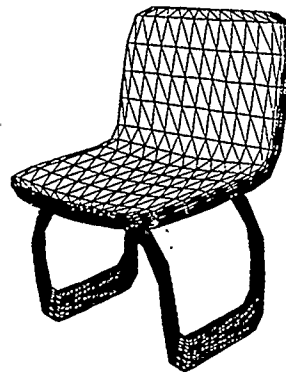


(a) Torus

(b) First bend



(c) Second bend



(d) Chair

Parameters	First bend	Second bend
y_{min}	0.5	-2.5
y_{max}	2.5	-0.5
y_0	0.5	-0.5
k	$-\frac{\pi}{4.0}$	$-\frac{\pi}{4.0}$

(e) Parametric input for bending chair legs

Parameters	Main bend	Minor bend
y_{min}	-0.2	-1.0
y_{max}	0.3	1.0
y_0	-0.2	0.0
k	$\frac{\pi}{1.2}$	$\frac{\pi}{16.0}$

(f) Parametric input for bending chair seat

Figure 4.8: Chair model

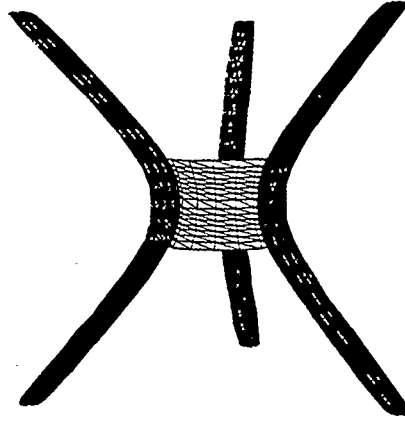


Figure 4.9: Ellipsoids sheared with superhyperbola

the parametric representation since the shear function must be expressed in terms of x , y , or z . GROOK implements the superellipse shear function:

$$f(z) = (1 - |z|^{\frac{2}{E1}})^{\frac{E1}{2}}$$

and the superhyperbola shear function:

$$f(z) = (1 + |z|^{\frac{2}{E1}})^{\frac{E1}{2}}$$

The legs of the table in Figure 4.9 were created by applying a superhyperbola shear to scaled superellipsoids. A single leg was created in GROOK and PG was used to make three instances of it.

4.4 Free form deformation

GROOK's implementation of free form deformation permits interactive manipulation. The following sections describe the process used to deform a superquadric, the

data structures required to store the information, the techniques used by GROOK to handle normal and tangent calculations, and continuity control between deformed and undeformed regions.

4.4.1 Interactive specifications

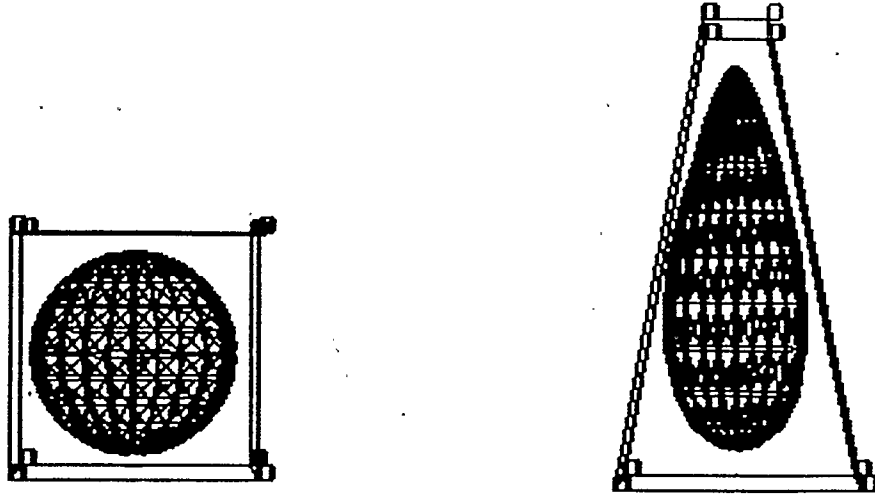
Free form deformation lends itself to interactive manipulation by allowing the user to place the initial parallelepiped over the desired volume to be deformed, and then drag the grid's vertices to new locations in space to deform the objects. The sequence of steps in a typical deformation is:

- the undeformed grid is specified (S, T, U to determine the grid length along each axis, and X_0 to determine its position in object space);
- the user adjusts the grid position by dragging it to the correct location;
- the grid is deformed by its vertices being picked and dragged to new locations;
- the surface is redrawn with the deformation applied to any points falling within the undeformed grid.

Figure 4.10 shows a superellipsoid before and after deformation.

The undeformed grid can be chosen in two ways: the user may either pick a default grid or specify S, T, U and X_0 interactively. The grid may be divided into sections along each axis by changing the values of l , m , and n of equation 3.5.

Direct manipulation of the grid vertices is used to deform the volume, and can be executed in any of the three orthogonal view windows. GROOK allows groups of



(a) undeformed superellipsoid and grid (b) deformed superellipsoid and grid

Figure 4.10: Free form deformation of superellipsoid

vertices to be chosen: vertices that lie “behind” one another can be moved simultaneously. For example, P_{000} , P_{100} , and P_{200} can be grouped in the window with a view down the x axis. Although this is the only type of grouping GROOK allows, other methods might prove useful, such as letting the user pick individual vertices to be grouped, or providing sets of groups (such as all vertices with $i=0$ and $j=2$).

The current implementation of the modeller applies the deformation only if the “deformation” button is turned on, and only when the surface is redisplayed. This was incorporated mostly for extended free form deformation, which is more computation-intensive than the original deformation technique.

The positions of the undeformed and deformed grid vertices need to be available when the surface is sampled. The former are used to map the surface point to grid space, and the latter are used in the Bernstein polynomial to map the deformed point back to object space. The position of the undeformed grid vertices can be calculated using S , T , U and X_0 along with l , m , and n as shown in equation 3.4. Since there is

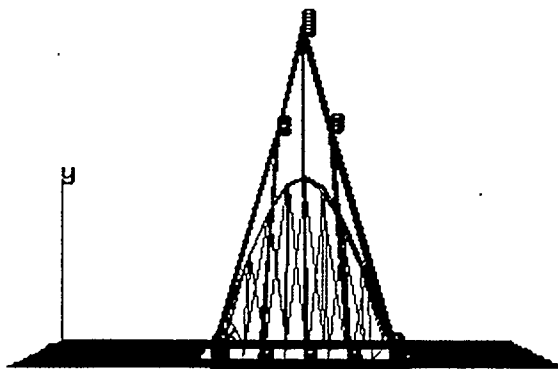


Figure 4.11: Grid volume does not closely fit grid

no such relationship between the deformed vertices, they must be stored individually. GROOK uses a three-dimensional array since the grid vertices are ordered, and their ordering relates directly to the array indexing.

Two further points are worth mentioning. First, since the mapping used is continuous, it does not make sense to cross sections of the grid with one another, for that would yield an impossible surface. Second, the grid volume is the Bezier volume defined by the control vertices of the grid, not that enclosed by the grid vertices themselves, and this sometimes causes unexpected results. Figure 4.11 shows a situation in which the user wishes to pull the surface to a point that is close to the deformed grid points, but since the Bezier curve described by the top five vertices is a shallow one, the effect is not as marked as desired. A familiarity with Bezier curves would, of course, benefit the user here. Using a grid that is larger than the object embedded in it accentuates this effect, and therefore grids are best chosen to fit the undeformed surface as closely as possible.

4.4.2 Normals and tangents

Tangents of surfaces deformed with free form deformation can be calculated using Barr's tangent transformation rule:

$$\begin{pmatrix} \frac{\partial B_x}{\partial \eta} \\ \frac{\partial B_y}{\partial \eta} \\ \frac{\partial B_z}{\partial \eta} \end{pmatrix} = \begin{pmatrix} \frac{\partial B_x}{\partial x} & \frac{\partial B_x}{\partial y} & \frac{\partial B_x}{\partial z} \\ \frac{\partial B_y}{\partial x} & \frac{\partial B_y}{\partial y} & \frac{\partial B_y}{\partial z} \\ \frac{\partial B_z}{\partial x} & \frac{\partial B_z}{\partial y} & \frac{\partial B_z}{\partial z} \end{pmatrix} \begin{pmatrix} \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \eta} \\ \frac{\partial z}{\partial \eta} \end{pmatrix}$$

The Bernstein polynomial (equation 3.5) needs to be defined in terms of x , y , and z instead of s , t , and u . This can be done by substituting s , t , and u with the linear equivalents in x , y , and z :

$$s = \left(\frac{pt.x - X_0}{S.x} \right) \quad t = \left(\frac{pt.x - X_0}{T.y} \right) \quad u = \left(\frac{pt.x - X_0}{U.z} \right) \quad (4.3)$$

The inverse Jacobian of the Bernstein polynomial is quite complex and has not been calculated. Instead, the normal at a point on the surface may be derived by taking the cross-product of the deformed tangents based on η and ω . The only problem associated with this step occurs if the two tangents have been mapped so that they are not linearly independent, which implies that the mapping is degenerate. This can be detected by checking the determinant of the Jacobian matrix: if it is zero the deformation is degenerate and cannot be performed.

Continuity

Continuity may be important in two situations: when two or more grids are joined together, and when a grid is placed over just part of a surface. Since GROOK only offers one grid, the first case does not arise (it has been investigated in [Par86]). The second case requires continuity to be maintained at the intersection of the grid and

the surface. By not altering the first plane in the grid that intersects the surface, C^0 continuity is maintained. C^k continuity can be achieved by not deforming the first k planes that intersect the surface [SP86]. This provides an easy way to control continuity interactively. GROOK leaves this control to the user; however, a fully developed system might automatically freeze k outer planes in the grid when C^k continuity is requested by the user.

4.5 Extended free form deformation

GROOK allows extended free form deformations to be specified interactively by menu selection. The following sections describe the differences between the free form deformation implementation and its extension, and discuss problems encountered with the extended deformation technique.

4.5.1 Freezing the grid

Extended free form deformation is more complex to implement than the original version, since the vertex positions must be stored for both deformed and undeformed grids. In the original (unextended) technique, the initial mapping to grid space coordinates uses the vectors S , T , U and X_0 (equation 3.3), and therefore the undeformed grid vertex positions need not be stored. The initial grid vertices for the extended version cannot be similarly related since the grid is arbitrarily shaped; hence the vertices must be stored in a second three-dimensional array.

The process for deforming a surface also changes. Rather than simply dragging the grid to the desired location and then deforming it, the user must now drag the

grid to the desired location, freeze it to store the vertex positions, and then deform it and apply the deformation to the surface. The grid, once frozen, cannot be dragged to a new location unless it is refrozen before deformation.

The process of deforming a surface using extended free form deformation in GROOK is:

- the undeformed grid is chosen (GROOK provides several predefined grids);
- the user adjusts the grid position by dragging it to the correct location, or adjusts any vertex by dragging it to a new location;
- when the grid is shaped and positioned as desired, the grid is frozen;
- the grid is deformed by directly manipulating its vertices, as in free form deformation;
- the surface is redrawn with the deformation applied to any points falling within the undeformed grid.

Each stage is controlled by buttons at the side of the interface. Modes are set so that manipulating the grid alters the initial array if the grid has not been frozen, and alters the deformed grid array if it has been frozen. Deformation of the grid is the same as for unextended free form deformation in that single points or groups of points may be repositioned.

4.5.2 Grid formulation

The extended free form deformation grid is equivalent to an (unextended) free form deformation grid where $l, m, n = 3$, and the grid may be nonparallelepiped. The

restriction in the number of sections simplifies the calculations used by Newton's method.

4.5.3 Normals and tangents

Tangents for a surface deformed with extended free form deformation cannot be calculated using Barr's tangent transformation rule since there is no analytic mapping between object and grid space. This means that the Bernstein polynomial cannot be expressed in terms of x , y , and z as required by the Jacobian. As a result, GROOK does not provide tangents and normals for superquadrics deformed with this technique. Approximation methods such as those outlined in [Blo88] should be further investigated.

4.5.4 Problems with Newton's iteration

The extensions added to free form deformation by Coquillart greatly expand the set of shapes attainable with the technique. However, the mapping from (x, y, z) coordinates to (s, t, u) coordinates is now nonlinear, and this not only requires much more computation than the linear mapping, but also does not guarantee a correct result.

GROOK uses multidimensional Newton's method to calculate the grid coordinates of the undeformed surface point. Although Coquillart, using the same technique, claimed that convergence problems did not arise, such problems were encountered while testing various grids with GROOK. Newton's method requires an initial guess to start the iteration process, and Coquillart used a starting value of 0.5 for s , t , and u . The grids shown in Figure 4.12 did not converge with these starting values,

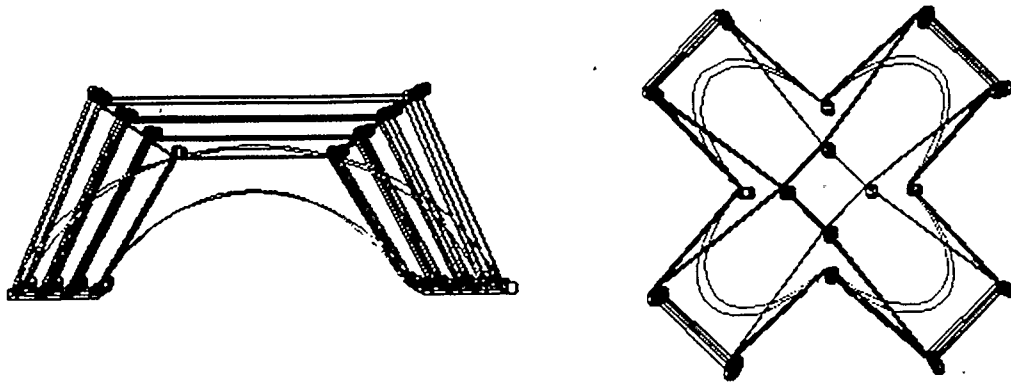


Figure 4.12: Grids for which Newton's iteration failed

although with better initial estimates the convergence problems disappeared. Good initial estimates are the key to effective use of multidimensional Newton's method.

Since the whole point of extended free form deformation is to allow arbitrarily shaped initial grids, it would defeat the purpose to restrict the grids to simple ones which were known to work with Newton's method. GROOK only permits one grid and relies on it to provide adequate complexity in shape, whereas a system that allowed multiple grids could afford to restrict each one to a simple shape. Techniques for determining good starting values should be investigated further, even if simple grids are used, since these will speed up the iteration process.

4.6 Combining deformation techniques

In contrast to free form deformation, Barr operators seem to be rather restricted, and it is worth asking whether they are still necessary, or whether free form deformation

might not offer enough shape control to render them redundant. Although Barr operators are efficient, the free form techniques provide a better balance between control of shape and ease of use. If they can perform the same operations as the Barr functions, then perhaps a system based solely on them could offer the user a more consistent interface with the same expressiveness.

Scaling, tapering, and bending can be accomplished with the free form techniques, but twisting is difficult. Applying Barr's twist function to the grid vertices overcomes the challenge of interactively twisting them, but because of the Bezier based definition of the volume, the twisted grid is much narrower along the twist direction than the original. Instead, the twist function can be applied to the grid space itself. In fact, both twist and taper functions can be applied to the grid volume. This has several benefits:

- the space is not narrowed as it is when the grid vertices are twisted;
- it provides the user with a consistent interface for deformations — the grid volume is always used to encompass the region to be deformed;
- twists and tapers can be applied along curves: since the functions are applied in (s, t, u) space, they will follow in the direction of the s , t , or u axes, which will be curved relative to object space if the grid is curved.

The last point describes a new type of shape control, since any function can be applied to the grid space as long as the deformed points remain in the grid volume. The procedure used with GROOK is outlined below, assuming a twist function is being applied to a superquadric:

- calculate the (s, t, u) coordinate of the sampled superquadric surface point;
- since the grid volume is defined from 0..1 along each axis, the surface point must be translated by -0.5 so that the twist is centered around zero;
- apply the twist function to the point;
- use the deformed (s, t, u) coordinate in the Bernstein polynomial to calculate the deformed position in object space.

The surface point in grid space must remain within the grid volume after deformation, so that its position in object space can be calculated by the Bernstein polynomial in the final step. This implies that the grid volume must be wide enough to contain not only the original surface, but also the deformed one. GROOK does not check for this, but any deformed (s, t, u) point outside of the range $(-0.5..0.5)$ could be flagged as an error, signalling the user to resize the grid.

Since the grid defines the region to be deformed, the user no longer needs to specify the range over which the twist should be applied. The functions are all similar to those defined in Section 4.3.2, except that specifying the number of twists per unit cannot be calculated since the twist is not in object space, and the relationship between grid space and object space is not easily obtained.

Figures 4.14 to 4.17 illustrate the effect of this new operation. The primitive is a superellipsoid which is variably sheared along the x axis. The initial grid used for all four deformations is cylindrical, and is aligned with the curved surface by applying the same varying shear function to its vertices. Interactive manipulation of some of the grid points then permits final adjustments to ensure that the grid volume closely

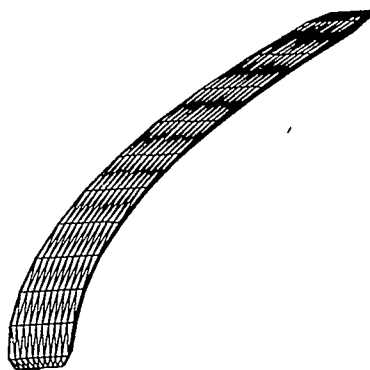


Figure 4.13: Curved surface: sheared by $f(y) = 0.3y^2$

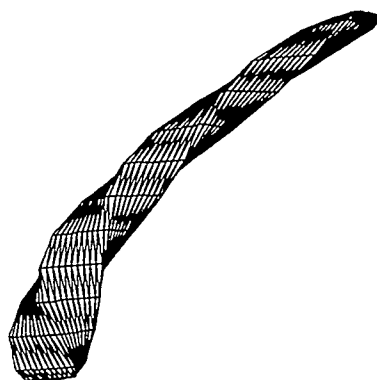


Figure 4.14: Twist along curved surface

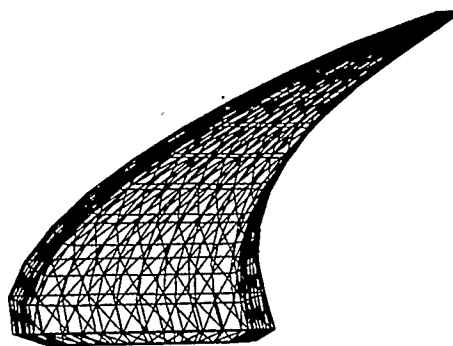


Figure 4.15: Taper along curved surface

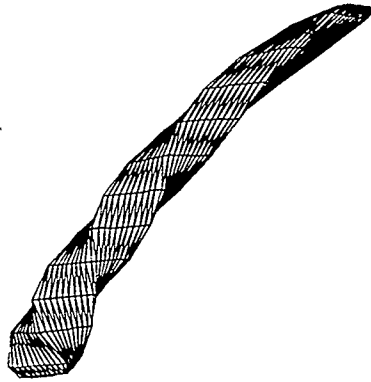


Figure 4.16: Nonconstant twist along curved surface

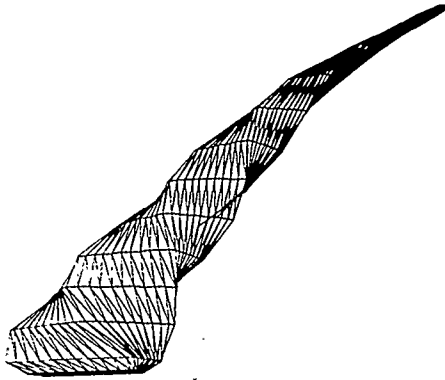


Figure 4.17: Taper and twist along curved surface



Figure 4.18: Twist using free form deformation

follows the curve of the surface. The superellipsoid shown in Figure 4.18 illustrates that twisting can be applied locally as well as globally with this method.

4.7 Summary

This chapter described GROOK, a system that implements three deformation techniques. Two polygonisation algorithms for its superquadric primitives were described, and each of the deformation techniques was discussed. A new technique for combining Barr operators and free form deformation methods was presented.

Chapter 5

Evaluation

This chapter shows how Barr operators, free form deformation, and extended free form deformation can be used to replicate, and extend, most of the operations listed in Chapter 2. When possible, GROOK was used to replicate the operations. Occasionally, GROOK did not offer an operation, although there was no inherent reason why Barr operators or free form deformations could not have been used. In these cases the details of such an implementation are discussed.

5.1 A metric for shape analysis

In order to analyse the shape control offered by GROOK a metric is required. It is not possible to design a metric and justify it on purely logical grounds [Koe90]. Instead, we illustrate the strength of deformation based modelling by showing that GROOK can provide the shape control offered by existing systems. However, a basis for evaluating the extent to which each operation provides flexibility is needed. Coquillart [Coq90] outlines four areas where operations should provide generality:

- the size of the deformed region;
- the position of the deformed region;
- the shape of the boundary of the deformed region;
- the shape of the deformed region itself.

Although she applies these criteria to warps, they can be used for all local deformations. They do not apply to global deformations, since it makes no sense to discuss the position, size, and boundary of a region when the entire surface is being deformed. Since there is no general way of evaluating global operations, each operation is discussed independently.

The operations in Table 5.1 are listed as either local or global. For each operation from Chapter 2, the deformation techniques that implement it are listed. In the sequel, the implementation of each operation is discussed in detail, and extensions are noted.

For the purpose of this chapter, “free form deformation” refers to both the original and its extension, unless otherwise noted. Since the original is faster, it is preferable in cases where both can be used. Only the original is discussed when no benefit is gained with the extension.

5.2 Global operations

5.2.1 Scale

Scaling can be achieved using Barr operators or free form deformation. The former allows the designer to scale a surface to some desired dimensions, whereas the latter does not easily permit such precision. In fact, the extra precision afforded by Barr operators over the free form techniques applies to many of the shape control operations. Since free form deformation embeds the surface in a volume, precision scaling can only be achieved if the volume fits the surface exactly. This is often difficult, especially when using interactive techniques where the grid may appear to fit, but

Type	Operation	Applicable Technique
Global	Scale	Barr, FFD, EFFD
	Bend	Barr, FFD, EFFD
	Thicken	Superquadrics
	Variable Shear	Barr
	Taper	Barr, FFD, EFFD
	Twist	Barr, Barr applied to FFD, EFFD
Local	Flatten	Barr
	Warp: square base	FFD, EFFD
	Warp: circular base	EFFD
	Warp: arbitrary base	EFFD
	Warp: skeletal base	EFFD
	Warp: beak	none
	Warp: scoop	FFD, EFFD
	Warp: bulge	none
	Freeze Region	FFD, EFFD
	Duplicate operation	Barr, FFD, EFFD
	Refinement	Superquadric sampling rate

Table 5.1: Summary of operations and applicable deformation techniques

does not.

Although Barr's scale operator is more precise, it does not always do exactly what is intended. The supertorus that serves as the "legs" to the chair model in Figure 4.8 illustrates this point. It is centered on the origin, and scaled using the scale operator as discussed in Section 3.3. The scale not only stretches the torus to make it longer, but also thickens it along the sides that run perpendicular to the direction of scaling. Although the result is aesthetically pleasing, it is not what was intended: the part of the legs that lies on the ground should have been the same thickness as the rest.

To overcome this problem, scaling was performed using free form deformation instead of the scale operator. A grid was placed over the entire supertorus, with eight sections dividing it along the z axis, as in Figure 5.1. The leftmost and rightmost sections of the grid were held a constant size, and merely moved out from the center of the supertorus, while the inner sections were expanded to simulate scaling. Figure 5.1 shows the grid after deformation. Notice that the torus need not be centered on the origin when scaling with free form deformation.

Extended free form deformation can be used to scale curved surfaces. In practice, models are often imported from other modellers, or have been created in the past with no history of their deformation process. In these cases, scaling along an axis is not general enough, as the scale may need to be applied along a curve. To scale the supertorus after the bend operations have been applied, a curved grid can be fitted to it and then scaled as needed. Since the grid can be curved in three dimensions, scaling along nonplanar curves is supported by this technique.

GROOK's scale operator extends previous operators by allowing the operation to

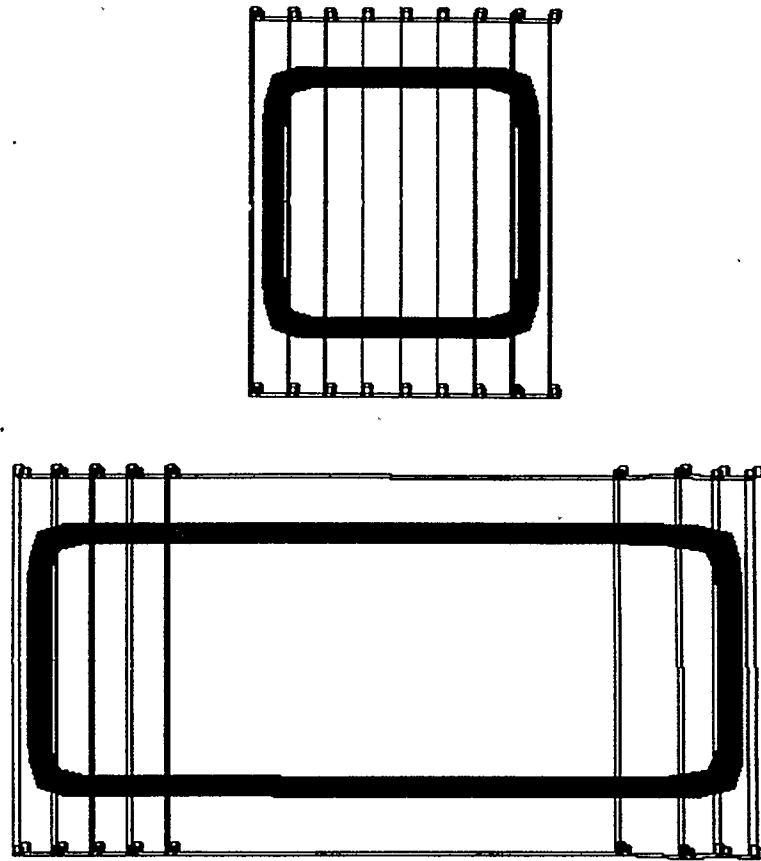


Figure 5.1: Torus scaled with free form deformation

be applied along three-dimensional curves.

5.2.2 Bending

Bending can be achieved using either Barr operators or free form deformation. The chair in Figure 4.8 was modelled on the Cobb chair and demonstrates how bend and scale can be used to deform superquadric primitives. The seat is modelled using a superellipsoid with exponents of 0.2 to square it, and then a scale to thin it along z . Two bends are used: one to create the sharp bend between the seat and the backrest, and a shallow one running from the front of the seat to the top of the backrest. Both were implemented using Barr operators.

Free form deformation can also be used to effect a bend, but lacks the precision of the Barr operator. It does, however, offer more flexibility in the types of bends permitted, since a bend in three dimensions can be executed in one application, whereas the Barr operator can only bend in a plane. Free form deformation also allows the surface to be bent and then tapered along that bend, which Barr operators cannot do. The extended version can be used to bend surfaces which are already bent.

GROOK's bend operator extends previous operators by allowing the bend to be defined in three dimensions.

5.2.3 Thicken

The variable lift and offset operator specified by Cobb allows a surface to become a solid by duplicating it and joining the copied surface to the original. This feature has not been introduced into GROOK because "thickening" surfaces is not needed: solid superquadrics can be deformed to achieve the desired models. For example, a variety of solids have been created simply by deforming superellipsoids: the agitator blades of Figure 4.6, the table legs of Figure 4.9, the chair seat of Figure 4.8, and the flashlight handle of Figure 5.2.

5.2.4 Twist and taper

A twisted or tapered surface can be created using Barr operators, or by using Barr operators with free form deformation. The blades for the washing machine agitator were created using the Barr twist and taper operators, while the horns in Figures 4.14 and 4.15 were executed by applying Barr operators to the space enclosed by an

extended free form deformation grid. In this case, the free form technique is not quite as intuitive as the Barr operator, given the details outlined in Chapter 4, but the extended version allows surfaces to be twisted and tapered along a curve, which other systems do not allow. It may be useful to apply other functions along a curved surface, although this has not been investigated.

The twist and taper operations must be applied over the entire cross-section of a surface, and in this sense are global. The operations can be applied locally by specifying a range over which they are applied when using Barr operators, or by placing the grid over the desired range when using free form deformation. This permits some flexibility in the position of the deformation. The shape of the deformed region can be altered by controlling the rate of tapering or twisting.

The twist and taper operations offered by GROOK extend previous operations by permitting the functions to be applied along three-dimensional curves.

5.2.5 Variable shear

Variable shearing is a new operation offered by GROOK, and is particularly useful for superquadric based modellers because conic shear functions can be applied to surfaces. Since superquadric silhouettes are defined by conics, surfaces can be made to align with each other, as in Figure 4.9. The table legs are modelled with a scaled superellipsoid which is sheared along a hyperbola to make it fit the centerpiece (a superhyperboloid of one sheet).

As this is a new operation, it extends the set of primitive operations. It extends the shear operation by including nonlinear functions, but is still applied along a straight line, in a plane.

5.3 Local deformations

5.3.1 Flatten

The flatten operator is implemented in GROOK, but is not a well defined operator. It bases the deformation function on the flatten operator as implemented by Cobb [Cob84]. The function is:

$$F(x, y, z) = \begin{cases} x + an_x \\ y + an_y \\ z + an_z \end{cases}$$

where n is the vector of projection. All surface points falling within a defined region are projected onto a user-defined plane in the direction of n . The constant a represents the distance between the sampled point and the plane along the direction of projection. The equation of the plane must be given:

$$Ax + By + Cz + D = 0$$

By substituting the equations of the projected point into the plane equation, the distance can be calculated:

$$a = \frac{-Ax - By - Cz - D}{An_x + Bn_y + Cn_z}$$

The flashlight handle in Figure 5.2 illustrates how a region of a superquadric can be flattened. Though this function works well with polygonised surfaces, it does not maintain a C^0 continuous surface as required by Barr operators. When the surface is lifted to the planar region, it is disconnected from the rest of the surface along the boundary. The polygonisation will create polygons to connect the regions, but the surface itself will no longer be connected. The position of the flattened region is

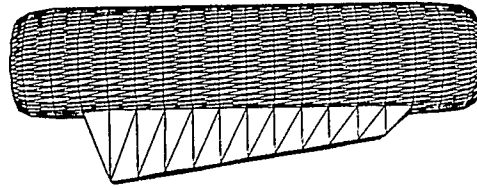


Figure 5.2: Flattening a region using Barr's technique

also restricted since deformed superquadric primitives may have regions that overlap along the line of projection. This causes the surface to self-intersect in the flattened region, and must be avoided. Cobb's flatten operator is made possible by applying the flatten operation to the control points of a B-spline surface, rather than to the surface itself. This not only maintains surface continuity but also provides a smooth transition at the boundary.

Flattening a curved surface with extended free form deformation also proves to be difficult. The method involves fitting the initial grid to the surface, then pulling control points in the grid up from the surface into a thin, flat parallelepiped shape. However, several problems are encountered:

- a close fit between the surface and the grid volume is crucial for the flatten to work properly;
- fitting the grid volume to the surface is not easy;
- more than one chunk ($3 \times 3 \times 3$ grid) is required to attain a flat volume, and GROOK only supports volumes defined by one chunk.

Since the grid is the control mesh for a Bezier volume, just fitting the grid to a surface does not mean that the volume defined by it fits the surface. If the surface is

at all complex, adjusting the grid vertices appropriately can be a tedious task. Even then, it is hard to determine whether the surface fits entirely within the volume, since the wire frame images of the grid volume and the surface do not provide adequate perceptual cues.

GROOK could not be used to test even simple cases since a $3 \times 3 \times 3$ grid cannot be used to flatten the surface. To maintain C^0 continuity at the boundaries, the outer planes of the grid must remain in their original positions, while the inner vertices are lifted. The Bezier definition of a $3 \times 3 \times 3$ grid causes the volume to be curved, as in Figure 3.13. An implementation that uses multiple chunks in one grid could likely overcome this by lifting entire chunks.

5.3.2 Warps

Extended free form deformation was designed to create arbitrarily shaped warps in surfaces with respect to the criteria provided by [Coq90]. The warps offered by GROOK provide the same level of generality. Although a functional specification for warping could be given, the inherent benefits of the free form technique make it the most feasible approach, except in the rare cases when the shape is easily described analytically. The following paragraphs describe how free form techniques can be used to implement each of the warps mentioned in Chapter 2.

Square base

Free form deformation allows bumps in surfaces to be created interactively. Figure 5.3 shows a grid being used to create a warp in a plane. The boundary between the deformed and undeformed areas is rectangular since the grid is parallelepiped. All

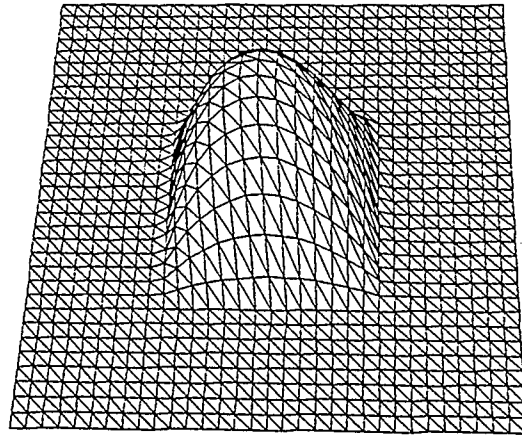


Figure 5.3: Deformation with square base

boundaries for free form deformations are restricted in this way.

Circular base

Figure 5.4 illustrates a circular warp, as described by Cobb. Extended free form deformation provides the necessary flexibility by allowing a nonparallelepiped grid. In this case, the grid approximates a circular base as shown in Figure 5.5.

Arbitrary base

Arbitrarily shaped initial grids can be used to deform various surfaces. Figure 5.6 shows some examples.

Skeletal base

The skeletal warps described by Cobb can be readily executed using free form techniques since an initial grid can be placed on the surface and then pulled up to raise the surface along the grid. A smooth deformation can be attained by only pulling up the grid's inner control points. Figures 5.7 and 5.8 show a skeletal warp with

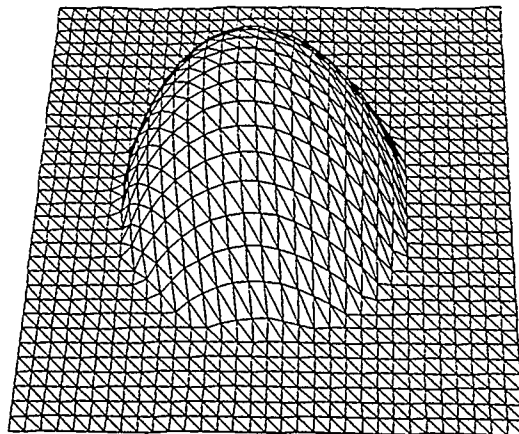


Figure 5.4: Deformation with circular base

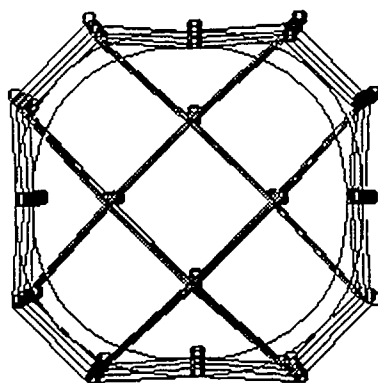


Figure 5.5: Grid used for warp with circular base

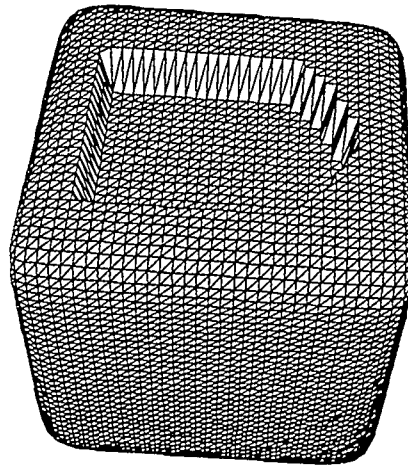


Figure 5.6: Warp with arbitrary base

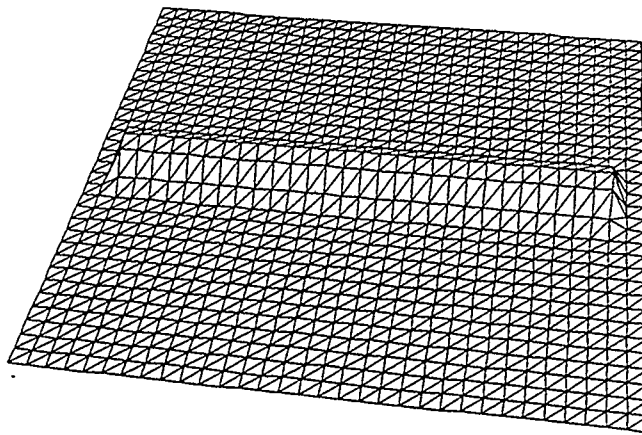


Figure 5.7: Skeletal warp using free form deformation: C^0 continuity

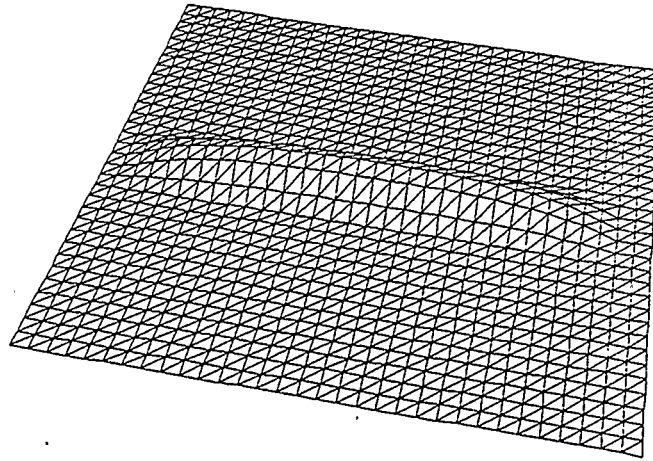


Figure 5.8: Skeletal warp using free form deformation: C^1 continuity

C^0 and C^1 continuity at its ends.

Beak — nonsmooth warps

The beak operation requires direct manipulation of the surface. Since this is not possible using Barr operators or free form techniques, GROOK cannot provide this operation. The angular warps resulting from the beak operation should, however, be achievable with GROOK. Sharp edges cannot be obtained with free form techniques—the deformed volume is based on Bezier surfaces, which are smooth. Figure 5.9 illustrates how even an extreme deformation in the grid still results in a smooth warp. A system permitting hierarchical application of free form deformation could likely attain angular warps by repeatedly applying grids until the required level of sharpness is achieved.

Scoop — inverse warps

The scoop operation offered by Form Synth can be attained using free form deformation by pushing the control points inward rather than outward from the surface.

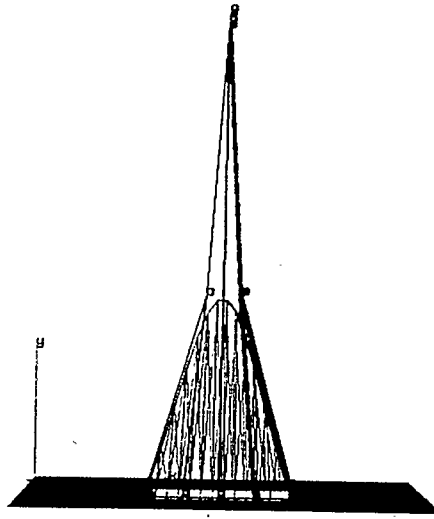


Figure 5.9: Creating a sharp tip in the warp using free form deformation

The position, size, and shape of the deformation are as flexible for the scoop as they are for the outward warp.

Bulge — spherical warps

Although free form deformation allows a multitude of warps, it has limitations. One is that the bulge operation of Form Synth cannot be simulated. The problem lies in the size of the warp at the tip: the fact that it has a narrow base and a spherical end makes it difficult to deform the grid enough to make the spherical expansion without having grid sections near the base of the warp intersect. A system allowing hierarchical applications of deformations could likely overcome this by pulling up the surface in the first application, and then expanding the region in the following one.

5.3.3 Freeze region

GROOK does not offer a way to freeze a region of a superquadric so that deformations will not affect that region. The operation used by Delta allows warps to have arbi-

trary boundaries by permitting parts of the warp region to be frozen. Since extended free form deformation permits arbitrary boundaries, it can be used to achieve the same effect: the grid is designed to cover only those regions that need to be deformed.

Freezing of a superquadric region, as in Delta, could not be implemented in any case because it relies on the underlying polygonal representation. Polygons connecting the deformed and the frozen regions are stretched to ensure surface continuity. The functional specification of the superquadric surface requires that the deformation provide a smooth connection to the frozen region, otherwise C^0 continuity would not be guaranteed.

5.3.4 Duplicate operation

GROOK does not allow duplicate operations to be applied to surfaces. Both Barr operators and free form deformation can be augmented to duplicate operations at other locations of the surface, but Barr operators are restricted in the orientation of the duplication. Since operators such as the taper function use the position of a point along one axis to determine the amount of deformation, they give different results when applied to identical surfaces at different locations. As a result, the surface must be translated and rotated to have the same orientation and position with respect to the deformation being applied. Assuming the restrictions for the specific operation being duplicated are met, Barr operators can be duplicated with no restriction on the size, position, boundary, or shape of the deformed region.

Duplicating an operation executed with free form techniques entails saving both the original and the deformed grids. The former must then be placed at its new location, and the latter aligned with it. This alignment must match exactly, and

cannot be done interactively. If the duplicate application has the same orientation the relative offsets of the deformed grid points from the original grid points could be stored. If a different orientation is desired, the offsets would need to be rotated the same amount as the grid.

The size, position, boundary, and shape of the region being duplicated with free form deformation is arbitrary.

5.3.5 Refinement

Since superquadrics are defined by continuous, functional specifications of a surface, they do not need a refinement operation. However, since GROOK uses polygonisation for rendering, there must be a way of controlling sampling. The user can set a “resolution” value which determines the sampling rate for the superquadric surface. The sampling is uniform in either parameter or object space, and there is no provision for highly deformed areas to be refined. Consequently, other rendering techniques such as adaptive subdivision should be explored (these are summarised in Chapter 6).

Since GROOK implements global refinement, the size, position, boundary, and shape of the deformed region is not arbitrary. Adaptive subdivision automatically refines the sampling at regions of high curvature, which does not provide arbitrary control of the region being refined, but is based on a reasonable assumption.

5.4 Summary and comparisons

This chapter compared the deformations achievable with GROOK with the operations listed in Chapter 2. Each operation as defined in GROOK was analysed with respect to the generality it provided. A yardstick for discussing generality in operations was provided for local deformations. The following questions arise:

- how well does GROOK compare to previous systems?
- how general are the operations it implements?

5.4.1 Comparison to existing systems

Barr operators and free form deformation can be used to replicate most of the operations in Chapter 2. All have been successfully modelled with GROOK with the following exceptions: flatten, beak and bulge warps, local refinement, and freezing a region. Of these operations, the flatten operator is the only one that is difficult to represent with the three deformation methods discussed in this thesis. Beak and bulge warps can likely be attained with a system that permits hierarchical application of free form deformation. Local refinement based on adaptive subdivision is documented in the literature [HB87], but its implementation is beyond the scope of this thesis. Freezing a region was shown to be redundant in a deformation based system.

5.4.2 Comparison to metric and previous capabilities

Global operations are extended by GROOK in several ways. First, surfaces may be scaled, bent, tapered, and twisted along a curve. This extends the original

operations, which are applied along axes. Second, surfaces may be scaled, bent, tapered, and twisted along three-dimensional curves. Rather than having to apply several planar operations, only one application of the deformation is needed. Scaling, bending, and tapering rely on extended free form deformation, while twisting requires a combination of Barr operators and extended free form deformation. Third, the deformation can be applied anywhere in the coordinate system, at any orientation. This is in contrast to many of the operations listed in Chapter 2, where the surface must be applied at or near the origin, along one of the axes. Finally, GROOK offers a new operation to variably shear surfaces. This permits superquadrics to be aligned with one another.

Local operations can all be implemented with deformation based techniques, except the flatten operator. As noted, each operation should provide generality in the size, position, boundary, and shape of the deformed region. The warp operation is enhanced by extended free form deformation to meet each of the four requirements. Local refinement can be implemented to provide adaptive subdivision, which does not provide generality in the four areas, but does provide refinement where it is most often needed. Duplication of both Barr operators and free form deformation can be implemented in an arbitrary manner.

Chapter 6

Conclusions

This thesis has shown that deformation based modelling is a viable modelling technique, in that it provides:

- a rich set of operations;
- operations that are general;
- an intuitive interface.

This has been substantiated as follows. Chapter 2 provides a survey of existing systems and summarises the operations they offer. Chapter 3 provides mathematical details of superquadrics, Barr operators, free form deformation, and its extension. A new operation to variably shear a surface is defined. Chapter 4 includes design and implementation details of the testbed program GROOK, and illustrates strengths and weaknesses in the deformation techniques and their superquadric primitives. Chapter 5 demonstrates that a deformation based modeller can perform most of the operations listed in Chapter 2, and extends many of them.

New contributions resulting from this work are:

- a comparison of three deformation based techniques;
- a method to combine Barr operators with free form deformation;
- the unification of the three approaches into one system, thus providing a spectrum of techniques that can be applied to the problem at hand;

- the variable shear operation, which can be used to align superquadrics;
- normal and tangent calculations for superquadrics deformed with free form deformation.

Several conclusions were drawn from experimentation with the three deformation based techniques. First, in terms of execution speed, Barr operators are the most efficient of the three techniques, while extended free form deformation is the least efficient. However, the extension provides considerably more expressive power than the original, unextended, free form deformation. Second, free form deformation and its extension provide more generality in shape control than Barr operators. However, Barr operators are indispensable as they provide precise specification of the operations, which free form deformation cannot. Third, Barr operators can provide deformations which free form deformation cannot, such as the variable shear function.

Several areas of deformation based modelling deserve further investigation. We briefly examine five: surfaces based on the spherical product, the user interface, rendering, extensions to each of the deformation techniques, and the design of a fully developed system.

This thesis described four superquadric surfaces. The spherical product can be used to describe many more surfaces by using different curves for $\vec{h}(\omega)$ and $\vec{m}(\eta)$. This extension fits nicely into a deformation based modeller since the spherical product is a deformation specification: it deforms a two-dimensional curve along an axis. Other quadrics such as the ones listed in [RA90] can create new superquadrics, while nonconic curves can be used to create nonquadric based surfaces.

The user interface for GROOK should permit free form deformation grids to be specified interactively. Coquillart's system [Coq90] provides lofting and extruding operations so that grids can be created from two-dimensional curves. Other possibilities should be investigated. Modes for manipulating the grids should be more flexible. A method for grouping vertices so that they can be moved simultaneously has been discussed in Section 4.4.1. Since grids may become complex, the user should be able to "hide" certain parts of the grid by not having them displayed. This technique is used by Forsey [FB88]. Finally, when Barr operators are applied to free form deformation volumes, the deformed surface must remain embedded in the grid. A way to check this before the entire surface is deformed would be of great benefit.

Without proper rendering techniques, the most impressive deformations will not be adequately illustrated. Two algorithms for deformed surfaces have been documented: one to ray trace parametric surfaces directly [Bar86] and another to calculate triangulations of them [HB87]. The triangulation method is more general as the output can be fed to scan-line algorithms or ray tracers.

Each of the deformation techniques deserves further investigation. More examples of Barr operators may prove useful, as did the variable shear function. A flatten operator, for example, that permits an arbitrary boundary may yet be accomplished with a Barr function.

Extended free form deformation warrants further research in several directions. An expanded formulation that includes multiple chunks per grid and hierarchical application of grids should be implemented. Multiple chunks permit a greater variety of shapes to be expressed, and would permit a better framework in which to test situations in which Newton's iteration fails to converge. Continuity between chunks

would need to be guaranteed, possibly with the tangent alignment method proposed by [Coq90]. This also affects the way Barr operators are applied to a grid: the deformation function must continue smoothly between chunks, each of which is defined by separate Bernstein polynomials. Unexpected results may occur if the grid chunks are not the same relative size, since a function applied over all chunks would not necessarily be applied evenly in object space.

Hierarchical application of grids would allow the beak and bulge operations to be tested. It may prove fruitful to permit multiple grids to be hierarchically organised into one “object,” much as primitives can be grouped hierarchically in systems such as PG [WMG86]. This would permit the designer to move them as one object, and would allow local deformations of other larger deformations. In this way, changes to the large deformation grid would also affect the local grid, as in [FB88]. Whether this is possible is not clear: the surface deformed by the large grid may not lie in the volume defined by the local one.

This thesis evaluated deformation based modelling according to its ability to create a wide variety of shapes. Since it has proven to be useful in this area, a complete system should be analysed with respect to other criteria such as those outlined in [All88]. In particular, efficiency of each technique should be further investigated. A great deal is known about algorithms for Bernstein polynomials [GR74] [FR87], and a faster, more efficient implementation based on these should be investigated. For example, Bernstein polynomials can be evaluated using the de Casteljau algorithm [Las85].

Finally, a fully developed modeller based on deformations should be developed. Grids could be used not only to deform surfaces, but also to position objects within

a scene: the objects are embedded in a grid, it is deformed or a Barr operator is applied to it, and the position vector of the object is deformed accordingly. Such a system presents interesting possibilities for animation, and an animated free form deformation has already been described [Coq91].

In 1989 I made a computer animated movie entitled “Snoozin’ Blues.” This depicted a saxophone that expanded and contracted as it snored. I wanted the keys, represented by a row of spheres, to be twisted around the bore of the saxophone. Since the instrument was conical and bent, the twist needed to be applied along a bend, and diagonally outward from the axis. I had to do this by trial and error—a very tedious process (I finally changed the script to avoid the twist). However, deformation based modelling would have been an ideal solution. By embedding the saxophone in a similarly shaped grid and aligning the spheres alongside it, a Barr twist function applied to the volume would calculate the new positions of the spheres.

Piet Hein’s superellipses initiated the developments that led to this thesis. His aim, to bridge the gap between Art and Science, is the essence of computer graphics. The challenge is to provide an expressive environment for artists and designers; this can only be achieved when the science behind the system is made transparent. Deformation based modelling is an important approach to this goal. The work presented in this thesis provides a sound basis for a powerful modelling technique in computer graphics.

Bibliography

- [All88] Jeffrey B. Allan. Polygon mesh modelling. Master's thesis, University of Calgary, September 1988.
- [Ant81] Howard Anton. *Elementary Linear Algebra*. John Wiley and Sons, New York, third edition, 1981.
- [Bar81] Alan H. Barr. Superquadrics and angle preserving transformations. *IEEE Computer Graphics and Applications*, 1(1):11–23, Jan 1981.
- [Bar84] Alan H. Barr. Global and local deformations of solid primitives. *ACM Siggraph*, 18(3):21–30, July 1984.
- [Bar86] Alan H. Barr. Ray tracing deformed surfaces. *ACM Siggraph*, 20(4):287–296, August 1986.
- [BBB87] Richard H. Bartels, John C. Beatty, and Brian A. Barsky. *Curves and Surfaces for Computer Aided Geometric Design*. Morgan Kaufmann Publishers Inc, Los Altos, California 94022, 1987.
- [BF91] P. J. Barry and D. R. Forsey. An introduction to spline curves and surfaces. Graphics Interface course notes, June 1991.
- [Bli86] James Blinn. Geometry for computer graphics and computer aided design. ACM SIGGRAPH Course Notes, Number 10, August 1986.
- [Blo88] J. Bloomenthal. Polygonization of implicit surfaces. *Computer Aided Geometric Design*, 5(4):341–355, 1988.

- [BW76] N. Burtnyk and M. Wein. Interactive skeleton techniques for enhancing motion dynamics in key frame animation. *ACM Siggraph*, 1976.
- [CGG⁺88a] Bruce W. Char, Keith O. Geddes, Gaston H. Gonnet, Michael B. Monagan, and Stephen M. Watt. *Maple Reference Manual*. Symbolic Computation Group, University of Waterloo, Canada, fifth edition, March 1988. Published by Watcom Publications Limited.
- [CGG⁺88b] Bruce W. Char, Keith O. Geddes, Gaston H. Gonnet, Michael B. Monagan, and Stephen M. Watt. *First Leaves: a tutorial Introduction to Maple*. Symbolic Computation Group, University of Waterloo, Canada, July 1988. Published by Watcom Publications Limited.
- [CHP89] John E. Chadwick, David R. Haumann, and Richard E. Parent. Layered construction for deformable animated characters. *ACM Siggraph*, 23(3):243–252, July 1989.
- [Cob84] Elizabeth Susan Cobb. *Design of Sculptured Surfaces Using the B-Spline Representation*. PhD thesis, The University of Utah, June 1984.
- [Coq90] Sabine Coquillart. Extended free-form deformation: A sculpturing tool for 3d geometric modeling. *Computer Graphics SIGGRAPH*, 24(4):187–193, August 1990.
- [Coq91] Sabine Coquillart. Animated free-form deformation: An interactive animation technique. *Computer Graphics SIGGRAPH*, 25(4):23–26, July 1991.

- [Far88] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, Inc, 1988.
- [FB81] Wm. Randolph Franklin and Alan H. Barr. Faster calculation of superquadric shapes. *IEEE Computer Graphics and Applications*, 1, 1981.
- [FB88] David R. Forsey and Richard H. Bartels. Hierarchical B-Spline refinement. *ACM Siggraph*, 22(4):205–212, August 1988.
- [FP79] I. D. Faux and M. J. Pratt. *Computational Geometry for Design and Manufacture*. Ellis Horwood Publishers, 1979.
- [FR87] R. T. Farouki and V. T. Rajan. On the numerical condition of polynomials in Bernstein form. *Computer Aided Geometric Design*, 4:191–216, 1987.
- [FvDFH90] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics Principles and Practice*. Addison-Wesley Publishing Company, New York, second edition, 1990.
- [FW83] A. Fournier and M. A. Wesley. Bending polyhedral objects. *Computer-Aided Design*, 15(2):79–87, March 1983.
- [Gar77] Martin Gardner. *Mathematical Carnival*, chapter 18, pages 240–254. Vintage Books, 1977.
- [Gar84] Geoffrey Y. Gardner. Simulation of natural scenes using textured quadric surfaces. *ACM Siggraph*, 18(3):11–20, July 1984.

- [Gla89] Andrew Glassner. *An Introduction to Ray Tracing*. Academic Press, 1989.
- [Gol83] Ronald N. Goldman. Two approaches to a computer model for quadric surfaces. *IEEE Computer Graphics and Applications*, Sept 21-24 1983.
- [GR74] William J. Gordon and Richard F. Riesenfeld. Bernstein-Bezier methods for the computer-aided design of free-form curves and surfaces. *Journal of the Association of Computing Machinery*, 21(2):293–310, 1974.
- [HB87] Brian Von Herzen and Alan H. Barr. Accurate triangulations of deformed, intersecting surfaces. *ACM Siggraph*, 21(4):103–110, July 1987.
- [HE89] Avon Huxor and Iain Elliot. Superquadric-based symbolic graphics for design. In *Computers in Art, Design and Animation*, pages 183–194. Springer-Verlag, 1989.
- [KAW91] Zoran Kacic-Alesic and Brian Wyvill. Controlled blending of procedural implicit surfaces. *Graphics Interface*, pages 236–245, June 1991.
- [Koe90] Jan J. Koenderink. *Solid Shape*. The MIT Press, Cambridge, Massachusetts, 1990.
- [KPC91] James R. Kent, Richard E. Parent, and Wayne E. Carlson. Establishing correspondences by topological merging: A new approach to 3-D shape transformation. *Graphics Interface*, pages 236–245, June 1991.

- [Las85] Dieter Lasser. Bernstein-Bezier representation of volumes. *Computer Aided Geometric Design*, 2:145–149, 1985.
- [Las87] John Lasseter. Principles of traditional animation applied to 3D computer animation. *ACM Siggraph*, 21(4):35–44, July 1987.
- [Lat89] William Latham. Form synth: The rule-based evolution of complex forms from geometric primitives. In John Lansdown and Rae A. Earnshaw, editors, *Computers in Art, Design and Animation*, pages 80–108. Springer-Verlag, 1989.
- [Lor53] G. G. Lorentz. *Bernstein Polynomials*. University of Toronto Press, 1953. Mathematical Expositions, No. 8.
- [McP90] Craig McPheeters. *The Joy Manual*. Computer Graphics Laboratory, University of Calgary, Canada, October 1990.
- [Mor85] Michael E. Mortensen. *Geometric Modeling*. John Wiley and Sons, New York, 1985.
- [MTT89] Nadia Magnenat-Thalmann and Daniel Thalmann. *State-of-the-Art in Computer Animation*. Springer-Verlag, New York, 1989.
- [MTT91] Nadia Magnenat-Thalmann and Daniel Thalmann. *New Trends in Animation and Visualization*. John Wiley and Sons, New York, 1991.
- [Par77] Richard E. Parent. A system for sculpting 3-d data. *Computer Graphics*, 11(2):138–147, July 1977.

- [Par86] Scott R. Parry. *Free-Form Deformations in a Constructive Solid Geometry Modeling System*. PhD thesis, Brigham Young University, April 1986.
- [PB88] John C. Platt and Alan H. Barr. Constraint methods for flexible models. *ACM Siggraph*, 22(4):279–288, August 1988.
- [Pen86] Alex P. Pentland. Perceptual organization and the representation of natural form. *Artificial Intelligence*, 28:293–331, 1986.
- [PFTV88] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, 1988.
- [RA90] David F. Rogers and J. Alan Adams. *Mathematical Elements for Computer Graphics*. McGraw-Hill Publishing Company, second edition, 1990.
- [RO87] Alyn P. Rockwood and John C. Owen. Blending surfaces in solid modeling. In Gerald E. Farin, editor, *Geometric Modelling: Algorithms and New Trends*, pages 367–383. Society for Industrial and Applied Mathematics, 1987.
- [Smi89] Gillian Crampton Smith. Computer graphics and graphic design: Too costly, too complex, too cryptic. In John Lansdown and Rae A. Earnshaw, editors, *Computers in Art, Design and Animation*, pages 225–234. Springer-Verlag, 1989.

- [SP86] Thomas W. Sederberg and Scott R. Parry. Free-form deformation of solid geometric models. *ACM Siggraph*, 20(4):151–160, August 1986.
- [TPBF87] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. *ACM Siggraph*, 21(4):205–213, July 1987.
- [WMG86] Brian Wyvill, Craig McPheeters, and Rick Garbutt. The University of Calgary 3D Computer Animation System. *Journal of the Society of Motion Picture and Television Engineers*, 95(6):629–636, 1986.