SIGGRAPH 1996

Course Notes

Implicit Surfaces for Geometric Modeling and Computer Graphics

Co-Chairs

Jai Menon
IBM T.J. Watson Research Center

Brian Wyvill
University of Calgary

Lecturers

Chandrajit Bajaj
Purdue University

Jules Bloomenthal
Microsoft Corporation

Baining Guo
York University

John Hart
Washington State University

Geoff Wyvill
University of Otago

◊

SIGGRAPH 1996

Implicit Surfaces for Geometric Modeling and Computer Graphics

Welcome to Implicit Surfaces for Geometric Modeling and Computer Graphics.

In this course we will survey implicit surfaces, discuss their usefulness, describe their advantages and disadvantages relative to other modeling techniques, and present the latest techniques for their design. Until recently, implicit surfaces have received little attention, partly due to the difficulties in visualizing them interactively. From the moment one realizes that it is easier to draw a circle with $(r \cos \theta, r \sin \theta)$ than it is with $(x^2 + y^2 = r^2)$, one is slowly led away from the world of implicit surfaces.

Welcome back!

Implicit surfaces are different from parametric surfaces: the latter, in use in many commercial modeling systems, are familiar to most of the computer graphics community. Implicit surfaces aren't necessarily less practical; they are simply different. They require different techniques for their creation, modification and visualization and have different properties and applications from their parametric counterparts.

The speakers in this course will discuss their current work in developing techniques to make implicit surfaces practical in modeling and animation. By definition, implicit surfaces embrace an extremely large set of surfaces. Undoubtedly, as they receive increased use in computer graphics, concepts will be developed that unify and distinguish various implicit forms. We hope the variety of approaches, applications and results presented in this course will stimulate interest in this exciting branch of modeling.

Courses on Implicit Surfaces were previously offered at SIGGRAPH in 1990 and 1993, co-organized by Jules Bloomenthal and Brian Wyvill.

Jai Menon, *IBM T.J. Watson Research Center*
Brian Wyvill, *University of Calgary*
1996

SIGGRAPH 1996

Implicit Surfaces for Geometric Modeling and Computer Graphics

Speaker Biographies

Chandrajit Bajaj

Professor
Department of Computer Science
1398 CS Bldg
Purdue University
West Lafayette, IN 47907
http://www.cs.purdue.edu/people/bajaj
bajaj@cs.purdue.edu

Chandrajit Bajaj graduated from the Indian Institute of Technology, Delhi in 1980 with a Bachelor's Degree in Electrical Engineering. Subsequently he received his M.S. and Ph.D. degrees in Computer Science from Cornell University, Ithaca, New York in 1984. Bajaj is currently a Professor in the Computer Science Department of Purdue University, West Lafayette, Indiana and directs the Collaborative Modelling and Visualization Laboratory which houses the SHASTRA projects. He also directs the Purdue Center of Computational Image Analysis and Data Visualization. His research are in the areas of Computational Geometry, Geometric Modeling, Computer Graphics, Scientific Visualization and Distributed and Collaborative Synthetic Environments.

Jules Bloomenthal

Member, Advanced Technology
Microsoft Corporation, Building 10N
1 Microsoft Way, Redmond, WA 98052
julesb@microsoft.com

Jules Bloomenthal studied computer graphics at the University of Utah, and recently received his Ph.D. from the University of Calgary. Dr. Bloomenthal has conducted research at the New York Institute of Technology and at Xerox PARC, and has taught computer graphics at George Mason University and UC Santa Cruz. He is presently with Microsoft Corporation.

Contending that implicit blends usefully represent natural forms, Dr. Bloomenthal has published on several implicit surface topics, including uniform and adaptive polygonization methods, polygonization of non-manifold surfaces, convolution of skeletons, bulge elimination in implicit blends, specification of volume/surface blends, definition of branching structures, interactive design and display techniques, and procedural methods.

Baining Guo

Assistant Professor
Department of Computer Science
York University
North York, Ontario
CANADA M3J 1P3
`guo@cs.yorku.ca`

Baining Guo is currently an assistant professor in the Computer Science Department at York University in Toronto (CANADA). He received his B.S. from Beijing University (PRC) in 1982 and his M.S. and Ph.D. from Cornell University in Ithaca, New York (USA) in 1989 and 1991. Prior to joining York, he worked for France Telecom (FRANCE) and The University of Colorado (USA). Guo was a visiting assistant professor at the University of Toronto, in the Department of Computer Science, where he still actively participate research activities.

Guo's research interests include volume visualization, geometric modeling, and computer vision. In geometric modeling, his work addresses issues in modeling with low degree implicit surfaces for CAD/CAM applications. In volume visualization, he is developing structure-based volume rendering techniques that combine volume rendering with feature extractions. Recently, he has started to construct direct solvers for early vision problems. Guo is a member of ACM.

John Hart

Assistant Professor
School of EECS
Washington State University
Pullman, WA 99164-2752
`hart@eecs.wsu.edu`

John C. Hart is an Assistant Professor in the School of Electrical Engineering and Computer Science at Washington State University. Hart received his B.S. in Computer Science from Aurora University, and his M.S. and Ph.D. in Computer Science in the Electronic Visualization Laboratory at the University of Illinois at Chicago. He also interned in Alan Norton's group at the IBM T.J. Watson Research Center, and at AT&T Pixel Machines (R.I.P.).

In 1993, Dr. Hart received an NSF Research Initiation Award to explore new modeling, rendering and animation techniques for implicit surfaces. This research resulted in new techniques for rendering skeletal models, volume visualization, implicit modeling of geometric detail and the interactive modeling of implicit surfaces. His implicit surface rendering algorithm was used to demonstrate the removal of a 720° twist in a ribbon in the SIGGRAPH '93 Electronic Theater animation "Air on the Dirac Strings." Dr. Hart is a co-chair of Implicit Surfaces '96 — the 1996 Eurographics/SIGGRAPH Workshop on Implicit Surfaces. He is also a member of ACM, SIGGRAPH and the IEEE Computer Society. At the time of this writing, he serves on the SIGGRAPH Executive Committee as a Director-at-Large, and is a candidate for Director of Communication.

Jai Menon

Project Leader, 3D Exploitation
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
`menon@watson.ibm.com`

Jai P. Menon is a Project Leader at the IBM T.J. Watson Research Center, where he currently manages R&D on geometry-based and image-based graphics systems – the *(IBM 3D Interaction Accelerator)* and *(IBM SurroundVR)* respectively – with a focus on PC platforms. Menon received his B.Tech. from the Indian Institute of Technology (Delhi), his M.S. and Ph.D. from Cornell University, and joined IBM Research immediately after. He is also an Adjunct Professor at the New York Polytechnic University. He serves on an industrial advisory board at the University of Wisconsin (Madison), and is a member of a doctoral thesis committee at SUNY (Stony Brook).

Menon's research interests lie in four broad areas. His work on "implicit surfaces" has focused on algebraic patches, where he has pioneered their use in exact CSG schemes. In his work on "massively parallel processing" for solid modeling, he has developed (in collaboration with Cornell and Duke Universities) custom-VLSI RayCasting Engine (RCE) and ray-rep technologies to support hitherto intractable applications, such as general sweeps, and Minkowski operations. His work in "manufacturing" has produced the P2NC automatic verification system (from Cornell) for Numerical Control (NC) machining programs. His work on "polyhedral graphics" (at IBM) has resulted in two releases of the IBM 3D Interaction Accelerator (3DIX) product for real-time visualization of and communication with complex (multi-million polygon) databases. He has authored several patents and has received several IBM awards for his work on the 3DIX system. He has participated in the production of IBM TV Olympic commercials aired on ABC, NBC, and so forth.

Menon has published a number of papers in all these areas. He serves as an area chair at the 1996 ASME Design for Manufacturing Conference, and is general co-chair at the 1997 conference. He has been invited to the editorial board of the *International Journal of Manufacturing Engineering* and will be a guest editor for the *Computer-Aided Design* journal. He serves on an ASME Executive Committee, and is a member of Phi Kappa Phi, ACM, SIGGRAPH and the IEEE Computer Society.

Brian Wyvill

Professor
Dept. of Computer Science
University of Calgary,
2500 University Dr. NW
Calgary, Alberta, T2N 1N4, Canada
`blob@cpsc.ucalgary.ca`

Brian Wyvill is a full professor in the Department of Computer Science at the University of Calgary where he heads the GraphicsJungle research group. After gaining his PhD in the UK in 1975, he worked at the Royal College of Art as a post doc. in London to produce a computer animation system. Since coming to Calgary in 1981, Brian's research has concentrated on building the Graphics and Animation and visualization system. Brian has directed several animations (two shown at SIGGRAPH) that feature implicit surfaces. Recent work is in the areas of implicit surface modeling, animation techniques and scientific visualization. Currently he is interested in very efficient adaptive tiling algorithms for implicit surfaces and CSG, as well as new techniques for warping, blending and collision detection using implicit surfacs. Brian is a member of ACM, SIGGRAPH, and on the editorial board of the Visual Computer as well as the Journal of Animation and Scientific Visualization.

Geoff Wyvill

Associate Professor
Department of Computer Science
University of Otago
Box 56, Dunedin
New Zealand
`geoff@otago.ac.nz`

Geoff Wyvill is an Associate Professor at the University of Otago, New Zealand and Director of Animation Research Limited. He is known for pioneering work in polygonizing implicit surfaces and ray tracing, especially the efficient ray tracing of CSG systems. He is an Executive editor of 'Virtual Reality' and serves on the editorial boards of 'The Visual Computer', 'Computer Graphics Forum' and 'Vizualisation and Computer Animation'. He contributed to the films 'Soft' (1985) 'Great Train Rubbery' (1988) and 'Fashion Show' (1992) as well as numerous TV commercial and channel animations. His company, ARL, has won eleven national and international awards for animation all of which has been produced using 'Katachi' Geoff's CSG, ray tracing and animation software. He has a BA in physics from Oxford University and MSc and PhD degrees in computer science from the University of Bradford.

SIGGRAPH 1996

Implicit Surfaces for Geometric Modeling and Computer Graphics

Table of Contents

SECTION A
Basic Building Blocks

Abstract

The first section will build some basics. It will contrast implicit and parametric methods, and overview the broad range of implicit surfaces used today. It will develop fundamental properties of quadratic implicit polynomial surfaces, and illustrate how these surfaces have been traditionally used in CAD systems for handling "prismatic" objects (e.g. piston rods), and how recent research on algebraic patches has extended their power to now support "free-form" objects (e.g. bones). This is followed by an introduction to the class of implicit skeletal methods that build shapes using distance functions and blends. As a new twist, this section will develop formal notions of representation schemes (particularly, Brep and CSG), conversions (fundamental problems of separation and describability), and their impact on rendering algorithms. This leads to an exercise in designing direct rendering hardware for quadratic surfaces, both for Brep-based and CSG-based parallel processing. In particular, the architecture of the prototype RayCasting Engine (RCE), with over 2K parallel processors, for direct CSG ray-casting will be discussed.

An Introduction to Implicit Techniques

Jai Menon
IBM T.J. Watson Research Center

Abstract

This chapter provides an introduction to implicit techniques used in geometric modeling and computer graphics. We begin by contrasting implicit and parametric methods. We then summarize three classes of techniques – algebraic, blobby, and functional. Functional (F-rep) techniques represent some of the recent work on using a single function to represent complex shapes.

1 Introduction

There are two main techniques for representing surfaces in geometric modeling and computer graphics – *parametric* and *implicit*. Parametric representations typically define a surface as a set points $\mathbf{p}(s,t)$, i.e.

$$\mathbf{p}(s,t) \quad = \quad (x(s,t),\ y(s,t),\ z(s,t)). \tag{1}$$

Implicit representations typically define a surface as the zero contour of a function $F(\mathbf{p}) = 0$, i.e.

$$F(\mathbf{p}) \quad = \quad F(x,\ y,\ z) \quad = \quad 0. \tag{2}$$

Parametric methods (such as non-uniform rational B-splines or NURBS) were motivated by properties of coordinate system independence, single-valued functions, ease of handling vertical slopes, and efficient evaluation of points on the surface ... the last being critical for image rendering; ergo its popularity in computer graphics [7]. Implicit methods, however, provide mathematical tractability and are becoming extremely useful for modeling operations such as blending, sweeping, metamorphosis, intersections, boolean operations, ... and even image rendering.

Explicit methods that express, for example, $y = f(x)$ and $z = g(x)$ are quite limiting. For example, it is impossible to get multiple values of y for a given x; hence circles and ellipses must be represented with multiple curve segments. Furthermore, such representations are not rotationally invariant and describing curves with vertical tangents is difficult, because a slope of infinity is difficult to represent [7]. This leaves parametric and implicit methods are the two key approaches.

In this chapter, we will take a brief tour through some of the key concepts of three major lines of work in implicit methods:

- *algebraic surfaces* (Section 2),
- *blobby objects* (Section 3), and
- *functional representations* (Section 4).

2 Algebraic Methods

Algebraic methods describe surfaces as implicit polynomials, i.e. $F(x, y, z)$ is a polynomial in x, y, and z. These are typically low degree (2, 3, 4) polynomials, and the most popular ones are quadratic implicit (degree 2) surfaces, often referred to as "quadrics". We will first make some observations on converting from parametric to algebraic representations and then devote the rest of this section on some basic concepts of quadrics.

2.1 A Note on Parametric Surfaces

We observe that parametric surfaces can be *implicitized*, i.e. converted to an implicit polynomial (or algebraic form) [17]. However, the resulting algebraic equation $F(x, y, z) = 0$ could have a degree as high as $2mn$ for a tensor product patch with rational functions of degree m and n, or n^2 for a triangular patch with rational functions of degree n. For example, a bi-cubic tensor-product patch could yield an algebraic patch of degree 18, and a quadratic triangular (Steiner) patch could yield a quartic (degree 4) algebraic equation.

In general, the intersection of a degree m algebraic surface with a degree n algebraic surface could result in a curve of degree mn. Hence the intersection of two bi-cubic patches could result in a curve of degree 324. Similarly for curve/patch intersections: Bezout's theorem states that a space curve of degree l intersects an algebraic surface of degree n in exactly ln points (counting complex, infinite, and multiple intersections) or else it intersects the surface infinitely often, i.e. a component of the curve lies entirely in the surface [18]. Since the implicit representations of parametric patches are usually of high degree, even the simplest curve/patch intersection is expensive to compute (for example, a line can intersect a bi-cubic patch in 18 points).

This explains why parametric patches (such as NURBS), albeit popular and flexible, are computationally so intractable. Parametric patches are often 'tamed' using linear (tesselated) approximations.

2.2 Basics of Quadric Surfaces

A quadric surface is the set $\{\mathrm{x} \mid F(\mathrm{x}) = \mathrm{x}^T Q \mathrm{x} = 0\}$, where

$$Q = \begin{bmatrix} A & D & E & G \\ D & B & F & H \\ E & F & C & J \\ G & H & J & K \end{bmatrix} \quad \text{and} \quad \mathrm{x} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}. \tag{3}$$

Expanding the equation $\mathrm{x}^T Q \mathrm{x} = 0$, we get

$$Ax^2 + By^2 + Cz^2 + 2Dxy + 2Exz + 2Fyz + 2Gx + 2Hy + 2Jz + K = 0 \tag{4}$$

which is a polynomial of degree 2. Nine popular types of quadric surfaces produced by varying the coefficients of this algebraic equation are sketched in Fig. 1. These are: ellipsoid, elliptic cone, cylinder (elliptic, parabolic, hyperbolic), hyperboloid (of 1 sheet, of 2 sheets), and paraboloid (elliptic, hyperbolic). These are referred to as *general quadrics*. Certain combinations of coefficients of the polynomial equation can also result in non-curved shapes (point, plane, parallel planes, coincident planes) or in invalid shapes (imaginary quadric, intersecting imaginary planes, imaginary parallel planes).

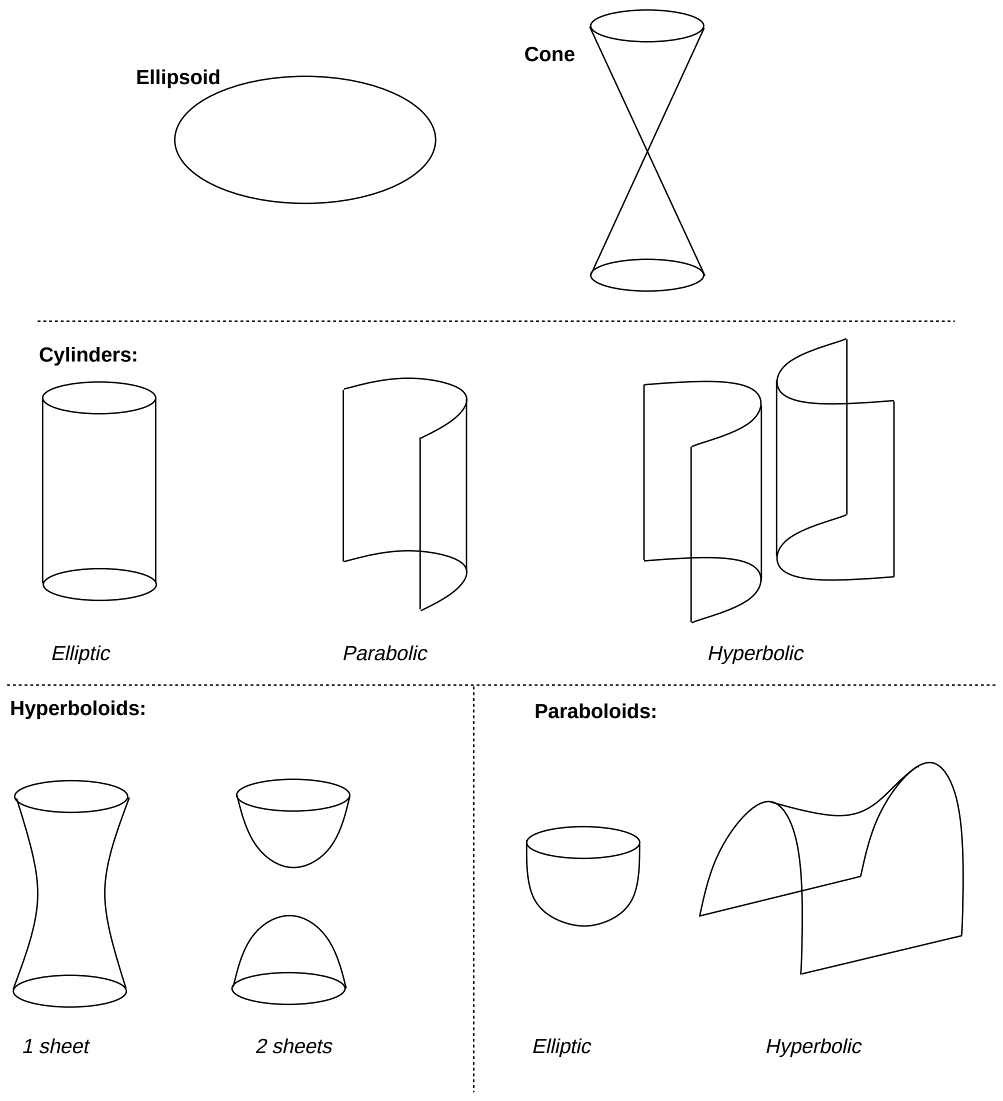


Figure 1: Sketches of general quadric surfaces.

(a)

(b)

Figure 2: Quadric surfaces may be used to describe (a) prismatic part (pump assembly) – traditional uses, or (b) free-form object (vase) – modern uses.

A particular subset of general quadrics is called *natural quadrics*; these include sphere (special case of ellipsoid), right circular cone (special case of elliptic cone), right circular cylinder (special case of elliptic cylinder), and planes. Natural quadrics are by far the most popular set of quadric surfaces used in modeling systems. Objects modeled with natural quadrics are often referred to as "prismatic" solids, such as in Fig. 2a, used mainly in mechanical CAD domains. Free-form objects were traditionally not modeled with quadric surfaces; in fact, parametric (NURBS) patches were most popularly used. Recently however, there has been a growing body of work on using subsets of general quadric surfaces – called algebraic patches – pieced together with tangent plane (G^1) continuity to create free-form shapes, as shown in Fig. 2b [11, 9, 12, 13].

2.3 Properties of Quadric Surfaces

General quadrics exhibit several interesting mathematical properties; a few of which are summarized below; see [5, 10] for more details. Let M denote an affine transformation describing transformed coordinates B in terms of reference frame A. Furthermore, let Q_A denote the matrix representing a quadric surface in frame A, and similarly Q_B in frame B. Then, the following describes the transformation relation for quadrics:

$$Q_B = M^T Q_A M. \tag{5}$$

Let Q_u denote the upper-left 3x3 sub-matrix of Q, i.e.

$$Q_u = \begin{bmatrix} A & D & E \\ D & B & F \\ E & F & C \end{bmatrix}. \tag{6}$$

The ranks of both Q and Q_u are invariant under affine transformations. Furthermore, the type of quadric surface is invariant under affine transformations. The quadric surfaces are invariant under rigid motions, and in particular, the characteristic functions

$$\mathrm{Det}(Q - \lambda I), \quad \text{and} \quad \mathrm{Det}(Q_u - \lambda I) \tag{7}$$

are invariant under rigid motion ("Det" denotes "determinant of").

Invariance of the characteristic equation provides algebraic methods for classifying the type of quadric surface. Specifically, the expansion of the characteristic equations yields coefficients D_1, D_2, D_3, D_4 and T_1, T_2, T_3 as follows.

$$\mathrm{Det}(Q - \lambda I) = \lambda^4 + D_1\lambda^3 + D_2\lambda^2 + D_3\lambda + D_4 \tag{8}$$

where

$$\begin{aligned}
D_1 &= A + B + C + K && (9)\\
D_2 &= AB + BC + CK + AK + AC + BK \\
&\quad - D^2 - E^2 - F^2 - G^2 - H^2 - J^2 && (10)\\
D_3 &= ABC + ABK + ACK + BCK + 2(DEF + FGJ + DGH + EHJ) \\
&\quad - (C + K)D^2 - (A + K)E^2 - (B + K)F^2 \\
&\quad - (B + C)G^2 - (A + C)H^2 - (A + B)J^2 && (11)\\
D_4 &= \mathrm{Det}(Q) && (12)
\end{aligned}$$

Similarly, the following relations follow for the Q_u submatrix.

$$\mathrm{Det}(Q_u - \lambda I) = \lambda^3 + T_1\lambda^2 + T_2\lambda + T_3 \tag{13}$$

where

$$\begin{aligned}
T_1 &= A + B + C && (14)\\
T_2 &= AB + BC + AC - D^2 - E^2 - F^2 && (15)\\
T_3 &= \mathrm{Det}(Q_u) && (16)
\end{aligned}$$

From the invariance property, these coefficients can be used to determine the type of quadric surface, using a decision tree due to Levin [10], shown in Fig. 3.

Certain quadric surfaces have the property that a family of straight lines can be found which lie entirely on the quadric surface. Such surfaces are called *ruled* quadric surfaces. Cylinders and cones are common examples of ruled quadric surfaces (Fig. 1). We leave it as an exercise to determine the family of straight lines that show how a hyperboloid of one sheet as well as a hyperbolic paraboloid are ruled quadrics.

If $F_1(x,y,z)$ and $F_2(x,y,z)$ are two quadric surfaces, there exists a family of quadric surfaces which are linear combinations of the two surfaces. This family constitutes the *pencil* of the two original surfaces, and is described by $F_1(x,y,z) + \alpha F_2(x,y,z)$ where α is an arbitrary real scalar. Since every point on the curve of intersection of F_1 and F_2 has to satisfy equations

$$F_1(x,y,z) = 0 \quad \text{and} \quad F_2(x,y,z) = 0 \tag{17}$$

it follows that the equation

$$F_1(x,y,z) + \alpha F_2(x,y,z) = 0 \tag{18}$$

is satisfied at the curve of intersection for all values of α. In other words, every quadric surface in the pencil intersects the surfaces F_1 and F_2 along their curve of intersection. Another property of interest is that there is at least one ruled quadric in the pencil of any two general quadrics. This property is useful because ruled quadrics are easily parameterized and the quadric surface intersection curves can be expressed parametrically using the parameterization scheme for the ruled quadric in the pencil.

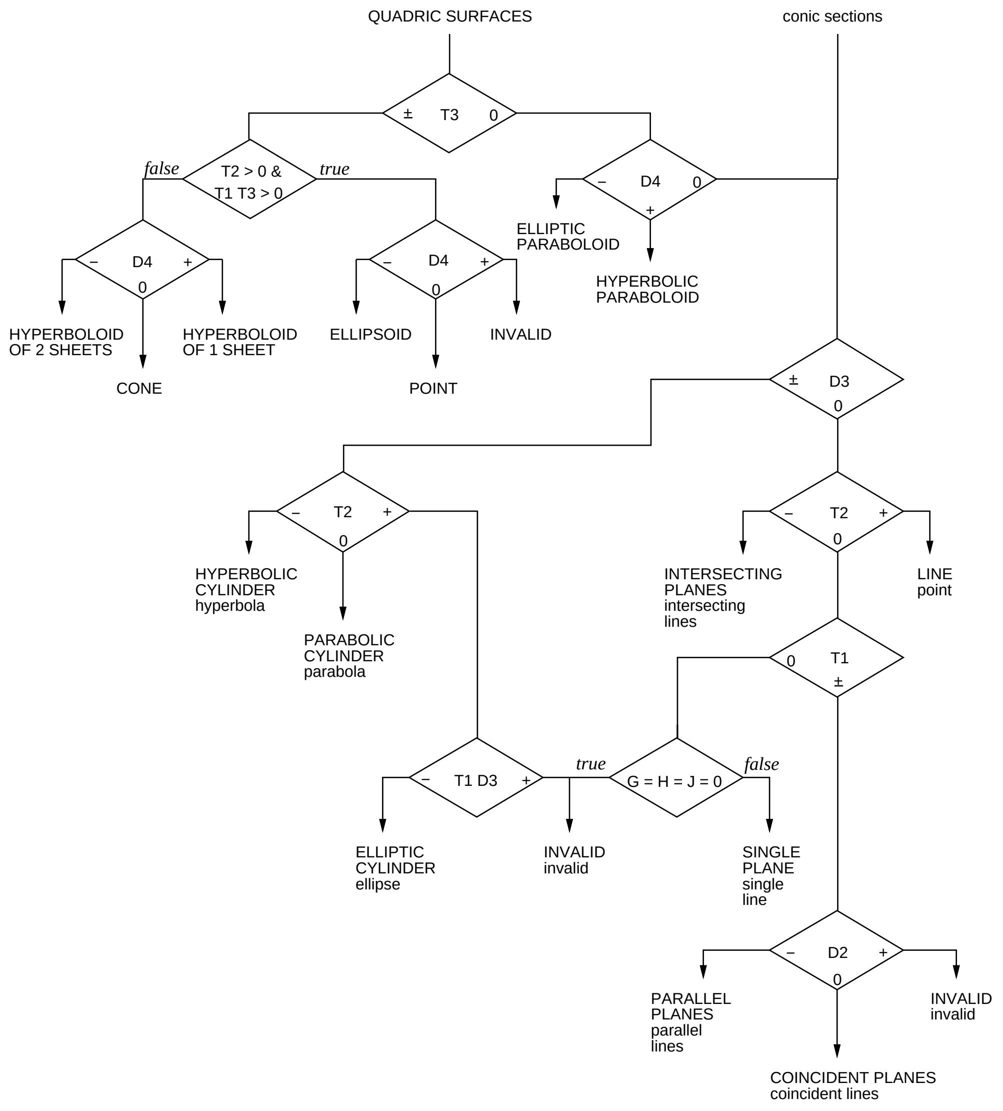


Figure 3: Levin's decision tree for classifying quadrics.

Type	*Point*	*Vectors*	*Scalars*
plane	on the plane	normal	none
sphere	center	none	radius
cylinder	on the axis	axis direction	radius
cone	vertex (tip)	axis direction	half-angle

Table 1: Geometric representations of natural quadrics.

2.4 Representation of Quadric Surfaces

Traditionally, there have two competing representation techniques for quadric surfaces – *algebraic* and *geometric* [8].

The algebraic approach represents the ten coefficients A, B, C, D, E, F, G, H, J, K of the quadratic implicit polynomial equation $F(x, y, z)$. The advantage of such this approach is that a common set of routines can be written for handling all types of general quadric surfaces. The major disadvantage is the lack of computational robustness. In this approach, certain critical decisions are based on floating point data of imperfect accuracy, e.g. the determination of the type of quadric surface (Fig. 3) depends critically on whether certain invariants are positive, negative, or zero. The algebraic model also, generally speaking, lacks internal consistency. For example, if a rigid transformation is applied to a cylinder to position it in space and then if an inverse of the same rigid transformation is applied, one would not necessarily get back the original cylinder due to round-off (or other) errors.

In the geometric approach, every quadric surface can be represented by:

(1 point, 2 orthogonal unit vectors, 3 scalars).

The point fixes the position of the surface, the vectors define its orientation or axes, and the scalars determine its dimensions. For example, an ellipsoid can be completely described by specifying its center (a point), two of its three orthogonal axes (two orthogonal unit vectors), and the three lengths (radii) along its three axes (three scalars). Table 1 gives a geometric description of the natural quadrics. It is worth noting that the main advantages of the geometric approach over the algebraic approach are its robustness and internal consistency. The disadvantage, however, is that a large number of routines are needed to handle all the special cases that arise as a result of each type of surface being treated as a separate entity (e.g. a problem of combinatorial explosion in the number of intersection routines).

2.5 Applications of Quadric Surfaces

As noted earlier, surface/surface or curve/surface intersection calculations are much more computationally (numerically as well as topologically) tractable as compared to parametric patches. For example, the intersection of a line L with a quadric surface $F(x, y, z)$ can be computed by substituting the parametric equation of the line $L:\ (x(t),\ y(t),\ z(t))$ in the implicit representation of the surface, to get $F(x(t),\ y(t),\ z(t))$, which reduces to a uni-variate quadratic $G(t)$. Roots of $G(t)$ can now be found using well known closed form solutions for a uni-variate quadratic function.

This simplicity has, for example, made it possible to architect and implement a quadric-based special purpose custom-VLSI highly parallel computer – the RayCasting Engine (RCE) – that efficiently processes complex geometries composed of quadric surfaces [6, 14]. With the use of recently developed quadratic algebraic patch based methods for meshing tiny subsets of quadric surfaces with smooth continuity, this technology has been seamlessly extended to support complex free-form shapes [11, 12, 13].

3 Blobby Methods

One way to think about blobby objects it to begin with physical ball-and-stick models for molecules. From physics, we know that electron clouds around each atom are not spherical, but rather are distorted by the electron clouds around other atoms. To generate iso-surfaces (those with identical electron densities), one would therefore have to consider the effects all neighboring atoms.

The computation of exact iso-surfaces is expensive, and several good approximations have been made. Blinn [3] used exponentially decaying (with distance) fields created by each atom, and defined the iso-surface as those points where a "density" function D equals some threshold amount T, i.e.

$$F(x, y, z) \;=\; D(x, y, z) \;-\; T. \tag{19}$$

The function D took the form

$$D(x, y, z) \;=\; \sum_i b_i \; e^{-a_i r_i^2}. \tag{20}$$

The exponential term is a simple Gaussian bump centered at r_i, with height b_i and standard deviation a_i. Different effects of "blobbiness" can be achieved for the same arrangement of atoms by adjusting the a_i and b_i parameters. Similar methods were also developed independently by Nishimura *et. al.* for use in the LINKS project[15].

Wyvill *et. al.* modify Blinn's method and create "soft objects" by distributing field sources in space and computing a field value at each point of space. The field value is a sum of field values contributed by each source and the value from each source is a function of distance only. This function $C(r)$ of distance r decays completely in a finite distance R, unlike Blinn's exponentially decay, with the properties [20]:

$$C(0) \;=\; 1, \;\; C(R) \;=\; 0, \;\; C'(0) \;=\; 0, \;\; C'(R) \;=\; 0, \;\; \text{and} \;\; C(\frac{R}{2}) \;=\; 0.5. \tag{21}$$

Thus the fields have finite extent, and smooth joints are obtained when the functions are blended together. They compute a number m, such that the volume of the set where $C(r) \;\geq\; m$ is exactly one-half the volume of the set where $2C(r) \;\geq\; m$. This property implies that the level-m iso-surface computed from two sources at the same location has twice the volume of the iso-surface for a single source. Thus when soft objects are merged, their volumes add. Furthermore, if two sources are far apart, the iso-surface may have two disconnected pieces.

Field sources could be points, or lines, or more complex geometric structures. Iso-surfaces are computed by algorithms that resemble the marching cubes techniques used frequently in computer graphics. A closely related method are "distance constrained" implicit models [4] that use skeleta (points, lines, curves, and such) and define surfaces in terms of distances to the skeleta. Skeletal design provides an intuitive and interactive specification for many forms found in a natural environment. Implicit techniques are particularly suited for relating the skeleton to the surface; they enable the smooth, seamless, bulge-free blend of components. Skeletal design also permits the smooth embedding of volume within a surface, given certain extensions to implicit techniques.

Implicit surfaces of the blobby kind are rendered typically after polygonization or via ray tracing; Fig. 4 provides two examples of complex shapes modeled using CSG combinations of implicit soft objects and other primitives.

(a)

(b)

Figure 4: Objects resulting from a CSG combination of implicit soft objects and other primitives.

4 Functional Methods

4.1 What is F-rep?

The *function representation* (or *F-rep*) defines a whole geometric object by a single real continuous function of several variables as $F(X) \geq 0$ [1]. F-rep is an attempt to step to a more general modeling scheme using real functions. Functions are not restricted very much - they only have to be at least C^0 continuous. The function can be defined by a formula or by an evaluation procedure. In this sense, F-rep combines many different models like classic implicits, skeleton based implicits, set-theoretic solids, sweeps, volumetric objects, parametric and procedural models [1].

4.2 Basic operations

- **Set-theoretic operations** are closed on this representation with the use of R-functions - C^k continuous definitions introduced by Rvachev [16] (see survey in [1]). The main restriction of well-known min/max operations is that they are C^1 discontinuous. This can yield unexpected results in further operations on the object.

  The simplest R-functions are

$$f_1 \& f_2 = f_1 + f_2 - \sqrt{f_1^2 + f_2^2} \tag{22}$$

  for the intersection of two objects described by f_1 and f_2, and

$$f_1 \& f_2 = f_1 + f_2 + \sqrt{f_1^2 + f_2^2} \tag{23}$$

  for the union. Note that these function have C^1 discontinuity only in the points where $f_1 = f_2 = 0$. There are C^k continuous R-functions as well.

- **Blending set-theoretic operations** generate smooth edges of constructive solids with added or subtracted material. These are proposed to be

$$B(f_1, f_2) = R(f_1, f_2) + d(f_1, f_2), \tag{24}$$

where $R(f_1, f_2)$ is a corresponding R-function and $d(f_1, f_2)$ is a Gaussian-like displacement function with three parameters controlling the overall shape.

- **Offsetting** generates a constructed or expanded version of an initial object. Pasko *et. al.* discuss in [1] its three different definitions: *iso-valued offsetting* defined as $f_1 + const$, *offsetting along the normal* defined as $f_1(X + N)$ (where N is a gradient vector) and *constant radius offsetting* defined as a maximal function value on a sphere of a given radius.

- **Cartesian product** is a dimension increasing operation. It is possible to generate a 3D solid as a Cartesian product of a 2D solid (plane area) and a line segment. It can be expressed in terms of R-functions as

$$f_3(x, y, z) = f_1(x, y) \& f_2(z) \tag{25}$$

where $f_1(x, y)$ describes a 2D solid and $f_2(z) = (z - z_1) \& (z_2 - z)$ describes a segment $[z_1, z_2]$ on z-axis.

- **Bijective mapping** serves for deformation of initial objects. The transformed object is described as $f_1(T^{-1}(X))$, where T^{-1} defines arbitrary inverse space mapping. Savchenko *et. al.* have developed a method to control the deformation by arbitrary control points linked to the features of the object. A volume spline is used to interpolate the displacements of the control points.

- **Metamorphosis** produces an intermediate shape between two given objects. With real functions it can be expressed as the weighted interpolation between two defining functions.

4.3 Advanced topics

- **Relations** *Inclusion* (binary) and *membership* (in/boundary/out) relations for a point and an object can be directly expressed in terms of the defining function. The *intersection* (*collision*) relation can be evaluated as a non-emptiness of the intersection of two objects.

- **Sweeping by a moving solid** is one of the long standing problems in solid modeling. The problem of a swept solid description has been reduced to a one-dimensional global extremum search by a parameter of movement. It allows the user to apply arbitrary variable-shape and CSG solids as generators, arbitrary parameterized movement and self-intersections.

- **Deformation with algebraic sums** Algebraic sums are used to control deformations by positions of arbitrary points (not their displacements):

$$f_3(x, y, z) = f_1(x, y, z) + disp(x, y, z), \tag{26}$$

where $disp$ is a displacement function and $disp(x_i, y_i, z_i) = -f_1(x_i, y_i, z_i)$ in every control point. Different interpolation techniques are used to ensure this property of the displacement function. This definition allows the new surface to pass through all given control points. Subtle local deformations, pinching, pricking and scratching effects can be handled in this way.

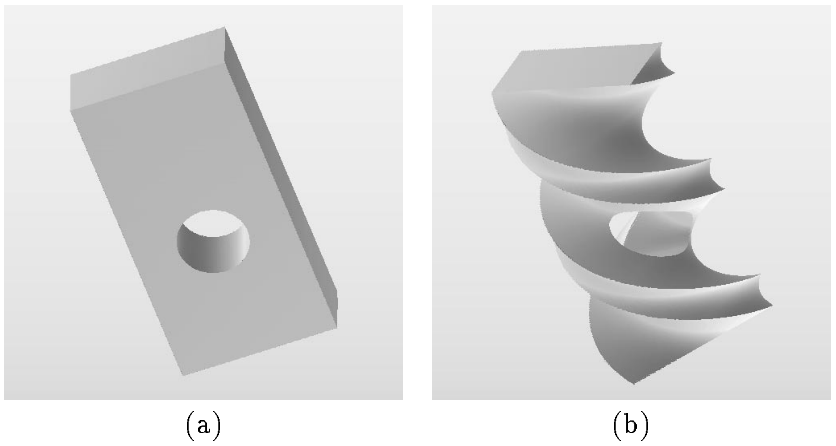


Figure 5: F-rep: An original set-theoretic solid and its twist with an inverse mapping.

- **Three-dimensional texture modeling** is based on the extensive use of the *solid noise* primitive defined by well-known *solid noise* functions. If a solid noise function is continuous, it defines some 3D primitive which can be an argument of all operations closed on F-rep. Thus, different 3D textures can modeled: moss, snow, fur or hair [10]. Different hairstyles have been modeled by procedurally defined real functions with the use of "solid noise", sweep-like technique, offsetting and set-theoretic operations, and non-linear space mappings.
- **Visualization** An algorithm of iso-surface polygonization is implemented for rendering [2]. It is free of topological ambiguities essential to the marching cubes algorithm. Trilinear interpolation is used for the hyperbolic arcs detection at faces of a cubic cell and for the construction of the edges connection graph. Its parallel version has been implemented on the workstations network with PVM system. Traditional ray-marching algorithms are also used to generate halftone images.
- **Interaction** A high-level modeling language is used to support so-called exploratory (or empirical) style of geometric modeling. The language provides full system extendibility by symbolic input of defining functions for new primitives and operations.

Figs. 5 – 7 give some examples of objects modeled with F-rep techniques.

5 Conclusions

This chapter provided a brief introduction to some of the main areas of work in implicit techniques for geometric modeling and computer graphics. These were presented in an order that is not only consistent with the chronology of development, but also with the mathematical complexity of the basic implicit equation $F(x, y, z)$, i.e. we progressed from algebraic to blobby to functional approaches.

An interesting twist is the combination of generalized interval arithmetic and implicit surface rendering techniques to "plot" 2D graphs, resulting in images such the one in Fig. 8 [19]. Implicit methods continue to stir interest in the research community, and could one day become as popular as (if not supersede) parametric patch methods used so rampantly in contemporary modeling and graphics systems.

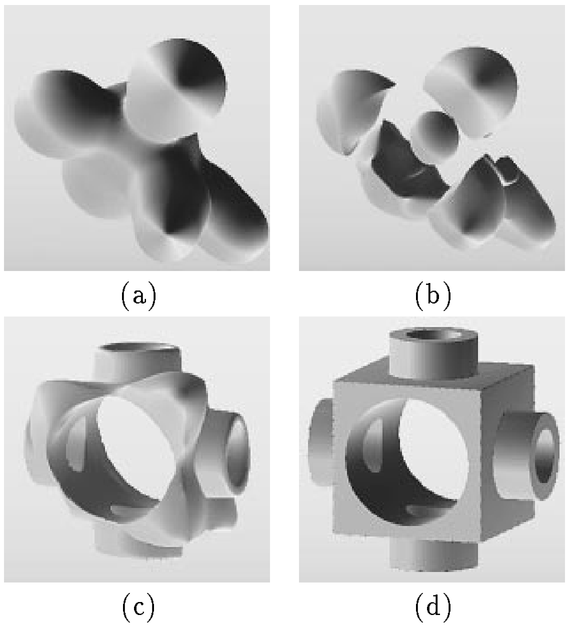


Figure 6: F-rep: Frames of metamorphosis.

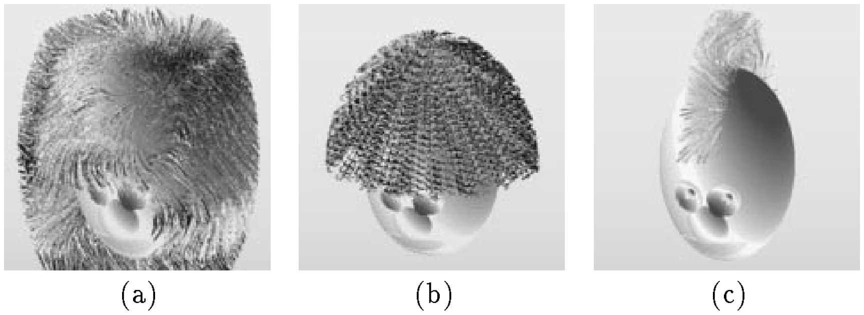


Figure 7: F-rep: Hairstyles modeled with solid noise function, sweep-like technique, set-theoretic operations and non-linear mappings.

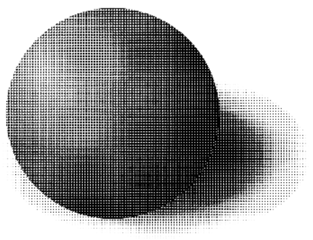


Figure 8: Combining interval arithmetic and implicit rendering techniques.

6 Acknowledgements

The author would like to thank Herb Voelcker and Gene Hartquist of Cornell University for introducing him to the theoretical and practical aspects of quadratic surfaces. Many thanks to Geoff Wyvill (University of Otago, New Zealand) and Brian Wyvill (University of Calgary, Canada) for providing the images for the soft objects. The author also thanks Alexander Pasko (University of Aizu, Japan) and his collaborators – V. Adzhiev (Moscow Engineering Physics University, Russia), V. Savchenko (University of Aizu, Japan), and A. Sourin (Nanyang Technological University, Singapore) – for providing a description of their work on F-reps.

References

[1] Pasko A., Adzhiev V., Sourin A., and Savchenko V. Function Representation in Geometric Modeling: Concepts, Implementation and Applications. *The Visual Computer*, 11(8):429–446, 1995.

[2] Pasko A., Pilyugin V.V., and Pokrovskiy V.V. Geometric Modeling in the Analysis of Trivariate Functions. *Computers and Graphics*, 12(3/4):457–465, 1988.

[3] J.F. Blinn. A Generalization of Algebraic Surface Drawing. *ACM Transactions on Graphics*, 1(3):235–256, 1982.

[4] J. Bloomenthal. Poligonisation of Implicit Surfaces. *Computer Aided Geometric Design*, 5:341–355, 1988.

[5] A. Dresden. *Solid Analtic Geometry and Determinants.* John Wiley and Sons INc., New York, 1930.

[6] J.L. Ellis, G. Kedem, T.C. Lyerly, D.G. Thielman, R.J. Marisa, J.P. Menon, and H.B. Voelcker. The RayCasting Engine and Ray Representation: A Technical Summary. *International Journal of Computational Geometry and Applications*, 4(2):347–380, December 1991.

[7] J. D. Foley, A. van Dam, S. K. Feiner, and J. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley, Reading, Mass., 1992.

[8] R. N. Goldman. Two Approaches to a Computer Model for Quadric Surfaces. *IEEE Computer Graphics and Applications*, pages 21–24, 1983.

[9] B. Guo and J.P. Menon. Local Shape Control for Free-Form Solids in Exact CSG Representation. *Computer Aided Design*, to appear, 1996. (also available as IBM Research Report 20051, IBM T.J. Watson Research Center, 1995.).

[10] J. Levin. Mathematical Models for Determining the Intersections of Quadric Surfaces. *Computer Graphics and Image Processing*, pages 73–87, 1979.

[11] J.P. Menon. Constructive Shell Representations for Free-form Surfaces and Solids. *IEEE Computer Graphics & Applications*, 14(2):24–36, March 1994.

[12] J.P. Menon and B. Guo. Free-form Modeling with Low Degree Algebraic Patches in Bilateral Brep and CSG Schemes. *IEEE Transactions on Visualization and Computer Graphics (submitted)*, 1995. (available as IBM Research Report RC 20050, IBM T.J. Watson Research Center, 1995.).

[13] J.P. Menon and B. Guo. A Framework for Sculptured Solids in Exact CSG Representation. In *CSG96: Set-theoretic Solid Modeling Techniques and Applications*, pages 141–157, Winchester, U.K., April 17-19 1996. Information Geometers Publishers.

[14] J.P. Menon, R.J. Marisa, and J. Zagajac. More Powerful Solid Modeling Through Ray Representations. *IEEE Computer Graphics & Applications*, 14(3):22–35, May 1994.

[15] H. Nishimura, H. Ohno, T. Kawata, I. Shirakawa, and K. Omura. LINKS-1: A Parallel Pipelined Multimicrocomputer Syetem for Image Creation. In *Tenth International Symposium on Computer Architecture, ACM SIGARCH Newsletter*, pages 387–394, 1983.

[16] V. L. Rvachev. *Methods of Logic Algebra in Mathematical Physics.* Naukova Dumka Publishers, Kiev, Ukraine (in Russian), 1974.

[17] T.W. Sederberg. Techniques for Cubic Algebraic Surfaces: Tutorial Part Two. *IEEE Computer Graphics and Applications*, 10(5):12–21, September 1990.

[18] J. G. Semple and L. Roth. *Introduction to Algebraic Geometry.* Clarendon Press, Oxford, UK, 1949.

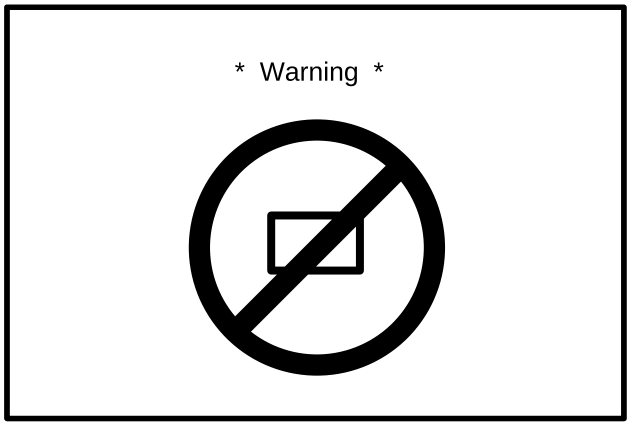
[19] J. A. Tupper. Graphing Equations with Generalized Interval Arithmetic. In *MS Thesis, Computer Science, University of Toronto*, 1996.

[20] G. Wyvill, C. McPheeters, and B. Wyvill. Data Structure for Soft Objects. *The Visual Computer*, 2(4):227–234, April 1986.

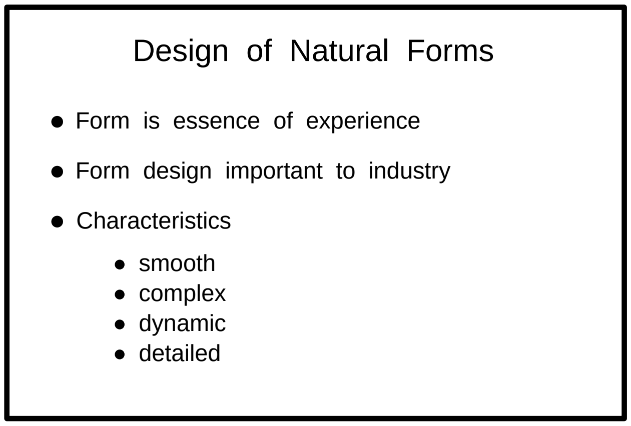
Intuitive Implicit Skeletal Design

Jules Bloomenthal

In this course notes chapter we relate several forms observed in nature to methods of representation using an inner structure, or skeleton. We develop techniques to define both manifold and non-manifold implicit surfaces skeletally. Along the way we demonstrate the techniques with example surfaces that are smooth.

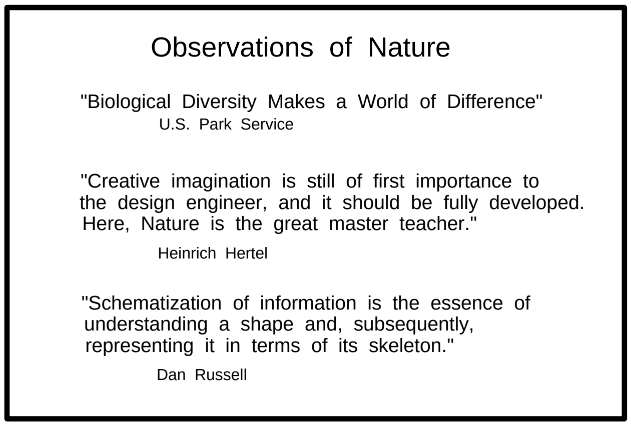


* Warning *



Design of Natural Forms

- Form is essence of experience
- Form design important to industry
- Characteristics
  - smooth
  - complex
  - dynamic
  - detailed

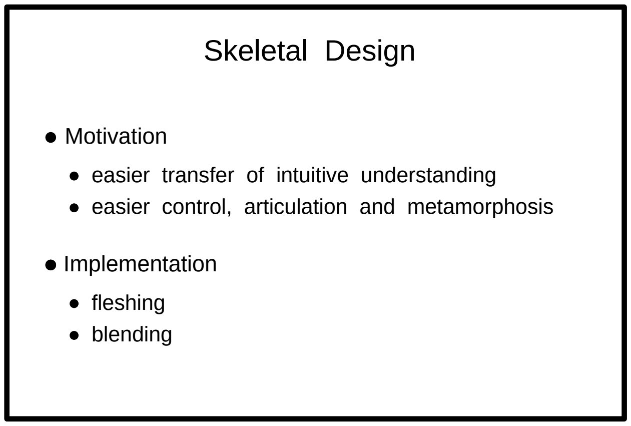


Observations of Nature

"Biological Diversity Makes a World of Difference"
U.S. Park Service

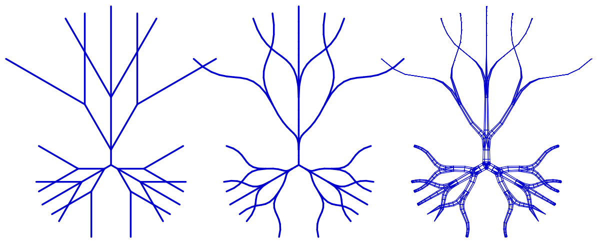
"Creative imagination is still of first importance to the design engineer, and it should be fully developed. Here, Nature is the great master teacher."
Heinrich Hertel

"Schematization of information is the essence of understanding a shape and, subsequently, representing it in terms of its skeleton."
Dan Russell



Skeletal Design

- Motivation
  - easier transfer of intuitive understanding
  - easier control, articulation and metamorphosis
- Implementation
  - fleshing
  - blending



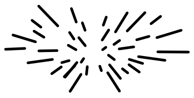
Skeletal Progression: Straight, Curved, and Tapered

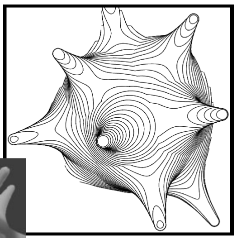
Skeletal Design :: Implicit Modeling

- Skeletons imply volume, $f(x, y, z) < 0$
- Implicit modeling indifferent to topology
- Reasonable to tile (non-adaptive)
- Difficulties
  - Details
  - Mixed Dimensions
  - Control
  - Texture

Direct Imaging

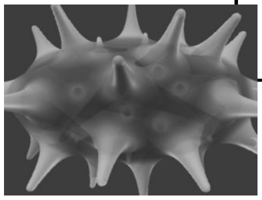
Skeleton



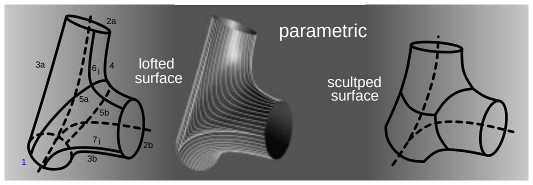


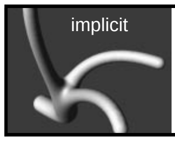
Contour
Line Drawing

Ray-
Tracing

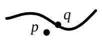


Ramiform Definitions

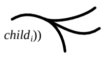




$$f_{limb}\,(\boldsymbol{p},\, limb) = \frac{\|\boldsymbol{p}-\boldsymbol{q}\|^2}{r_{limb}{}^2}$$

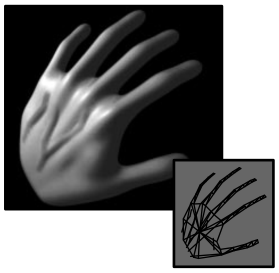


$$f(\boldsymbol{p}) = \; max\; (f_{limb}\,(\boldsymbol{p},\, parent), \sum_{i}^{n} f_{limb}\,(\boldsymbol{p},\, child_i))$$

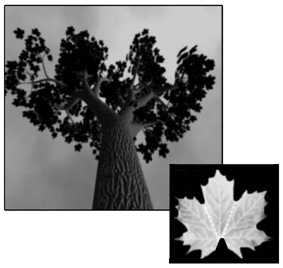


Motivation

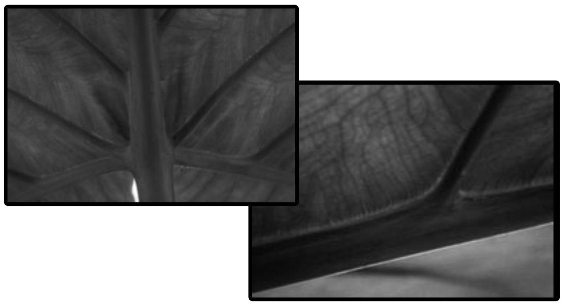
many natural forms are smooth, and



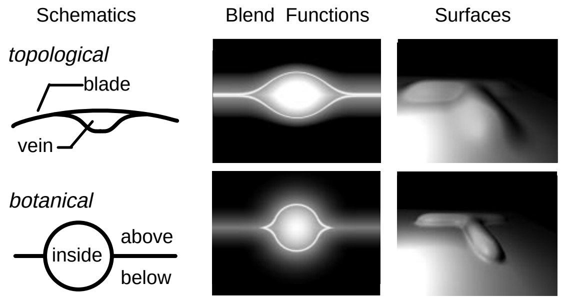
some natural forms are flat, but . . .



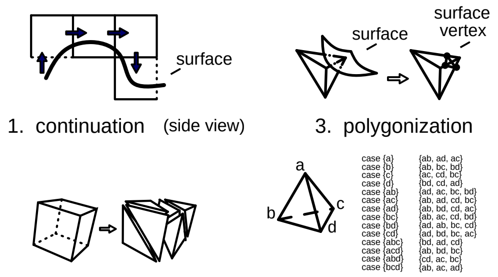
'Flat' Leaves are Not



Leaf Cross-sections



Polygonization

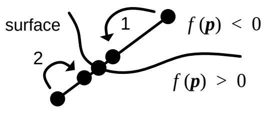


case {a}	{ab, ad, ac}
case {b}	{ab, bc, bd}
case {c}	{ac, cd, bc}
case {d}	{bd, cd, ad}
case {ab}	{ad, ac, bc, bd}
case {ac}	{ab, ad, cd, bc}
case {ad}	{ab, bd, cd, ac}
case {bc}	{ab, ac, cd, bd}
case {bd}	{ad, ab, bc, cd}
case {cd}	{ad, bd, bc, ac}
case {abc}	{bd, ad, cd}
case {acd}	{ab, bd, bc}
case {abd}	{cd, ac, bc}
case {bcd}	{ab, ac, ad}

tetrahedron table

Visualization: *Binary Sectioning*

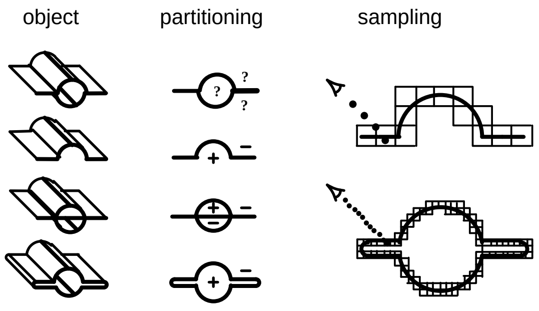
applicable if surface
separates inside
from outside



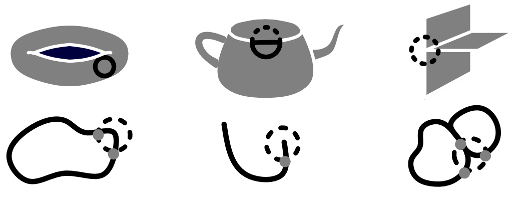
Implicit Surface Theorem

if *f* is continuous and
changes sign whenever zero,
then the surface defined by *f* is a manifold

Possible Implicit ($f(\boldsymbol{p}) = 0$) Definitions

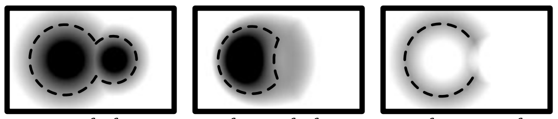


Terminology



Non-Degenerate and Degenerate Volumes

two circles: $f_1 = \frac{\|\boldsymbol{p}-\boldsymbol{c}_1\|}{r_1} - 1$ and $f_2 = \frac{\|\boldsymbol{p}-\boldsymbol{c}_2\|}{r_2} - 1$

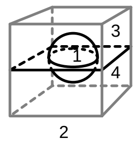


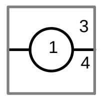
- (union)
- inside/outside
- manifold

- (subtraction)
- inside/outside
- manifold

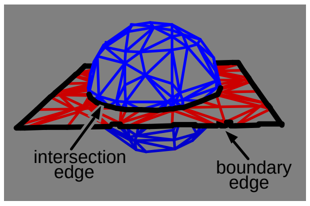
- *f* non-negative
- no inside/outside
- manifold w/ bound.
- no binary sectioning

Non-Manifold Implicit Definition and Surface





side view



Requirements for Non-Manifold Implicit Surfaces

Implicit Definition

- Must specify non-manifold and manifold with boundary
- Must accommodate binary sectioning

Polygonizer

- Must accommodate above definition

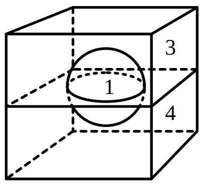
Complications

- need for *face vertex*
- need for *multiple edge vertices*
- need for *inner vertex*
- need for *disjoint surfaces*

Summary

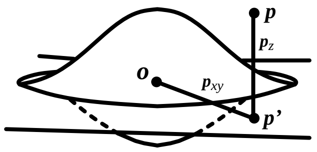
- Parametric surfaces good for tangential and positional control
- Represent sheets except where in volume
- Integer-valued implicit surface function
- Surface defined as separation between arbitrary regions
- Surface defined as separation between arbitrary regions

Surface / Volume Blend

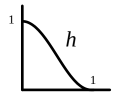


$region = 1$ if $\| \boldsymbol{p} \| < r$
or, *e.g.*, $h(\| \boldsymbol{p} \|) > c$

hard edge

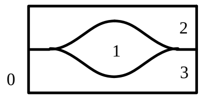


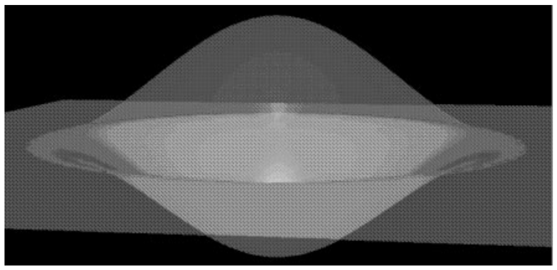
$region = 1$
if $h(\| \boldsymbol{p}_{xy} \|) > \boldsymbol{p}_z$
(h is cubic poly.)



blend

Saucer

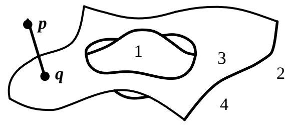




Blend to Bicubic Patch

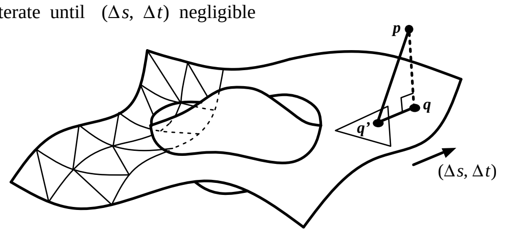
given ***q*** on *patch* nearest ***p***

1 if *h* (|| ***p*** ||) > || ***p* - *q*** ||
2 if ***q*** on *patch boundary*
3 or 4 determined by *surface normal* at ***q***

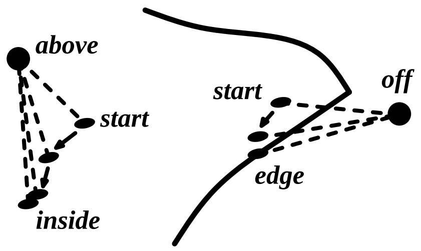


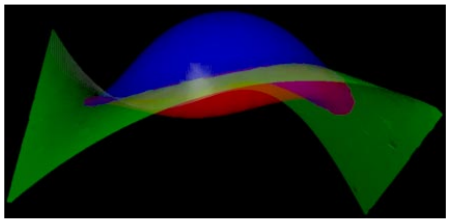
Distance to Patch

determine ***q'***, point on mesh nearest ***p***
calculate (s, t) from barycentric coordinates of ***q'***
compute (Δs, Δt) from projection of ***q'* - *p***
iterate until (Δs, Δt) negligible



Closest Point on Patch





References

Much of the above material may be found in my own works, the most relevant of which are listed below. In these works are found numerous references to related material.

A detailed review of skeletal methods:

Skeletal Design of Natural Forms by J. Bloomenthal
Ph.D. Dissertation, University of Calgary, 1995.

Implementation details for implicit surface polygonizers (manifold and non-manifold):

Polygonization of Non-Manifold Implicit Surfaces
by Jules Bloomenthal and Keith Ferguson
SIGGRAPH'95. In Computer Graphics 29, 4.

An Implicit Surface Polygonizer, by J. Bloomenthal
in Graphics Gems IV (Paul Heckbert, editor)
Academic Press, New York, 1994.

An Evaluation of Implicit Surface Tilers
by Paul Ning and Jules Bloomenthal
Computer Graphics and Applications, Nov., 1993.

Polygonization of Implicit Surfaces
by Jules Bloomenthal
Computer Aided Geometric Design 5, 4, Nov., 1988.

Detailed discussions of implicit surface blends:

Convolution Surfaces
by Jules Bloomenthal and Ken Shoemake
Proc. SIGGRAPH'91. In Computer Graphics 25, 4.

Bulge Elimination in Implicit Surface Blends
by Jules Bloomenthal
Computer Graphics Forum (to appear, 1996).

Some useful techniques for spline based skeletons:

Calculation of Reference Frames along a Space Curve
by Jules Bloomenthal
in Graphics Gems (Andrew Glassner, editor)
Academic Press, New York, 1990.

An early example (the tree trunk) of implicit and skeletal techniques:

Modeling the Mighty Maple
by Jules Bloomenthal
Proc. of SIGGRAPH'85. In Computer Graphics 19, 3.

Representation Schemes and Impact on Algorithms

Jai Menon
IBM T.J. Watson Research Center

Abstract

This chapter develops formal concepts of complete representation schemes, and explains the two – Constructive Solid Geometry (CSG) and Boundary representation (Brep) – that are used most frequently. Auxiliary representations, computed typically from complete representations, are used to support a variety of applications. The chapter explains the theory of two closely related auxiliary representations – z-buffers and ray-reps – that are used frequently in computer graphics and geometric modeling applications respectively. A case study of a particular parallel hardware architecture – the RayCasting Engine (RCE) – is presented as a contrast to contemporary polygon-pushing based 3D graphics hardware; the RCE processes CSG of implicit quadratics to produce ray-reps, while polygon-pushers process (loosely) Breps of linear polyhedra to produce z-buffers. The chapter concludes with a plethora of applications supported with ray-reps, and two specific case studies – one for Numerical Control (NC) machining and another for molecular modeling in chemistry.

1 Introduction

Surfaces defined using implicit or parametric techniques are combined typically in some manner to represent more complex shapes. When these shapes correspond to *solid* objects, the representation schemes must be rich enough to capture the physical properties of rigidity, boundedness and so forth. The field of solid modeling refers to the theory, techniques and systems that provide *unambiguous* or informationally *complete* representations of solids. A complete representation is one that permits any well defined geometric property to be computed automatically (or at least in principle) [16].

Complete representations are processed subsequently to compute *auxiliary* representations, which in turn support target applications; see Fig. 1. For example, the two most popular complete representation schemes are: Boundary representation *(Brep)* and Constructive Solid Geometry *(CSG)*. Graphics rendering algorithms, typically z-buffer based display hardware, in essence process Breps of polyhedra (flat-faced objects), and compute an auxiliary representation – the z-buffer – which represents the first intercept along every ray from the eye into the scene. Hidden surface (occlusion) calculations in turn are based on simple z sorting.

In this chapter, we will study:

- *what are the key complete representation schemes?*
  they are Brep and CSG (Section 2).
- *what are the close counterparts of the z-buffer auxiliary representation in geometric modeling?*
  they are ray-reps (Sections 3, 4).
- *how do complete representations influence processing hardware?*
  we will examine a CSG-based parallel processing engine, called RCE, that processes implicit quadratic algebraics to compute ray-reps (Sections 5, 6).

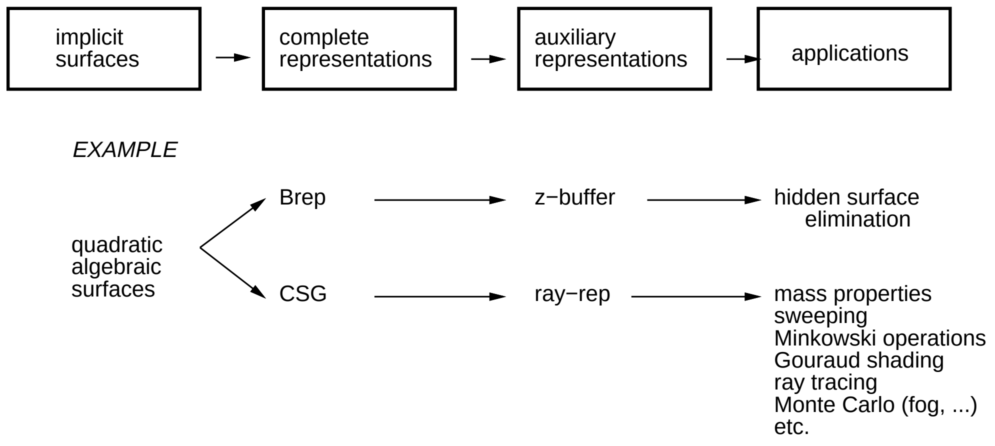


Figure 1: Implicit surfaces are embedded in a higher logic of complete representation schemes. Complete representations are often converted to auxiliary representations, which in turn support a wide variety of applications.

- *what are the applications of ray-reps?*
  we will see how ray-reps are versatile for a whole range of applications in geometric modeling and computer graphics (Sections 7, 8).

2 Complete Representations

The following progression serves as a useful framework for understanding how solids are represented in the computer, and how computational algorithms are implemented to achieve target applications [17, 23].

$$\hat{P} \mapsto \hat{M} \mapsto \hat{R} \mapsto \hat{A} \tag{1}$$

Here $\hat{P}$ is the physical space of real-world objects, $\hat{M}$ is the space of mathematical models, $\hat{R}$ is the space of complete representations, and $\hat{A}$ is the space of computational algorithms. Thus math models are representation-free (existential) elements whose mathematical properties capture important physical properties of solids objects in $\hat{P}$. Complete representations are symbol structures, e.g. Brep or CSG, that designate elements of $\hat{M}$. Every complete representation in $\hat{R}$ unambiguously corresponds to one and only one element in $\hat{M}$, although an element in $\hat{M}$ can have several non-identical representations in $\hat{R}$. Finally, computational algorithms in $\hat{A}$ depend on the particular representation scheme in $\hat{R}$.

For example, if A is a block, then A can be represented as:

- $\mathrm{CSG}_1(A)$, which is the intersection of six linear halfspaces, or
- $\mathrm{CSG}_2(A)$, which is the intersection of two larger blocks, or
- $\mathrm{Brep}(A)$, which is the collection of six square faces.

This example shows that A could not only have two different kinds of representations (Brep and CSG), but also have non-identical representations within a single scheme (CSG). The following sections will provide a brief overview of $\hat{M}$, $\hat{R}$, and $\hat{A}$.

physical	*mathematical*
entity	set of points in E^n
rigidity	shape invariance (fixed distances and signed angles between all pairs and triples of points)
solidity	homogeneous interior
boundedness	finite size
Boolean closure	regularized operations

Table 1: Conditions established from point-set topology.

2.1 Math Models $\hat{M}$

The elements of $\hat{M}$ are sets of points (often called point sets) in n-dimensional Euclidean space (E^n). To model rigid solids, the following mathematical criteria restrict the admissible point sets to those that capture the physical essence of a solid [16, 23].

The conditions in Table 1 are established from the theory of general point-set topology, and result in restricting $\hat{M}$ to the set of *compact regular* sets, with associated operations consisting of rigid motions (translation, rotation) and regularized set union, intersection and difference. Compact regular sets are bounded, closed and regular. Closed regular sets satisfy:

$$X = kiX = r_n X \tag{2}$$

where X is regular in E^n, i denotes interior, and k is a topological closure operator which, in essence, adds limit points to iX. The notation r_n denotes a regularization operator denoting the ordered application of i then k in a topological space of dimension n. Intuitively, a regular set has a homogeneous "solid" interior covered with a tight bounding "skin". The operator r_n regularizes a non-regular set by trimming subsets that have no interiors in the usual topology of E^n ("dangling" points, edges or faces), and then supplying a boundary for the open set that remains.

Regular sets are not algebraically closed under conventional set operations ($\cap, \cup, -$). To guarantee algebraic closure, we use *regularized Boolean* operators $\cap_n, \cup_n$ and $-_n$, where

$$A\ op_n\ B \ = \ r_n(A\ op\ B), op \ \in \ \{\cap, \cup, -\}. \tag{3}$$

Two more properties in Table 2, drawn from combinatorial topology, adds to those in Table 1 to complete the mathematical modeling space $\hat{M}$ for solids. However, compact regular sets do not meet these new criteria, and smaller classes – *semi-analytic* compact regular sets – are needed. This class of semi-analytic compact regular sets is dubbed *r-sets*, and serves as the most widely used mathematical model for solids. Semi-analytic (or semi-algebraic) sets can be thought of as Boolean compositions of halfspaces whose boundaries are defined by analytic or algebraic functions, and thus do not vary "wildly". An algebraic halfspace is the set $\{(x, y, z) \mid f(x, y, z,) \leq 0\}$, where f is a polynomial function.

R-sets are generally accepted as suitable math models for solids, although some prefer the more restrictive sub-class of r-sets that are *manifolds*. Each edge of a manifold solid is shared precisely by two faces, and each vertex has precisely one "cone" of adjacent faces. Manifold models are a restricted class of r-sets that is *not closed* under regularized or conventional set operations.

physical	*mathematical*
finite describability	finite triangulability, in the sense of simplicial complexes
boundary determinism	a set X in E^n is determined uniquely by ∂X, its boundary

Table 2: Conditions established from combinatorial topology.

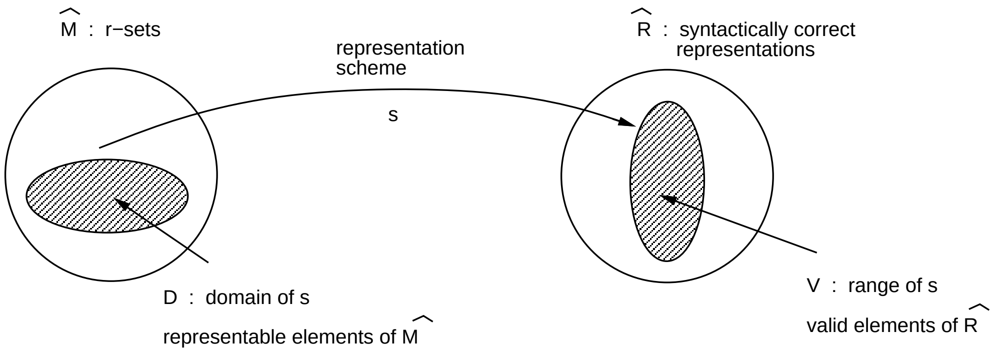

$\widehat{M}$: r–sets
representation scheme
s
$\widehat{R}$: syntactically correct representations
D : domain of s
representable elements of $\widehat{M}$
V : range of s
valid elements of $\widehat{R}$

Figure 2: Math models to complete representations

In summary, it is worth observing that point-set topology and combinatorial (or algebraic) topology provide complementary perspectives and tools for solid modeling [23]. Regularity is defined naturally in either perspective, and is consistent across both. In practice, combinatorial notions are useful in dealing with Breps and intrinsically topological properties such as connectedness and number of connected components. The local, neighborhood view provided by point-set topology is useful for establishing properties for closed regular sets, and in designing CSG algorithms.

2.2 Representation Schemes $\hat{R}$

The mapping from $\hat{M}$ to $\hat{R}$ is illustrated in Fig. 2, and the formal definable properties for representations are summarized in Table 3 [23]. These properties are discussed in details in [16, 17], together with properties of consistency and equivalence for multiple representations and such informal properties as conciseness and efficiency.

Completeness is the most important property, and six known schemes [17] for unambiguously representing solids have been developed, and a seventh one (Sreps) [14] has recently begun to evolve.

- *Primitive Instancing:* a solid is represented as a particular member of a family (e.g. bolt with hexagonal head) of parts, by specifying values for certain family-specific parameters (e.g. diameter and height).

- *Sweeping:* a solid is represented as the spatial region swept out by a *generator* (e.g. cutter) as it moves on a spatial trajectory, called the *director* (e.g. cutter feed path).

property	*description*
domain	D is the set of entities describable in scheme s
validity	elements $r \in V$ are valid representations, i.e. image of elements of D; other elements, while syntactically correct, represent invalid solids
uniqueness	a scheme s is unique if s is a function, i.e. each $m \in D$ has but a single corresponding $r \in V$
completeness	a scheme is complete or unambiguous if s^{-1} is a function, because each $r \in V$ then associates with a single $m \in D$

Table 3: Properties of representations.

- *Spatial Enumeration:* a solid is usually approximated as a union of quasi-disjoint box-shaped cells of uniform (e.g. voxels) or varying (e.g. derived from recursive space partitioning in quadtrees and octrees) sizes.

- *Cell Decomposition:* a solid is again represented as a union of quasi-disjoint cells, this time each cell has a distinct shape (e.g. triangulations or finite-element meshes).

- *Boundary Representation (Brep):* a solid is represented by the set of bounding faces that enclose the solid.

- *Constructive Solid Geometry (CSG):* a solid is represented as a binary tree whose nodes are regularized Boolean operators ($\cap_n, \cup_n, -_n$) and leaves are simple primitives (blocks, cylinders, ..., or just halfspaces).

- *Sampling Representation (Srep):* a solid is represented by a set of spatial samples (e.g. line segments) augmented with some symbolic data (e.g. halfspace equations at endpoints).

Of these, the Brep and CSG schemes, illustrated in Fig. 3, are the most popular ones, and used most frequently. In the 1970s, most modeling system architectures fell into one of two categories: (1) *single* representation, using either Brep or CSG as its primary scheme, or (2) *dual* representation using both [Brep,CSG], supporting CSG-to-Brep conversion. These evolved into multi-representation systems, using auxiliary representations (e.g. planar approximations, or approximate spatial enumerations) to support a variety of applications. As these systems evolved into the 1990s, they support editing in one or two primary schemes, usually Brep or CSG. The key issue continues to be the development of *representation conversion* modules that maintain consistent Brep and CSG schemes, in particular when the CSG representation is maintained as a Boolean composition of halfspaces.

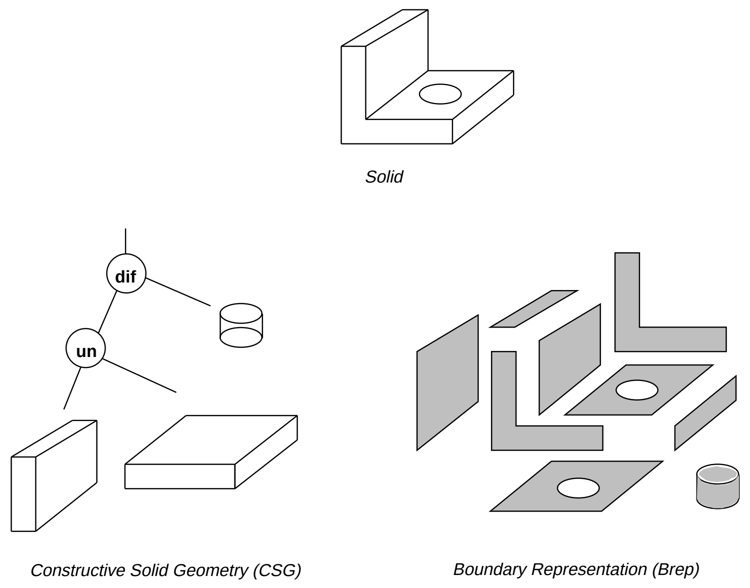

Solid
dif
un
Constructive Solid Geometry (CSG)
Boundary Representation (Brep)

Figure 3: CSG and Brep schemes.

2.3 Algorithms $\hat{A}$

The algorithms that implement target application functions depend very closely on the representation scheme. Consider the problem of classifying a line l against a solid A to determine the portion of l that lies inside A, denoted $l\text{in}A$, and that lies outside A, denoted $l\text{out}A$. Refer to Fig. 4. Below, we sketch the algorithm for CSG(A) and Brep(A) respectively.

CSG's natural divide-and-conquer recursive classification algorithm [21] first computes classification results with respect to the leaves of the CSG tree. Then the algorithm ascends the tree by combining classification results of the subtrees, based on the Boolean operation at every node. This is illustrated through a simple two-leaf CSG tree in Fig. 4b, where line/primitive classification yields $l\text{in}B$ and $l\text{in}C$ for the two leaves B and C. The combination of these results $l\text{in}A$ is merely their set union, since the node is a union operator.

The basic procedure for classifying the line with respect to the Brep is to compute the intersection of the line with every face in Brep(A), sort the list of intersection points padded with $\pm\infty$, and infer the classification from the sorted list as alternating "out" and "in" segments. For example, in Fig. 4c, the intersection points are p_1 and p_2; the "in" and "out" segments follow trivially after sorting.

This geometric query, termed *line membership classification* (LMC), is a specific instance of *set* membership classification (SMC) [21], denoted as $\mathcal{M}$. The classification function

$$\mathcal{M}(X, S) = (X\text{in}S, X\text{out}S, X\text{on}S) \tag{4}$$

partitions a candidate set X with respect to a reference set S into subsets that lie entirely inside $X\text{in}S$, outside $X\text{out}S$, or on $X\text{on}S$ the reference set.

To be precise, X, the candidate set, is closed and regular in a topology W' of dimension m. The reference set S is closed and regular in a topology W of dimension n, where $n \geq m$ and $W \supset W'$. The classification results are then defined mathematically as:

$$X \text{ in } S \quad = \quad r_m(X \cap iS) \tag{5}$$

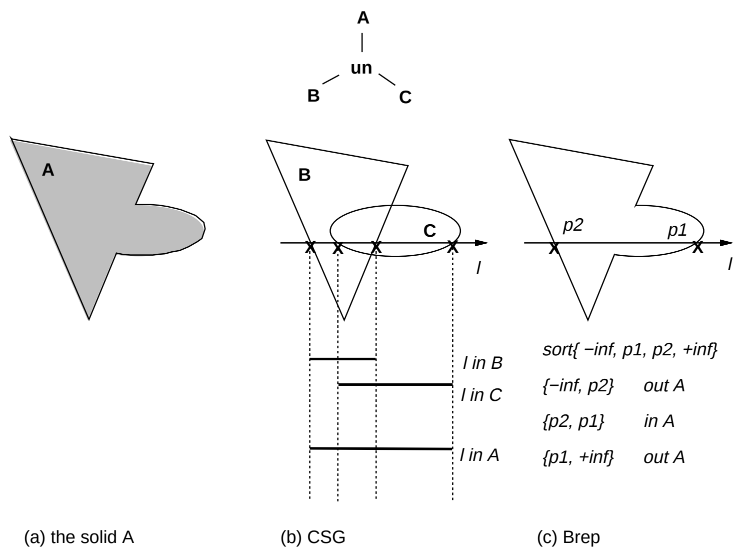


Figure 4: A 2D example of line/solid classification for CSG and Brep.

$$X \text{ on } S = r_m(X \cap \partial S) \tag{6}$$

$$X \text{ out } S = r_m(X \cap cS) \tag{7}$$

where ∂S denotes the boundary of S, and c denotes the complement of S. Thus, when X is a point, the geometric query is called *point membership classification* (PMC), and when X is a line, we get LMC.

The "on" cases, i.e. $X \text{on} S$, are quite tricky. For example, consider the problem of PMC of a point p with respect to a CSG tree $A \cap_2 B$ shown in the two cases in Fig. 5. While p is clearly "out" of the solid for case 1, it is "on" for case 2. However, the results of primitive classification tells us that p is "on" A as well as "on" B. Combining the results of these "on/on" results is therefore not possible by just looking at the classification results with respect to the primitives. In fact, we need additional information about points in the immediate vicinity of the point being classified. This leads to the definition of a regular *neighborhood* of a point p with respect to a set S as

$$N(p, S; r) = B(p; r) \cap_n S \tag{8}$$

where $B(p; r)$ is an open ball of radius r centered at p (see Fig. 5b). Thus for case 1 in Fig. 5a, the regularized intersection of the two neighborhoods is null, thus classifying p as "out"; similarly in case 2, the two neighborhoods overlap, their regularized intersection is a half closed ball, and p is "on" the resulting solid.

Although general primitive classifications, e.g. point/primitive or line/primitive classifications, cannot be combined in a divide-and-conquer manner, their neighborhoods can. This retains the divide-and-conquer character of SMC for CSG.

In this section we have studied SMC as one class of geometric queries on complete representations, and have seen how the algorithm for implementing SMC varies from one representation to another. While there are a wide range of geometric queries that can (in principle) be answered automatically from a complete representation, the general principle is that: *algorithms are designed on a representation-specific basis.* Furthermore, SMC is also perhaps the most

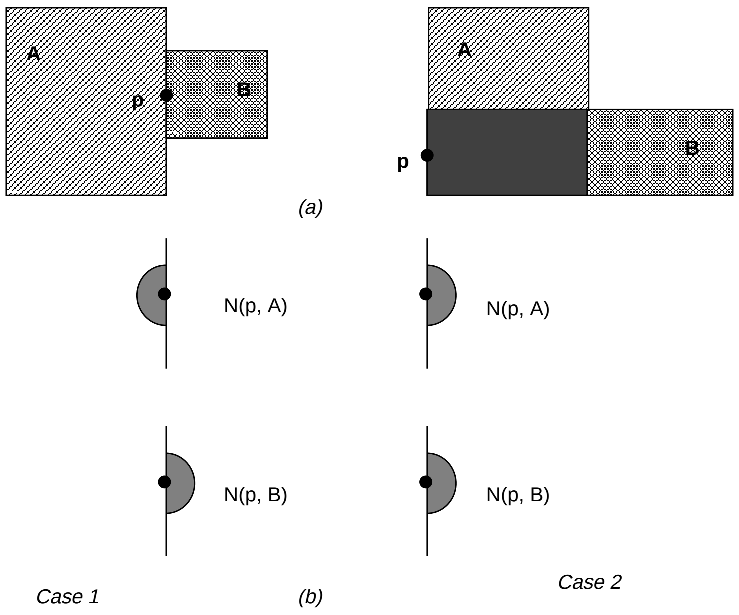


Figure 5: On/on ambiguities for point p with respect to the solid $A \cap_2 B$, and their resolution through neighborhoods.

frequently employed geometric query in modeling – we will illustrate this through the topic of auxiliary representations and applications thereof in the following sections.

3 Auxiliary Representations

Geometric algorithms are run on complete representations to often compute intermediate *auxiliary* representations that in turn support target applications. For instance, consider a grid G of parallel rays (lines) shown in Fig. 6a. Now consider the set of "in" segments resulting from LMC of every ray of G with respect to solid A, shown in Fig. 6b.

Rendering typically requires the computation of the first z intercept along the ray, cast from the eye into the scene, for every pixel in the scene. Most z-buffer based graphics hardware systems essentially implement this technique on polyhedral models. Thus the z-buffer would be nothing but the set of first intercepts of $l \text{in} A$ for every l in G. The z-buffer serves as an auxiliary representation to do hidden surface elimination on a pixel-by-pixel basis, and obviates expensive polygon sorting procedures.

If however, all the z-intercepts are retained, we get what is known as a *ray-representation*, or simply *ray-rep* for brevity [10]. Ray-reps serve as auxiliary representations for geometric modeling, because they not only support graphics rendering, but also prove extremely versatile for a wide range of other applications, as explored in the following sections.

4 Ray-reps

The result of classifying a ray grid G (a rectangular array of finitely spaced parallel lines) against a solid A, $\mathcal{M}(G, A)$ is a set of "in", "on", and "out" segments of regularly spaced parallel lines.

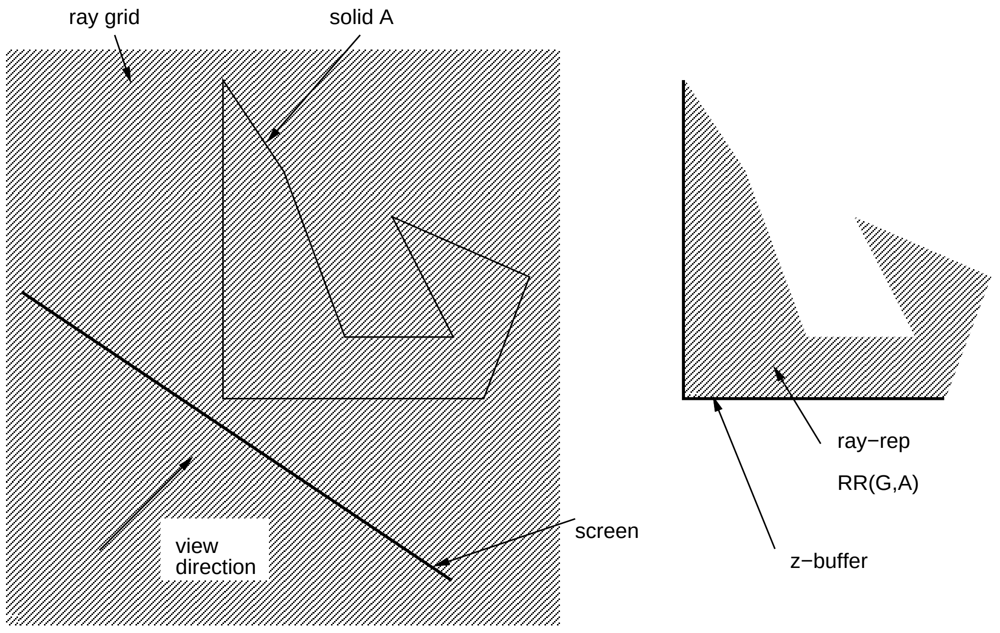


Figure 6: A 2D ray-rep.

The set consisting on $G\text{in}A$ and $G\text{on}A$ segments, whose endpoints lie in the boundary of the solid (∂A), is the ray-rep of A, denoted $\text{RR}(G, A)$.

As we noted in the above section, "on" segment processing requires additional neighborhood information, and so it is desirable that "on" segments are avoided for practical use. This is accomplished through s a proper of ray direction; see [12] for details. Henceforth, we assume that $G\text{on}A = \emptyset$, i.e. $\text{RR}(G, A) = G\text{in}A$.

A ray-rep may also contain *tags*, i.e. descriptive symbolic information appended to the "in" segments. Tags may be used to carry properties of the interior of the solid, or characteristics of the solid's boundary in the region of the segment's endpoint, or for other purposes. Ray-reps have several important properties [12], some of which are summarized below.

4.1 Boolean simplification

A ray-rep of a Boolean composition of solids A and B in E^n can be obtained as

$$\text{RR}(G,\ A\ op_n\ B) \ = \ \text{RR}(G,\ A)\ op_{r1}\ \text{RR}(G,\ B) \tag{9}$$

where op_n denotes a Boolean operation ($\cap, \cup, -$) on n-dimensional solids regularized in the topology of E^n, $n = 2, 3$, and op_{r1} denotes op applied to "in" segments of each ray and regularized in the 1D relative topology of the ray. For rays represented parametrically, 1D regularization can be accomplished via simple sorting of parameter values. The Boolean simplification property is important because it reduces an n-dimensional (usually 3D) problem into a series of independent 1D problems.

4.2 Rigid motions

Let M denote a rigid motion and @ denote application of the motion to a set. Then

$$\mathrm{RR}(G,\ A@M) \;=\; \mathrm{RR}(G@M^{-1},\ A)@M \tag{10}$$

This is useful especially when the representation of A has a primitive large number of halfspaces, each requiring transformation by M. For example, if CSG(A) has N_A halfspaces, then computing RR($G, A@M$) would require N_A rigid motion transformations, whereas $G@M^{-1}$ and the final $@M$ transformations are single rigid motions.

4.3 Discrete translations

Intuitively, discrete translations are motions by integer distances that correspond to ray grid spacings. Formally, let us define grids G_1 and G_2 to be equivalent ($G_1 \equiv G_2$) if their extended infinite versions are indistinguishable; a motion M is a discrete translation *iff* $G@M \equiv G$. It is easy to show that

$$G@M \;\equiv\; G \quad\Rightarrow\quad \mathrm{RR}(G, A@M) \;=\; \mathrm{RR}(G, A)@M \tag{11}$$

Note that the relation on the right hand side does not hold for an arbitrary rigid motion M, even if M is a translation. But, for those M's that can produce equivalent grids, this relation is important, because it allows RR($G, A@M$) to be computed from RR(G, A) through a simple shift of indices and addition of a constant to every parameter value in RR(G, A); no new classification $\mathcal{M}$ is needed.

4.4 Representation conversion and completeness

Approximate representations of solids can be constructed easily from ray-reps; columnar decompositions, faceted Breps, and octrees are examples. As the spacing between the rays increases, the quality of the approximation decreases and small features are lost. Less obviously, under suitable conditions, ray-reps with tag information can be *complete* representations in the sense indicated below, and as such may be members of a seventh family of unambiguous representations, called *sampling representations* (Sreps) [14].

To see the implications, let $\mathrm{CSG}_i(A)$ be a particular CSG representation of A, let f and g be representation conversions, and let $[X]$ denote the spatial set represented by X. Then, if RR(G, A) is complete

$$[\mathrm{CSG}_i(A)] \;=\; [\mathrm{CSG}_j(A)], \tag{12}$$

where

$$\mathrm{CSG}_j(A) \;=\; g(\mathrm{RR}(G, A)) \tag{13}$$

and

$$\mathrm{RR}(G, A) \;=\; f(\mathrm{CSG}_i(A)). \tag{14}$$

A discussion of the conditions governing ray-rep completeness and the conversion g can be found in [14].

5 Computing Ray-reps

Ray-reps can be computed from Breps or CSG, using the appropriate algorithm sketched in Fig. 4. Contemporary graphics hardware that essentially scan convert Breps of polyhedra (flat-faced solids

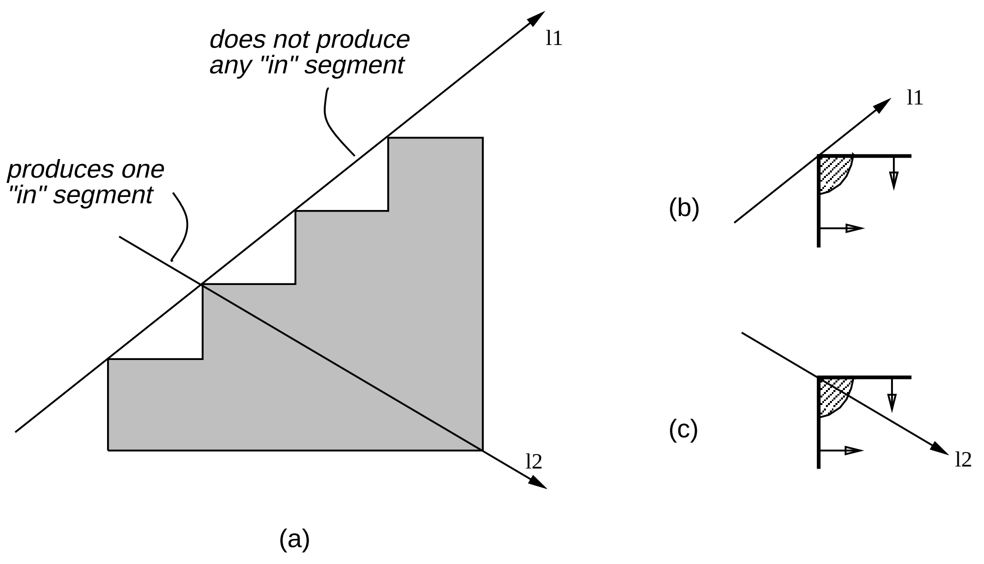


Figure 7: Resolving singularities through neighborhoods.

bounded by polygons) do in essence compute all the requisite z intercepts for the ray-rep. Broadly speaking however, the following pieces of technology are currently missing for producing ray-reps from such polygon rendering hardware.

- Multiple z-buffers to store all z intercepts for any ray.
- Techniques for handling *singular* points [18]. Intuitively, a singular point is one that generates a z intercept, but does not contribute to the ray-rep. For example, a ray (l_1 in Fig. 7a) passing through the edges of a staircase produces z intercepts, but does not generate an "in" segment. Resolution of singular points typically requires additional neighborhood processing [18]. For example, the neighborhood of the intercept in Fig. 7c intersects l_2, but not l_1 in Fig. 7b. Neighborhood information could be inferred from the sense of the face, i.e. from its normals, as marked in Fig. 7b,c.
- Boolean operations between intermediate ray-reps, which essentially reduces to simple sorting operations.

CSG processing would require hardware architectures quite different from contemporary Brep (boundary) based rendering machines. The next section provides an in-depth analysis of one such CSG processing machine.

6 The RCE: Case Study of a CSG Processing Hardware

6.1 Hardware accelerator families

At least four known types of hardware accelerators are ostensibly applicable to geometric modeling in general, be it based on Brep or CSG or any other representation scheme.

- Contemporary "3D graphics hardware" (both for PCs and workstations), that are nothing but fast line- and surface-rendering display engines based on custom-VLSI chips for homogeneous transformation and "polygon pushing" (scan conversion and such).

- Smart frame buffers, as exemplified by the Pixel Planes machines [3], wherein a small amount of processing power is assigned to each pixel.
- Octree machines, i.e. integer machines for parallel calculations on octree representations [5].
- Classification machines, which are discussed below, and currently the RayCasting Engine (RCE).

The first two families are display-oriented and linked intimately to Breps; they contribute little to the central mathematical problems in solid modeling, such as swift evaluation of Boolean compositions. The third family – octree machines – can provide some major speedups, but requires a special approximate representation scheme to do so. The fourth family, classification machines, seems to offer most promising possibilities, for reasons that will become evident.

6.2 Logical Evolution of the RCE

Fig. 8 summarizes the logical evolution of the RCE and provides a roadmap for the discussion below. More details can be found in [1, 2].

6.2.1 Classification Machines for CSG Solids

Many important calculations can be defined and implemented using set membership classification, and thus $\mathcal{M}(X, R)$ provides an obvious starting point for accelerator design studies. R is chosen to be a solid S represented in CSG, because CSG's natural divide-and-conquer classification algorithms offer great scope for parallelism. Specifically, from Tilove's work on set membership classification [21]:

```
Classify($X$: X_Rep, $S$: CSGsolid): X_Subset_Set;
  begin
    if ($S$ is a Primitive) then return Prim($X$,$S$)
    else return Combine(Classify($X$, $S$.Left), Classify($X$, $S$.Right), $S$.Operator);
  end


Prim($X$: X_Rep, $H$: Primitive); X_Subset_Set;
  Classifies the candidate $X$ against the primitive solid $H$


Combine($M1$, $M2$: X_Subset_Set, $op$: Operation): X_Subset_Set;
  Computes the set operation $op$ on the two sets $M1$ and $M2$ of X-subsets
```

An 'unwinding' of the recursive algorithm shows that the candidate:primitive classification function **Prim**(X, H) must be evaluated for all primitives $\{H_i\}$ in the representation of the solid. Further, the classification of X against a primitve H_i is independent of the classification of X against any other primitive H_j, hence all such calculations may be done in parallel. This suggests a specialized computer acrhitecture in which a "**Prim**(X, H) processor" – or Primitive Classifier (PC) – is provided for every primitive in the representation of the solid. Further inspection of the 'unwound' algorithms shows that pairs of outputs from PCs are combined in parallel (function **Combine** – and hence suggests "Classification Combine (CC)" processors in the specialized

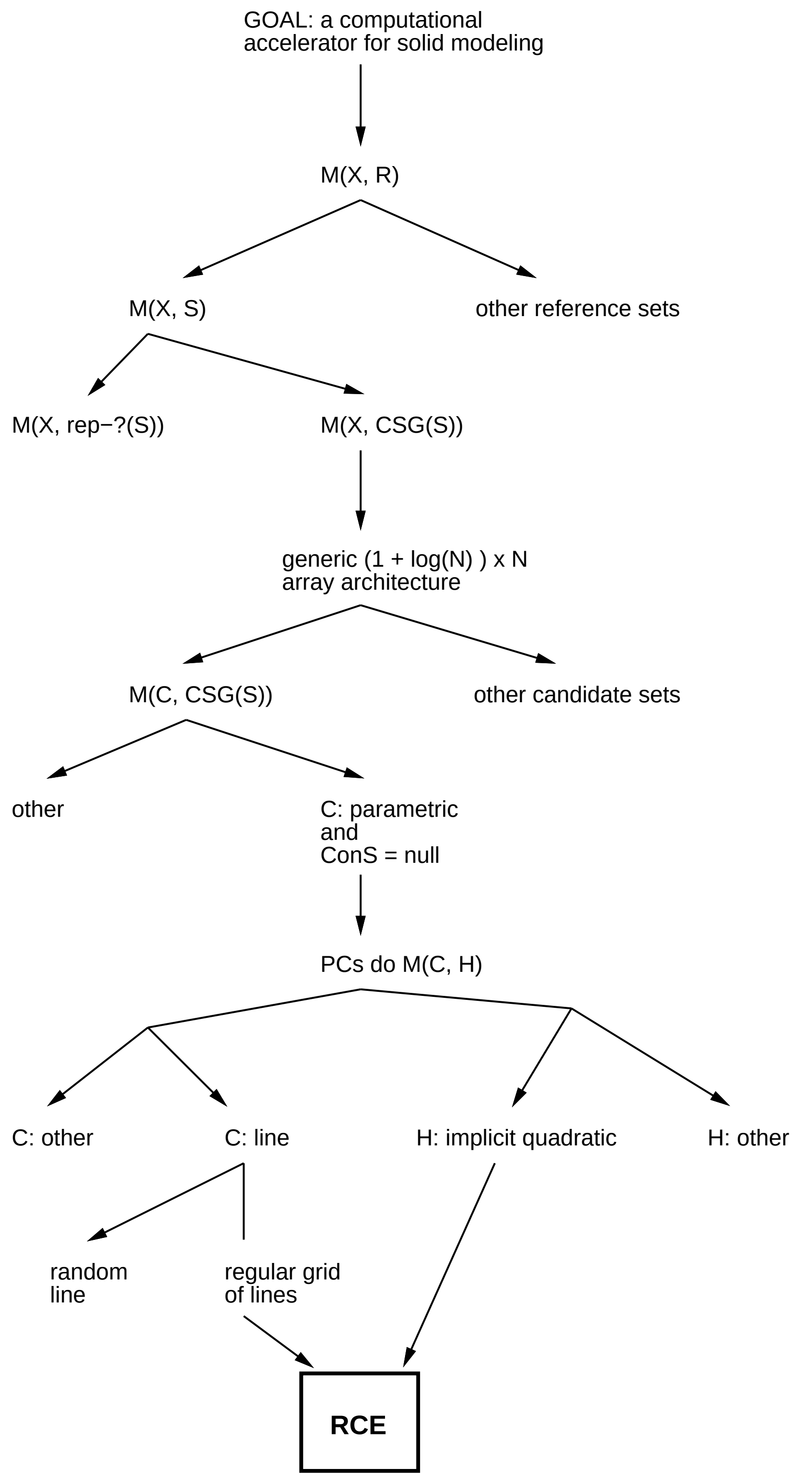


Figure 8: Logical evolution of the RCE architecture.

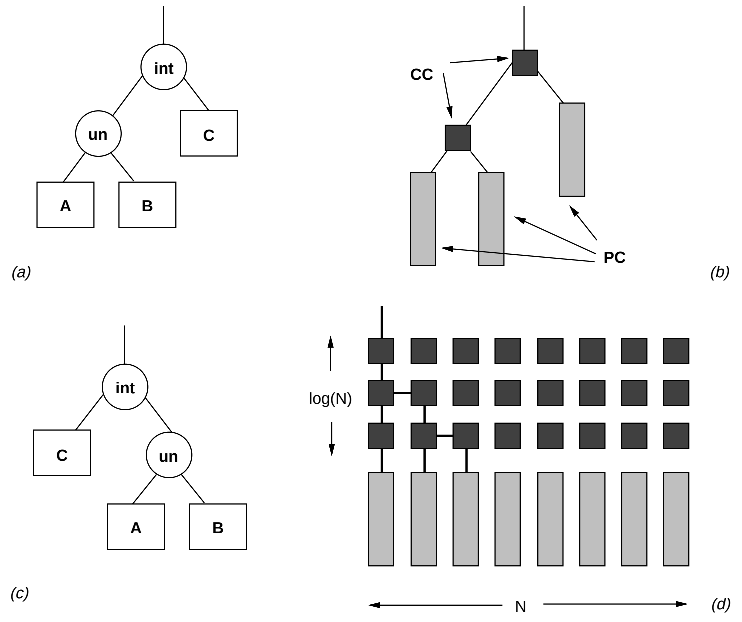


Figure 9: Mapping a CSG classification algorithm into hardware.

architecture. The combine step in each node of the CSG tree depends only on the results of the combine steps at the left and right sub-trees, and therefore the combine computation can be pipelined. The next section shows how these notions lead to a pipelined programmable tree architecture.

6.2.2 Machine Architecture

Our design strategy was based on providing a programmably configurable tree of processors that can mimic the CSG tree representing the reference solid S. For practical reasons dictated by VLSI technology, the CSG tree is embedded in a grid of processors with all leaves of the tree on the edge of the grid. Two key results drive the architecture:

- any CSG tree may be reorganized as a right-heavy tree, and
- the nodes of any N-leaf right-heavy tree may be mapped into an N x $\log_2 N$ grid.

Fig. 9 shows such a mapping. A CSG tree (Fig. 9a) is associated with a programmably reconfigurable hardware tree (b) whose leaves are primitive classifiers (PCs) and whose nodes are classification combiners (CCs); any CSG tree may be represented by a right-heavy tree as in (c), and any N-leaf right heavy tree may be mapped into an N x $(1 + \log_2 N)$ array as in (d).

In the array architecture of Fig. 9d, each PC processes the candidate X simultaneously, and classified X-subsets are pipelined bottom-up through the grid. The result – the classification of X with respect to the CSG solid S – is 'returned' by the root processor. The generic architecture has two important properties.

1. Efficient communication: Data in the array flow 'up and left', to nearest neighbors, through nodes that may operate as active CCs or passive conductors. Thus the data paths are 'soft silicon' customized for the particular reference solid, and more efficient than paths in general purpose

parallel computers.

2. Recirculation to process large CSG trees: The maximum practical size of an accelerator is governed by VLSI technology and economics. While a machine may be large enough to process some mechanical parts in a single pass through the machine, complicated parts and many assemblies will exceed any reasonable machine size and hence cannot be handled in a single pass; they must be segmented for sequential processing through a single hardware array, or for parallel processing through several hardware arrays. Because a CSG tree may be partitioned easily into subtrees that fit into the machine, classifications with respect to 'large' objects may be *accumulated* by combining 'old' results with the current subtree classification. The RCE employs a recirculation buffer storage and a direct-entry path into the CC-array for processing large CSG trees.

6.2.3 Curve/Solid Classification

At this point we are about halfway down the decision tree in Fig. 8, and must now make decisions about the candidate X. In principle, X could be solid, a surface or face, a curve, or a point: important applications requiring the classification of each of these entities are known. However, our experience with algorithms led us to focus on curves, and thus we narrow the discussion to curve/solid classification – 'CSC'. (Essentially all algorithms we know reduce classification with respect to a solid of another solid, or of a surface subset, to a series of simpler CSC problems, as in, for example, boundary evaluation [18]. PMC is easy to implement through CSC.)

The next two decisions – that curve C be represented parametrically, and that $C \mathrm{on} S = \emptyset$ – were strongly driven by hardware considerations. A parametric representation of a curve is of the form $(x(t),\ y(t),\ z(t))$, where t is the parameter along the curve. Curve segments are represented by two numbers (endpoint t-values), and segments may be combined (unioned, intersected, differenced) by operations akin to sorting that are easy to implement in hardware. The requirement that C have no 'on$S$' segments is more restrictive, and in the RCE described below requires special preprocessing to ensure that it is met. Again, this requirement was driven by practical considerations. Proper handling of 'on' segments requires that domain-dependent neighborhood information (described in earlier sections) be represented and processed in ways that complicate PC and CC hardware considerably.

In return for these restrictions, we gain the following significant properties for generic CSC machines.

3. Domain independence of the CCs: The ($\log_2 N$ x N) grid of CC processors is independent of the spatial domain, i.e. the particular type of curve being classified and the particular primitives $\{H_i\}$ in the representation of S. This follows because the CCs need only do regularized set operations on intervals represented by endpoints. (The CCs do some additional processing on 'tags', as explained later, but this also domain independent.)

4. Domain dependence of PCs: PCs must be designed to implement $\mathcal{M}(C,\ H)$, which mainly requires that the intersection of curve C with primitive solid H be computed and regularized in the relative (1-D) topology of the curve. The nature of C and H dictate the hardware design of the PCs. Thus, the domain dependence of the architecture is concentrated in the N-array of PC processors. The PCs must be customized to accommodate particular classes of curves and primitive solids, but the CCs – which occupy the bulk of the silicon in any VLSI implementation by a factor of $\log_2 N$ – need not be changed.

6.3 A Specialized CSC Machine

We must now specify the geometric character of the curves and primitives that the PCs process. Once the overall architecture has been set, these decisions embody the hard tradeoffs between functionality and hardware complexity. To understand the issues, assume that the curve C is represented parametrically by a polynomial of degre $\leq n$, and that every primitive H is a halfspace whose boundary is defined implicitly by a polynomial of degree $\leq m$. The essential operation in C/H classification is intersection, and this requires that the equations be solved simultaneously – typically by substituting the parametric equation into the implicit equation and finding the roots of the resulting polynomial of degree $\leq mn$. Thus one is faced with designing a hardware root-finder cheap enough to be replicated in each of the N PCs, and fast enough to keep the CC pipeline filled. (The CCs are fast because they do mainly comparisons of numbers.)

These are difficult requirements, and given the experimental nature of the work, we opted for a machine wherein

- $n = 1$, i.e. curves are straight lines, and
- $m = 2$, i.e. solids are bounded by quadric surfaces.

These decisions reduced the root-finding problem to non-iterative quadratic-equation solving, but this is still hardware-critical if one seeks to match the speed of the CCs. The key to PC speed and simplicity lies in solving quadratic equations *incrementally*, through the difference equation methods summarized below. The underlying principle is not to compute classifications for single 'random' lines, but rather to classify large grids of parallel lines. Most of the 'heavy' quadratic calculation can then be done once for the whole grid, as a preprocessing step, and only small incremental calculations are needed for each line in the grid. This approach reduces the PC hardware to a few adders and a square-root unit (no multipliers), and by careful design these can be matched in speed to the CCs.

Thus evolved the *RayCasting Engine* (RCE), a machine for classifying grids of parallel lines against solids represented in CSG that have quadratic bounding faces. The RCE is almost the simplest CSC machine imaginable. (The simplest would be an RCE for flat-faced solids.) The performance of the RCE relative to serial computation for a CSG tree of size $N_A \leq N$ range from $O(\log_2 N_A)$ to $O(N_A)$, while it is constant for asymptotically large ($N_A \gg N$) CSG trees [9].

Some important technical characteristics are summarized below.

6.3.1 Primitive Classifiers (PCs)

Each PC does line/halfspace classification for each line l_{ij} in an n_1 x n_2 grid of lines. This grid $G = \{l_{ij} \mid i = 1 \ldots n_1;\ j = 1 \ldots n_2\}$ is parameterized on t and indexed on (i, j). The grid has its origin $O = (o_x,\ o_y,\ o_z)$ and a direction defined by the vector $V = (v_x,\ v_y,\ v_z)$: see Fig. 10. Two vectors $H = (h_x,\ h_y,\ h_z)$ and $S = (s_x,\ s_y,\ s_z)$ define the spacing between parallel lines in the grid. Thus every line l_{ij} in G can be described as follows

$$l_{ij} = \begin{pmatrix} v_x t + h_x(j-1) + s_x(i-1) + o_x \\ v_y t + h_y(j-1) + s_y(i-1) + o_y \\ v_z t + h_z(j-1) + s_z(i-1) + o_z \end{pmatrix} = \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix}. \quad (15)$$

Line/halfspace classification is easily inferred from the set of line/halfspace-boundary intersection points. To find these points, the parametric $x(t)$, $y(t)$, $z(t)$ for the $(i,j)^{th}$ lines are

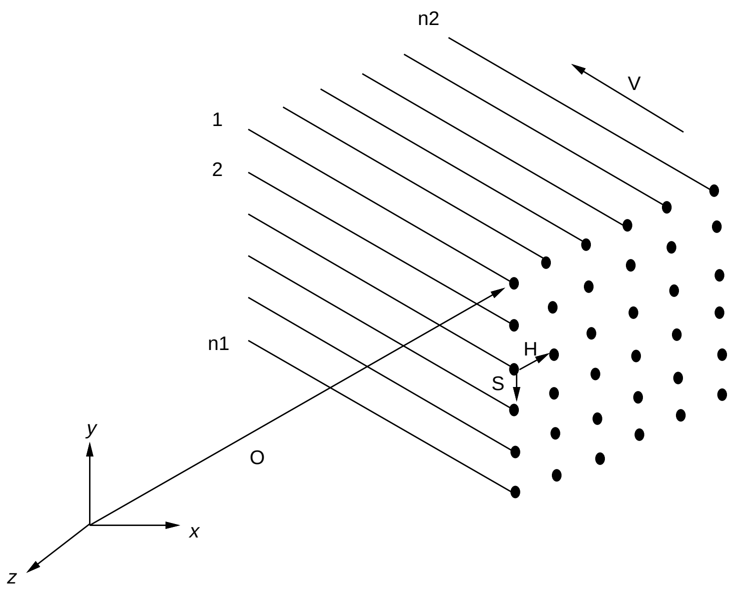


Figure 10: A ray grid in 3D.

substituted into the implicit representation of the boundary to obtain a quadratic in t. The two roots of the quadratic correspond to the the lead ($\mathcal{L}_{ij}$) and trail ($\mathcal{T}_{ij}$) for the line, i.e.

$$\mathcal{L}_{ij} = \mathcal{F}_{ij} - \sqrt{\mathcal{P}_{ij}}, \quad \mathcal{T}_{ij} = \mathcal{F}_{ij} + \sqrt{\mathcal{P}_{ij}}, \tag{16}$$

where $\mathcal{P}_{ij}$ is the value of a second degree polynomial at one of a sequence of a equally spaced points, and $\mathcal{F}_{ij}$ is the value of a linear polynomial at the same point. It is easy to see that any quadratic polynomial $\mathcal{P}_k$ and any linear polynomial $\mathcal{F}_k$ can be computed using finite differences by the following recurrence relations.

$$\Delta\mathcal{P}_{k+1} = \Delta\mathcal{P}_k + \Delta^2\mathcal{P} \tag{17}$$

$$\mathcal{P}_{k+1} = \mathcal{P}_k + \Delta\mathcal{P}_{k+1} \tag{18}$$

$$\mathcal{F}_{k+1} = \mathcal{F}_k + \Delta\mathcal{F} \tag{19}$$

Thus every line/halfspace classification requires four additions, one subtraction and one square-root operation. Each PC implements these calculation in fixed-point hardware using a square-root unit, adders, comparators, I/O registers, and appropriate control logic.

6.3.2 Classification Combiners (CCs)

The CCs do Boolean operations on intervals essentially by sorting the endpoints – the $\mathcal{L}_{ij}$, $\mathcal{T}_{ij}$ tuples – of the 'in' segments. The CC also supports a *tag calculus* for manipulating an integer tag appended to each segment. Tags typically carry symbolic names of surfaces or solids.

6.3.3 Scaling

Current generation PCs and CCs use fixed-point arithmetic. This imposes a *dynamic range* (the range of integers representable in the machine) within which all computations must proceed, and a requirement that the following conditions be met.

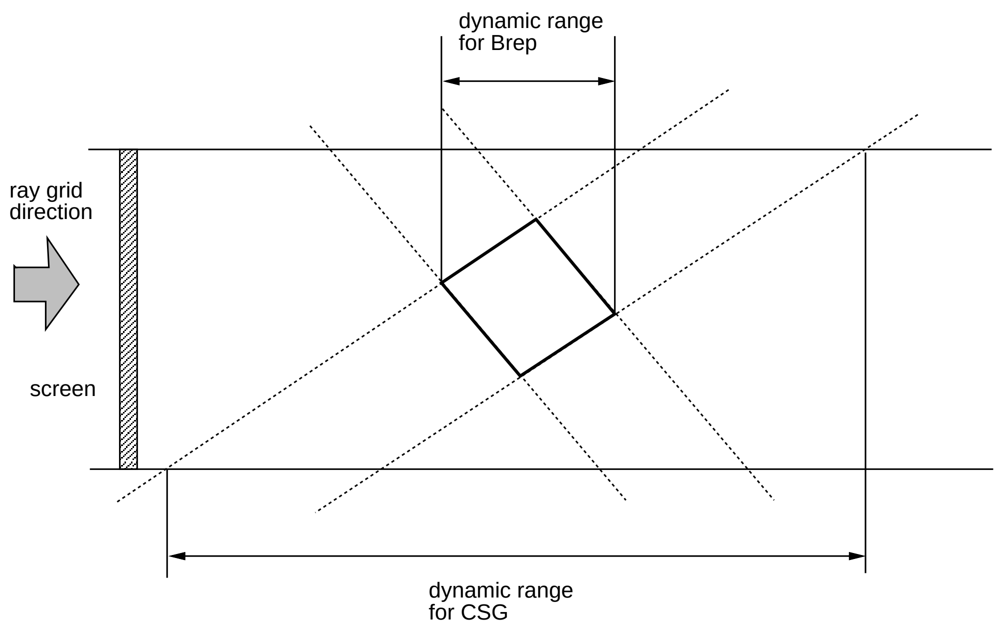


Figure 11: Optimizing machine dynamic range with scaling: comparison of Brep and CSG implications.

- The input data must be scaled to insure that all geometric calculations lie within the dynamic range of the machine.
- The scaling must be optimized to insure efficient use of the dynamic range, because overly conservative scaling leads to serious loss of resolution.

Fig. 11 explains the scaling problem. Assume the ray direction to be along the z-axis. For computing a ray-rep from the Brep, the dynamic range will correspond to the min/max z-coordinates of the vertices, marked on the top. For CSG however, the min/max corresponds to the min/max intercept that can be generated by any ray that intersects with the four linear halfspaces, marked below. Thus the dynamic range computation is trickier in the case of CSG, particular for curved (quadratic and beyond) halfspaces.

6.3.4 Programming the machine

The PCs and CCs operate asynchronously, and the classified line-segments are pipelined bit-serially bottom-up through the arrays using simple handshake protocols. The RCE system is programmed by an address-tagged data stream which flow upwards through the PCs into the CCs. The stream carries the numeric constants needed to initialize each PC, and the CC control codes which specify whether a particular CC is to be passive or do a specific set operation.

6.3.5 Status of the experimental machine

The RCE/1.0 at $N = 256$ (256 PCs, 2048 CCs) was running (at Cornell and Duke Universities) from 1989-1994, and since then an experimental RCE/1.5 (with better I/O channels) has been under study. A new *Distributed RayCaster* (DRC) currently implements an RCE-like architecture

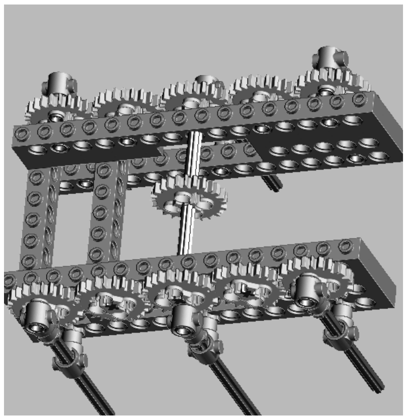


Figure 12: A six-legged walking machine constructed as a LEGO assembly with input torque provided at the central gear on the horizontal shaft.

on a cluster of workstations using the MPI (Message-Passing Interface) protocols. MPI lets such an implementation run on several individual workstations on a network, or on the IBM SP-2 distributed parallel computer. The DRC implements RCE technology without the need for custom hardware, and also provides a rich environment to experiment with more twists to ray-reps, e.g. sophisticated tag calculus, dynamic load balancing, and so forth.

7 A Plethora of Applications

Ray-reps have a wide range of applications in computer graphics and geometric modeling. This does not really depend on how the ray-reps were computed, or from what representation (e.g. Brep or CSG) they were generated. While the RCE provides one convenient means of computing ray-reps (from CSG trees of quadratic implicit surfaces), multiple z-buffer based polygon rendering from Breps could generate ray-reps as well. In fact, some of the *alhpa-blending* operations performed in contemporary display hardware do, in essence, implement ray-reps. This section summarizes several exemplary applications supported with ray-reps. We discuss these under three main topics:

- core geometric utilities,
- procedural geometry creation, and
- high-level applications (two case studies in Section 8).

Photographs of examples computed using ray-reps are accompanied by a halfspace count, e.g. 612h in Fig. 13, which indicates the number of leaves in the CSG tree processed on the RCE.

7.1 Core Geometric Utilities

Basic graphics: Color-graphic renderings of solids are produced easily via simple algorithms, e.g. Phong shading, on tiled ray-reps (Fig 12). “In”-solid segments along a ray provide sufficient information for transparencies (Figs. 16, 18). Cross-sectioning is available at almost zero cost, and approximate line drawings can be produced from ray-reps containing tags.

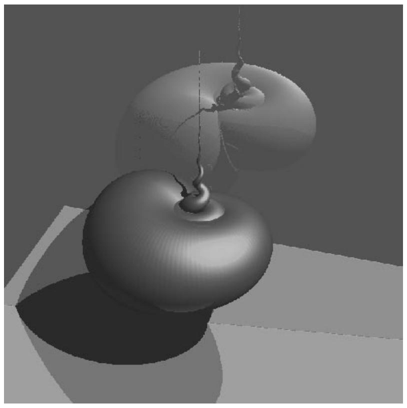


Figure 13: A ray-traced non-rigid sweep. (612h)

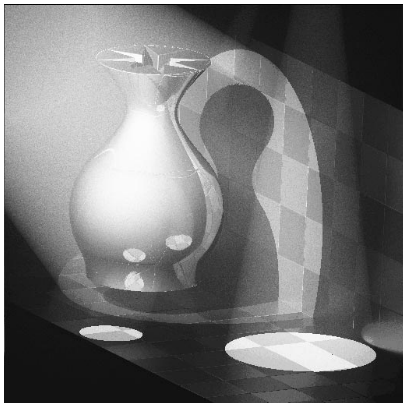


Figure 14: A vase (free-form solid modeled with CSRs and implicit quadratic algebraic patches) in participating media, rendered by solving a transport rendering equation. (2,880h)

Realistic rendering: Ray tracing is computationally intensive, and the RCE is not directly useful as an accelerator because it cannot handle efficiently collections of rays having different directions. However, the spatially discrete and indexable structure of ray-reps makes it easy to track rays using *2.5 dimension* DDA (digital differential analyzer) techniques. The surface gradients that determine the directions of reflected and refracted rays may be estimated from tilings of the ray-rep of the object being rendered, or may be computed more exactly from information contained in the the tags (Fig. 13). Fast 2.5 DDA algorithms allows for more than one ray to be bounced off a surface point, which when coupled with a Monte Carlo logic for statistical particle tracing, supports more realistic rendering (Figs. 14, 15) [24].

Mass properties: Integral properties – volume, centroid, moments of inertia, and such – are calculated directly and swiftly from ray-reps by inferring a columnar decomposition that is natural to ray-reps. The accuracy is governed by the grid density, with high accuracy being almost free on the RCE. In addition, variation of material properties can be tracked via tags in ray-reps. Fig. 12 shows a mechanical assembly, and volumes and centroids of every gear and every support structure

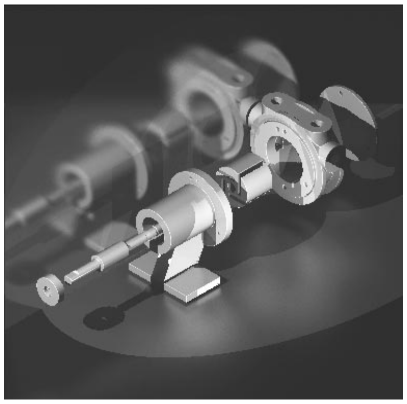


Figure 15: An exploded pump assembly rendered with hazy reflections using Monte Carlo ray tracing (one billion boundary samples) on its ray-rep. (200h)

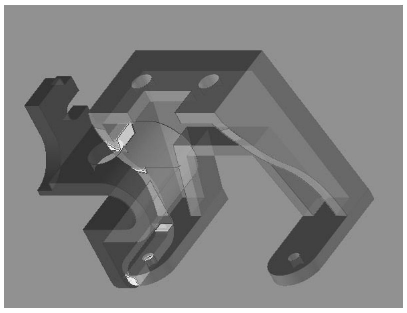


Figure 16: Interference detection between two static objects. (462h)

in the assembly can be computed very swiftly.

Static interference detection: Interference between stationary objects A and B can be tested by examining RR(G, $A\ \cap_3\ B$) for nullity. Tags in non-null ray-reps can be examined to localize the intersection (Fig. 16). Coarse grids may sometimes yield erroneous results, but very sensitive tests can be made by exploiting the RCE's variable grid and local bounding capabilities.

Tagging and graphic interaction: Tags in a ray-rep provide support for graphic interaction ('screen poking'). They also provide mechanisms for distinguishing components in assemblies, for detecting surface-surface mating conditions, and so forth.

Dynamic interference detection: The goal here is to determine whether two moving solids collide, and the location and nature of the collision if one occurs. One solution is to implement this as a series of static interference tests, using ray-reps, over a range of closely spaced instances of the moving solids (Fig. 17). A much simpler method may be used when the relative motion between the solids is a linear translation. Specifically, one need only examine a ray-rep of the union of the solids in a grid aligned with the motion vector. Fig. 18a shows an example of the linear motion problem, and Fig. 18b shows the computed solution.

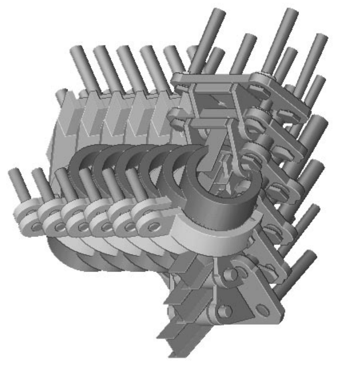


Figure 17: Six instances during motion. A static interference check at each instance helps to detect dynamic interference. (3,414h)

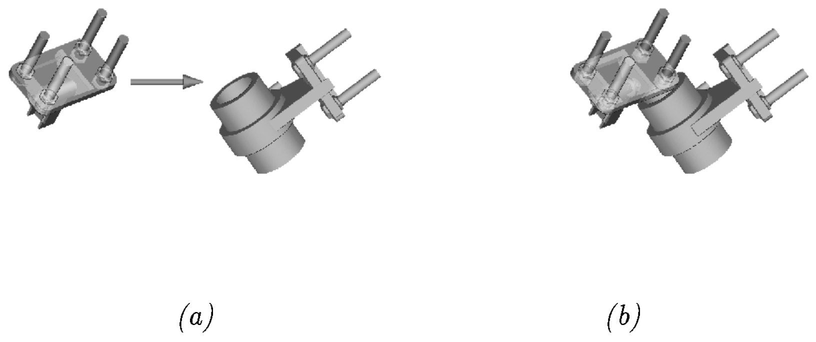


Figure 18: (a) The linear motion problem, and (b) the solution showing contact. (569h)

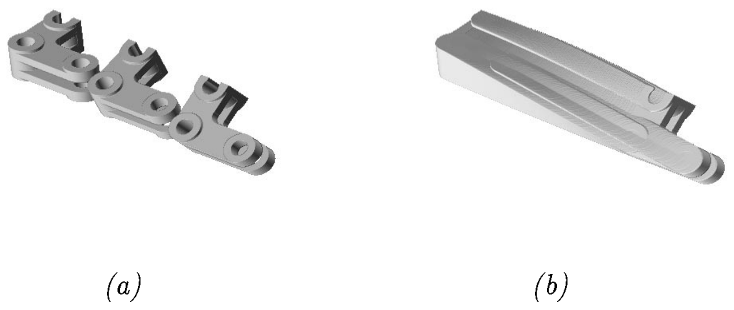


Figure 19: (a) Three instances during the sweep of a shifter link. (b) A swept shifter link with rotations and translations, using 3.001 instances. (195,065h)

7.2 Procedural Geometry Creation

General sweeps: The Boolean simplification property of ray-reps allows any solid that can be modeled as Boolean compositions to be handled effectively via ray-reps. Repeated Boolean operations arise in the mathematical definition of the sweep of a solid S (a *generator*) along some rigid motion (a *director*) $M(t)$ that is parameterized in time t. The swept solid $\text{Sweep}(S, M)$ can be defined as

$$\text{Sweep}(S,\ M) \ = \ \bigcup_t S_t \ = \ \bigcup_t S@M(t) \tag{20}$$

which is an infinite union of instances of S along $M(t)$. The sweep may be approximated by the following discrete union of a finite number of instances of S

$$\text{Sweep}(S,\ M) \ \approx \ \bigcup_{i=1,k} S_i \tag{21}$$

Now the ray-rep of the sweep may be computed in a "brute-force" manner by applying the Boolean simplification property to the discrete union approximation, i.e.

$$\text{RR}(G,\ \text{Sweep}(S, M)) \ \approx \ \text{RR}(G,\ \bigcup_{i=1,k} S_i) \ = \ \bigcup_{i=1,k} \text{RR}(G,\ S_i) \tag{22}$$

For example, Fig. 19a shows three instances during the sweep of a shifter link, and Fig. 19b shows the sweep computed as the union of 3,001 instances (corresponding to a CSG tree with 195,065 halfspaces). A similar formulation is employed to permit the generator to deform during a sweep as show, for example, in Fig. 20. Yet another example of deforming generator, or *non-rigid sweep* is shown in Fig. 21 – here a sphere is shrunk during its sweep along a logarithmic spiral and Boolean operations are performed subsequently on the swept solid in order to obtain the disconnected component.

Minkowski operations: Minkowski additions and subtractions of solids create new solids that are (loosely) 'grown' or 'shrunk' versions of the initial solids. The Minkowski sum (M-sum) of two

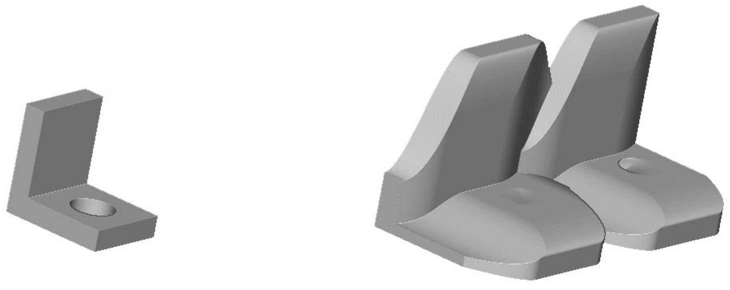


Figure 20: (a) An L-bracket, and (b) L-bracket sweep with sinusoidal deformation along directions normal to linear translation. (90,015h)

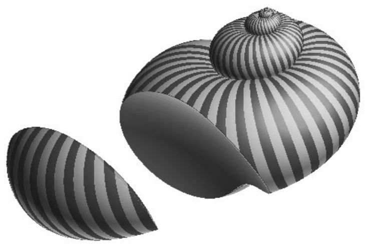


Figure 21: A nonrigid sweep of a sphere along a logarithmic spiral, using Boolean intersection and difference to produce the disconnected piece. (601h)

sets A and B is the set $A \oplus B$ defined by

$$A \oplus B = \{\mathbf{a} + \mathbf{b} \mid \mathbf{a} \in A, \mathbf{b} \in B\} = \bigcup_{\mathbf{a} \in A} B@M(\mathbf{a}) = \bigcup_{\mathbf{b} \in B} A@M(\mathbf{b}) \tag{23}$$

where $\mathbf{a} + \mathbf{b}$ denotes vector addition of points $\mathbf{a}$, $\mathbf{b}$, and $M(\mathbf{a})$ denotes a translation by vector $\mathbf{a}$. Its dual, called Minkowski difference (M-dif) and denoted $A \ominus B$, is defined as

$$A \ominus B = \bigcap_{\mathbf{b} \in B} A@M(\mathbf{B}) = (A^c \oplus B)^c. \tag{24}$$

An M-sum of two solids always results in a regular set, i.e. $A \oplus B$ will not have dangling faces/edges. However, and M-dif of two solids could result in a nonregular set, the dangling faces/edges of which are generally lost in the ray-rep scheme. Therefore, we introduce a new operator, called the regularized Minkowski difference, denoted $\ominus_n$, and defined as

$$A \ominus_n B = r_n (A \ominus B). \tag{25}$$

Regularized M-dif will eliminate all dangling faces/edges that may arise during M-dif.

When one of the sets (say, A) is a solid and the other (B) is a space curve, then their M-sum corresponds to a translational sweep of generator A along director B. When both sets are solids, the M-sum grows solid A by B, the M-dif shrinks A by B, and the regularized M-dif produces the regularized version of A shrunk by B. In the special case where B is a sphere centered at the origin, M-sum results in positive offsets and (regularized) M-difs in (regularized) negative offsets. When A is an arbitrary solid and B a sphere centered at the origin, the sequences

$$A_r = (A \ominus B) \oplus B \tag{26}$$

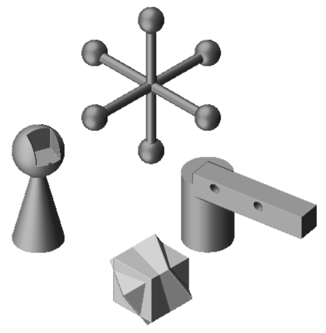
and

$$A_f = (A \oplus B) \ominus B \tag{27}$$

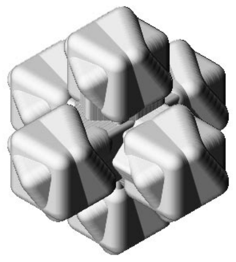
result in global blending of the solid. Specifically, A_r is a *rounded* version of A, and A_f is a *filleted* version. Several applications, such as modeling tolerance zones, configuration space modeling for robot motion planning, and so forth, require Minkowski operations. A set-theoretic reformulation of Minkowski operations is summarized in the algorithms **Msum-solids** and **Mregdif-solids**; see [12] for details.

```
**Msum-solids**($A$, $B$, $M_s$)
**begin**
  $M_s \leftarrow \emptyset$
  $C \leftarrow \bigcup_{\mathbf{a} \in \partial A} B@M(\mathbf{a})$
  **for** every connected component $B_i$ of $B$ **do**
    $k_i \leftarrow$ any point contained in $B_i$
    $M_s \leftarrow M_s \cup (A@\, M(k_i))$
  **end_for**
  $M_s \leftarrow M_s \cup C$
**end**

**Mregdif-solids**($A$, $B$, $M_d$)
**begin**
  $M_d \leftarrow$ universe
```



(a)



(b)

Figure 22: (a) A simple path-planning environment with a cube-like robot and three obstacles. (b) The M-sum of the robot and the cross-handle obstacle. (242,247h)

```
    $C \leftarrow \bigcup_{\mathbf{a} \in \partial A} B @ M(\mathbf{a})$
    for every connected component $B_i$ of $B$ do
      $k_i \leftarrow$ any point contained in $B_i$
      $M_d \leftarrow M_d \cap_n (A @ M(k_i))$
    end_for
    $M_d \leftarrow M_d -_n C$
  end
```

Ray-reps are used to compute M-ops (M-sum and regularized M-dif) by exploiting its Boolean simplification property. In other words, the infinite unions in the two algorithms above are replaced by finite unions, just as in the case of general sweeps. For example, Fig. 22b shows the M-sum of the robot (bottom) and an obstacle (top) in Fig. 22a – these are used in configuration space based motion planning algorithms. Fig. 23 shows a blending operation, Fig. 24 shows a "non-rigid" M-sum (analogous to non-rigid sweep in Fig. 20), and Fig. 25 shows an exhaust manifold modeled as combinations of sweeps and M-ops (successive growing and shrinking on the blends).

8 High-level Application Case Studies

8.1 NC Verification

Numerical Control (NC) machining programs essentially specify a sequence of commands to control cutter motions. Cutters spin at specified rotary velocities and are feed into a stock to remove material and produce the final "machined" part. The machining process can be modeled mathematically as:

$$W_i = W_{i-1} -_n C_i, \tag{28}$$

where W_i represents a workpiece after the execution of the i-th NC command, C_i is the spatial region swept by the cutter on the i-th command, and $-_n$ is the regularized set difference operator. Fig. 26a,b shows an example for a single cut, and Fig. 27a,b shows the result of machining with over 2,000 circular interpolation cuts – all calculations done with ray-reps.

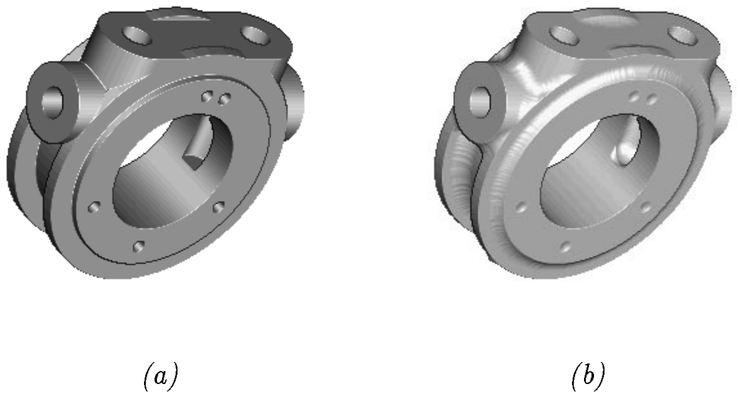


Figure 23: A pumpbody in (a) which is blended automatically in (b) using successive growing and shrinking by a sphere of blend-radius. (over 1 million halfspaces)

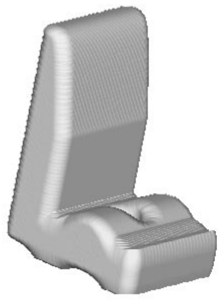


Figure 24: A nonrigid M-sum of a sphere with an L-bracket, with the sphere being scaled vertically in a sinusoidal fashion. (65,015h)

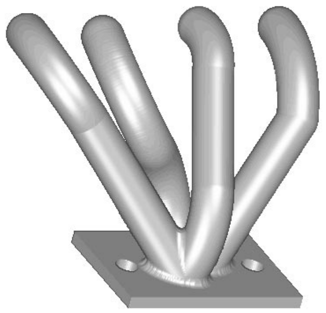


Figure 25: An exhaust manifold modeled using sweeps for pipes and blends for the flange weld. (197,477h)

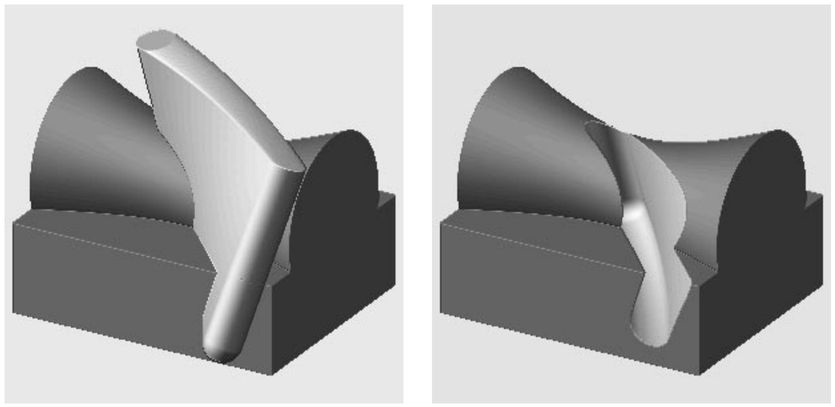


Figure 26: (a) A stock (union of a block and a hyperboloid of one sheet) with a five-axis cutter swept region, and (b) the result of machining obtained by a Boolean subtraction.

NC programs may contain errors of several types. Errors may range from trivial syntax errors to subtle interactions between objects, e.g. cutter/fixture collisions or over/under machining. Error detection via “proofing” run on soft material such as wood or foam is not only time consuming and expensive but also inadequate for predicting cutting forces and possible cutter breakage due to excessive cutter deflections. Hence there is a need for systems that verify NC programs by determining computationally whether an NC program, when executed in a known machining environment, will produce a specified part from specified stock without undesirable side effects.

We distinguish between *NC simulators* and *NC verifiers*. The former uses primarily computer graphics methods and relies on human interpretation and judgment to detect errors, while the latter is more precise since it uses mathematical predicates that are tested automatically in order to detect errors. Automatic NC verifiers [4, 6, 22, 11, 13] are more comprehensive and reliable. Furthermore, automatic predicate testing modules developed for an NC verifier are also useful in building robust process planning systems that automatically generate error-free NC programs. Four classes of machining phenomena are addressed, and computations are done on ray-reps:

- nominal spatial,
- tolerance,
- dynamical, and
- on-line sensing.

Nominal spatial verification does a first-order check to test for the following conditions: (a) stock sufficiency, (b) cutter collisions with the fixture, (c) over-machining (gouging), (d) under-machining, (e) cutter clearance, and (f) ineffective machining (cutting “air”). For example, a cutter collides with a fixture F during the i-th motion if the predicate

$$C_i \cap_n F \neq \emptyset \tag{29}$$

is true. Observe that this test can be conducted without rendering; graphics would be used only as an aid for visual understanding and subsequent problem resolution. For example, Fig. 28a shows

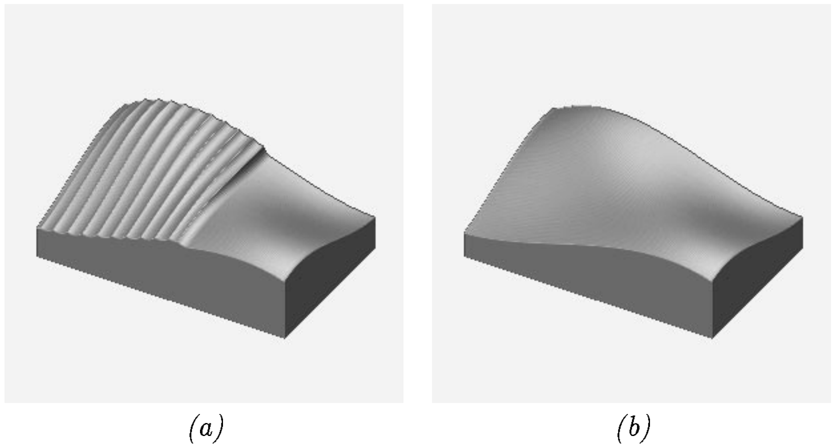


Figure 27: (a) Mid-way through a finish pass showing previous roughing cuts, and (d) the final machined surface after more than 2,000 cuts.

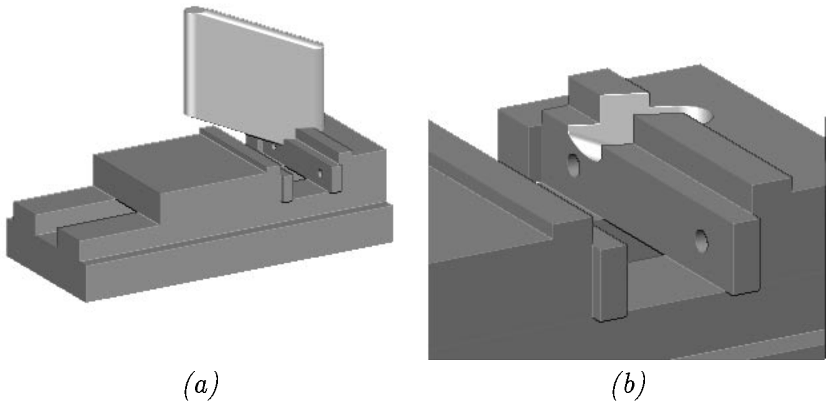


Figure 28: Fixture collision detection: (a) the cutter swept region and the fixture, and (b) the gouged fixture. Collision occurs if the Boolean intersection of the swept region and the fixture is not null.

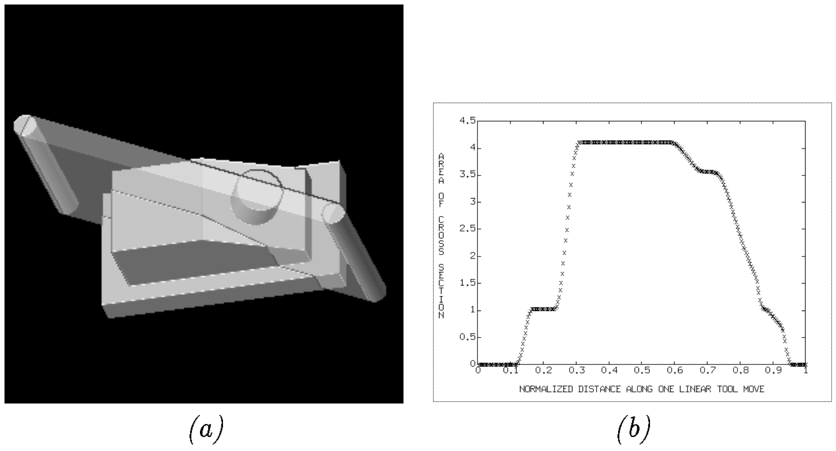


Figure 29: (a) A single NC cut showing start and end positions of the cutter with a transparent swept region, and (b) variation of removal rates tracked within the cut. Peak removal rates, detected here, could far exceed average estimates.

C_i and F, and Fig. 28b shows that the cutter would have collided with the fixture and removed material (or broken) – this would be detected automatically since $C_i \cap_n F \neq \emptyset$ in this case. One of the research thrusts is to speed up these tests, especially for the case of five-axis machining that generate complex cutter swept regions.

Tolerance verification pertains to checking for second order variations arising from: (a) cutter path approximation, (b) cutter envelope approximation, (c) cutter deflections, and (d) process uncertainties. While zones may be used sometimes to model permissible variations, the prediction of actual deviations during machining is hard. A zone for a line segment may be constructed by translating a sphere with radius corresponding to allowable tolerance along the segment. Similarly, a part zone may be defined as

$$\bigcup_i (f_i \oplus B(0, t_i)) \tag{30}$$

where f_i is a face of the part, and $\oplus$ denotes the Minkowski sum of a face f_i and a closed ball B at the origin whose radius is the face tolerance t_i. While computational zone-based tests are still largely unsolved, special case solutions via surface discretization and linear approximation are currently provided for single surface gouge determination.

Dynamical verification predicts and checks applied and reactive forces associated with cutting. A simple mathematical model starts with the assumption that the total energy to remove material in one NC cut is directly proportional to the volume of the material removed. This leads to a first order approximation

$$P \approx K_p K_c \mathcal{R} \ , \qquad P \approx F_c c \tag{31}$$

for the machining power P. The first relation approximates the machining power as a product of the material removal rate $\mathcal{R}$ and constants K_p, K_c found in machining handbooks. $\mathcal{R}$ is estimated from the geometric model of the machining process. From this, the tangential or cutting force F_c is estimated for a given cutting (or tangential or peripheral) speed c of the tool tip for the particular

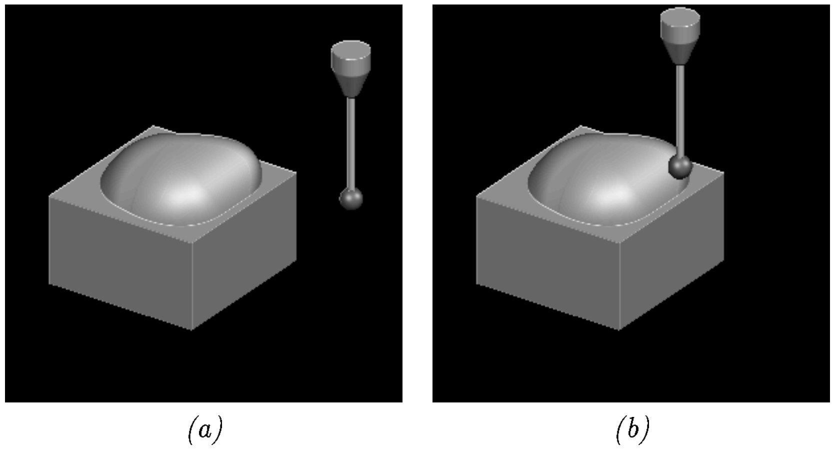


Figure 30: Modeling touch-sense probing: (a) Free-form stock, and a probe at the beginning of the probing motion; (b) Modeling physical contact between the probe and the stock, as the probe moves in a specified direction until it contacts the part.

NC command. This in turn provides an estimate for the magnitude of the cutter deflection δ as:

$$F_t \approx 0.5\ F_c\ , \qquad \delta = \frac{F_t L^3}{3EI}\ , \tag{32}$$

where F_t is the radial or thrust force (approximately half of F_c), L is the effective cutter overhang, E is the elastic modulus of the cutter material, and I is the cross sectional moment of inertia of the cutter. Thus simple estimates of the machining dynamics can be obtained for every NC cut, and accurate variations of removal rates can be tracked within every cut (Fig. 29) using solid models.

On-line sensing commands are increasingly becoming part of NC code. These commands typically contain instructions for touch-sense probes to establish a datum or find the center of a hole and so forth. A probe is a precise and very sensitive electromechanical switch that is triggered upon mechanical contact. Probe command verification requires a solid model of the probe and a series of spatial reasoning modules that: (a) compute the contact point(s) during probe motion (Fig. 30), (b) check for collisions (just as in cutter collisions), (c) check for erroneous triggers due to supporting rod contact or contact with unintended surface, and (d) check for orthogonal tip contact to be consistent with calibrated values.

8.2 Molecular Modeling

Space filling models are often invoked to rationalize the chemical behavior of proteins [19]. Prisant and co-workers have shown how ray-reps can be adapted to the treatment of solid geometric problems in molecular modeling [15].

Computation of Molecular Ray-Reps

The version of the ray-reps developed by Prisant's group for molecular modeling associates a material tag with each ray entry and exit point. In a minimal implementation, the tag identifies which atom gave rise to a particular point by referencing an an Protein Data Bank (PDB) atom

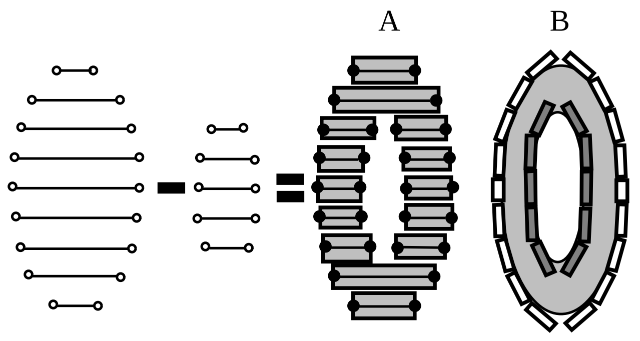


Figure 31: **LEFT:** Construction of the ray-rep of a composite object from simpler primitives. The torus ray-rep is computed by taking the ray by ray Boolean difference of two ellipses. **RIGHT:** Examples of unsophisticated algorithms for calculating volume and surface area. A) "Pile-of-Bricks" volume calculation and B) "Collocation of Tiles" area calculation.

identity number. In the present implementation, an extended tag also stores other information derived from the parenting primitive including a surface normal and differential area.

Calculation of physical properties from the ray-rep is conceptually simple. Consider two naive algorithms for volume and surface area of a compound object using the ray-rep. In the *pile-of-bricks* volume calculation shown on the right side of Fig. 31A, the lengths of all line segments connecting ray entry and exit points are computed. The scaled sum of these lengths estimates the volume of the compound object. In the *collocation-of-tiles* area calculation shown in Figure 31B, an area associated with each entry and exit points is derived from the placement of the point on the parenting primitive. The sum of these tile areas estimates the surface area of the compound object. In a similar fashion, visualization of the final ray-rep is straightforward because a surface normal is associated with each point and the points are already sorted front to rear.

The Fused Sphere Model

The fused sphere model represents a protein molecule as a union of spheres. Each atom in the molecule is taken to be a hard sphere of given van der Waals radius Fused-sphere models define a spatial boundary of a molecule or *van der Waals exclusion volume* with respect to non-bonding interactions. This spatial boundary is an energy surface for non-bonding interactions within the protein and with other molecules *in the absence of solvent.*

The straightforward algorithms presented in the previous section are the starting point for simple, accurate, and rapid calculations. Prisant has written a short *C* program ($<$ 300 lines of code) to implement the ray-casting calculations. Program accuracy and timing are shown in Fig. 32 as a function of casting density for myoglobin, a 1261 atom protein. The bottom panel provides timing information for the calculations on a 133 Mhz Pentium processor. Code execution time is linear with respect to the number of *non-empty rays* and the average number of atom spheres intersected by each ray. In general, the average number of atom spheres intersected by each ray is independent of protein size. At a casting density of 25 $rays/\AA^2$, the ray-rep computation takes approximately 10 seconds. This density provides accuracies of parts per million and parts per thousand in volume and area calculations respectively.

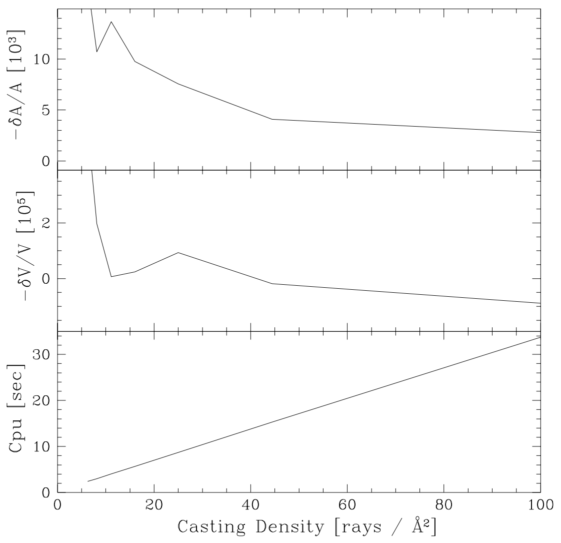


Figure 32: Ray-rep calculations of solvent accessible surface as function of grid density for myoglobin. The top, middle, and bottom panels display respectively: Collocation of tiles area calculation, pile of bricks volume calculation, and single processor computation times.

Solvent Exclusion Volume and Surface

The geometric construct of a surface boundary through which solvent cannot penetrate or *solvent exclusion volume* is key to many descriptions of how a protein interacts with surrounding water solvent. The surface of the fused sphere solid model of the protein does not correspond to this outer solvent accessible boundary. This is because the solvent is of finite size and therefore cannot come into contact with all portions of the fused sphere solid model's reentrant surface.

Ray-rep formalism can greatly simplify computing the protein solvent exclusion surface. Fig. 33 shows the Minkowski dilation and erosion steps in the process [12]. The procedure begins with the original collocation of fused van der Waals spheres. These atomic spheres are bloated by a radius corresponding to a probe sphere of water. The bloating of these spheres by the radius of the probe or solvent molecule fills in all crevices which cannot accommodate solvent molecules. Next, the bloated object is ray-cast and a casing of solvent spheres is located at the ray entrance and exit points. A compound object corresponding to the union of these casing spheres is constructed. Finally, a *new* ray-rep is computed by taking the Boolean difference of the probe sphere casing and the bloated protein. This difference has the effect of blending re-entrant concave surfaces on the protein according to the probe size while recovering the convex surfaces of the original unbloated fused sphere model. The surface of the final blended object encloses the true solvent excluded volume of the protein.

Computation times for the Minkowski procedure have a linear dependence on the number of segments to be subtracted and thus a quadratic dependence on casting density. Single processor execution times for myoglobin on a Pentium 133 range from 0.730 seconds at 4 $rays/\AA^2$ to 378.300 seconds at 100 $rays/\AA^2$. Volume calculations attain better than 5 parts per thousand convergence at 25 $rays/\AA^2$ and area calculations at 44.5 $rays/\AA^2$.

Description of Internal Cavities

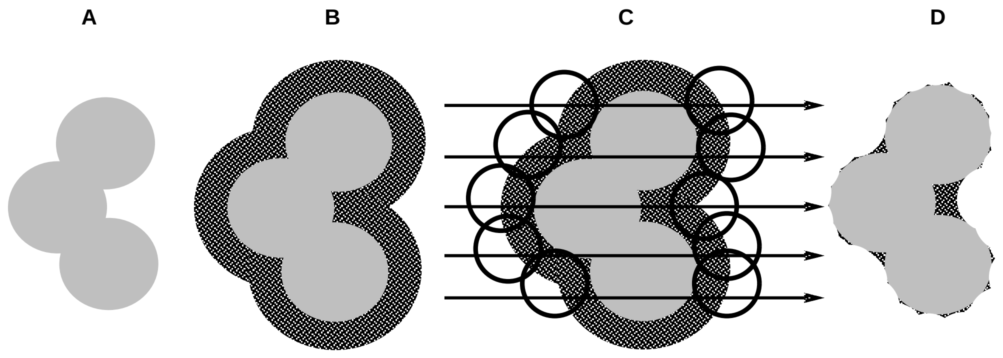


Figure 33: Pictorial depiction of the Minkowski operation: A) Fused VDW spheres are the starting point; B) The spheres are bloated by the solvent radius; C) The bloated object is ray-cast and a casing of solvent radius spheres are positioned at the ray entry and exit points; D) The casing spheres are subtracted from the bloated object.

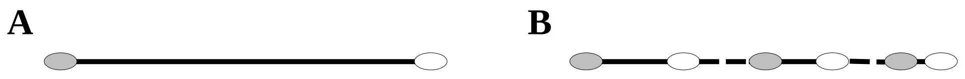


Figure 34: A) Ray traversing one pair of entry and exit points contains solid space. B) Ray traversing more than one pair of entry and exit points contains internal free spaces.

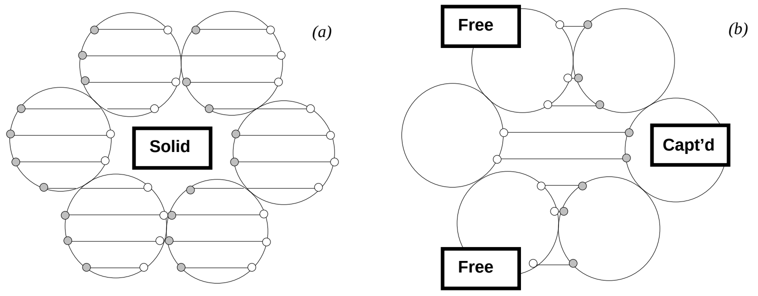


Figure 35: (a) Solid segments. (b) Void segments are either captured or free.

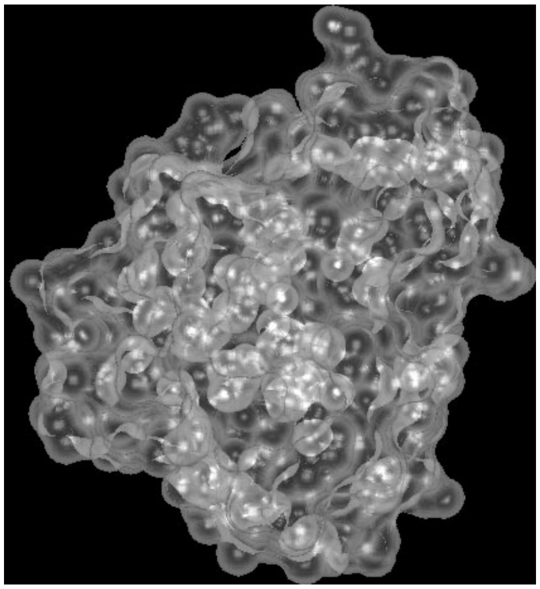


Figure 36: A transparent myoglobin, rendered from its ray-rep.

The Minkowski surface of a protein defines the protein's boundary and surface area with respect to solvent. We are interested in learning about cavities in the protein contained by this Minkowski surface. If totally enclosed, we might imagine that these cavities are important structural elements in protein folding.

Ray-rep can be used to identify and describe cavities in the protein interior. As shown in Fig. 34, individual rays can be classified by the number of times they enter and exit an object. If the ray has 0 sets of entry and exit points, then it is **empty**. If the ray has 1 set of entry and exit points then it is **solid**. If the ray has multiple sets of entry and exit points, then it contains **void** segments.

As shown in Fig. 35, void segments can be further classified into two categories according to the neighborhood of the ray to which they belong. Void segments which overlap empty space on adjoining rays are categorized as *free* void segments; those which do not overlap empty space are categorized as *captured* segments. We are now ready to map out the enclosed and open caverns in the protein. Algorithmically this means that segments which are connected through a series of overlaps with neighboring segments can be grouped together using standard clustering algorithms [20]. Each equivalence set of void segments defines a "cavern" in the protein. Now, if any one void segment in an equivalence set overlaps empty space, then that cavern opens to the outside. Conversely if no one void segment in an equivalence set overlaps empty space, then that cavern is completely enclosed within the protein. (Fig. 36 shows an example of myoglobin, rendered from ray-reps with transparency.)

9 Conclusions

This chapter formalized the notion of representations and made a distinction between complete and auxiliary representation schemes. Two members of each kind were studied: Brep and CSG for complete schemes and z-buffer and ray-rep for auxiliary schemes. Algorithms and consequent hardware architectures depend intimately on the representation scheme being employed. In particular, we studied the RCE – a CSG based massively parallel machine that produces ray-reps. The broad range of applications supported by ray-reps (when processed on the RCE in this case) showed how simple extensions to traditional z-buffer graphics can be extremely versatile in geometric modeling. The basic calculations are performed on quadratic implicit algebraic surfaces. Recent work on dual [Brep,CSG] representations of free-form geometries [7, 8] modeled with quadratic algebraic *patches* shows how all the techniques applied here can be extended in a straight forward manner to more complex shapes.

10 Acknowledgements

The author would like to thank Herb Voelcker (Cornell University), Gershon Kedem (Duke University), and their research teams for collaborative research on most of the topics presented here. Many thanks to Michael Prisant (Duke University) for providing a description of his work on molecular modeling using ray-reps.

References

[1] J.L. Ellis, G. Kedem, T.C. Lyerly, D.G. Thielman, R.J. Marisa, J.P. Menon, and H.B. Voelcker. The RayCasting Engine and Ray Representation: A Technical Summary. *International Journal of Computational Geometry and Applications*, 4(2):347–380, December 1991.

[2] J.L. Ellis, G. Kedem, R.J. Marisa, J.P. Menon, and H.B. Voelcker. Breaking Barriers in Solid Modeling. *ASME Mechanical Engineering*, 113(2):28–34, February 1991.

[3] J. Goldfeather and H. Fuchs. Quadric Surface Rendering on a Logic-enhanced Frame-buffer Memory. *IEEE Computer Graphics and Applications*, 6(1):48–59, January 1986.

[4] E. E. Hartquist, J. P. Menon, and U. A. Sungurtekin. Solid Modeller Based Machining. In *National Machine Tool Builders' Association 4-th Biennial International Manufacturing Technology Conference*, pages 7.23–7.38, Chicago, Illinois, September 7-15 1988.

[5] D.J. Meagher. The Solids Engine: A Processor for Interactive Solid Modeling. In *Nicograph '84*, Tokyo, Japan, November 1984.

[6] J.P. Menon. P2NC: Automatic NC Simulator/Verifier. Technical Report CPA User's Manual UM-11/2.2, Cornell University, 1988.

[7] J.P. Menon. Constructive Shell Representations for Free-form Surfaces and Solids. *IEEE Computer Graphics & Applications*, 14(2):24–36, March 1994.

[8] J.P. Menon and B. Guo. A Framework for Sculptured Solids in Exact CSG Representation. In *CSG96: Set-theoretic Solid Modeling Techniques and Applications*, pages 141–157, Winchester, U.K., April 17-19 1996. Information Geometers Publishers.

[9] J.P. Menon, R.J. Marisa, and H.B. Voelcker. Theoretical Relative Efficiency of the RayCasting Engine, A Parallel CSG Classification Computer. In *CSG94: Set-theoretic Solid Modeling Techniques and Applications*, pages 225–250, Winchester, U.K., April 13-15 1994. Information Geometers Publishers.

[10] J.P. Menon, R.J. Marisa, and J. Zagajac. More Powerful Solid Modeling Through Ray Representations. *IEEE Computer Graphics & Applications*, 14(3):22–35, May 1994.

[11] J.P. Menon and D.M. Robinson. Advanced NC Verification via Massively Parallel RayCasting: Extensions to New Phenomena and Geometric Domains. *ASME Manufacturing Review*, 6(2):141–154, June 1993.

[12] J.P. Menon and H.B. Voelcker. Mathematical Foundations I: Set Theoretic Properties of Ray Representations and Minkowski Operations on Solids. *Cornell Technical Report CPA91-9*, 1991. Cornell University, Ithaca, NY.

[13] J.P. Menon and H.B. Voelcker. Toward a Comprehensive Formulation of NC Verification as a Mathematical and Computational Problem. *Journal of Design and Manufacturing*, 3(4):263–278, December 1993. Chapman and Hall Publishers, London.

[14] J.P. Menon and H.B. Voelcker. On the Completeness and Conversion of Ray Representations of Arbitrary Solids. In *Third ACM/IEEE Symposium on Solid Modeling and Applications*, pages 175–186, Salt Lake City, Utah, May 17-19 1995.

[15] M. G. Prisant. Applications of the ray-representation to problems of protein structure and function. In *Proceedings of ACM Workshop on Applied Computational Geometry.* ACM, May 1996.

[16] A.A.G. Requicha. Mathematical Models of Rigid Solid Objects. Technical Report Tech. Memo. 28, Production Automation Project, University of Rochester (available at CPA, Cornell University), November 1977.

[17] A.A.G. Requicha. Representations for Rigid Solids: Theory, Methods and Systems. *ACM Computing Surveys*, 12(4):437–464, December 1980.

[18] A.A.G. Requicha and H.B. Voelcker. Boolean Operations in Solid Modeling: Boundary Evaluation and Boundary Merging. *Proceedings of the IEEE*, 73(1):30–44, January 1985.

[19] F. M. Richards. Folded and unfolded proteins: An introduction. In Thomas E. Creighton, editor, *Protein Folding*, pages 1–58. W. H. Freeman and Company, New York, 1992.

[20] Dietrich Stauffer and Amnon Aharony. *Introduction to Percolation Theory.* Taylor and Francis, Washington, DC, 1992.

[21] R.B. Tilove. Set Membership Classification:A Unified Approach to Geometric Intersection Problems. *IEEE Transactions on Computers*, 29(20):874–883, October 1980.

[22] H.B. Voelcker and J.P. Menon. Contemporary CNC Machining Technology. In *International Symposium on Steel Product-Process Integration*, pages 12–33, Halifax, Nova Scotia, Canada, August 20-24 1989.

[23] H.B. Voelcker and A. A. G. Requicha. Research in Solid Modeling at the University of Rochester: 1972-87. In *Fundamental Developments of Computer-Aided Geometric Modeling*, pages 203–254. Academy Press Limited, 1993.

[24] J. Zagajac. A fast Method for Estimating Discrete Field Values in Early Engineering Design. *Third ACM/IEEE Symposium on Solid Modeling and Applications*, May 1995.

SECTION B
Implicit Algebraic Methods

Abstract

The second section will focus on implicit algebraic methods that build free-form surfaces using low degree (2, 3) algebraic patches in the Bernstein-Bezier form. This section will provide techniques to construct free-form solids through polyhedral smoothing, guarantee continuity between patches, and avoid topological anomalies, such as multi-sheetedness. This leads to methods for maintaining dual Brep and CSG representations for arbitrary topology in a consistent manner. Concepts of trunctets, CSRs, finite extent decompositions, and incremental CSG updates under surface tweaking will be discussed. As a consequence, bones, vases, and such free-form objects can be rendered through either a Brep tesselation on contemporary workstations or direct CSG processing on the RCE. Details on performing these shape control operations are covered next, starting with a dual control polygon model that establishes a relation somewhat similar to that between NURBS surfaces and their control points/weights. Methods to obtain local and global shape tweaking effects will be illustrated.

Free-Form Modeling with A-Patches

Chandrajit L. Bajaj [*]
Department of Computer Science,
Purdue University,
West Lafayette, Indiana 47907
http://www.cs.purdue.edu/people/bajaj
bajaj@cs.purdue.edu

1 Introduction

While it is possible to model a general closed surface of arbitrary genus as a single implicit surface patch, the geometry of such a global surface is difficult to specify, interactively control, and polygonize. The main difficulties stem from the fact that implicit representations are iso-contours which generally have multiple real sheets, self–intersections and several other undesirable singularities. In this chapter I shall present details of several implicit polynomial surface splines (one is termed the A-patch) which overcome the above difficulties, and show how these are used in C^1 and C^2 interpolation/approximation and interactive free-form modeling schemes. The potential of implicit patch splines remains largely latent and virtually all commercial and many research modeling systems are based on parametric spline representations. An exception is SHASTRA which allows modeling with both implicit and parametric splines [4].

The important issues in free-form patch modeling of shapes with arbitrary topology are:

1. the patch representation
2. the polynomial degree of the patches
3. the number of patches per face of some input or benchmarking polyhedron
4. functional connectivity and nonsingularity of the patches
5. conditions for the desired continuity between adjacent patches (How the patches "stitched" together to form a "smooth" surface)
6. curvature and higher derivative variation of the patches, especially around the "stitches"

A significant amount of recent research has focussed on these questions with varying emphasis on non-tensor product patches, multivariate generalizations B-splines, geometric continuity, approximation order, and the fairness of fit. Common free-form patch modeling patch schemes include convex combinations of blending functions, and local interpolation of a mesh of curves, simplex and box based schemes, and stationary / non-stationary subdivision. In this chapter I address these issues with different implicit surface patches in a variety of algorithms. There are three broad categories of implicit

[*]Supported in part by NSF grant CCR 92-22467, AFOSR grant F49620-94-1-0080, and ONR grant N00014-94-1-0370

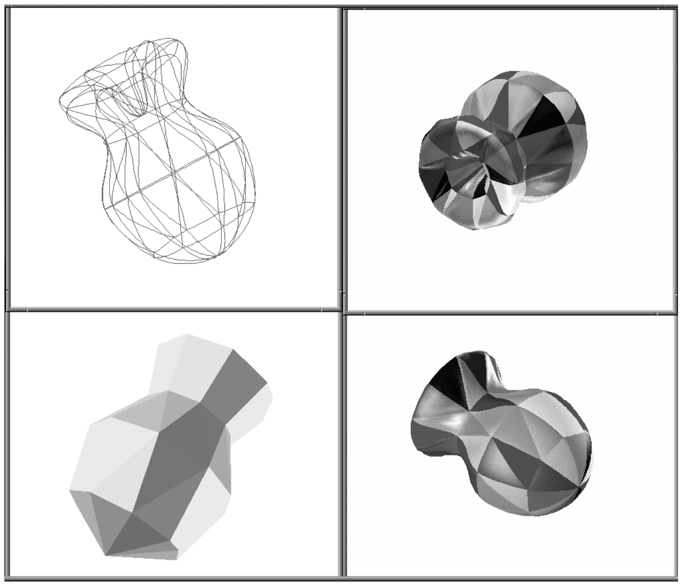


Figure 2.1: C^1 Implicit Splines over a Spatial Triangulation

patch schemes : *curvlinear-mesh* based, *simplex or box* based, and *subdivision* based. These make up the three subsequent sections.

2 Curvlinear Mesh Scheme

Bajaj, and Ihm [3, 12] construct implicit surfaces to solve the scattered data fitting problem. The resulting surfaces approximate or contain with C^1 continuity any collection of points and algebraic space curves with derivative information. Their Hermite interpolation algorithm solves a homogeneous linear system of equations to compute the coefficients of the polynomial defining the implicit surface. Bajaj, Ihm and Warren [14] extend this idea to C^k (rescaling continuity) interpolation or least squares approximation of a mesh of implicit or parametric curves in space. They also show that this problem can be formulated as a constrained quadratic minimization problem, where algebraic distance is minimized instead of geometric distance.

In a *curvlinear-mesh* based scheme, Bajaj and Ihm [13] construct low–degree implicit polynomial spline surfaces by interpolating a mesh of curves in space using the techniques of [3, 12, 14]. They consider an arbitrary spatial triangulation $\mathcal{T}$ consisting of vertices in $I\!R^3$ (or more generally a simplicial polyhedron $\mathcal{P}$ when the triangulation in closed) with possibly normal vectors at the vertex points. Their algorithm constructs a C^1 continuous mesh of real implicit polynomial surface patches over $\mathcal{T}$ or $\mathcal{P}$. The scheme is local (each patch has independent free parameters) and there is no local splitting. The algorithm first converts the given triangulation or polyhedron into a curvilinear wireframe with at most cubic parametric curves which C^1 interpolate all the vertices. The curvilinear wireframe is then fleshed to produce a single implicit surface patch of degree at most 7 for each triangular face $\mathcal{T}$ of $\mathcal{P}$. If the triangulation is convex then the degree is at most 5. Furthermore, the C^1 interpolation scheme is local in that each triangular surface patch has independent degrees of freedom which may be used to provide local shape control. Extra free parameters may be adjusted and the shape of the patch controlled by using weighted least squares approximation from additional points and normals, generated locally for each triangular patch. See also Figure 2.1. Similar techniques exist for parametrics[23, 36, 43] however

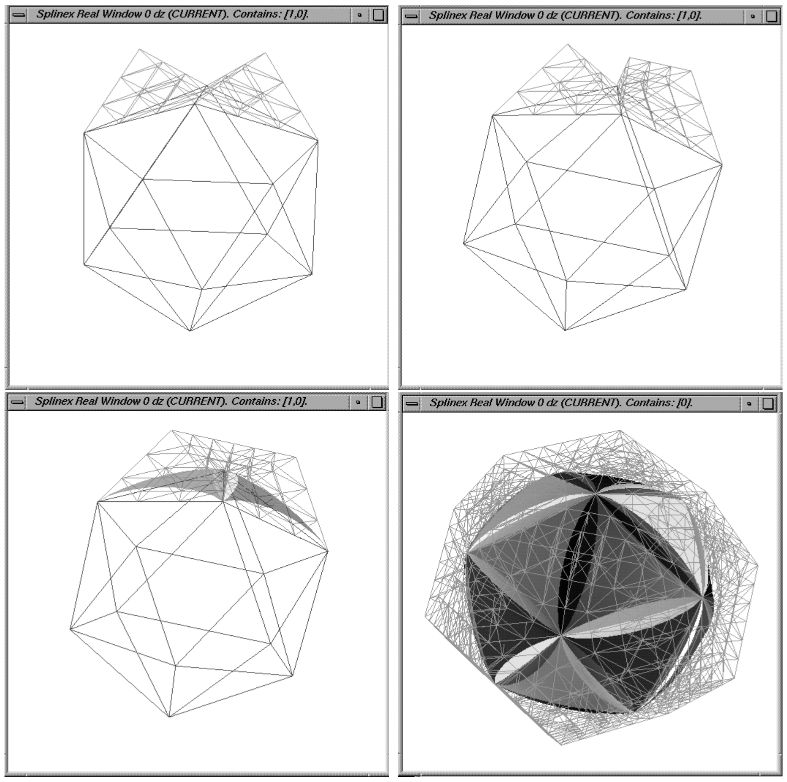

Splinex Real Window 0 dz (CURRENT). Contains: [1,0].
Splinex Real Window 0 dz (CURRENT). Contains: [1,0].
Splinex Real Window 0 dz (CURRENT). Contains: [1,0].
Splinex Real Window 0 dz (CURRENT). Contains: [0].

Figure 3.2: The construction of a simplicial hull and C^1 cubic A-patches interpolating the vertices of the input icosahedron.

the geometric degree of the solution surfaces tend to be very high.

3 Simplex and Box Based Schemes

In a *simplex* based approach one first constructs a tetrahedral mesh (called the simplicial hull) conforming to a surface triangulation $\mathcal{T}$ of a polyhedron $\mathcal{P}$. See Figure 3.2. The implicit piecewise polynomial surface consists of the zero set of a Bernstein-Bézier polynomial defined within each tetrahedron (simplex) of the simplicial hull. A simplex based approach enforces continuity between adjacent patches by enforcing that vertex/edge/face-adjacent trivariate polynomials are continuous to each other.

Similar to the trivariate interpolation case, Powell-Sabin or Clough-Tocher splits are used to introduce degree-bounded vertices to prevent the continuity system from propagating globally. Such splitting, however, could result in a large number of patches. A full trivariate Clough-Tocher split would split a tetrahedron into 12 sub-tetrahedra. However, as only the zero set of the polynomial is of interest, one does not need a complete mesh covering the entire space. Furthermore, a vertex does not need to be fully covered by the trihedral corners of the incident tetrahedra. A "incomplete" vertex helps introduce degree-bounded vertices as well. An effective simplicial hull construction of this kind first appears in Dahmen [19], and subsequently developed and used by Guo, Lodha, Dahmen and Thamm-Scharr, Bajaj, Chen and Xu[9, 20, 24, 29].

Given a triangulated polyhedron, the simplicial hull construction builds a tetrahedron on each triangular face (sometimes a pair of tetrahedra one on each side of a face). See Figure 3.4. The surface

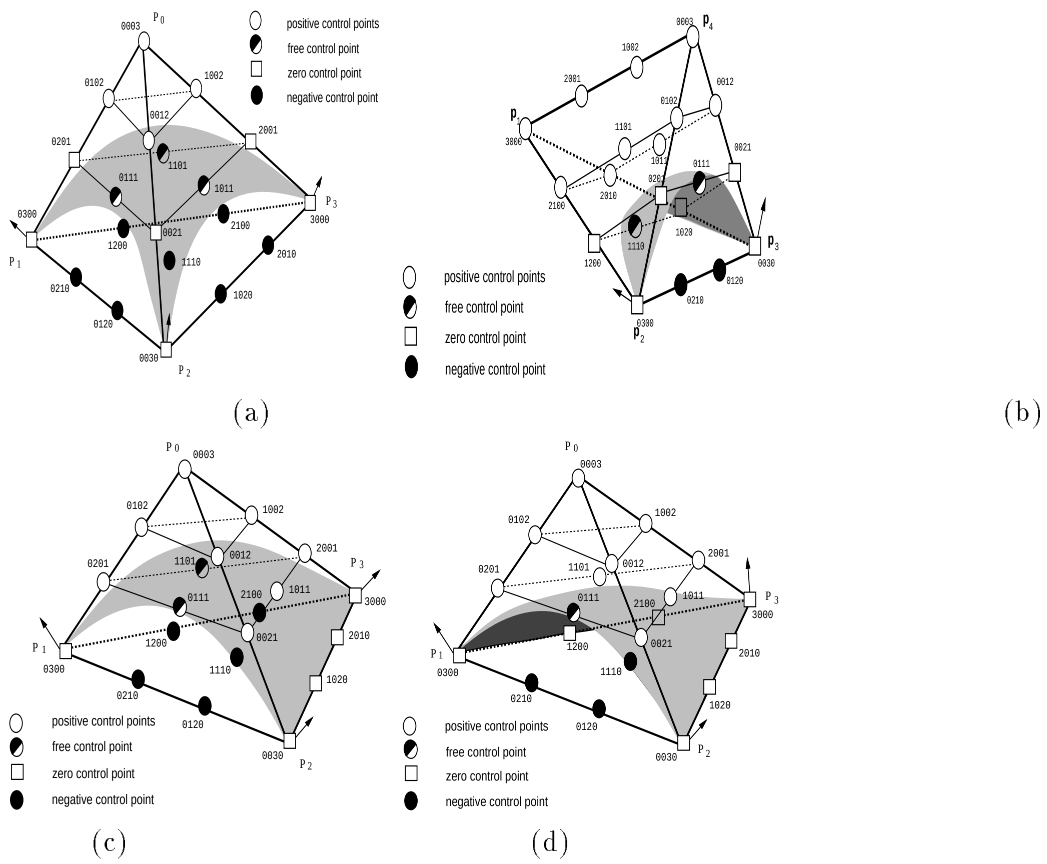


Figure 3.3: A-patches defined by a single change of coefficients in preferred directions. (a) A three sided A-patch tangent at p_1, p_2, p_3. (b) A degenerate four sided A-patch tangent to face $[p_1p_2p_4]$ at p_2 and $[p_1p_3p_4]$ at p_3. (c) A three sided A-patch interpolating edge $[p_2p_3]$. (d) A three sided A-patch interpolating edges $[p_2p_3]$ and $[p_1p_3]$.

is set to interpolate the three "bottom" vertices of the face tetrahedron. Hence a tetrahedron (over an edge) is enough to fill in the gap between tetrahedra built on adjacent faces. See Figure 3.2. The "top" of a face tetrahedron is thus degree-bound to just four. Hence, a face Powell-Sabin split or a face Clough-Tocher split (the bivariate analogue to the trivariate case), suffices. In a face Clough-Tocher split, a tetrahedron is split into only 3 sub-tetrahedra and therefore the number of micro patches are greatly reduced. An edge tetrahedron is usually split into two in a C^1 or C^2 scheme.

A disadvantage of an interpolating simplicial hull is that, as the surface is pinned to the vertices of the hull, a modification of the surface would involve both changes of the coefficients and the simplicial hull, whereas if the surface does not have to pass through any hull vertex, surface modification can be done solely by changing the coefficients of the polynomial.

3.1 Smooth Interpolation of a Polyhedron with C^1 A-patches

Dahmen [19] presents a simplicial hull scheme for constructing C^1 continuous piecewise quadric surface patches for a triangulation $\mathcal{T}$ of a polyhedron $\mathcal{P}$. In his simplicial hull construction, each triangular face is covered by a tetrahedron. Similar to the Powell–Sabin split [40], the tetrahedron is then face-split and replaced by six subtetrahedra, each of which defines a micro quadric triangular patch. Additional

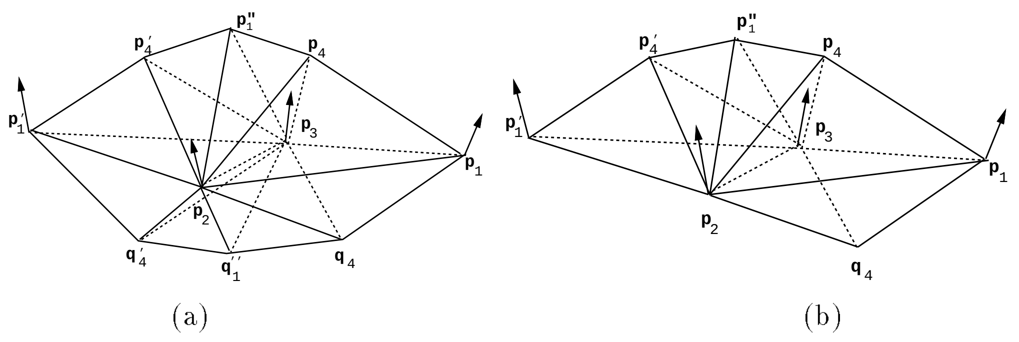


Figure 3.4: The construction of double tetrahedra for (a) adjacent 'non-convex'/'non-convex' faces and (b) 'convex'/'non-convex' faces

simplices are then used to fill in the gaps between the simplices built on adjacent triangular faces. Dahmen's technique however works only if the original triangulation of the data set allows a transversal system of planes, and hence is quite restricted. A major contribution of this scheme, however, is the construction of the simplicial hull.

Guo [24] uses cubics in an interpolating simplicial hull approach, to create free–form geometric models and enforces monotonicity conditions on a cubic polynomial along the direction from one vertex to a point of the opposite face of the vertex. This condition is difficult to satisfy in general, and even if this condition is satisfied, one still cannot avoid singularities on the zero contour. A face Clough-Tocher split is used to subdivide each tetrahedron of the simplicial hull. Dahmen and Thamm-Scharr [20] independently develop a similar scheme utilizing a single cubic patch per tetrahedron at the cost of less locality and extra perturbation when adjacent faces of $\mathcal{P}$ are coplanar.

Moore and Warren [33] extend the marching-cubes box based scheme to a generic simplex-based scheme, and compute a C^1 piecewise triquadratic B-spline approximation. An effective technique called "signed distance" is used to prevent the multiple-sheeted problem with implicit surfaces when employing least square approximation. Auxiliary data points are evenly added in the domain dpace with values being the signed distances to the input data points. The technique, although a heuristic rather than a guarantee, works quite well in preventing unwanted branches. However, the auxiliary data points with values of signed distances forces the surface patch to be monotonic in one direction, a condition which is not necessary and overly constraining.

Lodha [29] constructs low degree surfaces with dual parametric and implicit representations and investigates their properties. A method is described for creating a quadratic triangular parametric Bézier surface patch which is the parametric dual of an implicit quadric surface. Another method is described for creating a biquadratic tensor product Bézier surface patches which is the parametric dual of an implicit cubic surface. The resulting patches satisfy all the standard properties of parametric Bézier surfaces, including interpolation of the corners of the control polyhedron and the convex hull property.

Papers [13, 19, 20, 24, 25, 33], propose heuristics based on trivariate coefficient monotonicity, and least square approximation of convex quadrics to circumvent the multiple sheeted and singularity problems of implicit patches. Bajaj, Chen and Xu [7, 9] construct 3- and 4-sided A–patches that are implicit surfaces in Bernstein–Bézier(BB) form that are guaranteed smooth and single-valued. See Figure 3.3. For a three–sided A-patch any line segment passing through a vertex of the tetrahedron and its opposite face intersects the surface patch at most once ; and for a four–sided A-patch where any line segment connecting two points on opposite edges intersects the patch at most once. Instead

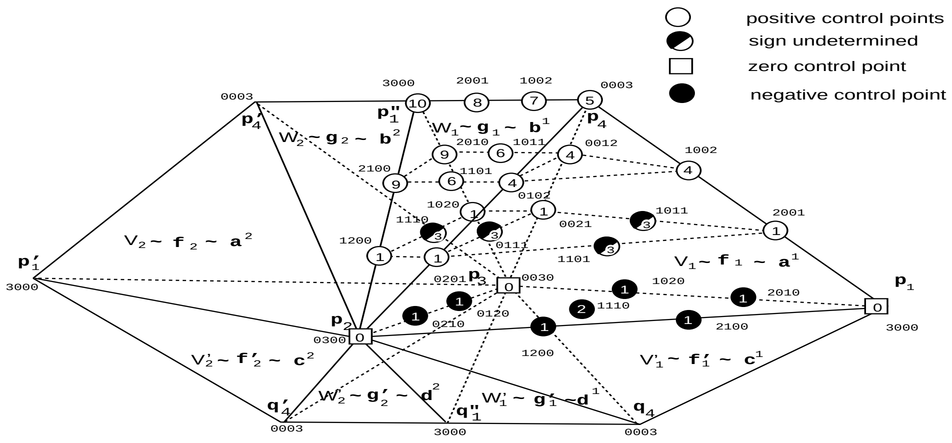


Figure 3.5: Adjacent double tetrahedra, coefficient signs and C^1 continuity relations for two cubic A-patches defined on non-convex adjacent faces

of having patch coefficients be monotonically increasing or decreasing there is now only a single sign change condition. There are also free parameters for both local and global shape modification of the patch. Papers [7, 9] also show how A-patches of cubic degree can be used to interpolate the vertices of any polyhedron $\mathcal{P}$ and yield a globally C^1 surface. This scheme is also extended to a C^2 version using A-patches of degree five [8]. Details are provided in the next subsection.

In these algorithms [9, 7] they first specify unique "normals" (tangent planes) on the vertices of $\mathcal{P}$, then build a simplicial hull surrounding the surface triangulation T of $\mathcal{P}$ and satisfying vertex tangent plane containment, and finally construct cubic A-patches within each tetrahedron of the simplicial hull. Different configurations of vertex "normals" for edges and faces of $\mathcal{T}$ are categorized as 'convex' and 'non-convex' . The edges and faces together with their normals are thus tagged as 'convex' and 'non-convex' . As part of the simplicial hull, a single tetrahedron is constructed for a 'convex' face while a pair of tetrahedra are constructed (one on each side of the face) for a 'non-convex' face (Figure 3.4). C^1 continuity conditions are next set up and satisifed between coefficients (control points) of all adjacent tetrahedra in the simplicial hull. Figure 3.5 shows the control points, their signs and their relations (like numbers) by C^1 continuity conditions in neighboring edge and face tetrahedra for the most difficult of cases, viz. two adjacent 'non-convex' faces. The C^1 continuity conditions are all linear and shown to be always satisfiable while maintaining the single sign change conditions of the A-patches. Details are in [9]. The resulting mesh of A-patches is thus guaranteed to be globally C^1 continuous. They also show how to adjust the free parameters of the A-patches to achieve both local and global shape control (bottom of Figure 3.6).

They use a single cubic A-patch per face of T except for the following two special cases. For a 'non-convex' face, if additionally the three inner products of the face normal and its three adjacent face normals have different signs, then in this case one needs to subdivide the face using a single face Clough-Tocher split, yielding C^1 continuity with the help of three cubic A-patches for that face. Furthermore for coplanar adjacent faces of T, they show that the C^1 conditions cannot be met using a single cubic A-patch for each face. Hence for this case they again use face Clough-Tocher splits for the pair of coplanar faces yielding C^1 continuity with the help of three cubic A-patches per face.

3.2 Smooth Interpolation with C^2 A-patches

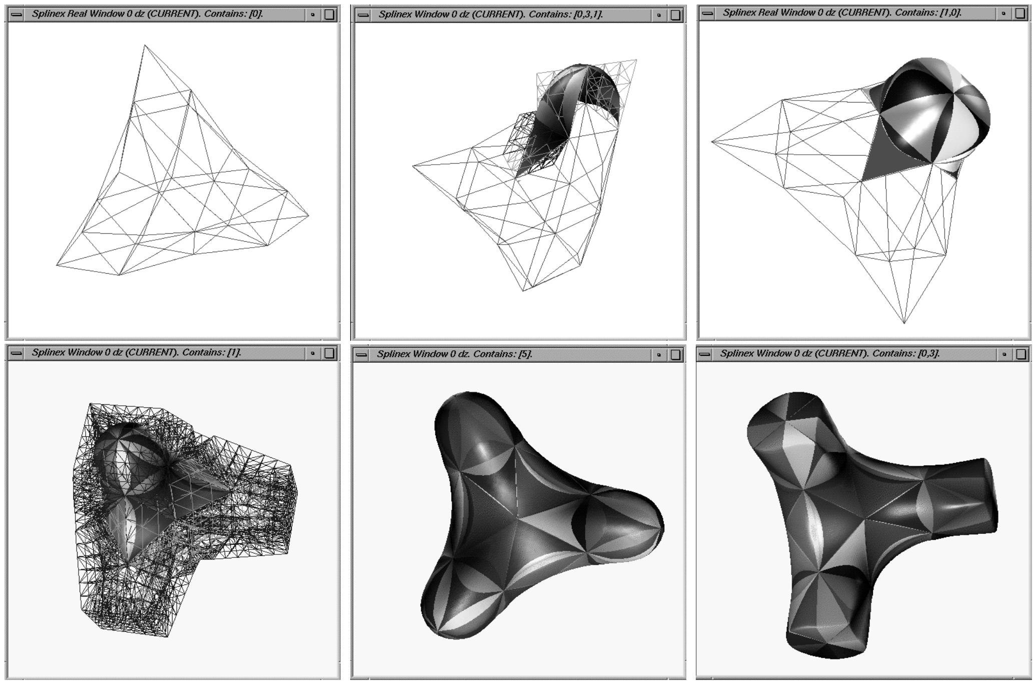

Splinex Real Window 0 dz (CURRENT). Contains: [0].
Splinex Window 0 dz (CURRENT). Contains: [0,3,1].
Splinex Real Window 0 dz (CURRENT). Contains: [1,0].
Splinex Window 0 dz (CURRENT). Contains: [1].
Splinex Window 0 dz. Contains: [5].
Splinex Window 0 dz (CURRENT). Contains: [0,3].

Figure 3.6: A face triangulated polyhedron, the simplicial hull and different interpolating C^1 cubic A-patches. Shape modification is achieved by adjusting the free parameters of the A-patches.

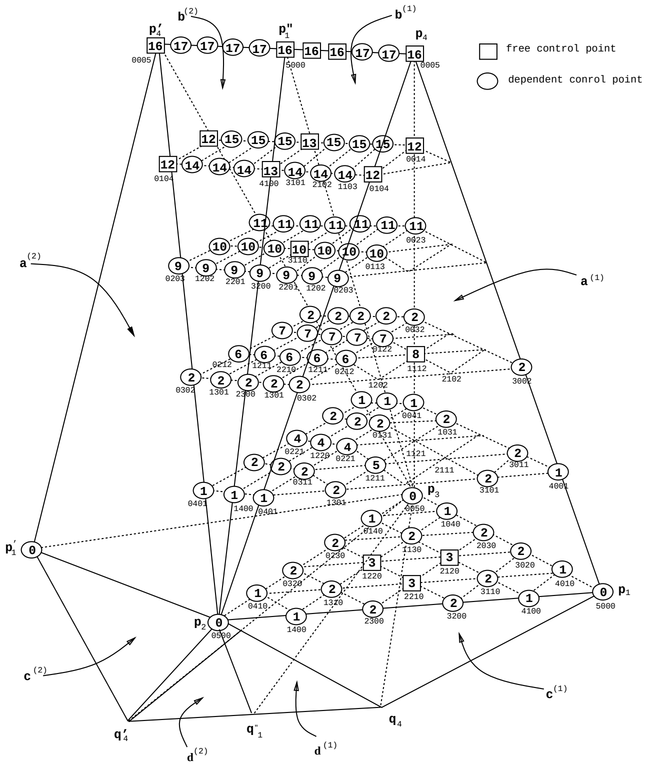


Figure 3.7: Adjacent double tetrahedra, coefficient signs and C^2 continuity relations for two quintic A-patches defined on ‘non-convex’ adjacent faces

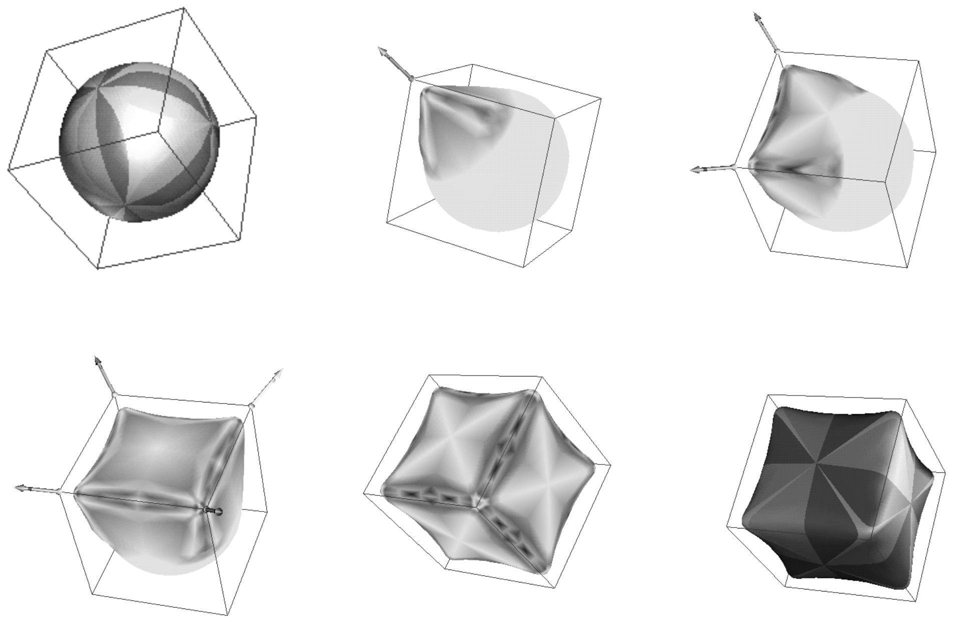


Figure 3.8: Interactive deformation of a sphere defined by C^2 continuous quintic A-patches.

Bajaj, Chen and Xu [8]. present a scheme for building a C^2 patch complex with quintic A-patches. Similar to the C^1 scheme, a simplicial hull Σ is constructed conforming to a face triangulated polyhedron $\mathcal{P}$ and a quintic A-patch is defined within each tetrahedron of Σ and made C^2 continuous across their share boundaries. The one-sign change condition, which is used in the C^1 scheme, turns out to be too constraining, and is thus relaxed by subdivisions for the C^2 case. The C^2 continuity conditions are all linear and involve groups of coefficients (control points) across common tetrahedral vertices, edges and faces. Figure 3.7 shows the control points, their signs and their relations (like numbered control points) by C^2 continuity conditions in neighboring edge and face tetrahedra for the most difficult of cases, viz. two adjacent 'non-convex' faces.

The number 0, 1, and 2 coefficients are given by the C^2 data values at the vertices. The number 4, 5, 6, 7, 9 , 11, and 17 coefficients are determined by C^2 continuity conditions and involve all the tetrahedra surrounding the vertex or edge. The number 3, 8, 12, 13, and 16 coefficients can be freely specified and adjusted for local or global shape control.

The number 0, 1, and 2 coefficients are given by the C^2 data values at the vertices. The number 4, 5, 6, 7, 9 , 11, and 17 coefficients are determined by C^2 continuity conditions and involve all the tetrahedra surrounding the vertex or edge. The number 3, 8, 12, 13, and 16 coefficients can be freely specified and adjusted for local or global shape control. One approach to setting default values for all the free coefficients is to first construct a desirable C^1 surface with cubic A-patches. Next one degree-raises the entire simplicial hull to quintic patches. These values become default for all the control points, some of which are subsequently modified by the C^2 continuity conditions as specified above. Figure 3.8 shows the modeling and interactive deformation of a sphere defined by C^2 continuous quintic A-patches by adjusting groups of free control points. Starting from the free-form model of the sphere, Figure 3.8 (a) the surface is first pulled towards a vertex of the cube (Figure 3.8 (b)). The Mean curvature map is displayed. The surface is next pulled towards an edge of the cube (Figure 3.8 (c)). The Gaussian curvature map is displayed. The surface is subsequently pulled towards a face of the cube showing the Mean curvature map and then towards all the other faces (Figure 3.8 (e)) showing the Gaussian curvature map. Figure 3.8 (f) show the final deformed surfaces with shades highlighting the C^2 continuous A-patches.

3.3 Smooth Reconstruction from Scattered Data

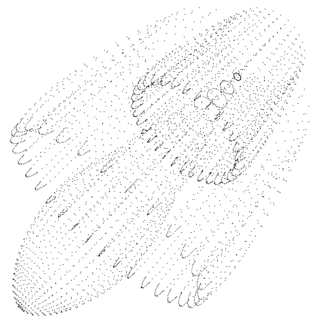
The problem here is the reconstruction of surfaces and scalar fields defined over it (surface-on-surface), from scattered trivariate data. The data points are assumed sampled from the surface of a 3D object, and the sampling is assumed to be *dense* for unambiguous reconstruction. Laser range scanners are able to produce a dense sampling, usually organized in a rectangular grid, of an object surface. Some 3D scanners are also able to measure the RGB components of the object color (i.e. three scalar fields) at each sampled point. When the object has a simple shape, this grid of points can be a sufficient representation. However, multiple scans are needed for objects with more complicated geometry, e.g. objects with holds, handles, pockets cannot be scanned in a single pass. Other applications, for example recovering the shape of a bone from contour data extracted from a CT scan, require reconstruction of a surface from data points organized in slices. The approach of considering the input points as unorganized has the advantage of generating cross-derivatives by a uniform treatment of all spatial directions.

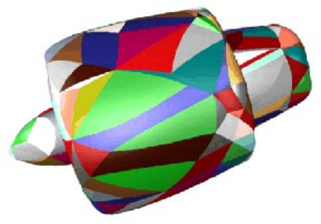
Bajaj, Bernardini and Xu [6] reconstruct the sampled surface using A-patches. Their scheme effectively utilizes an incremental Delaunay 3D triangulation for a more adaptive fit; the dual 3D Voronoi diagram for efficient point location in signed distance computations and cubic implicit surface patches. Furthermore, in the same time they also compute a C^1 smooth approximation of the sampled surface-on-surface. Bajaj, Bernardini and Xu [5] have also developed a similar method based on tensor-product Bernstein-Bézier patches.

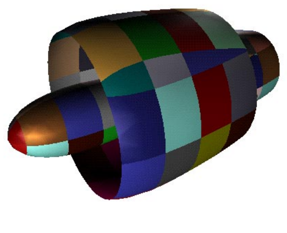
A different, three-step solution is given by Hoppe et al.[27, 28, 26]. In the first phase, a triangular mesh that approximates the data points is created. In a second phase, the mesh is optimized with respect to the number of triangles and the distance from the data points. A third step constructs a smooth surface from the mesh.

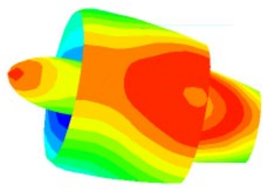
The problem of modeling and visualizing function-on-surface arises in several physical analysis application areas: characterizing the rain fall on the earth, the pressure on the wing of an airplane and the temperature on the surface of a human body. A number of methods have been developed for dealing with this problem.

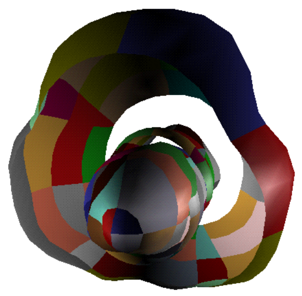
Currently known approaches for approximating function-on-surface data however possess restrictions either on the domain surfaces or the surface-on-surface. The domain surfaces are usually assumed to be spherical, convex or genus zero. The function-on-surface is not always polynomial [16, 35], or rather higher order polynomial [42], or a large number of pieces [1] compared to the approach of [6]. The method of [1] is a C^1 Clough-Tocher scheme that splits a tetrahedron into 4 subtetrahedra, uses quintic polynomials and requires C^2 data on the vertices of each subtetrahedron. Another Clough-Tocher scheme [44] requires only C^1 data at the vertices, for again constructing a C^1 function which is a cubic polynomial over each subtetrahedron, however splits the original tetrahedron into 12 pieces. A C^1 scheme [42] that does not split each tetrahedron uses degree 9 polynomials and requires C^4 data at the vertices. In extending the method of [42] to a C^2 scheme, requires degree 17 polynomials and C^8 data at the vertices of each tetrahedron. Compared to these approaches, the C^1/C^2 construction of [15] has no splitting and uses much lower degree polynomials (cubic/quintic) requiring only C^1/C^2 data respectively, at the vertices of each tetrahedron.


(a)


(b)


(c)


(c)


(d)

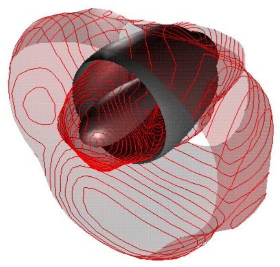

(e)

Figure 3.9: Jet engine model and associated pressure (scalar) field reconstruction from scattered data (a) Input point data (b) reconstructed model with cubic A-patches within a 3D triangulation (c) reconstructed model with bicubic A-patches over a box decomposition (d) isocontours of a pressure field displayed on the jet engine surface (e) reconstructed model of the pressure field with bicubic A-patches over a box decomposition (f) pressure field with iso-contours displayed surrounding the jet engine

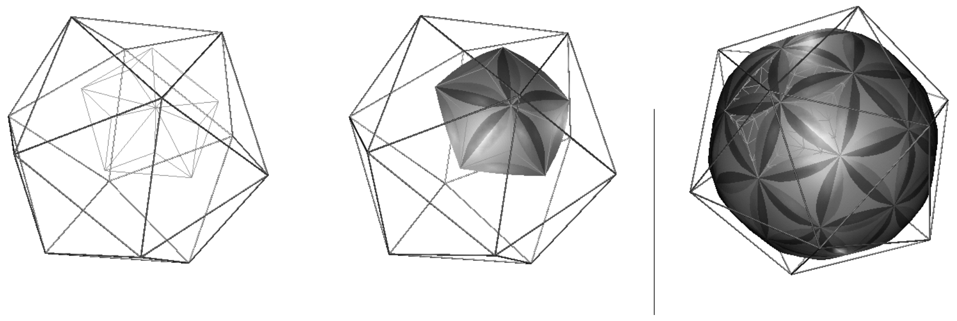


Figure 4.10: Corner Cuts, Inner Simplicial Hull and C^1 smooth A-patches

4 Subdivision Based Schemes

The third class of popular approaches of free-form modeling of shapes are in conjunction with subdivision methods. The input mesh is averaged and split before surface patches are fit in. The subdivision steps are sophistically designed so that the surface patches are bereft of difficulties such as *curve compatibility* and/or *vertex enclosure*. The idea behind subdivision is somewhat similar to that of Clough-Tocher and Powell-Sabin interpolants: introducing new vertices that are degree-bounded. The earliest of these approaches are the recursive subdivision schemes of Chaikin, Doo, Sabin, Catmull and Clark [17, 18, 21, 22]. These algorithms generate C^1 surfaces that interpolate the centroids of all faces at every step of subdivision.

Nasri [34] describes a recursive subdivision surface scheme that is capable of interpolating points on irregular networks as well as normal vectors given at these points. The subdivision scheme developed by Loop [30] splits each triangle of a triangular mesh into four triangles. Each new vertex is positioned using a fixed convex combination of the vertices of the original mesh. The final limit surface is tangent plane continuous. Hoppe et al. [26] extends Loop's method to incorporate sharp edges into the final limit surface. The vertices of the initial polyhedron are tagged as belonging on a face, edge, or vertex of the final limit surface. Based on this tag different averaging masks are used to produce new polyhedra. Reif [41] presents a unified approach to subdivision algorithms for meshes with arbitrary topology and gives a sufficient condition for the regularity of the surface. The existence of a smooth regular parameterization for the generated surface near the point is determined from the leading eigenvalues of the subdivision matrix and an associated characteristic map.

As subdivision techniques do not necessarily yield surfaces with analytical representations. However, by careful arrangement, after initial steps of subdivision, the mesh can be used as control nets for piecewise parametric Bézier patches or B-splines [31, 32, 37, 38, 39] or using implicit surface A-patches[10, 11]. The subdivision flavor is thus "taken over" by the subdivision algorithms of the particular parametric representation.

Bajaj, Chen and Xu [10, 11] construct an "inner" simplicial hull after one step of subdivision of the input polyhedron $\mathcal{P}$. See Figure 4.10. Similar to traditional subdivision schemes, $\mathcal{P}$ is used as a control mesh for free-form modeling while an inner surface triangulation $\mathcal{T}$ of the hull can be considered as the second level mesh. Both a C^1 smooth mesh with cubic A-patches and a C^2 smooth mesh with quintic A-patches can be constructed to approximate the given polyhedron $\mathcal{P}$. See Figures 4.11, 4.12.

Also similar to traditional schemes, simple editing of $\mathcal{P}$ can impose some interesting constraints on the surface, such as interpolating a line or a region. Furthermore, the free wieghts of neighboring patches and cutting ratios control how deep a corner is smoothed. For a trihedral corner, we control the

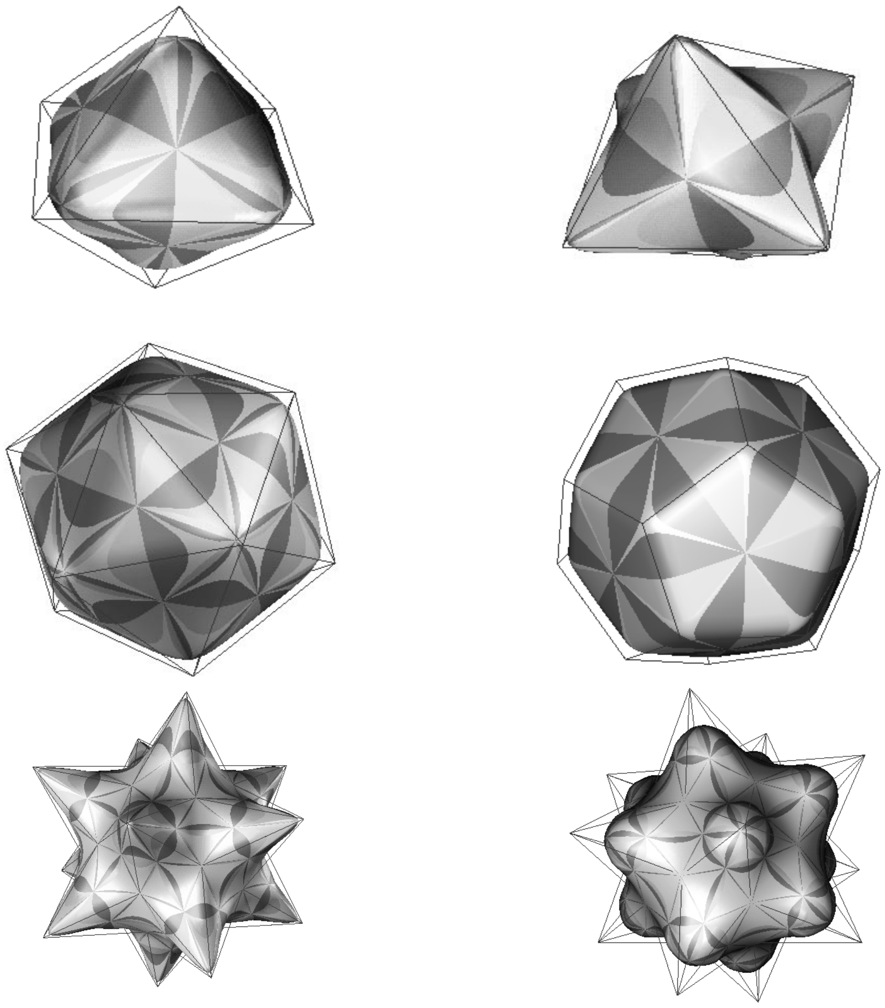


Figure 4.11: Smoothed Octahedron, icosahedron, dodecahedron and stellated dodecahedra using cubic A-patches. Corners of the dodecahedron are trihedral. Those of the others are non-trihedral.

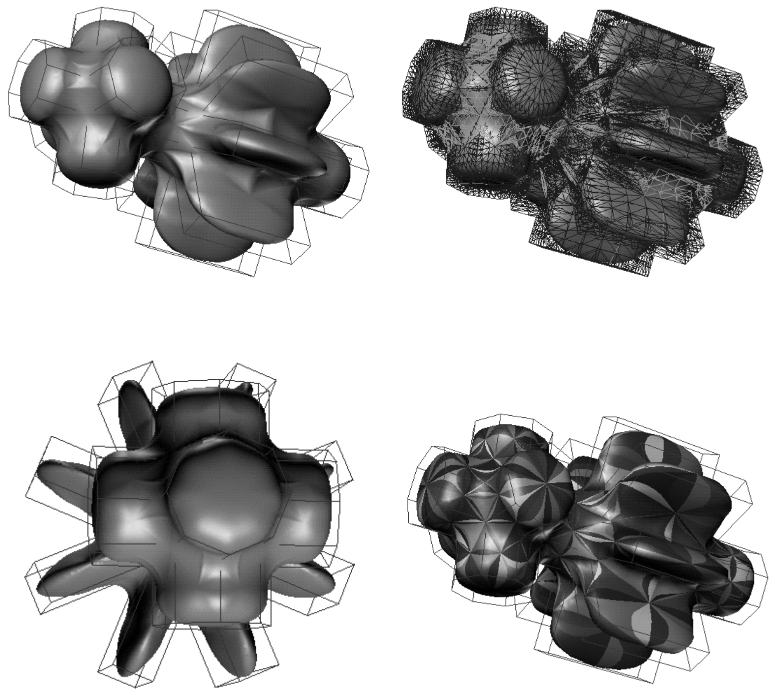


Figure 4.12: Subdivision based smoothing of a satellite-like object with cubic A-patches. Upper right also shows the simplicial hull, while bottom right shows the individual cubic A-patches.

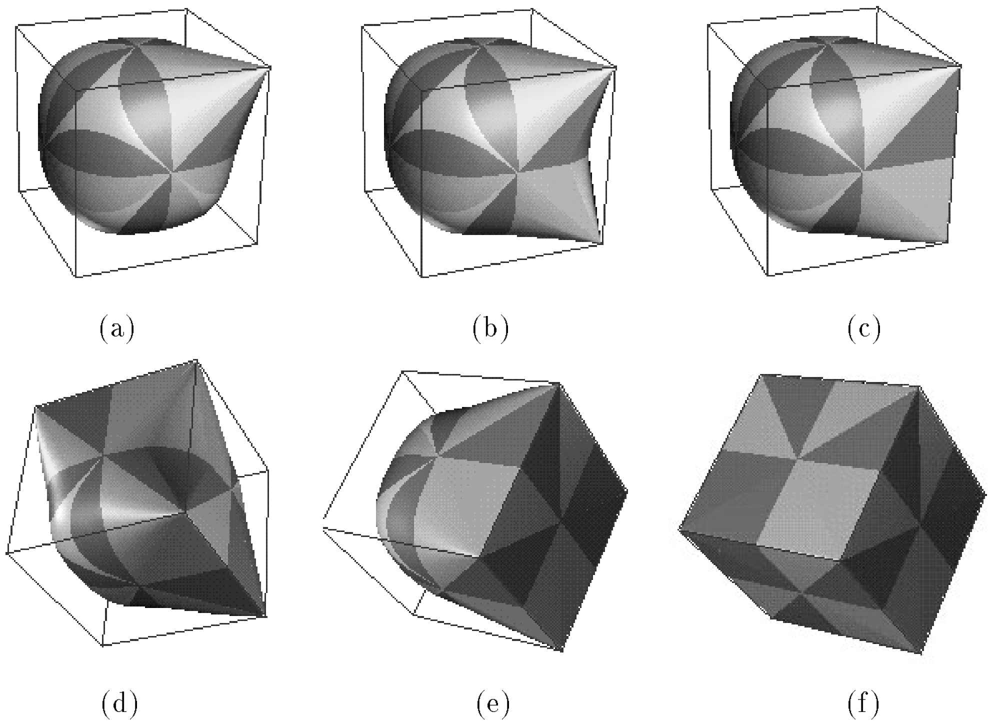


Figure 4.13: Modeling with singular A-patches. (a) Interpolating a vertex with a singular point. (b) Interpolating two vertices. (c) Interpolating an edge with a singular edge on the surface. (d) Interpolating two edges. (e) Interpolating a face of a cube. (f) The A-patch surface degenerates into the cube. All the edges are now singular.

shape of the neighboring surface by changing only the free weights in the face patch built at the corner and the surrounding edge patches. For a non-trihedral corner, we control the shape of the neighboring surface by changing the corner cutting ratio. See Figures 4.11 and 4.12. C^0 and C^1 features can be mixed into the same model, by allowing zero cutting ratios and patches with singular vertices/edges (weights around the vertices/edges are all zero, compared to coincident control points in the parametric case). Figure 4.13 shows how C^0 and C^1 features can be mixed to approximate a cube in different shapes.

5 Conclusion

All the algorithms of the previous sections have been implemented in the SPLINEX and SHILP toolkits of the distributed and collaborative geometric design environment SHASTRA [2]. SHILP is an X11 and Motif based, interactive solid modeling system, which is used to create a simplicial (face triangulated) polyhedral model of a desired shape. This model could also be the triangulation of an arbitrary surface in three dimensions. This triangulation is C^1 smoothed by client/server calls to SPLINEX processes using inter process communication. SPLINEX is an X11 and Motif based, interactive surface modeling toolkit for arbitrary algebraic surfaces (implicit or parametric) in BB form. It allows for the creation of simplex chains (for example, the simplicial hull of the triangulation) and the interactive change of control points of the A-patches for shape control. SPLINEX also has the ability to distribute its rendering tasks (for the display of the individual A-patches) on a network of workstations, to achieve maximal display parallelism.

References

[1] P. Alfeld. A trivariate Clough-Tocher scheme for tetrahedral data. *Computer Aided Geometric Design*, 1:169–181, 1984.

[2] V. Anupam and C. Bajaj. SHASTRA: Collaborative Multimedia Scientific Design. *IEEE Multimedia*, 1(2):39–49, 1994.

[3] C. Bajaj. Surface fitting with implicit algebraic surface patches. In H. Hagen, editor, *Topics in Surface Modeling*, pages 23–52. SIAM Publications, 1992.

[4] C. Bajaj. The Emergence of Algebraic Curves and Surfaces in Geometric Design. In R. Martin, editor, *Directions in Geometric Computing*, pages 1–29. Information Geometers Press, 1993.

[5] C. Bajaj, F. Bernardini, and G. Xu. Adaptive resconstruction of surfaces and surface-on-surface from dense scattered trivariate data. Technical Report Computer Science Technical Report, CS-95-028, Computer Sciences Department, Purdue University, 1994.

[6] C. Bajaj, F. Bernardini, and G. Xu. Automatic reconstruction of surfaces and scalar fields from 3D scans. In R. Cook, editor, *Annual Conference Series. Proceedings of SIGGRAPH 95*, pages 109–118. ACM SIGGRAPH, Addison Wesley, August 6-11 1995.

[7] C. Bajaj, J. Chen, and G. Xu. Free form surface design with A-patches. In *Proceedings of Graphics Interface '94, Banff, Canada*, pages 174–181, 1994.

[8] C. Bajaj, J. Chen, and G. Xu. Free form modeling with C^2 quintic A-patches. Presented in the Fourth SIAM Conference on Geometric Design, September 1995.

[9] C. Bajaj, J. Chen, and G. Xu. Modeling with cubic A-patches. *ACM Transactions on Graphics*, 14(2), April 1995.

[10] C. Bajaj, J. Chen, and G. Xu. *Smooth Low Degree Approximations of Polyhedra (Part 1).* Manuscript, September 1995.

[11] C. Bajaj, J. Chen, and G. Xu. *Smooth Low Degree Approximations of Polyhedra (Part 2).* Manuscript, September 1995.

[12] C. Bajaj and I. Ihm. Algebraic surface design with Hermite interpolation. *ACM Transactions on Graphics*, 11(1):61–91, January 1992.

[13] C. Bajaj and I. Ihm. C^1 Smoothing of Polyhedra with Implicit Algebraic Splines. *SIGGRAPH'92, Computer Graphics*, 26(2):79–88, July 1992.

[14] C. Bajaj, I. Ihm, and J. Warren. Higher order interpolation and least squares approximation using implicit algebraic surfaces. *ACM Transactions on Graphics*, 12(4):327–347, Oct. 1993.

[15] C. Bajaj and G. Xu. Modeling Scattered Function Data on Curved Surface. In J. Chen, N. Thalmann, Z. Tang, and D. Thalmann, editor, *Fundamentals of Computer Graphics*, pages 19 – 29, Beijing, China, 1994.

[16] R. E. Barnhill, K. Opitz, and H. Pottmann. Fat surfaces: a trivariate approach to triangle-based interpolation on surfaces. *Computer Aided Geometric Design*, 9:365–378, 1992.

[17] E. Catmull and J. Clark. Recursively Generated B-spline Surfaces on Arbitrary Topological Meshes. *Computer Aided Design*, 10(6):350–355, 1978.

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

[19] W. Dahmen. Smooth piecewise quadratic surfaces. In T. Lyche and L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design*, pages 181–193. Academic Press, Boston, Massachusetts, 1989.

[20] W. Dahmen and T-M. Thamm-Schaar. Cubicoids: modeling and visualization. *Computer Aided Geometric Design*, 10(2):89–108, Apr. 1993.

[21] D. Doo. A Subdivision Algorithm for Smoothing Down Irregular Shaped Polyhedrons. In *Proceedings of Interactive Techniques in Computer Aided Design, Bologna*, pages 157–165, 1978.

[22] D. Doo and M. Sabin. Behaviour of recursive division surfaces near extraordinary points. *Computer Aided Design*, 10(6):356–360, November 1978.

[23] G. Farin. Triangular Bernstein-Bézier Patches. *Computer Aided Geometric Design*, 3(00):83–127, 1986.

[24] B. Guo. Surface generation using implicit cubics. In N.M. Patrikalakis, editor, *Scientific Visualizaton of Physical Phenomena*, pages 485–530. Springer-Verlag,Tokyo, 1991.

[25] B. Guo. Non-splitting Macro Patches for Implicit Cubic Spline Surfaces. *Computer Graphics Forum*, 12(3):434 – 445, 1993.

[26] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweizer, and W. Stuetzle. Piecewise smooth surface reconstruction. *Computer Graphics*, 28:295–302, 1994.

[27] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. *Computer Graphics*, 26(2):71–78, 1992.

[28] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimzation. *Computer Graphics*, 27:19–26, Aug 1-6 1993.

[29] S. Lodha. *Surface Approximation with Low Degree Patches with Multiple Representations.* PhD thesis, Computer Science, Rice University, Houston, Texas, 1992.

[30] C. Loop. Smooth Subdivision based on triangles. Master's thesis, University of Utah, 1987.

[31] C. Loop. A G^1 triangular spline surface of arbitrary topological type. *Computer Aided Geometric Design*, 11:303–330, 1994.

[32] C. Loop. Smooth Spline Surfaces over Irregular Meshes. In *Proceedings of SIGGRAPH'94*, pages 303–310, July 1994.

[33] D. Moore and J. Warren. Approximation of dense scattered data using algebraic surfaces. In *Proceedings of the twenty-fourth Hawaii International Conference on System Sciences*, volume 1, pages 681–690, Kauai, Hawaii, 1991. IEEE, IEEE Computer Society Press, Los Alamitos, California.

[34] A. Nasri. Surface interpolation on irregular networks with normal conditions. *Computer Aided Geometric Design*, 8:89–96, 1991.

[35] G. Nielson, T. Foley, B. Hamann, and D. Lane. Visualizing and Modeling Scattered Multivariate Data. *IEEE Computer Graphics And Applications*, 11:47–55, 1991.

[36] J. Peters. Smooth interpolation of a mesh of curves. *Constructive Approximation*, pages 221–246, July 1991.

[37] J. Peters. Free-form surface splines. Technical Report CSD-TR-93-019, Computer Sciences Department, Purdue University, March 1993.

[38] J. Peters. Smooth free-form surface over irregular meshes generalizing quadratic splines. *Computer Aided Geometric Design*, pages 347–361, October 1993.

[39] J. Peters. C^1 surface splines. *SIAM Journal of Numerical Analysis*, 32(2):645–666, April 1995.

[40] M. Powell and M. Sabin. Piecewise quadratic approximations on triangles. *ACM Transactions on Mathematical Software*, 3:316–325, 1977.

[41] U. Reif. A unified approach to subdivision algorithms. Technical report, Mathematisches Institut A, Universität Stuttgart, 1992. Preprint 92-16.

[42] K. Rescorla. C^1 Trivariate Polynomial Interpolation. *Computer Aided Geometric Design*, 4:237–244, 1987.

[43] R. Sarraga. G^1 Interpolation of generally unrestricted Cubic Bezier Curves. *Computer Aided Geometric Design*, 4(00):23–29, 1987.

[44] A. Worsey and G. Farin. An n-dimensional Clough-Tocher element. *Constructive Approximation*, 3(2):99–110, 1987.

CSG Constructs for Free-form Solids Bounded by Implicit Algebraic Patches

Jai Menon
IBM T.J. Watson Research Center

Abstract

This chapter covers dual-representation [Brep, CSG] systems and summarizes algorithms for converting between Brep (Boundary representation) and CSG (Constructive Solid Geometry). Hard problems (such as separation) in Brep-to-CSG conversion, and the lack of exact CSG support for free-form solids are identified as the key limitations that led to the decline of dual [Brep, CSG] systems. The chapter then addresses these limitations and covers in details the topics of exact CSG and efficient Brep-to-CSG conversion for free-form solids bounded by implicit algebraic patches. A related CSG construct – Constructive Shell Representation (CSR) – provides an alternate complete representation scheme with potential applications that exploit CSR's hybrid Brep/CSG character. All CSG constructs can now be processed on massively parallel machines based on a CSG-architecture, In particular, several applications are run on the RayCasting Engine (RCE), including shading, Monte Carlo based realistic rendering, Booleans, sweeping, Minkowski operations, and Numerical Control (NC) machining.

1 Introduction

Free-form (sculptured) surfaces are modeled typically as a finite union of patches represented in the traditional parametric or the recently developed algebraic form. Parametric surfaces can be *implicitized*, but this results in surfaces of extremely high degrees, for example, a degree 18 algebraic surface from a bi-cubic parametric patch [26]. As a result, computing on parametric patches poses fundamental problems. For example, the intersection of two bi-cubic patches (degree 18 algebraic each) could result in a space curve of degree 18 x 18 = 324. Similar problems arise during curve/patch intersection calculations.

These and other limitations of parametric patches led to a recent line of work that seeks to construct free-form surfaces as a collection of algebraic patches. Each patch is defined as a low-degree implicit polynomial (degree 2 or 3) that is clipped by the walls of a tetrahedron, as shown in Fig. 1a. Typically, the vertices of the tetrahedron and additional points in the boundary of the tetrahedron prescribe the control points for the patch. A weight is associated with each control point; for example, changing the weight controls the shape.

In this chapter, we will study:

- *what are dual-representation systems, and why have these faded?*
  these systems support consistent [Brep, CSG] representation schemes; good in principle, but faded away due to lack of appropriate technology, especially for free-form solids (Brep: Boundary representation; CSG: Constructive Solid Geometry). (Section 2.)

- *how does one interconvert between Brep and CSG?*
  through generic generate and test paradigms, with some subtleties. (Section 3.)

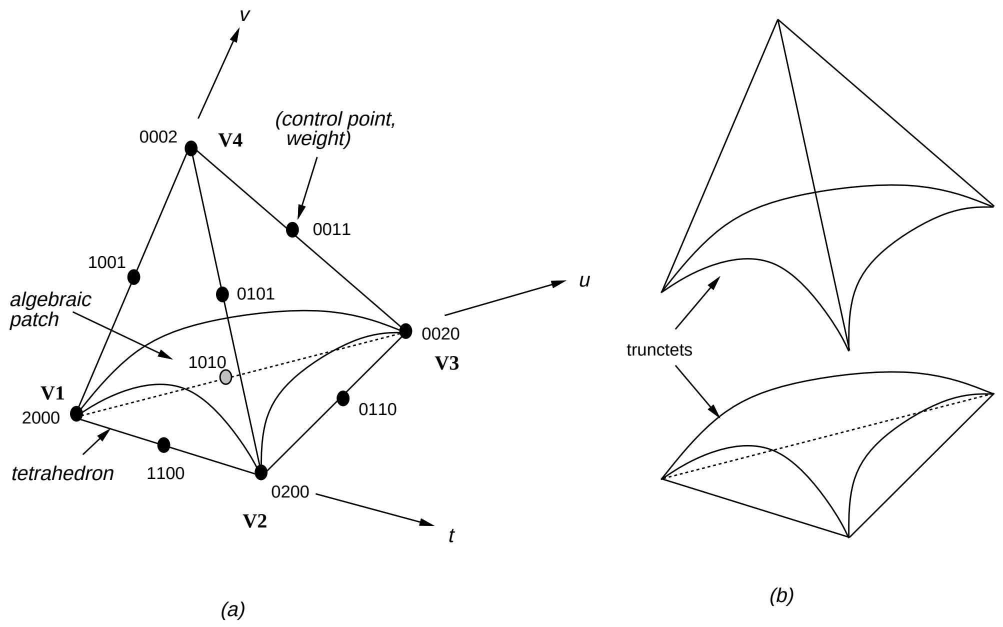


Figure 1: (a) An algebraic patch and (b) associated trunctet solids.

- *can dual-rep systems be resurrected for free-form solids?*
  yes (in principle); implicit algebraic patches can be used to construct Breps (we provide a brief overview), and efficient procedures can be developed to maintain consistent CSG (we provide details in Sections 4 – 8).

- *what are the applications of CSG constructs for implicit algebraic patches?*
  CSG constructs provide direct access to (for example) massively parallel processing on the RCE (RayCasting Engine) for a wide variety of applications (shading, ray tracing, sweeping, machining, etc.); furthermore, new techniques based on hybrid Brep/CSG processing on CSRs (CSR: Constructive Shell Representation) support another class of applications. (Sections 9, 10.)

2 Modeling System Architectures

From the mid-1970s until the later 1980s, two generic architectures of solid modeling systems predominated: systems in which all applications are based on Boundary Representations (Breps), and systems maintaining two or more consistent representations, usually Brep and Constructive Solid Geometry (CSG) [31]. Dual representation [Brep, CSG] systems were deemed promising in the early 1980s since they exploit complementary strengths of Brep and CSG schemes [22]. For example, Breps provide natural means to interrogate or tweak (edit) boundaries, while CSG alleviates the problems related to validity, non-manifold topology, and robustness. Yet, CSG and [Brep, CSG] systems were abandoned commercially, mainly for the following reasons [31]:

Domain limitation: CSG technology could accommodate prismatic solids (e.g. piston rods), but not free-form geometry (e.g. turbine blades). Parametric patches, used pervasively in Brep

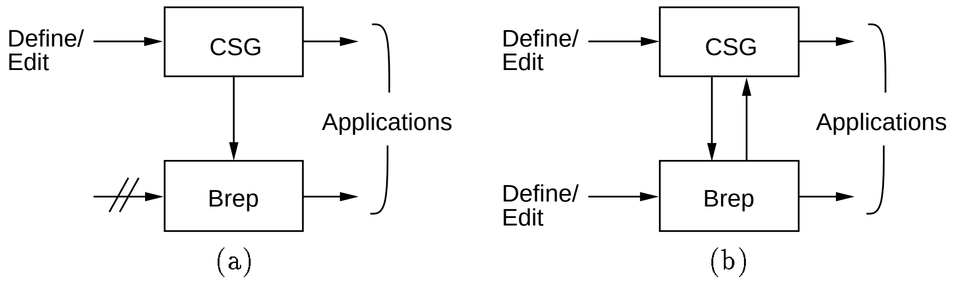


Figure 2: Dual representation modeling system architectures: (a) unilateral, (b) bilateral.

systems, induce halfspaces of high implicit degrees (e.g. degree 18 halfspace for a bi-cubic tensor product patch [26]), thereby leading to intractable separation [28], combinatorial, and numerical problems in Brep-to-CSG conversion. This partially explains the surge in single-representation Brep systems in the later 1980s and early 1990s.

System limitation: The development of reliable CSG-to-Brep conversion algorithms [23] spurred the emergence of solid modeling in the 1970s, resulting in *unilateral* dual representation systems with the architecture shown in Figure 2a. It was in the early 1990s that mathematically sound inverse conversion algorithms were developed, and experimental versions of *bilateral* dual representation systems (Figure 2b) are only now under development [27, 28]. Yet, these systems are limited to prismatic solids bounded by natural quadrics.

3 Converting Between Brep and CSG

This section summarizes the key concepts for inter-converting between Brep and CSG representations.

3.1 CSG-to-Brep conversion

It is useful to distinguish between two forms [23]:

- *boundary evaluation* and
- *boundary merging.*

For a solid A, boundary evaluation does a CSG(A) $\rightarrow$ Brep(A) conversion, i.e. it computes a boundary representation of solid A from a CSG representation of A. Two forms of boundary evaluators are worth distinguishing. The first operates on the CSG tree of A and its primitives, and is called "non-incremental" boundary evaluation. The second is called "incremental", because it uses information contained in the Breps of the subtrees of A. Incremental evaluation is more suitable for interactive modeling, but non-incremental evaluation is appropriate when, for example, a CSG definition is retrieved from archival memory.

Boundary merging computes (Brep(A), Brep(B), op_n) $\rightarrow$ Brep(A op_n B), i.e. a Brep of a regularized Boolean composition [22] from Breps of its operands. Here op denotes a conventional set operation: $\cup$, $\cap$, or $-$ (union, intersection, or difference).

The classical embodiment of boundary evaluation is a generate-and-test paradigm: to build a Brep, generate a sufficient set of candidate entities from the CSG, and then test and trim each

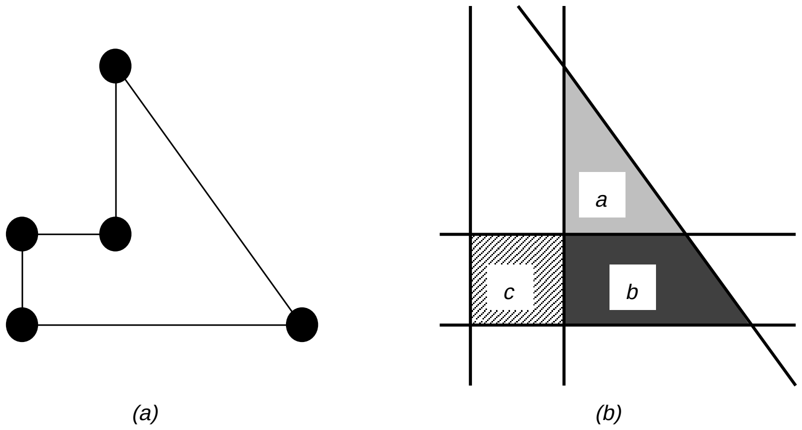


Figure 3: (a) Brep of a solid, and (b) cellular decomposition induced by halfspaces.

entity for inclusion in the resulting Brep. The candidates in this case are the boundaries of all halfspaces h_i in the CSG representation of A, because it is easy to prove that

$$\bigcup \partial h_i \supset \partial A, \quad \forall\, h_i \in \mathrm{CSG}(A), \tag{1}$$

where ∂ is the "boundary of" operator; thus ∂A denotes the boundary of A.

The critical calculation is the classification [30] of a halfspace boundary, i.e. a surface, with respect to a solid; see [23] for details.

3.2 Brep-to-CSG conversion

The key ideas follow from the observation that two linear halfspaces h_1, h_2 in E^2 decompose the world W into four *cells*, each described by a "product" term $g_1 \cap g_2$, where $g_i \in \{h_i, \overline{h_i}\}$ and $\overline{h_i}$ is the regularized complement of h_i. Thus, the two linear halfspaces induce a *disjunctive decomposition* of W as

$$W = (h_1 \cap h_2) \cup (\overline{h_1} \cap h_2) \cup (h_1 \cap \overline{h_2}) \cup (\overline{h_1} \cap \overline{h_2}). \tag{2}$$

The cell-based method for converting from Brep-to-CSG follows immediately as [27]:

1. Induce a set of halfspaces $H = \{h_i\}$ from the faces of the Brep.

2. Represent the cells of the disjunctive decomposition of W induced by H.

3. Classify each cell (e.g. by classifying any interior point) with respect to the Brep to determine if the cell is "in" or "out" of the solid represented by the Brep (in Fig. 3b, only cells *a, b, c* are "in").

4. Construct a canonic CSG representation as the union of the product terms representing the "in" cells.

5. Simplify (minimize) the CSG representation using procedures in Boolean logic.

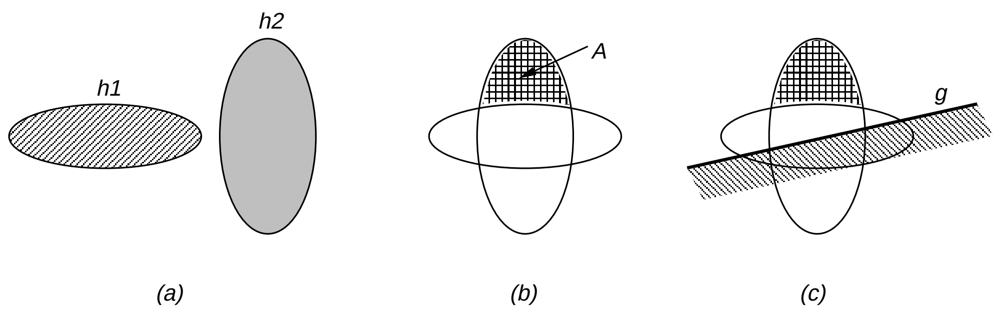


Figure 4: (a) Two halfspaces, (b) the solid, and (c) an additional non-unique separating halfspace necessary for describing the solid in CSG.

Subtleties arise for the case of curved solids. Fig. 4a shows two elliptical halfspaces h_1 and h_2 in E^2. When they overlap, as in Fig. 4b, space is divided into six disjoint cells but the disjunctive decomposition of W expressed as combinatorial intersections of the elliptical halfspaces (and their complements) contains only four terms. Two of these – $h_1 \cap \overline{h_2}$ and $\overline{h_1} \cap h_2$ – represent disconnected regions which each contain two connected cells. As a consequence, the hatched cell A in Fig. 4b is not representable in CSG using only $\{h_1,\ h_2,\ \overline{h_1},\ \overline{h_2}\}$; an additional halfspace g (Fig. 4c) is needed to *separate* the cells represented by $\overline{h_1}\ \cap\ h_2$. Note that many halfspaces could play this role. The requirement for separability dictates an expansion of Step 1 above, into three parts:

- Induce a set of *natural* halfspaces $H_N\ =\ \{h_i\}$ from the Brep.
- Construct a sufficient set of separating halfspaces $H_S\ =\ \{g_j\}$ from H_N (and the Brep, if useful).
- Construct the set H as the union of H_N and H_S.

Finding good methods to construct H_S is still a major research problem, and methods are known only for some cases of quadratic implicit surfaces [28]. Methods for reducing the computation needed in Steps 2, 3, and 5 of the above procedure are other research challenges.

4 Free-form Boundary Construction: A Quick Overview

This section develops the basic concept of an algebraic patch, and then briefly summarizes the key concepts for meshing these patches with prescribed continuity.

4.1 Bernstein-Bezier (BB) Form

Let N denote the degree of an implicit algebraic surface, e.g $N\ =\ 2$ for a quadric surface. Consider a tetrahedron with vertices $\mathbf{V}_{N000}$, $\mathbf{V}_{0N00}$, $\mathbf{V}_{00N0}$, and $\mathbf{V}_{000N}$, where the $\mathbf{V}$s are non-coplanar points in E^3 (Fig. 1a). Let (s,t,u,v) denote the barycentric coordinates in the tetrahedron. By definition, the barycentric coordinates of a point $\mathbf{q}$ are the values of (s,t,u,v) such that

$$\mathbf{q}\ =\ s\mathbf{V}_{N000}\ +\ t\mathbf{V}_{0N00}\ +\ u\mathbf{V}_{00N0}\ +\ v\mathbf{V}_{000N}\ ,s+t+u+v\ =\ 1. \qquad (3)$$

Let $a(s,t,u,v)$ denote a polynomial scalar function. A contour surface of the function comprises all points for which a is constant. The algebraic patch is defined as the zero contour of the function that is clipped by the tetrahedron. The Bernstein-Bezier polynomial provide a convenient basis to control the behavior of the zero contour within the tetrahedron [25]. Specifically, a degree N algebraic patch can be defined by first imposing a lattice of $(N+1)(N+2)(N+3)/6$ control points $\mathbf{c}_{ijkl}$ such that

$$\begin{aligned} \mathbf{c}_{ijkl} &= \frac{i}{N}\mathbf{V}_{N000} + \frac{j}{N}\mathbf{V}_{0N00} + \frac{k}{N}\mathbf{V}_{00N0} + \frac{l}{N}\mathbf{V}_{000N} \\ i,j,k,l &\geq 0, \quad i+j+k+l = N. \end{aligned} \tag{4}$$

Fig. 1a shows the lattice of control points for a quadratic algebraic patch. This lattice defines the control net for the patch, and its convex hull is the tetrahedron itself. Next, we assign a weight w_{ijkl} to each control point using Bernstein-Bezier basis functions as

$$\begin{aligned} a(s,\ t,\ u,\ v) &= \sum_{i,j,k,l\geq 0} w_{ijkl}\frac{N!}{i!\ j!\ k!\ l!}s^i t^j u^k v^l \\ i+j+k+l = N,&\quad s,t,u,v = 1-s-t-u \geq 0. \end{aligned} \tag{5}$$

Non-tetrahedral techniques for constructing algebraic patches may also be used; this chapter however focuses on tetrahedron-based patches.

4.2 Mesh of Patches

Algebraic patches are meshed together to form extended smooth surfaces. A Brep consisting of algebraic patches is usually obtained by smoothing an input polyhedron P. Some methods *interpolate* vertices and vertex normals of P (e.g. [16]), while others *approximate* P (e.g. [1]). In this chapter, we shall focus on polyhedral smoothing via interpolation of vertices of P.

Most existing methods for meshing algebraic patches in the Bernstein-Bezier form (BB-form) are tetrahedron-based: these methods first build a *polyhedral hull* [4] consisting of tetrahedra, and then construct an algebraic patch inside each tetrahedron. Popular methods include the following:

- **Quadratic methods:** Dahmen uses quadratic patches to construct interpolants to the vertices of a piecewise linear surface while matching prescribed normals [3]. He was the first to present the idea of a polyhedral hull, an idea that has a fundamental influence on later work. Unfortunately, Dahmen's quadratic method is based on 'transversal systems', which are difficult to construct. We mesh quadratic patches with G^1 continuity using the method described in [8]. This method requires a set of *compatible* vertex normals.

- **Cubic methods:** Both [7] and [26] use Clough-Tocher splitting to construct piecewise cubic interpolants. Dahmen and Thamm-Schaar show that the splitting can be avoided in many cases [4]. A difficulty with cubic patches is that they can easily have multiple sheets inside their bounding tetrahedra. To eliminate the extraneous sheets, Bajaj, Chen, and Xu introduce the A-patch technique [1]. We mesh cubic patches with C^1 continuity [10, 7].

The interpolative smoothing of P proceeds as follows. The sculptured surface S which constitutes the Brep of solid A, denoted Brep(A), is constructed by interpolating the vertices and vertex normals of an input polyhedron P having triangular faces. Figure 5a provides a simple 2D example. Given P, we build a polyhedral hull H as a triangulated polytope consisting of *face* and *edge* (or *wedge*) tetrahedra associated respectively with the faces and edges of P, as shown in Figure 5b. The hull is divided into *exterior* and *interior* parts, depending on the sense of the

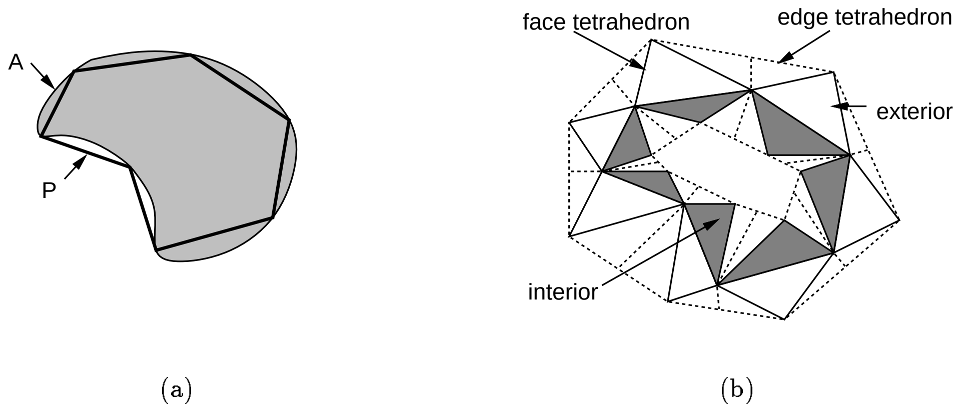


Figure 5: In 2D: (a) polyhedral smoothing, (b) polyhedral hull with face tetrahedra (exterior - unshaded, interior - shaded) and gap-filling edge tetrahedra (dotted).

tetrahedra relative to the material side of P. The control net for the surface consists of *interpolative* control points (vertices of P) and *non-interpolative* control points (apexes of tetrahedra).

For the specific case of quadratic algebraic patches, we require *compatible normals*, i.e. the average normal $\frac{1}{2}(\mathbf{n}_i + \mathbf{n}_j)$ is perpendicular to the edge $[\mathbf{v}_i\mathbf{v}_j]$ of a triangle with vertices $\mathbf{v}_k$ and normals $\mathbf{n}_k$ $(k = 1, 2, 3)$. We use subdivision methods [16] to generate a new polyhedron $\widehat{P}$ with more number of faces that: (a) retains the original vertices and normals of P, and (b) guarantees normal compatibility.

The surface S is obtained by constructing algebraic patches inside the tetrahedra of H. A polynomial $a_i(\mathbf{x})$ is defined on each tetrahedron T_i, and the polynomials of all tetrahedra determine a piecewise polynomial function f with the following properties: (a) f is continuously differentiable on H, (b) the boundary of A is $S = \{\mathbf{x} \in \mathsf{H} \mid f(\mathbf{x}) = 0\}$, (c) at each point $\mathbf{x} \in S$, the gradient $\nabla f(\mathbf{x})$ points to the outside of A, and (d) at each vertex $\mathbf{v}_i$ of the input polyhedron P, $f(\mathbf{v}_i) = 0$, i.e. S interpolates the vertices of P. If a tetrahedron T_i of H contains a nontrivial portion of the zero contour of the corresponding polynomial $a_i(\mathbf{x})$ (i.e. $T_i \cap_{k-1} S(a_i) \neq \emptyset$), then we say that T_i is *patch-containing*; otherwise T_i is referred to as an *empty tetrahedron*. The algebraic patches inside the face tetrahedra are called *face patches*, while those inside the edge tetrahedra are *blend patches*.

4.3 Hull Validity

Given a valid hull, constructing a G^1 (C^1) continuous mesh of quadratic (cubic) algebraic patches, reduces to the determination of appropriate Bezier coefficients of face and edge tetrahedra; see [16] and references therein. However, constructing a valid polyhedral hull from P (and similarly from $\widehat{P}$) that guarantees a sculptured surface to smooth a given P is a hard problem. We use local and global constraints to guarantee this. While local constraints are necessary (except visibility), but not sufficient, the combination of local and global constraints are sufficient but not necessary.

Local validity imposes the following three constraints related to tetrahedra that share a vertex or edge of P (see Figure 6a). By symmetry, we list these only for the exterior hull; similar conditions apply for the interior hull.

- *Tangent containment:* For each vertex $\mathbf{x}_i$ of a face F_k of P, the apex $\mathbf{v}_k^+$ of the exterior face

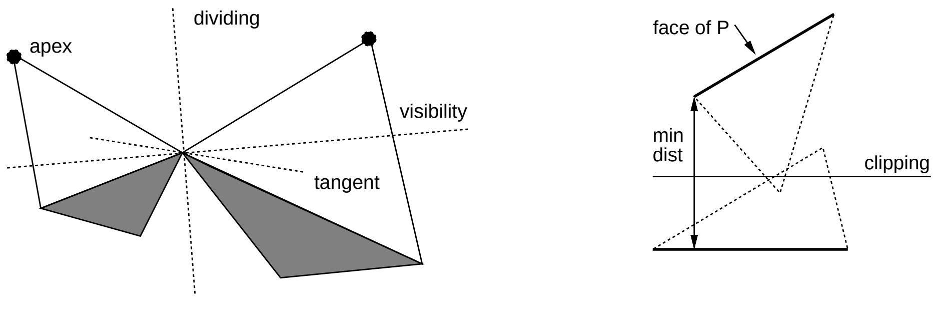


Figure 6: In 2D: (a) local hull validity planes, and (b) global hull validity planes.

tetrahedron satisfies $(\mathbf{v}_k^+ - \mathbf{x}_i) \cdot \mathbf{n}_i > 0$.

- *Wedge validity:* For each pair of faces F_i and F_k sharing an edge E_{ik}, the corresponding exterior face tetrahedra intersect only along the edge E_{ik}.
- *Visibility:* For each pair of faces F_i and F_k, the corresponding apexes $\mathbf{v}_i^+$ and $\mathbf{v}_k^+$ are mutually visible.

We construct a feasible apex set region from the *tangent*, *dividing*, and *visibility* planes that are used to satisfy the above three conditions respectively; see Figure 6a. For each edge E_{ik} of the input polyhedron P, we set the dividing plane to pass through E_{ik} and bisect the dihedral angle formed by the two faces F_i and F_k sharing E_{ik}. The visibility plane is chosen to pass through the edge E_{ik} and be perpendicular to the dividing plane. This leads to ten linear inequality constraints (3-tangent, 3-dividing, 3-visibility, 1-face) for each face F of P. The apex of F is the point closest to the centroid of F in the convex set defined by the ten linear inequality constraints. This apex is found through solving a least-distance programming problem [10]. In case the apex reaches the centroid, we elevate the apex slightly to avoid a flattened tetrahedron. In addition, the *existence* of a valid polyhedral hull also requires the satisfaction of a linear form of Kuhn-Tucker conditions for the interpolative control points [10]. If this condition is violated, we use subdivision methods to "smooth" out the sharp features of P.

Global constraints require that all pairs of tetrahedra in the polyhedral hull be quasi-disjoint. This guarantees that the free-form surface does not self-intersect and also simplifies the CSG computation (below). We use *clipping* planes, as shown in Figure 6b, to satisfy this global condition. The goal is to select clipping planes that maximizes the volume of the polyhedral hull in order to provide maximum freedom in constructing the free-form surface. For a pair of faces (or edges) of P, we choose the clipping plane to be the perpendicular bisector of the line segment joining the closest points on the pair.

5 Other Approaches for Free-form CSG

Very little of the work on parametric patches has proven applicable to CSG representations. The main reason is that halfspaces induced via implicitization of parametric patches produce high degree polynomials (e.g. a degree 18 halfspace for a bi-cubic tensor product patch [26]), leading to difficult separation [28], combinatorial, and numerical problems in Brep-to-CSG conversions. Chan [2] represents free-form solids by extending the leaves of CSG trees to include parametric patches, but this hybrid representation fails to fully exploit CSG's elegant divide-and-conquer algorithms, often degrading CSG's robustness. Dunnington, Saia and de Pennington [5] construct polyhedral approximations using an Inner Set Outer Set (ISOS) approach, yielding polyhedra that are either contained in the solid or that contain the solid. The ISOS methods are not applicable to free-form solids of arbitrary topology because of the restrictions imposed by a 'radial visibility criterion'.

Metaballs provide a different approach for representing free-form shapes, with widespread commercial implementation. However, metaballs have some limitations, since not every algebraic surface can be represented therewith. Furthermore, we are not aware of direct *low-degree* algebraic halfspace based CSG formulations using metaballs. As an example, Wyvill and Van Overveld [32] use boolean combinations of 'soft' objects (objects bounded by general implicit surfaces). They demonstrate that the boundaries of solids in constructive 'soft' geometry can be polygonalized efficiently. However, in general, bounding faces are not even algebraic (let alone high degree polynomial), leading to complicated halfspace separation problems. Furthermore, primitives have a global effect, thus making local shape control operations difficult.

6 Free-form CSG Computation from Algebraic Patches

This section develops a general CSG solution and derives the *trunctet-subshell* conditions under which CSG constructions become simple and efficient. The general solution is complete in that it will always provide a CSG representation for a solid (in both normal and abnormal constructions), as long as the solid has a valid Brep and every patch satisfies an *orientation consistency* condition. The trunctet-subshell conditions are found to be satisfied in most examples, thus making CSG computation simple and efficient in practice. These conditions can either be built into the algorithms for meshing algebraic patches, or can be applied dynamically.

6.1 Normal and Abnormal Constructions

Let the set of nonempty (patch-containing) tetrahedra be $\{T_i\}$. We distinguish between a *normal construction* where all tetrahedra are quasi-disjoint ($T_i \cap_k T_j = \emptyset, \forall ij, i \neq j$) and an *abnormal construction* where at least one pair of tetrahedra overlap ($T_i \cap_k T_j \neq \emptyset$, for some $i \neq j$). A 2D analog of a free-form solid with a normal construction that interpolates the vertices of a polygon is shown in Figure 7a; the absence of blend patch constructs makes it easier to follow the figure. For clarity, we shall use mainly 2D illustrations henceforth, although all arguments hold for 3D solids, including face and blend patches. Figure 7b thus gives an example of an abnormal construction.

6.2 Trunctets

Consider a tetrahedron T containing an algebraic patch $\widetilde{F}$ belonging to the boundary of a free-form solid A. This patch can be expressed as $\widetilde{F} = S(a) \cap_{k-1} T$, where $S(a)$ is the algebraic surface $\{\mathrm{x} \mid a(\mathrm{x}) = 0\}$ and $a(\mathrm{x})$ is a polynomial in BB-form. The tetrahedron T can be expressed in the CSG scheme as the intersection of four linear halfspaces whose boundaries contain the triangular faces of T. The algebraic surface $S(a)$ decomposes the tetrahedron into two point sets: $T \cap_k \mathtt{A}$ and $T \cap_k \overline{\mathtt{A}}$, where $\mathtt{A} = \{\mathrm{x} \mid a(\mathrm{x}) \leq 0\}$ is an *algebraic patch halfspace* (or simply algebraic halfspace) and its complement $\overline{\mathtt{A}} = \{\mathrm{x} \mid a(\mathrm{x}) \geq 0\}$. Each point set may be viewed as a tetrahedron that is truncated (or capped) by the patch, and therefore termed as a *trunctet*, denoted $\mathcal{T}$ (recall Figure 1). Since our polyhedral smoothing procedure ensures that $T \cap_k \mathtt{A}$ has the same sense as the inside of

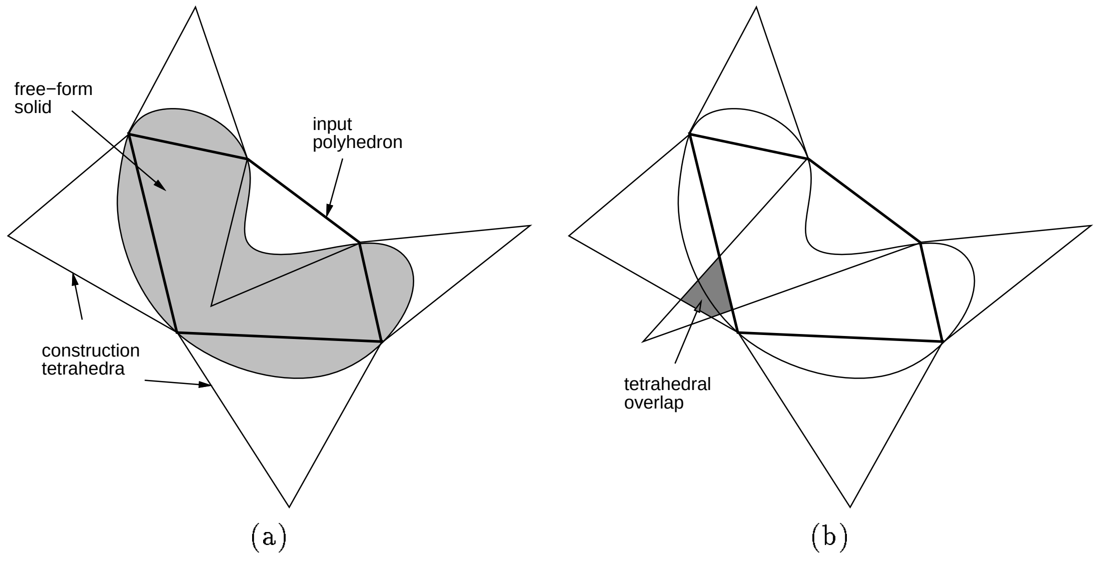


Figure 7: (a) A normal construction, and (b) an abnormal construction in 2D.

the solid A, we define patch $\widetilde{F}$ to be associated with an *inner trunctet* ${}^{\mathcal{I}}\mathcal{T} = T \cap_k \mathtt{A}$ and an *outer trunctet* ${}^{\mathcal{O}}\mathcal{T} = T \cap_k \overline{\mathtt{A}}$. Inner and outer trunctets are quasi-disjoint (${}^{\mathcal{I}}\mathcal{T} \cap_k {}^{\mathcal{O}}\mathcal{T} = \emptyset$) and their union is the tetrahedron itself (${}^{\mathcal{I}}\mathcal{T} \cup_k {}^{\mathcal{O}}\mathcal{T} = T$).

An algebraic patch $\widetilde{F}$ on the boundary of the solid A is either a protrusion or a depression relative to the input polyhedron P. The protrusions and depressions are associated with *polyhedral trunctets*, each denoted as ${}^{\mathcal{P}}\mathcal{T}$. A protrusion corresponds to the inner trunctet ${}^{\mathcal{I}}\mathcal{T}$ of an exterior (face or edge) tetrahedron, while a depression corresponds to the outer trunctet ${}^{\mathcal{O}}\mathcal{T}$ of an interior tetrahedron (see Figure 8). Two related concepts are *inner polyhedral trunctet* ${}^{\mathcal{IP}}\underline{\mathcal{T}}$ and *outer polyhedral trunctet* ${}^{\mathcal{OP}}\underline{\mathcal{T}}$, defined as follows.

$$
{}^{\mathcal{IP}}\underline{\mathcal{T}} = \begin{cases} \emptyset & \text{if } {}^{\mathcal{P}}\mathcal{T} = {}^{\mathcal{O}}\mathcal{T} \\ {}^{\mathcal{P}}\mathcal{T} & \text{if } {}^{\mathcal{P}}\mathcal{T} = {}^{\mathcal{I}}\mathcal{T} \end{cases} \quad \text{and} \quad {}^{\mathcal{OP}}\underline{\mathcal{T}} = \begin{cases} \emptyset & \text{if } {}^{\mathcal{P}}\mathcal{T} = {}^{\mathcal{I}}\mathcal{T} \\ {}^{\mathcal{P}}\mathcal{T} & \text{if } {}^{\mathcal{P}}\mathcal{T} = {}^{\mathcal{O}}\mathcal{T} \end{cases}
$$

Thus a protrusion is always associated with a ${}^{\mathcal{IP}}\underline{\mathcal{T}}$ and a depression with a ${}^{\mathcal{OP}}\underline{\mathcal{T}}$. While every patch is associated with an inner, outer, and polyhedral trunctet, a patch cannot be associated simultaneously with an inner and an outer polyhedral trunctet. We use underlines in the notation to highlight this mutually exclusive character of trunctets.

6.3 Separation for Trunctets

In general, there is no restriction on the number of connected components of the zero contour of $a(\mathbf{x})$ in T. For example, T may contain 1 (Figure 1a), or 2 (T^- in Figure 9a), or 4 (T^+ in Figure 9a) connected components of the zero contour. Of these, some connected components may be valid, i.e. contribute to ∂A, while others may be extraneous (do not belong to ∂A), as shown in Figure 9a. In cases of extraneous zero contours, additional separating halfspaces (e.g. $\mathtt{G}^+, \mathtt{G}^-$ in Figure 9) are required to describe the inner and outer trunctets in the CSG scheme.

In general, a *cell* induced from a set $G = \{\mathtt{H}_1, \mathtt{H}_2, \cdots, \mathtt{H}_m\}$ of halfspaces is defined as the point set represented by the regularized intersection of the form $\mathtt{G}_1 \cap_k \ldots \cap_k \mathtt{G}_m$, where the halfspace

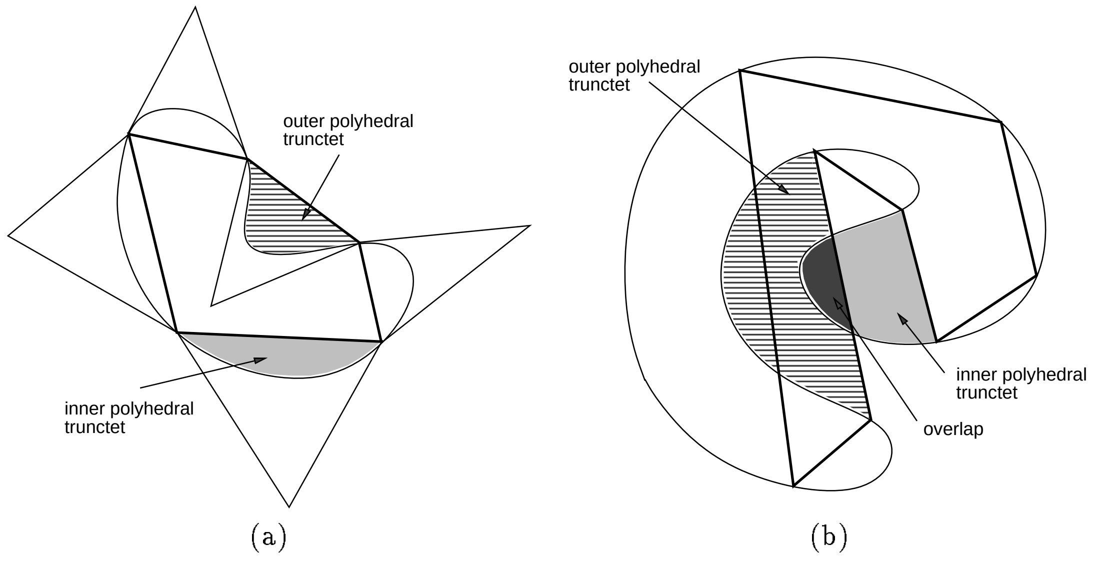

outer polyhedral trunctet
inner polyhedral trunctet
(a)
outer polyhedral trunctet
inner polyhedral trunctet
overlap
(b)

Figure 8: Polyhedral trunctets: (a) normal construction, and (b) abnormal construction where a patch may not lie entirely inside or outside P.

$\mathtt{G}_i \in \{\mathtt{H}_i, \overline{\mathtt{H}}_i\}, i = 1, m$. The following theorem [28] gives a condition for describing any solid in CSG using the halfspaces in G.

Describability Theorem for CSG: Given a set $G = \{\mathtt{H}_1, \mathtt{H}_2, \cdots, \mathtt{H}_m\}$ of halfspaces, and a solid B such that $\partial B \subseteq \partial\mathtt{H}_1 \cup \partial\mathtt{H}_2 \cup \cdots \cup \partial\mathtt{H}_m$, there exits CSG($B$) if and only if all connected components of the cells of B have the same classification ('in' or 'out') with respect to B. For brevity, B is said to be *describable* by G.

When B is a polyhedron, G is simply the set of *natural* halfspaces, which in this case are the linear halfspaces whose boundaries contain the faces of B. On the other hand, if B is a curved solid, then G contains both the set of natural halfspaces and additional *separating* halfspaces, which are needed to ensure that all connected components of a cell have identical classification with respect to B.

Figure 9 shows how additional separating halfspaces are used in CSG descriptions of trunctets. Observe that trunctets may have multiple connected components, e.g. the inner trunctet of T^+ in Figure 9b. In this example, $\mathtt{G}^+$ would have been unnecessary if there was no extraneous zero contour in T^+, even though the zero contour has several connected components in T^+. The following properties hold for separation in trunctets.

Linear separation for trunctets: For an algebraic patch $\widetilde{F}$ of any degree, there exists a sufficient set of linear separating halfspaces for describing CSG($^{\mathcal{ANY}}\mathcal{T}$) and CSG($^{\mathcal{ANY}}\underline{\mathcal{T}}$).

Self-separating trunctets: For an algebraic patch $\widetilde{F}$ of any degree, if a patch $\widetilde{F}$ is single-sheeted, no additional separating halfspace is required for describing CSG($^{\mathcal{ANY}}\mathcal{T}$) and CSG($^{\mathcal{ANY}}\underline{\mathcal{T}}$).

The linear separation property follows because the tetrahedron-based algebraic patch techniques build patches whose edges are planar (e.g. an edge of $\widetilde{F}$ lies in a face of T). For any solid whose boundary faces have only planar edges, there exists a sufficient set of linear separators [28].

Further simplification can be achieved for algebraic halfspaces induced from single-sheeted patches. Following Bajaj, Chen, and Xu [1], we define a face patch $\widetilde{F}$ bounded by a tetrahedron T to be *single sheeted* if any ray fired from the apex of T intersects $\widetilde{F}$ at most once inside T. A single-sheeted blend patch is similarly defined; see [1] for details. When $\widetilde{F}$ is a single-sheeted patch, the corresponding halfspace $\mathtt{A}$ decomposes T into exactly two cells, and all connected components of a cell either belong to an inner trunctet or an outer trunctet. This obviates the computation of separating halfspaces, thus giving rise to *self-separating* trunctets.

6.4 Shells and Cores

A *shell* $\mathcal{S}$ is defined as the regularized union of trunctets, one associated with each patch, irrespective of whether the trunctet is inner or outer. Similarly, we define an *inner shell* ${}^{\mathcal{I}}\mathcal{S}$, an *outer shell* ${}^{\mathcal{O}}\mathcal{S}$, and a *polyhedral shell* ${}^{\mathcal{P}}\mathcal{S}$ (e.g. ${}^{\mathcal{I}}\mathcal{S} = {}^{\mathcal{I}}\mathcal{T}_1 \cup_k {}^{\mathcal{I}}\mathcal{T}_2 \cup_k \cdots \cup_k {}^{\mathcal{I}}\mathcal{T}_n$). An *inner polyhedral subshell* ${}^{\mathcal{IP}}\underline{\mathcal{S}}$ is the union of all the inner polyhedral trunctets, one may or may not be associated with a patch (hence the underline). The *outer polyhedral subshell* ${}^{\mathcal{OP}}\underline{\mathcal{S}}$ is similar. By definition, ${}^{\mathcal{IP}}\underline{\mathcal{S}} \subseteq {}^{\mathcal{P}}\mathcal{S}$ and ${}^{\mathcal{OP}}\underline{\mathcal{S}} \subseteq {}^{\mathcal{P}}\mathcal{S}$. Subshells are often quasi-disjoint (Figure 8a), although they could overlap sometimes, i.e. ${}^{\mathcal{IP}}\underline{\mathcal{S}} \cap_k {}^{\mathcal{OP}}\underline{\mathcal{S}} \neq \emptyset$, e.g. in Figure 8b. The subshell overlap region $\underline{\mathcal{O}}$ is defined as $\underline{\mathcal{O}} = {}^{\mathcal{IP}}\underline{\mathcal{S}} \cap_k {}^{\mathcal{OP}}\underline{\mathcal{S}}$.

A shell is naturally associated with a *core* $\mathcal{C}$, which is the region of solid A that is not contained in the shell, and defined as $\mathcal{C} = A -_k \mathcal{S}$. Thus, we have inner (${}^{\mathcal{I}}\mathcal{C}$), outer (${}^{\mathcal{O}}\mathcal{C}$), and polyhedral (${}^{\mathcal{P}}\mathcal{C}$) cores. Inner polyhedral (${}^{\mathcal{IP}}\underline{\mathcal{C}}$) and outer polyhedral (${}^{\mathcal{OP}}\underline{\mathcal{C}}$) subcores are similarly defined.

A *Constructive Shell Representation* of A is a CSG representation of a shell (not subshell):

$$\mathrm{CSR}(A) = \mathrm{CSG}({}^{\mathcal{ANY}}\mathcal{S}), \tag{6}$$

which is the union of CSG representations of trunctets, one for every patch. CSR(A) is a *complete* representation of A in that it can represent a free-form solid unambiguously and that it contains sufficient information to answer any geometric query about A [12]. A CSR is a non-unique representation, not only because it is a CSG tree [22], but also because the point sets represented by CSRs (shells in this case) are also non-unique [12].

A CSR is a hybrid Brep/CSG representation since characteristics of both Brep and CSG schemes are prevalent. Like CSG, it is a binary tree with primitive halfspaces for leaves and regularized boolean operations for nodes. Like Brep, a CSR may be thought of as representing a 'thick' boundary of A. This hybrid Brep/CSG character allows us to exploit the algorithmic conveniences of both schemes – specifically, the execution of boundary traversal algorithms (typical for Breps), and the use of divide-and-conquer methods (typical for CSG). For example, a new algorithm for classifying a line against CSR(A) exploits this hybrid Brep/CSG character [15], which in turn lends to massively parallel processing for supporting graphics and modeling applications [13].

6.5 A General CSG Expression

We use the above concepts to derive a CSG expression of A. Observe that a free-form solid A is constructed by essentially replacing the faces of the input polyhedron P by patches, with each patch introducing a protrusion or depression relative to P. Our approach to constructing CSG mimics this behavior, namely, we first union all protrusion trunctets (${}^{\mathcal{IP}}\underline{\mathcal{S}}$ subshell) to P, and then subtract all the depression trunctets (${}^{\mathcal{OP}}\underline{\mathcal{S}}$ subshell) from the union. However, if the subshells overlap (Figure 8b), subtracting ${}^{\mathcal{OP}}\underline{\mathcal{S}}$ will remove some extra material (Figure 10a).

In general, we have

$$(P \cup_k {}^{\mathcal{IP}}\underline{\mathcal{S}}) -_k {}^{\mathcal{OP}}\underline{\mathcal{S}} \subseteq A \tag{7}$$

$$(P -_k {}^{\mathcal{OP}}\underline{\mathcal{S}}) \cup_k {}^{\mathcal{IP}}\underline{\mathcal{S}} \supseteq A \tag{8}$$

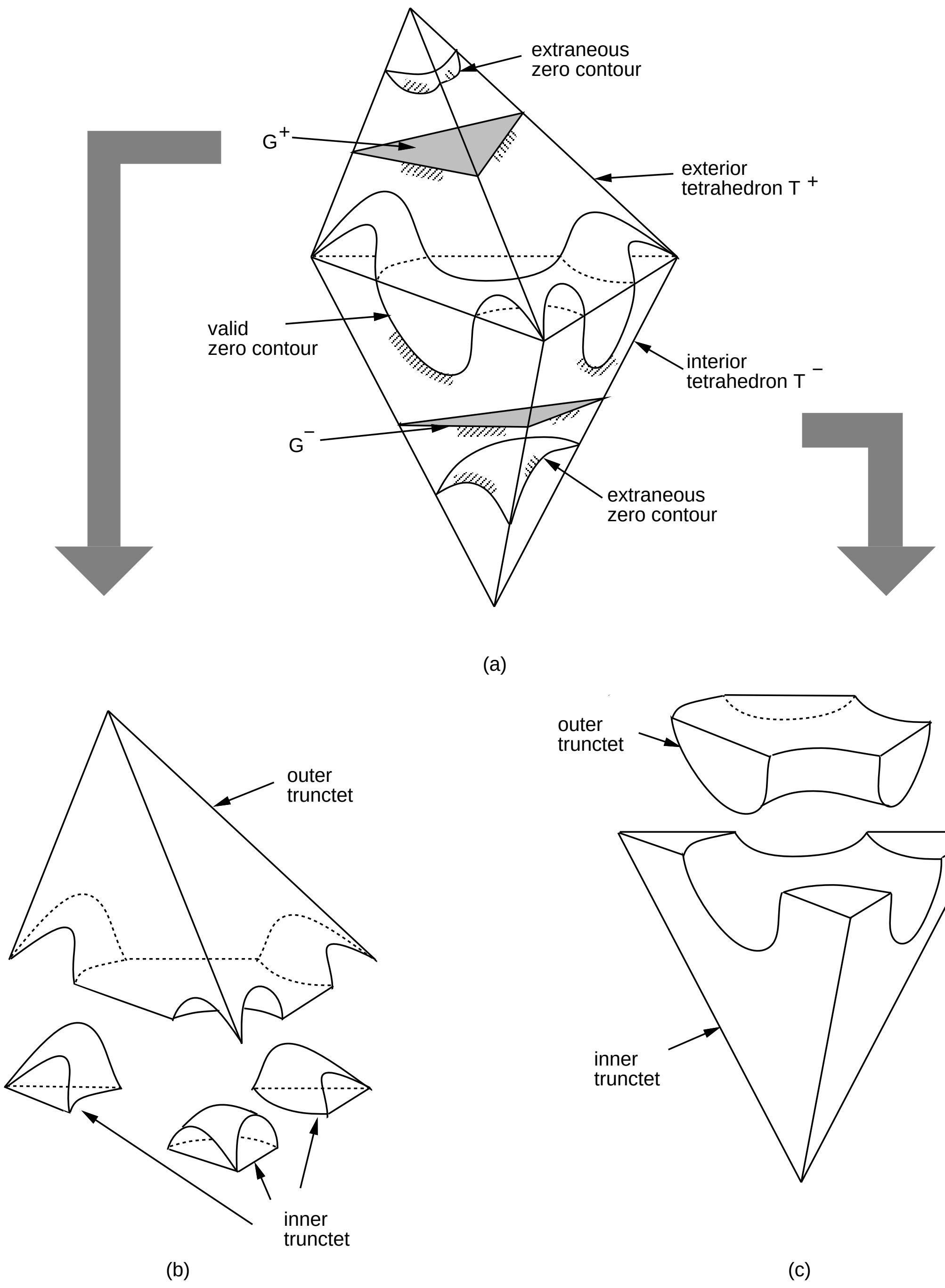


Figure 9: (a) A pair of face-adjacent exterior and interior tetrahedra, with a valid zero contour, and some extraneous zero contours. Separating halfspaces $\mathtt{G}^+, \mathtt{G}^-$ are required to describe the associated trunctets. Specifically, (b) in the exterior tetrahedron: $\mathrm{CSG}({}^{\mathcal{I}}\mathcal{T}) = T^+ \cap_k \mathtt{A} \cap_k \mathtt{G}^+$, $\mathrm{CSG}({}^{\mathcal{O}}\mathcal{T}) = (T^+ \cap_k \overline{\mathtt{A}}) \cup_k (T^+ \cap_k \mathtt{A} \cap_k \overline{\mathtt{G}^+})$, and (c) in the interior tetrahedron: $\mathrm{CSG}({}^{\mathcal{I}}\mathcal{T}) = (T^- \cap_k \mathtt{A}) \cup_k (T^- \cap_k \overline{\mathtt{A}} \cap_k \mathtt{G}^-)$, $\mathrm{CSG}({}^{\mathcal{O}}\mathcal{T}) = T^- \cap_k \overline{\mathtt{A}} \cap_k \overline{\mathtt{G}^-}$.

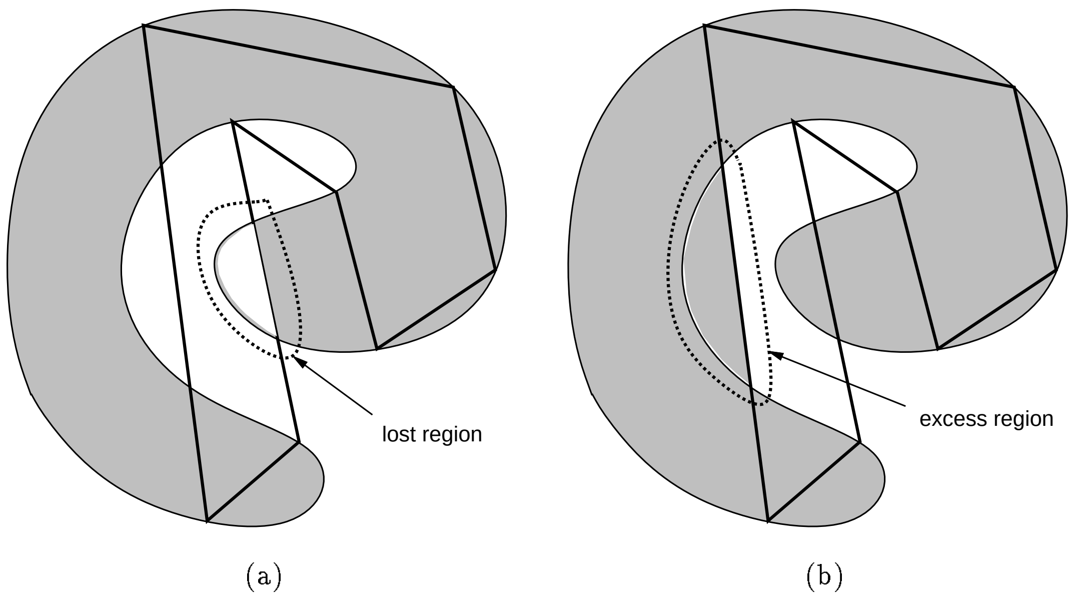


Figure 10: (a) Overlapping subshells resulting in regions (b) $(P \cup_k {}^{\mathcal{IP}}\underline{\mathcal{S}}) -_k {}^{\mathcal{OP}}\underline{\mathcal{S}} \subseteq A$, or (c) $(P -_k {}^{\mathcal{OP}}\underline{\mathcal{S}}) \cup_k {}^{\mathcal{IP}}\underline{\mathcal{S}} \supseteq A$.

as shown in the examples in Figure 10. It is worth observing that these containment relations would not hold in extreme cases of inadmissible boundary tweaking that violate the orientation consistency condition and flip the sense of the patches (e.g. Figure 13b). Such configurations are characterized mathematically by ${}^{\mathcal{IP}}\underline{\mathcal{S}} -_k {}^{\mathcal{OP}}\underline{\mathcal{S}} \not\subseteq A$ or ${}^{\mathcal{OP}}\underline{\mathcal{S}} -_k {}^{\mathcal{IP}}\underline{\mathcal{S}} \not\subseteq \overline{A}$. We guarantee the containment relations in Eqs. 1 and 2 by using locally valid polyhedral hulls that do not permit configurations with flipped patch orientations (such as Figure 13b). By adding appropriate *delta* terms to Eqs. 1 and 2, specifically Δ_l for the 'lost' region and Δ_e for the 'excess' region, we now obtain a general expression for CSG(A) as:

$$\mathrm{CSG}(A) = \underbrace{((\mathrm{CSG}(P) \cup_k \mathrm{CSG}({}^{\mathcal{IP}}\underline{\mathcal{S}})) -_k \mathrm{CSG}({}^{\mathcal{OP}}\underline{\mathcal{S}}))}_{direct} \cup_k \underbrace{\mathrm{CSG}(\Delta_l)}_{delta} \tag{9}$$

or similarly,

$$\mathrm{CSG}(A) = \underbrace{((\mathrm{CSG}(P) -_k \mathrm{CSG}({}^{\mathcal{OP}}\underline{\mathcal{S}})) \cup_k \mathrm{CSG}({}^{\mathcal{IP}}\underline{\mathcal{S}}))}_{direct} -_k \underbrace{\mathrm{CSG}(\Delta_e)}_{delta}. \tag{10}$$

For the *direct* term, CSG expressions of ${}^{\mathcal{IP}}\underline{\mathcal{S}}$ and ${}^{\mathcal{OP}}\underline{\mathcal{S}}$ are obtained from their respective definitions, and CSG(P) is computed from Brep(P) using known techniques (e.g. [28]). Computing CSG(P) does not pose any additional separation requirements, since P is a polyhedron with linear faces. The delta term is more complex, and we discuss it in the next section.

6.6 Delta Terms

Observe that the lost (excess) delta term recovers (removes) portions of the direct term that are removed (added) when ${}^{\mathcal{OP}}\underline{\mathcal{S}}$ (${}^{\mathcal{IP}}\underline{\mathcal{S}}$) is subtracted (added). Hence the point sets represented by the delta-terms need not be unique, let alone the fact that their CSG expressions need not be unique. We shall choose minimal (only what is required) delta point sets, that are also thereby unique. Specifically, the lost-delta point set belongs to A and is quasi-disjoint with the direct term, i.e.

$$\Delta_l \cap_k ((P \cup_k {}^{\mathcal{IP}}\underline{\mathcal{S}}) -_k {}^{\mathcal{OP}}\underline{\mathcal{S}}) = \emptyset, \Delta_l \subseteq A, \tag{11}$$

and the excess-delta point set does not belong to A but is contained within the direct term, i.e.

$$\Delta_e \cap_k \overline{((P -_k {}^{\mathcal{OP}}\underline{\mathcal{S}}) \cup_k {}^{\mathcal{IP}}\underline{\mathcal{S}})} = \emptyset, \Delta_e \subseteq \overline{A}. \tag{12}$$

An important property is that the delta point sets are contained in the subshell overlap region, i.e.

$$\Delta_x \subseteq \underline{\mathcal{O}}, x \in \{l, e\}. \tag{13}$$

Hence the delta terms – CSG(Δ_l) and CSG(Δ_e) – can be recovered from the non-null intersection ($\underline{\mathcal{O}}$) of the inner and outer polyhedral subshells. Δ_l consists of those portions of $\underline{\mathcal{O}}$ that are 'in' A and Δ_e consists of those portions of $\underline{\mathcal{O}}$ that are 'out' A. Our approach for computing CSG(Δ_x), $x \in \{l, e\}$, is to:

1. decompose $\underline{\mathcal{O}}$ as a union of quasi-disjoint *segments* that are describable in CSG,
2. establish that every segment has a unique classification with respect to A,
3. infer the classification of every segment with respect to A, and
4. compose Δ_l (Δ_e) as the union of all those segments that are 'in' ('out') A.

A solid bounded by n patches has n polyhedral trunctets, one associated with each patch. We define a segment σ_i to be the regularized intersection of n polyhedral trunctets or their complements, i.e.

$$\begin{aligned} \sigma_i \quad &= \quad {}^{x}\underline{\mathcal{T}}_1 \cap_k {}^{x}\underline{\mathcal{T}}_2 \cap_k \cdots \cap_k {}^{x}\underline{\mathcal{T}}_n, {}^{x}\underline{\mathcal{T}}_i \in \{{}^{\mathcal{IP}}\underline{\mathcal{T}}_i, \overline{{}^{\mathcal{IP}}\underline{\mathcal{T}}_i}, {}^{\mathcal{OP}}\underline{\mathcal{T}}_i, \overline{{}^{\mathcal{OP}}\underline{\mathcal{T}}_i}\} \\ &\quad\ \ \text{such that } \sigma_i \cap_k \sigma_j = \emptyset, i \neq j \end{aligned}$$

Because individual polyhedral trunctets are representable in CSG, CSG(σ_i) can be found easily. For n polyhedral trunctets, there are 2^n distinct segments, many of which are empty point sets. Since the subshell overlap $\underline{\mathcal{O}}$ is describable in CSG as a boolean combination of polyhedral trunctets (i.e. $\underline{\mathcal{O}} = ({}^{\mathcal{IP}}\underline{\mathcal{T}}_1 \cup_k {}^{\mathcal{IP}}\underline{\mathcal{T}}_2 \cup_k \cdots) \cap_k ({}^{\mathcal{OP}}\underline{\mathcal{T}}_1 \cup_k {}^{\mathcal{OP}}\underline{\mathcal{T}}_2 \cup_k \cdots)$), it follows from the Describability Theorem that:

- each segment may be thought of as a cell that is induced from the set $G = \{{}^{\mathcal{P}}\mathcal{T}_1, {}^{\mathcal{P}}\mathcal{T}_2, \cdots, {}^{\mathcal{P}}\mathcal{T}_n\}$ of polyhedral trunctet primitives, and
- all connected components of a segment have unique classification ('in' or 'out') with respect to $\underline{\mathcal{O}}$.

We use the notation $\mathbf{M}(X, R)$ = 'in'/'on'/'out' to denote set membership classification of candidate set X with respect to reference set R [30]. Let us define a *delta-segment* ${}^{\Delta}\sigma_i$ to be a segment that classifies as 'in' $\underline{\mathcal{O}}$, i.e. $\mathbf{M}(\sigma_i, \underline{\mathcal{O}})$ = 'in' – the prefix 'delta' signifies that these segments will contribute to the appropriate delta terms. Thus,

$$\underline{\mathcal{O}} = {}^{\Delta}\sigma_1 \cup_k {}^{\Delta}\sigma_2 \cup_k \cdots {}^{\Delta}\sigma_l,$$

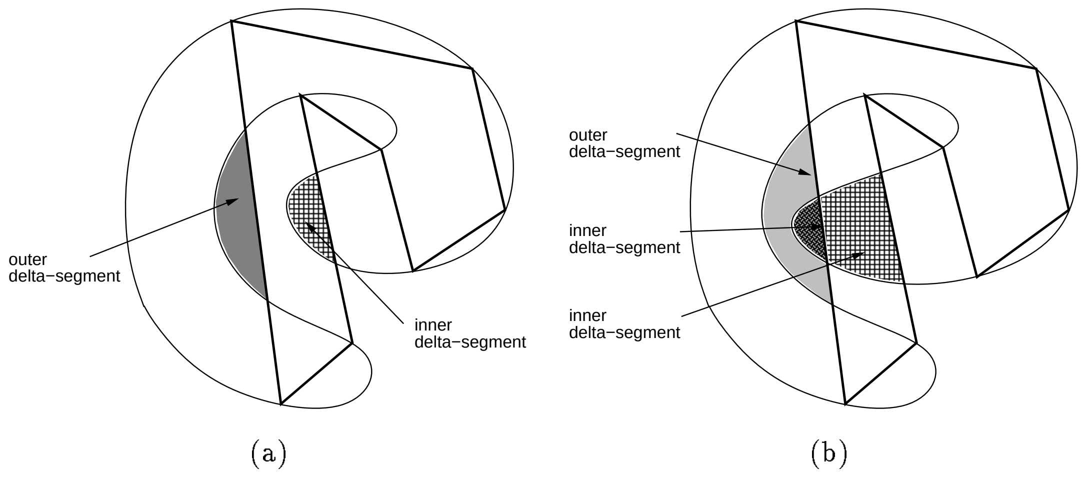


Figure 11: Decomposition of subshell overlap (a) two segments, (b) three segments.

and the set $\{{}^{\Delta}\sigma_i \mid i = 1...l\}$ of delta-segments may be viewed as the smallest spatial building blocks that may be glued together to form the subshell overlap region. For example, Figure 11a,b shows $\underline{\mathcal{O}}$ decomposed into two and three delta-segments respectively.

For a valid input polyhedron (P), the faces do not inter-penetrate. Similarly, for a valid Brep(A), the patches do not inter-penetrate. Consequently, the point sets represented by any boolean combination of a pair of overlapping inner and outer polyhedral trunctets would be either 'in' A or 'out' A, e.g. the two segments in Figure 11a. Through an inductive reasoning, this holds for all intersection regions (having single or multiple connected components) of polyhedral trunctets or their complements, e.g. the three segments in Figure 11b. Hence, segments have unique classification with respect to A, i.e. for every segment ${}^{\Delta}\sigma_i$, $\mathbf{M}({}^{\Delta}\sigma_i, A) =$ 'in' or $\mathbf{M}({}^{\Delta}\sigma_i, A) =$ 'out'. A delta-segment is called an *inner delta-segment* ${}^{I\Delta}\sigma$ if $\mathbf{M}({}^{\Delta}\sigma, A) =$ 'in', otherwise it is an *outer delta-segment* ${}^{O\Delta}\sigma$.

A CSG representation of the lost-delta Δ_l (or excess-delta Δ_e) region is now obtained by appropriate set membership classification tests of suitable point sets in known representation schemes, as follows:

1. determine whether a segment is a delta-segment by the test
   $\sigma_i = {}^{\Delta}\sigma_i \Leftrightarrow \mathbf{M}(\mathrm{CSG}(\sigma_i), \mathrm{CSG}(\underline{\mathcal{O}})) =$ 'in',

2. determine whether the delta-segment is inner or outer by the test
   ${}^{\Delta}\sigma_i = {}^{I\Delta}\sigma_i({}^{O\Delta}\sigma_i) \Leftrightarrow \mathbf{M}(\mathrm{CSG}({}^{\Delta}\sigma_i), \mathrm{Brep}(A)) =$ 'in' ('out'), and

3. assign
   $\mathrm{CSG}(\Delta_l) = \mathrm{CSG}({}^{I\Delta}\sigma_1) \cup_k \mathrm{CSG}({}^{I\Delta}\sigma_2) \cup_k \cdots \mathrm{CSG}({}^{I\Delta}\sigma_{l_I})$, and/or
   $\mathrm{CSG}(\Delta_e) = \mathrm{CSG}({}^{O\Delta}\sigma_1) \cup_k \mathrm{CSG}({}^{O\Delta}\sigma_2) \cup_k \cdots \mathrm{CSG}({}^{O\Delta}\sigma_{l_O})$,
   where $l_I + l_O = l$.

Since every segment has a unique classification with respect to $\underline{\mathcal{O}}$ and A, points in the interior of any segment define an equivalence class. Instead of enumerating all the 2^n segments and then classifying them, the enumeration and examination of null segments can be obviated by resorting to geometrical calculations that: (a) generate a *characteristic* point $\mathbf{q}_i$ that belongs to the interior of the segment σ_i, and (b) determine segment type by the classifications $\mathbf{M}(\mathbf{q}_i, \mathrm{CSG}(\underline{\mathcal{O}}))$ and $\mathbf{M}(\mathbf{q}_i, \mathrm{Brep}(A))$. One method to compute these points is to use offset halfspace intersections [27]; another is raycasting [15].

In general, delta term computation poses two classes of problems: (a) a combinatorial problem requiring the ability to reduce the processing of an exponential number (2^n)of segments, and (b) a numerical problem surrounding the generation of a sufficient set of characteristic points so as not to miss a required delta-segment.

6.7 Properties

The bounding tetrahedra of algebraic patches provide an elegant space partitioning that confines the effect of curved algebraic halfspaces to tetrahedra, and the separation problem outside the tetrahedra has a linear character. Compared to the general Brep-to-CSG conversion algorithm in [27], this partition significantly alleviates the combinatorial explosion of cells: it not only reduces the number, but also the degree[1] of the halfspaces participating in the disjunctive decomposition of space. Such bounding-box based spatial partitioning can be applied to convex hulls of general (e.g. spline) patches.

Thus we obtain the following main properties of the general CSG solution.

> **Finite extent decomposition:** For a free-form solid A whose boundary is composed of algebraic patches $\{\widetilde{F}_i \mid i = 1...n\}$, CSG($A$) can be computed without considering the decomposition of space induced by $\mathtt{A}_i$ outside T_i.
>
> **Locally-separating solid:** For a free-form solid A whose boundary is composed of algebraic patches $\{\widetilde{F}_i \mid i = 1...n\}$, separation problems, if at all, need to be solved locally for every $(T_i, \mathtt{A}_i)$ tuple.

Because all separation problems arising from curved faces are also localized to individual tetrahedra (or convex hulls), it follows that Breps satisfying the self-separating trunctet condition for every patch do not require *any* separating halfspace for CSG computation; such solids are called *self-separating solids.*

6.8 Towards Efficient, Robust CSG Constructions

In general, we have the following algorithm for CSG computation.

```
if ($\underline{\mathcal{O}} = \emptyset$)
    CSG($A$) is the direct term
else
    CSG($A$) is the combination of the direct and delta terms
```

Here we test whether CSG($\underline{\mathcal{O}}$) represents a null set or not, in order to decide if the delta term needs to be computed. In practice, nullity of CSG($\underline{\mathcal{O}}$) can be computed swiftly on the RayCasting Engine via ray-rep processing [18].

The general CSG solution simplifies considerably if subshells do not overlap, since the delta term would not need to be computed, namely,

$$\underline{\mathcal{O}} = \emptyset \Rightarrow A = (P \cup_k {}^{\mathcal{IP}}\underline{\mathcal{S}}) -_k {}^{\mathcal{OP}}\underline{\mathcal{S}} = (P -_k {}^{\mathcal{OP}}\underline{\mathcal{S}}) \cup_k {}^{\mathcal{IP}}\underline{\mathcal{S}}.$$

[1]The number of cells in a disjunctive decomposition of E^k induced from a set of m algebraic halfspaces of degree d is $\mathrm{O}(md)^k$ [11].

In such cases, CSG(A) can be obtained as the direct term in Eq. 3 or 4, thus obviating any combinatorial or numerical problems associated with the segment method. We summarize this important result as follows.

> **Quasi-disjoint subshells:** Consider a free-form solid A whose boundary is composed of algebraic patches $\{\widetilde{F}_i \mid i = 1...n\}$, such that inner and outer polyhedral subshells are quasi-disjoint, i.e. $\underline{\mathcal{O}} = \emptyset$. For such a solid, CSG($A$) can be expressed as the direct term in Eq. 3 or 4, thus obviating any delta term computation.

Note that $\underline{\mathcal{O}} \neq \emptyset$ does not always imply that a delta-term is required. We leave it as an exercise to the reader to construct a solid A with overlapping subshells, such that A can be expressed in CSG without a delta term (you may need Δ_l or Δ_e, but not both). In some situations (e.g. Figure 10), the delta term cannot be avoided. We are currently exploring conditions under which a delta term can be avoided even when subshells overlap.

Convex tetrahedral bounds on patches and trunctets can be exploited for reducing complexity through spatial localization. For example,

$$T_i \cap_k T_j = \emptyset \Rightarrow {}^{\mathcal{P}}\mathcal{T}_i \cap_k {}^{\mathcal{P}}\mathcal{T}_j = \emptyset,$$

and therefore polyhedral trunctets contained in disjoint tetrahedra need not be included in CSG($\underline{\mathcal{O}}$) or CSG(${}^{\Delta}\sigma$). This is particularly useful for tackling the combinatorial problems associated with delta term computation, since the number of segments is exponential in the number of polyhedral trunctets.

Normal constructions guarantee quasi-disjoint subshells, and therefore never require delta term computation. A different approach, called *shell/core* method [12], for normal constructions gives CSG(A) = CSG(${}^{\mathcal{I}}\mathcal{S}$) $\cup_k$ CSG(${}^{\mathcal{I}}\mathcal{C}$), where the first term is CSR(A) and the second term requires Brep-to-CSG conversion of a polyhedron ${}^{\mathcal{I}}\mathcal{C}$. Observe also that if subshells overlap, the given Brep is an abnormal construction; the inverse is not true, i.e. an abnormal construction can have non-overlapping subshells. Thus quasi-disjoint subshells provide a more relaxed condition for simple and efficient CSG computation since they accommodate several cases of abnormal constructions that arise in practical applications.

Finally, by combining the above discussions with earlier ones, we conclude that the two *trunctet-subshell* conditions:

- self-separating trunctets, and
- quasi-disjoint subshells

simplify CSG computation, by obviating: (a) separating halfspaces, and (b) delta term computation. A seamlessly integrated bilateral [Brep,CSG] system would incorporate the trunctet-subshell conditions as constraints in the Brep construction procedure – see Brep/CSG coupling below.

6.9 Brep/CSG coupling

To maintain consistent [Brep, CSG] representations efficiently, it is therefore desirable that the Brep methods guarantee the trunctet-subshell conditions. The methods described in [10, 16] guarantee this by

- using clipping planes for global hull validity to satisfy the quasi-disjoint subshell condition, and

- using appropriate apex weights to guarantee single-sheeted patches, which is turn provides self-separating trunctets.

See also [17] for a coherent mathematical framework with details on the Brep/CSG coupling.

6.10 Advantages over the generic algorithm

It is instructive to compare the CSR based methods for CSG computation with the 5-step cell-based generic Brep-to-CSG conversion algorithm, outlined in the beginning of this chapter. There are three salient features of the CSR-based approach.

1. *Natural halfspaces:* algebraic patches induce low-degree (typically 2, 3) halfspaces, as opposed to, for example, a degree 18 halfspaces induced from a bi-cubic parametric patch.

2. *Local separation:* the separation problem is localized to the construction tetrahedron for each patch, and requires only linear separating halfspaces. Furthermore, self-separating trunctets completely obviate the separation problem for CSG computation.

3. *Finite extent decomposition:* participation of halfspaces is localized to known regions in space, specifically the tetrahedra. which contrasts with the general "infinite" extent cell-based disjunctive decomposition techniques. Finite-extent not only reduces the size of the Boolean optimization problem for CSG computation, but also supports efficient incremental CSG updates using point-sampling reconstruction methods [14].

7 Shape Control

7.1 Dual Control Polygon

The control net of S is comprised of the interpolative (vertices of P) and non-interpolative (apexes of tetrahedra) *control points.* For each face F_i of P, both apexes $\mathbf{v}_i^+$ and $\mathbf{v}_i^-$ (of the exterior and interior tetrahedra respectively) have *control weights*

$$w_i^+ = \frac{1}{f(\mathbf{v}_i^+)} \quad \text{and} \quad w_i^- = -\frac{1}{f(\mathbf{v}_i^-)} \tag{14}$$

where f denotes the piecewise algebraic function whose zero contour defines S. The interpolative control points have weights ∞. The control points define the *dual control polygons* P^+ and P^-, which are the boundaries of the exterior and interior hulls respectively.

The boundary of the sculptured solid can be edited with local support by tweaking control weights associated with the non-interpolative control points or by moving the interpolative and/or non-interpolative control points of the dual P^+/P^- polygons. A change of weight modifies the Bezier coefficients of the associated face patch and its neighboring wedge patches (see [10] and references therein). Moving a control point requires maintaining validity of the polyhedral hull, which is obtained through a re-assertion of the hull validity constraints (below).

7.2 Re-asserting Hull Validity

A change of a non-interpolative control point (apex) triggers a local re-computation of the hull to satisfy local hull validity constraints. The final apex point tends to the specified position in a least distance sense. When the user wishes to relocate an apex to a target position, our system

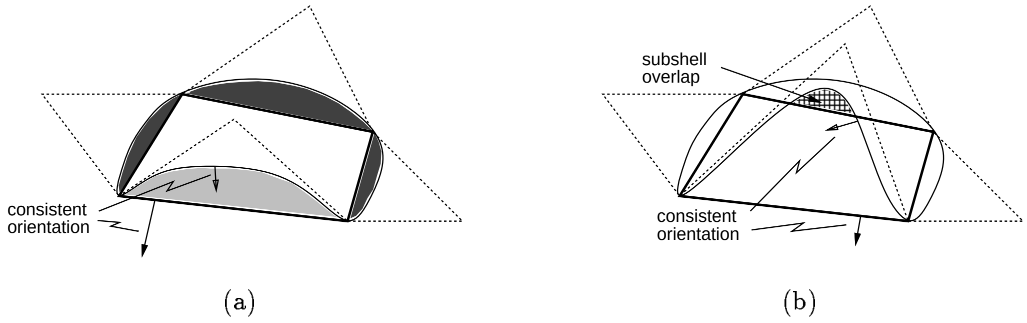


Figure 12: In 2D: (a) bean shaped solid with protrusions (dark), depressions (light), and (b) subshell overlap (shaded) in a slightly tweaked solid.

[10, 16, 17] will move the apex as close as possible to the target without violating the ten linear inequality constraints (Figure 6a) on the apex.

If a vertex **v** of the input polyhedron P is moved, then the linear inequality constraints for all faces incident to **v** may be affected, thus, perturbing the apexes of these faces. In both cases, the new apexes are found through solving a least-distance programming problem for every face.

7.3 Incremental CSG Updates

Given an initial CSG(A), incremental techniques are used for updating the CSG representation during shape control.

- Change of apex weight influences the corresponding trunctet and its immediate neighbors, thus affecting only CSG($^{\mathcal{IP}}\underline{\mathcal{S}}$) or CSG($^{\mathcal{OP}}\underline{\mathcal{S}}$) in the direct term.
- Moving non-interpolative control points requires trunctet updates, again affecting only CSG($^{\mathcal{IP}}\underline{\mathcal{S}}$) or CSG($^{\mathcal{OP}}\underline{\mathcal{S}}$).
- Moving interpolative control points require incremental Brep-to-CSG conversion [28] on the input polyhedron, i.e. the CSG(P) term. If apexes are moved as a side effect of re-asserting hull validity, the subshell terms would also have to be updated.
- The *delta* term needs updating only if the subshell overlap has changed, e.g. if the solid in Figure 12b is obtained as a result of apex movement on the solid in Figure 12a.

7.4 Analogy with Parametric Patches

Shape control with algebraic patches is somewhat similar to that with parametric patches. The surface S follows the control points in an intuitive manner with local support, except S does not interpolate a control point, unless it has a weight of ∞. Increasing a control weight "pulls" the surface towards the corresponding control point. Furthermore, the surface S always lies inside the polyhedral hull. However, there are three main differences:

- While control points can be moved freely in the parametric case, apex movement for the algebraic case is restricted to local hull validity constraints.

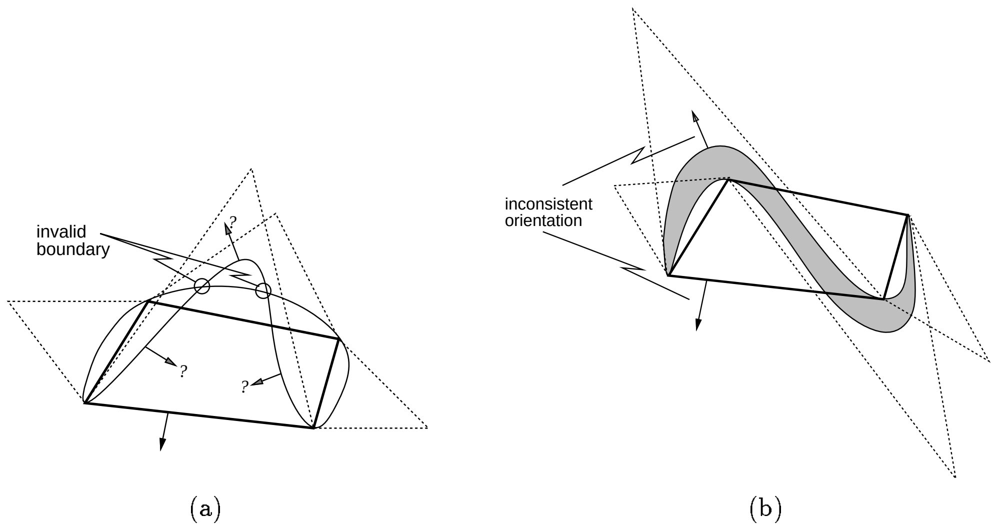


Figure 13: Extreme cases of shape control in 2D: (a) invalid boundary, and (b) inconsistent orientation for the shaded solid.

- Moving a control point does not have "side effects" on other control points in parametric patches. However, for algebraic patches, movement of interpolative control points can cause the non-interpolative (apex) control points to move during re-assertion of hull validity.

- The main difference however lies in the key observation that unlike parametric patches, algebraic patches *can* support bilateral [Brep, CSG] dual-rep systems in a computationally tractable manner.

8 Orientation Consistency

Boundary tweaking operations could result in transforming the solid in Figure 12a to that in Figure 12b. If continued, this could result in an invalid boundary, as in Figure 13a, or even in flipping the sense of the solid, as in Figure 13b. It turns out that the preceding CSG treatment does not hold for such a flipped solid. For example, Figure 14 shows the protrusion and depression trunctets for the flipped solid. Neither the direct nor the delta terms for the CSG expression applies to this case, and this is true even if one uses the complements of the protrusion/depression trunctets in Figure 14.

To formalize this behavior, we capture the relationship between the material sense of the input polyhedron P, the faces of the sculptured solid A, and the trunctets. The orientation of P, which is defined by the outward facing normals of its faces distinguishes between interior and exterior tetrahedra. The orientation of A, determined by the outward facing gradients of the patches, gives rise to the concept of inner and outer trunctets. We introduce an *orientation consistency* condition that makes these orientations coherent.

To describe this condition, we observe that each tetrahedron of the polyhedral hull has vertices not interpolated by the surface S. An edge tetrahedron has two non-interpolating vertices and a face tetrahedron has one (the apex). For a tetrahedron T with inner trunctet ${}^{\mathcal{I}}\mathcal{T}$ and outer trunctet ${}^{\mathcal{O}}\mathcal{T}$, the orientation consistency condition states that:

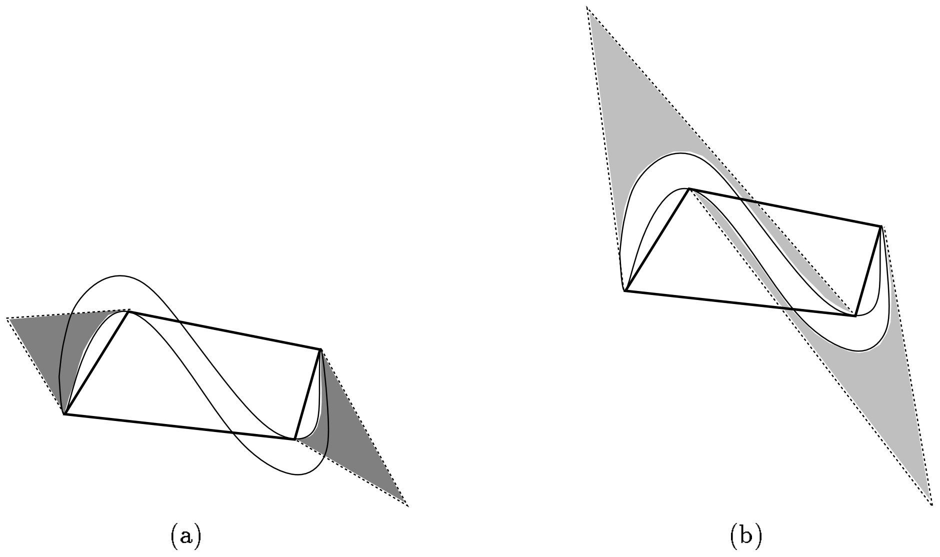


Figure 14: In 2D: (a) protrusion trunctets of the flipped solid, and (b) its depression trunctets.

if T is an exterior (interior) tetrahedron, then the non-interpolating vertices of T belong to ${}^{\mathcal{O}}\mathcal{T}$ (${}^{\mathcal{I}}\mathcal{T}$).

Observe that this condition holds for the examples in Figure 12, but not for the one in Figure 13b. Testing for orientation consistency for any surface S is therefore a linear-time algorithm in the number of patches of S. Every patch uses a constant cost test, which reduces to a single point membership classification ($\mathcal{M}$) if the patch is single-sheeted.

```
assume patch is consistently oriented
if $T$ is a face tetrahedron then $k \leftarrow 1$ else $k \leftarrow 2$
if $T$ is an exterior tetrahedron then $g \leftarrow \overline{a}$ else $g \leftarrow a$
for every non-interpolating vertex $\mathbf{v}_k^{niv}$ do
    if $\mathcal{M}(\mathbf{v}_k^{niv}, g)$ = OUT
    then patch is inconsistently oriented; break
end_for
```

Locally valid polyhedral hulls provide a built-in guarantee for orientation consistency. To verify this, we observe that inconsistent patch orientations arise only when the bounding tetrahedra of the patches overlap. Let T and T' be such bounding tetrahedra. If T and T' share no vertex of the input polyhedron P, then the inconsistent orientation must be accompanied by an invalid Brep (see Figure 13a). If T and T' share at least one vertex of P, then T and T' cannot overlap in a locally valid polyhedral hull. Thus, cases as illustrated in Figure 13b will never arise.

9 Applications of CSRs

Recall that a *Constructive Shell Representation* of A is a CSG representation of a shell (not subshell):

$$\mathrm{CSR}(A) = \mathrm{CSG}({}^{\mathcal{ANY}}\mathcal{S}), \tag{15}$$

which is the union of CSG representations of trunctets, one for every patch. CSR(A) is a *complete* representation of A in that it can represent a free-form solid unambiguously and that it contains sufficient information to answer any geometric query about A [12]. In this section we will briefly explore two applications of CSRs

- raycasting on CSRs (see [15] for details), and
- point sampling for Brep-to-CSG (see [14] for details.)

9.1 Raycasting on CSRs

The "in" segments resulting from the classification of a line L with respect to CSR(A) yields $L\text{in}^{\mathcal{ANY}}\mathcal{S}$ – the portions of L_2, L_3 in the shaded areas of Fig. 15a, and L_5 in Fig. 15b. Observe that the end-points of $L\text{in}^{\mathcal{ANY}}\mathcal{S}$ belong either to the tetrahedral planes or to the algebraic surface, marked with "x" or "•" respectively in Fig. 15. This observation leads to the following simple technique for inferring $L\text{in}A$ from $L\text{in}^{\mathcal{ANY}}\mathcal{S}$.

1. Find $L\text{in}^{\mathcal{ANY}}\mathcal{S}$ using divide-and-conquer on CSR(A).
2. Ignore all intercepts ("x") that lie in tetrahedral plane boundaries.
3. Infer $L\text{in}A$ by alternating "in"/"out" classifications resulting from the remaining intercepts (•) that lie in algebraic patch boundaries.

This algorithm can be used to make simple modifications to the RCE architecture, for parallel processing of ray-reps ("in" segments resulting from classifying a grid of regularly spaced parallel lines) directly on CSRs, thus obviating the need for exact CSG representation of A for ray-rep purposes. Several subtleties related to singular point processing (p, q in Fig. 15a) and associated impact on RCE processor modifications, to yield a new RCE*) are covered in [15].

9.2 Point sampling for Brep-to-CSG

The above procedure for generating ray-reps from CSRs, leads to a simple and efficient means for computing CSG(A) using a "point-sampling" approach. Fig. 16 gives a pictorial outline of the technique for the case of normal constructions (no overlapping tetrahedra). The CSR (CSG of the shell) is easy to compute, and we need only compute a CSG representation of the core so that CSG(A) = CSR(A) $\cup$ CSG(core). The main steps of the algorithm to compute CSG(core) are as follows.

1. Compute a ray-rep of the shell from CSR(A) (Fig. 16a).
2. From this, infer the ray-rep of A using the raycasting on CSR method above (Fig. 16b).
3. Subtract the ray-rep of the shell from that of A to obtain a ray-rep of the core (Fig. 16c).
4. Pick a sample point ($\mathbf{p_1}$) in the ray-rep of the core and classify it against all the halfspaces induced by the linear halfspaces of the tetrahedra (this set of halfspaces can be culled further, if need be). This generates an "in"-cell (as in the cell-based Brep-to-CSG algorithm outlined earlier in the chapter). Fig. 16d shows one in-cell (labelled a) generated in this manner. This constitutes the "evolving" solid E.

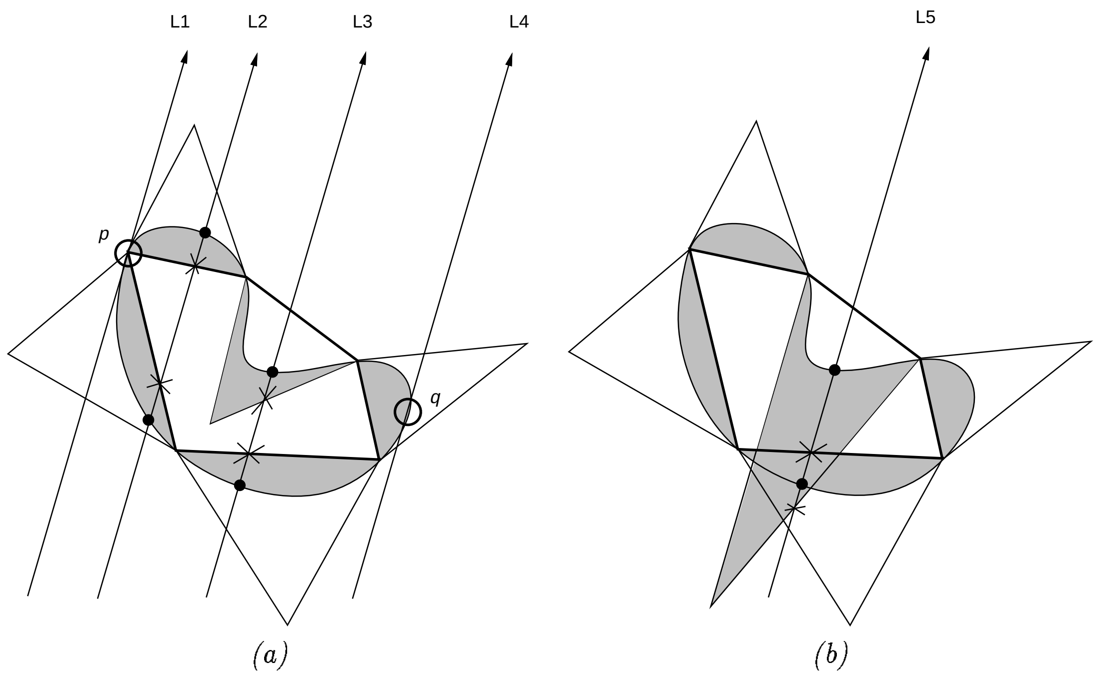


Figure 15: (a) Raycasting on normal constructions, and (b) abnormal constructions.

5. Two other cells (labelled b and c) constitute a "remainder" solid R. Check if R is null by subtracting the ray-rep of E from that of the core (note that we have a CSG expression of E, which can be processed on the RCE).
6. If the remainder solid R is not null, proceed with the above two steps, i.e. get another sample point, and generate cells b and c, until the remainder solid R is null.
7. The union of CSG representations of all "in"-cells is the CSG representation of the core.

The same technique extends to the more complex case of computing the "delta" terms, without combinatorial processing. Fig. 17 summarizes the technique, outlined below.

1. Compute the ray-rep of A from CSR(A).
2. Compute the ray-rep of the direct term.
3. Compute the ray-rep of Δ_l by subtracting the ray-rep of the direct term from the ray-rep of A.
4. Now use the ray-rep of Δ_l to generate a sample point. This time, classify all the polyhedral trunctets against this point to generate a CSG representation of an inner delta-segment.
5. Use the same notion of a remainder solid to determine when the algorithm terminates.

Thus, we can use point-sampling techniques to obviate combinatorial processing, especially for the delta-term calculations. This is particularly effective when the calculations can be done on an RCE (or RCE*) architecture. The results of the sampling technique are accurate to the spacing between rays, i.e. cells smaller than the ray grid spacing could be missed. Finely spaced ray grids, that are quite easily processed on the RCE, work well in practice.

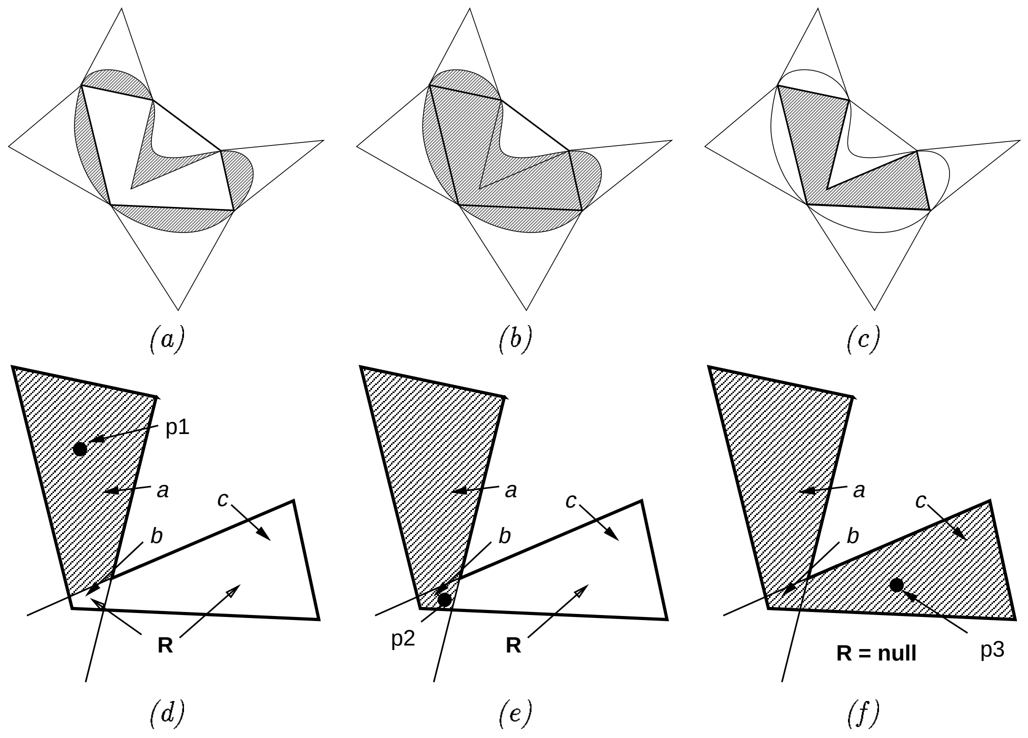


Figure 16: (a) Ray-rep of the shell, (b) of the solid (inferred from the ray-rep of the shell), (c) of the core (shell subtracted from solid). (d) Cell-a, (e) cell-b, and (f) cell-c generation. Remainder R is null at termination.

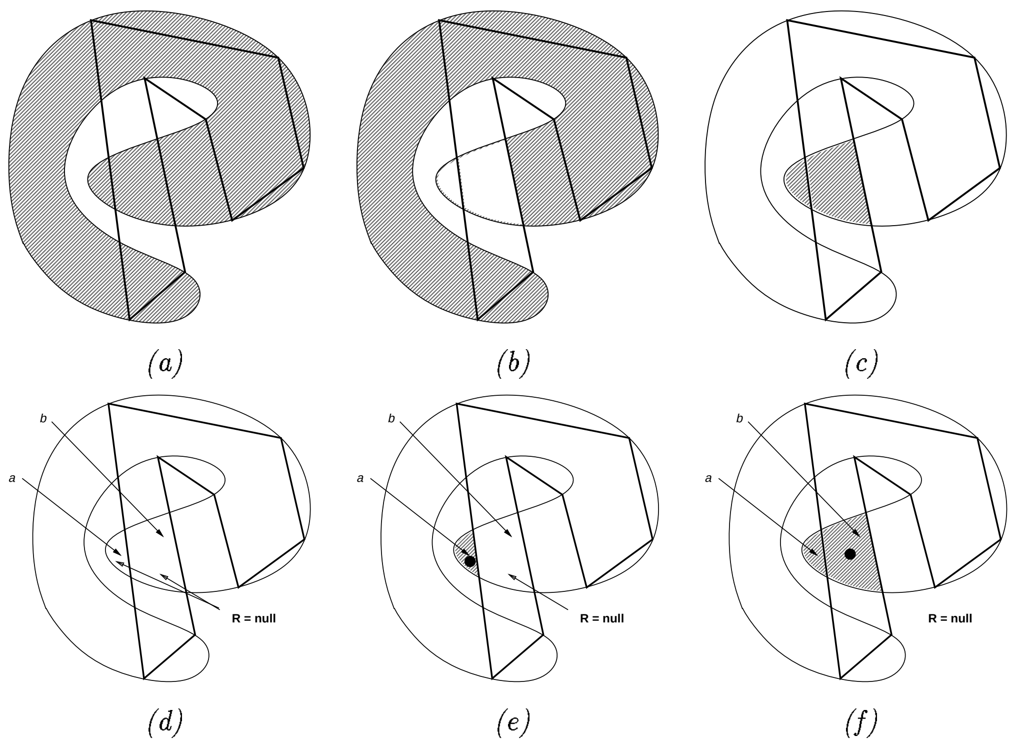


Figure 17: (a) Ray-rep of solid, (b) of the direct term, (c) of the delta term (direct term subtracted from solid). (d) No cells, remainder not null, (e) cell-a, and (f) cell-b generation. Remainder R is null at termination.

10 Sample Results

Fig. 18 shows the configuration of our experimental system [16]. Representation schemes are shown in boxes, conversions between representations are shown between arrows, and applications are shown in italics. The system has been decomposed into four logical components.

Algebraic Brep: Sample points and normals, generated from various sources, e.g. a Coordinate Measuring Machine (CMM), are smoothed to obtain tangent-plane continuous boundaries of free-form solids.

Algebraic CSG: CSRs are computed from the above Breps to be subsequently: (a) used in computing exact CSG representations for processing on the RayCasting Engine (RCE), or (b) processed directly on the RCE*. The RCE uses massively parallel computation [6] to classify dense grids of parallel lines against CSG trees (recall, a CSR is also a CSG tree) and produces a set of 'in' solid segments, called 'ray-rep'. Ray-rep based application modules [18] are used to support a variety of applications, few examples of which are presented below.

Polyhedral Visualization: Direct triangulation of the quadratic algebraic patches, based on Warren and Moore's algorithms [21], produces a linear triangulated approximation with exact vertex normals. These are imported into the IBM 3D Interaction Accelerator [24] or the IRIS Inventor systems for interactive visualization and Virtual Reality applications, e.g. for producing animations and video clips.

Parametric Brep: The bridge to existing parametric Brep technology is the key to the practical acceptance and applicability of the technology presented here. One-way bridge from quadratic algebraic to NURBS has been solved exactly: each quadratic algebraic patch we generate is a single sheet of surface without disconnected components or holes [9], and using a technique developed by Teller and Sequin [29], we can easily represent such a quadratic algebraic patch as a trimmed NURBS patch. The other way bridge is currently handled by sampling a NURBS surface to obtain a set of points and normals, which are subsequently approximated using the Algebraic Brep modules summarized above.

Figs. 19 – 31 show a range of examples generated in this system environment. Artifacts of fixed-point computation of the RCE may sometimes be visible in some of the photographs. We summarize these examples below; the figure captions contain more details.

- Fig. 19 shows how polyhedral smoothing is used to create free-form solids.
- Fig. 20 shows how trunctets are induced from algebraic patch halfspaces, and how these are maintained under shape control operations.
- Fig. 21 shows how the association between a free-form surface and its shell, represented as a CSR.
- Fig. 22 shows a simple example of CSG computation.
- Fig. 23 shows individual patches and resulting deformations during a simple shape control operation.
- Fig. 24 shows how the interpolation of the same polyhedron can result in normal or abnormal constructions, depending on the apex heights of the construction tetrahedra. CSG formulations presented here provide methods to handle both normal and abnormal constructions.

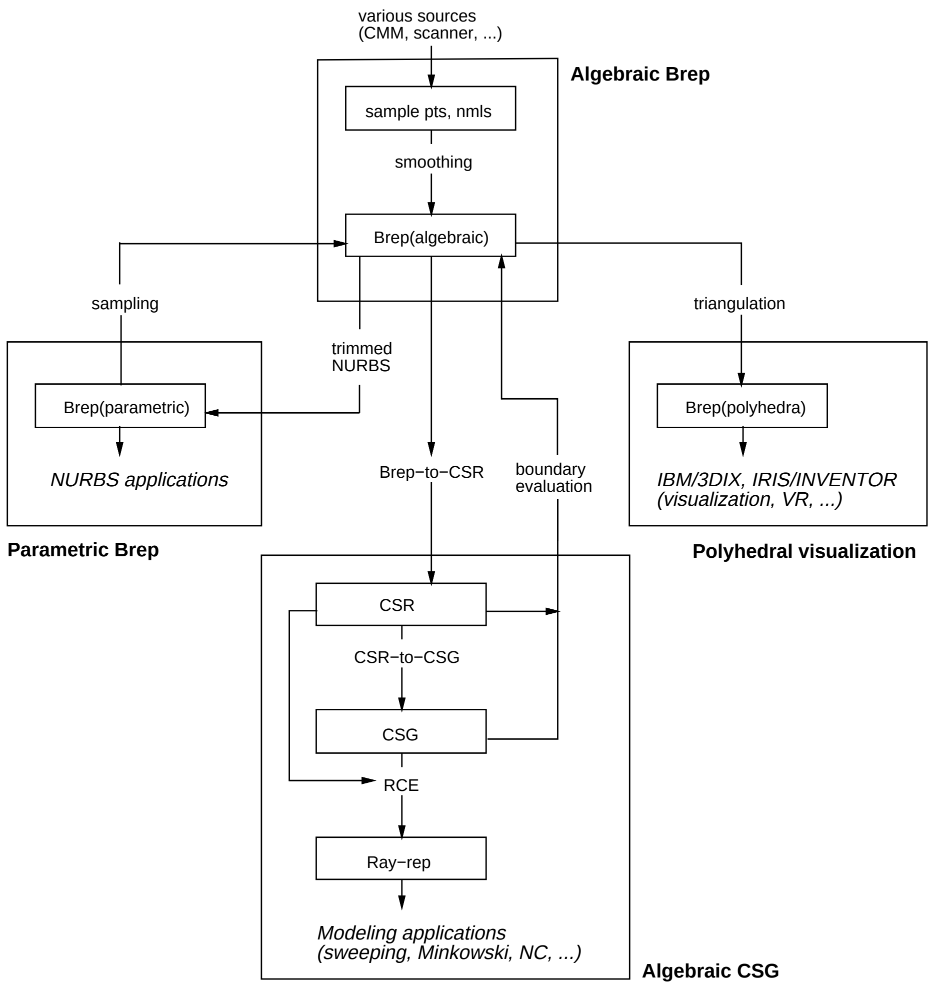


Figure 18: An overview of the experimental system.

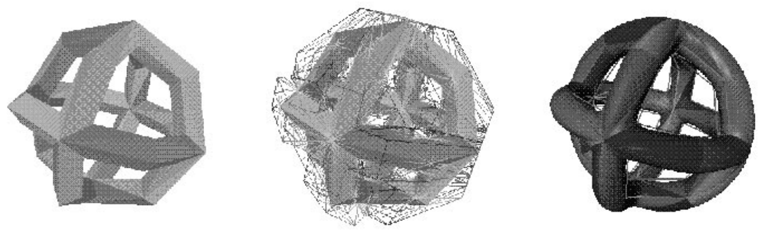


Figure 19: Free-form solid creation via interpolative polyhedral smoothing. The initial polyhedron (left) is used to construct a polyhedral hull (center). Algebraic patches are defined inside some of the tetrahedra of the hull and meshed with tangent plane continuity to produce a free-form solid (right) – a smoothed version of the input polyhedron (left).

- Fig. 25 shows the internal structure of trunctets, and how graphic rendering – transparencies or complex ray tracing – can be applied easily to such objects using CSG constructs.
- Fig. 26 shows an example of a shape control operation on the vase.
- Fig. 27 shows three examples of complex free-form solids constructed entirely with quadratic algebraic patches.
- Fig. 28 shows swept solid computation using the RCE and ray-rep technology [18] – this was feasible only because of the CSG constructs that could be computed easily using Brep-to-CSG conversion methods presented earlier.
- Fig. 29 shows Minkowski sum computation using the RCE and ray-rep technology [18] – again feasible because of the CSG constructs.
- Fig. 30 shows the application of CSG constructs in an NC machining simulation program [20, 19].
- Fig. 31 shows the application of CSG constructs in an NC probing simulation run [20, 19].

11 Conclusions

This chapter first covered the topic of dual representation [Brep, CSG] systems, and summarized known algorithms to inter-convert between Brep and CSG. While conversion from CSG to Brep is relatively straight forward, the inverse (Brep-to-CSG) is much harder particularly because of an additional problem of "separation" introduced by curvature. The lack of powerful Brep-to-CSG conversion, especially for the case of free-form solids was identified as one of the key reasons for the decline for dual-representation systems.

The chapter then provided a detailed explanation of new results on exact CSG for free-form solids, and efficient Brep-to-CSG conversion for such solids. Implicit algebraic patches of low degrees (typically quadratic or cubic) are meshed together smoothly (typically G^1 or C^1 continuity respectively) and algorithms are developed to efficiently compute the CSG representation and to update the CSG under shape control operations.

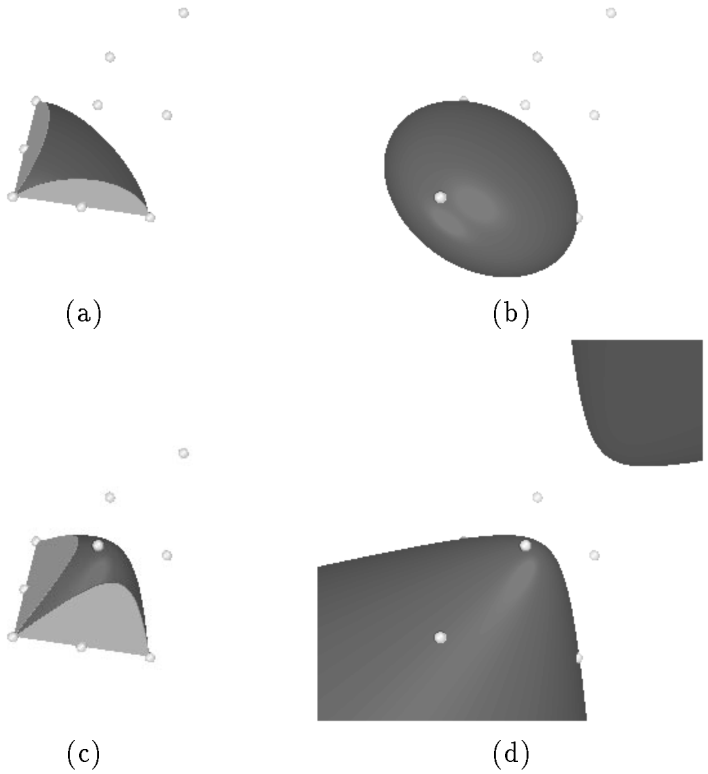


Figure 20: (a) A quadratic algebraic patch and its trunctet, along with prescribed control points; (b) the associated halfspace (ellipsoid). (c) A tweaked patch, and (d) the associated halfspace (a hyperboloid of two sheets).

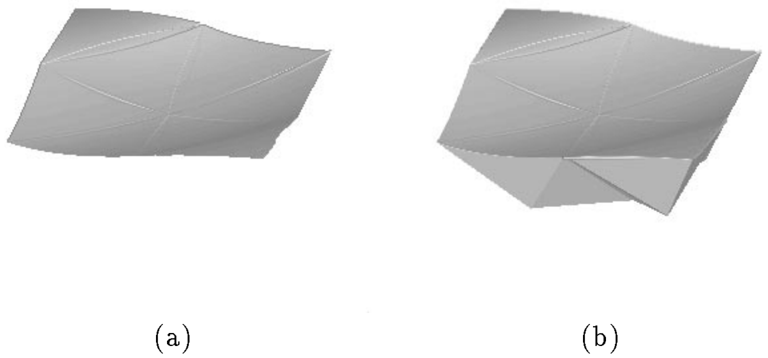
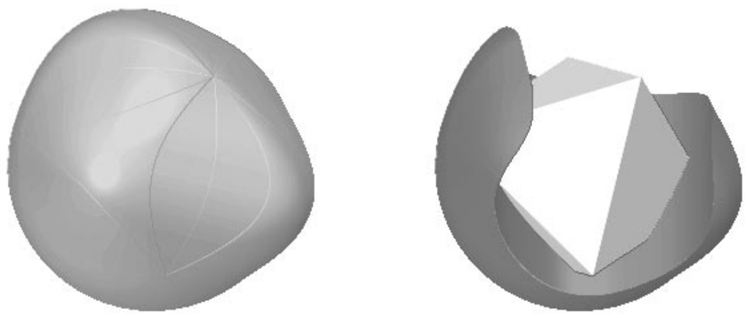
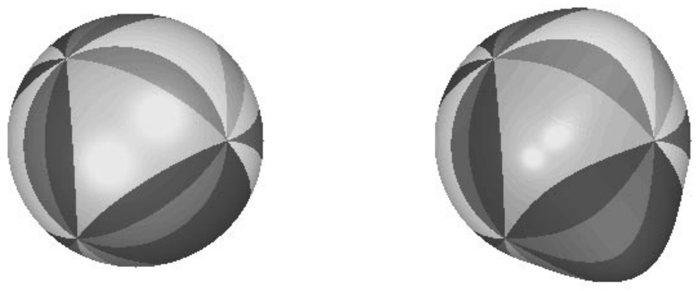


Figure 21: (a) A free-form surface with 26 quadratic algebraic patches, and (b) a "thick shell" represented by the CSR.



(a) (b)

Figure 22: (a) A free-form solid, whose boundary is modeled with 32 quadratic algebraic patches. (b) The input polyhedron and a portion of the inner polyhedral (protrusion) subshell – the CSG representation of the solid is the union of the subshell and the input polyhedron.



(a) (b)

Figure 23: Shape control on a 32-patch mesh of quadratic algebraic patches: (a) sphere, (b) tweaked sphere. Exact CSG representations of the solids were processed on the RCE to produce these images.

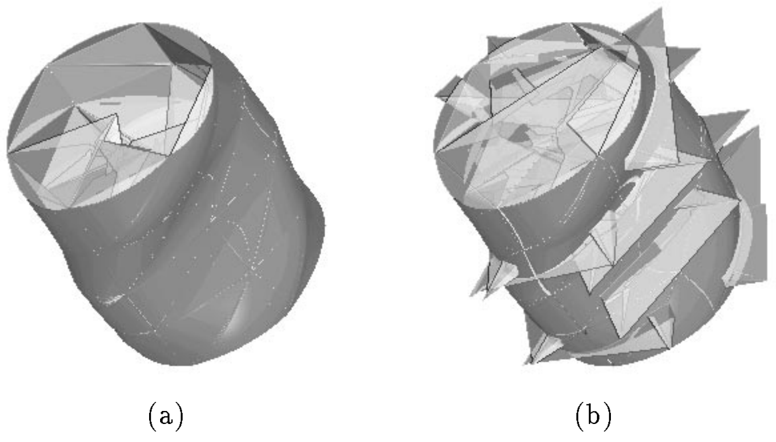


Figure 24: An inner shell for a surface with 204 quadratic algebraic patches: (a) normal construction, (b) abnormal construction.

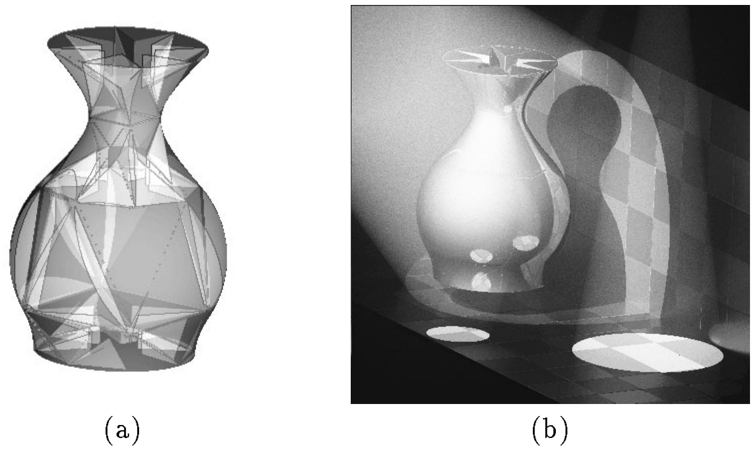


Figure 25: (a) A vase modeled with 336 quadratic algebraic patches, rendered with transparency to show some of its trunctets. (b) The same vase in an environment with multiple light sources and participating media – rendered with Monte Carlo ray tracing.

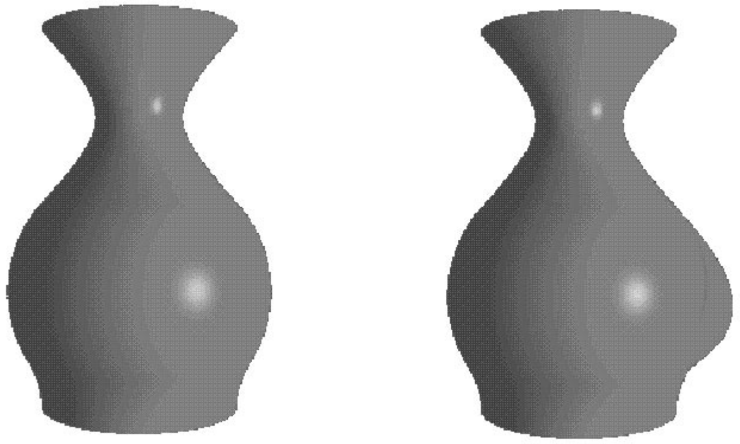


Figure 26: Local shape control operations to tweak the vase (modeled with 336 quadratic algebraic patches).

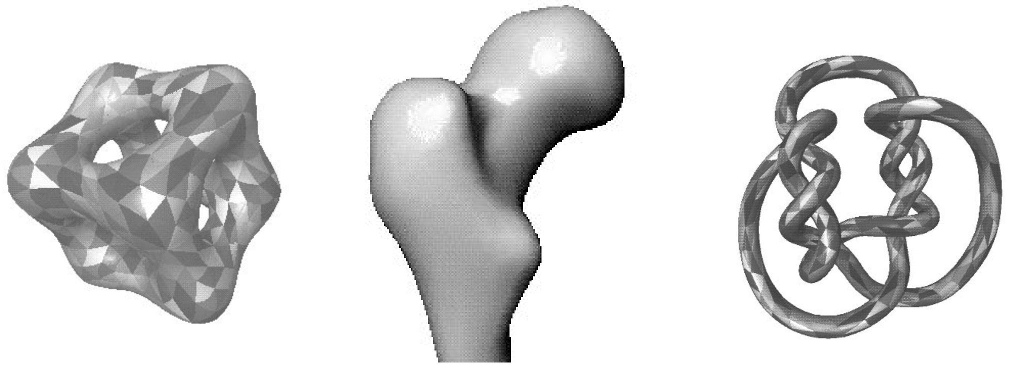


Figure 27: Some complex free-form solids modeled with algebraic patches: (a) a genus five solid with 2,400 quadratic algebraic patches, (b) a bone head, the entire bone was modeled with 5,696 quadratic algebraic patches, and (c) a knot modeled with 6,912 quadratic algebraic patches.

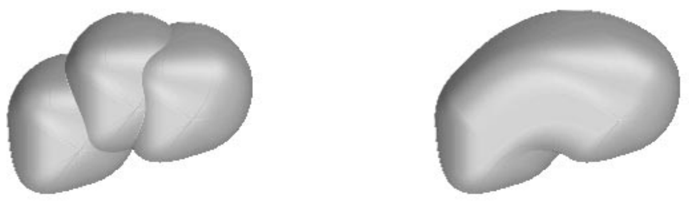

(a) (b)

Figure 28: Sweeping a free-form solid using the repeated union formulation of ray-reps, implemented on the RCE: (a) three instances along sweep, and (b) the swept solid (9,576 halfspaces).

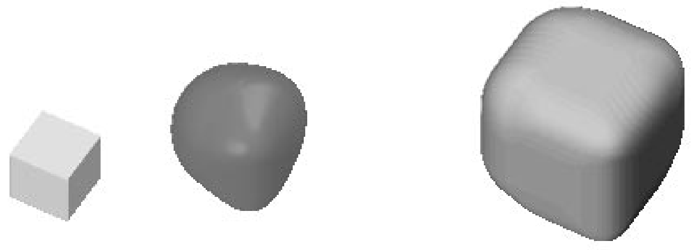
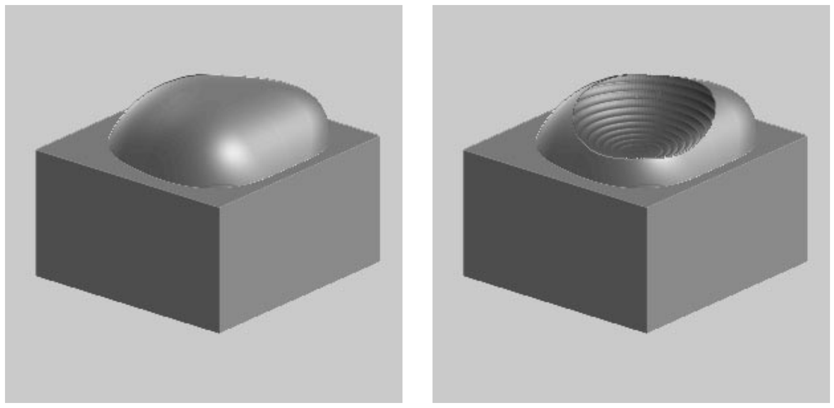
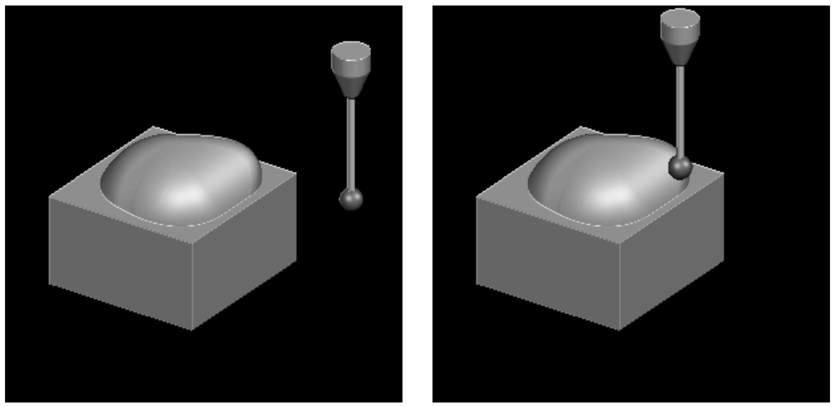

(a) (b)

Figure 29: (a) A free-form solid and a block, and (b) their Minkowski sum computed with the repeated union formulation of ray-reps, implemented on the RCE (123,660 halfspaces).



(a) (b)

Figure 30: NC machining of (a) a free-form stock created from a block and a mesh of quadratic algebraic patches, and (b) rough machining of a spherical pocket in the stock.



(a) (b)

Figure 31: Simulating touch-sense probing: (a) free-form stock and a probe at the beginning of probing motion, and (b) contact between probe and stock as the probe moves in a specified direction until it contacts the stock.

The CSG constructs make it possible to directly access the massively parallel processing power of the RayCasting Engine (RCE) for a wide variety of applications (rendering, sweeping, machining, etc.). A new representation scheme for such solids – Constructive Shell Representation (CSR) – was introduced, and applications for direct hybrid Brep/CSG processing on CSRs were explored. In particular, CSRs provide one way of handling "quadratic tesselations", and the RCE* provides one hardware architecture for processing these tesselations.

Can quadratic tesselations replace today's linear tesselations? Observe that they (quadratic tesselations) require lesser number of 'triangles' than conventional linear tesselations, e.g. a cylinder is a quadric to begin with, and a NURBS patch could be sampled and meshed smoothly with a few quadratic algebraic patches. Hence they produce lesser data to be stored and consequently can be processed efficiently. Continuing research may yield answers to these and related questions, and we might see practical systems that fully exploit the power of implicit algebraics in shape modeling.

12 Acknowledgements

The author would like to thank Herb Voelcker (Cornell University) for his advise and encouragement during the course of this work, as well as Gene Hartquist (Cornell University) for several technical discussions and useful feedback. Many thanks also to Baining Guo (York University, Canada) for close collaboration on many aspects of algebraic patch meshing.

References

[1] C. Bajaj, J. Chen, and G. Xu. Free Form Surface Design with A-patches. In *Graphics Interface '94*, pages 174–181, Banff, Canada, June 1994.

[2] K-C. Chan. *Solid Modeling of Parts with Quadric and Free-form Surfaces.* PhD thesis, Department of Mechanical Engineering, University of Hong-Kong, November 1987.

[3] W. Dahmen. Smooth piecewise quadric surfaces. In *Mathematical Methods in CAGD*, pages 181–193, eds. T. Lyche and L. Schumaker, Academic Press, 1989.

[4] W. Dahmen and T.-M. Thamm-Schaar. Cubicoids: Modeling and Visualization. *Computer Aided Geometric Design*, 10:89–108, 1993.

[5] A. Dunnington, D.R.and Saia and A. de Pennington. Constructive Solid Geometry with Sculptured Primitives using Inner and Outer Sets. In *Theory and Practice of Geometric Modeling*, pages 127–142, eds. W. Strasser and H.P. Seidel, Springer-Verlag, 1989.

[6] J.L. Ellis, G. Kedem, T.C. Lyerly, D.G. Thielman, R.J. Marisa, J.P. Menon, and H.B. Voelcker. The RayCasting Engine and Ray Representation: A Technical Summary. *International Journal of Computational Geometry and Applications*, 4(2):347–380, December 1991.

[7] B. Guo. *Modeling Arbitrary Smooth Objects with Algebraic Surfaces.* PhD thesis, Cornell University, August 1991.

[8] B. Guo. Representation of Arbitrary Shapes Using Implicit Quadrics. *The Visual Computer*, 9:267–278, 1993.

[9] B. Guo. Avoiding Topological Anomalies in Quadric Surface Patches. In *Mathematical Methods in CAGD III*, eds. M. Daehlen, T. Lyche and L. Schumaker, Academic Press, 1995.

[10] B. Guo and J.P. Menon. Local Shape Control for Free-Form Solids in Exact CSG Representation. *Computer Aided Design*, to appear, 1996. (also available as IBM Research Report 20051, IBM T.J. Watson Research Center, 1995.).

[11] J. Heintz. Definability and Fast Quantifier Elimination in Algebraically Closed Fields. *Theoretical Computer Science*, 24:239–278, 1983.

[12] J.P. Menon. Constructive Shell Representations for Free-form Surfaces and Solids. *IEEE Computer Graphics & Applications*, 14(2):24–36, March 1994.

[13] J.P. Menon. Massively Parallel Hybrid Brep/CSG Processing for Geometric Modeling and Graphics. *SIGGRAPH 94 Technical Sketches*, July 1994. (available as IBM Research Report RC 19669, IBM T.J. Watson Research Center, 1994.).

[14] J.P. Menon. Incremental Brep-to-CSG Conversion for Free-form Solids Using Point-sampling Methods. *International Journal of Shape Modeling (submitted)*, 1996. (available as IBM Research Report, IBM T.J. Watson Research Center, Yorktown Heights, NY, 1996.).

[15] J.P. Menon. Ray-reps for Free-form Modeling: Line Membership Classification on CSRs. In *Implicit Surfaces '96: The Second Eurographics Workshop on Implicit Surfaces (submitted)*, Netherlands, October 1996. (available as IBM Research Report, IBM T.J. Watson Research Center, Yorktown Heights, NY, 1996.).

[16] J.P. Menon and B. Guo. Free-form Modeling with Low Degree Algebraic Patches in Bilateral Brep and CSG Schemes. *IEEE Transactions on Visualization and Computer Graphics (submitted)*, 1995. (available as IBM Research Report RC 20050, IBM T.J. Watson Research Center, 1995.).

[17] J.P. Menon and B. Guo. A Framework for Sculptured Solids in Exact CSG Representation. In *CSG96: Set-theoretic Solid Modeling Techniques and Applications*, pages 141–157, Winchester, U.K., April 17-19 1996. Information Geometers Publishers.

[18] J.P. Menon, R.J. Marisa, and J. Zagajac. More Powerful Solid Modeling Through Ray Representations. *IEEE Computer Graphics & Applications*, 14(3):22–35, May 1994.

[19] J.P. Menon and D.M. Robinson. Advanced NC Verification via Massively Parallel RayCasting: Extensions to New Phenomena and Geometric Domains. *ASME Manufacturing Review*, 6(2):141–154, June 1993.

[20] J.P. Menon and H.B. Voelcker. Toward a Comprehensive Formulation of NC Verification as a Mathematical and Computational Problem. *Journal of Design and Manufacturing*, 3(4):263–278, December 1993. Chapman and Hall Publishers, London.

[21] D. Moore. *Simplicial Mesh Generation with Applications.* PhD thesis, Cornell University, August 1992.

[22] A.A.G. Requicha. Representations for Rigid Solids: Theory, Methods and Systems. *ACM Computing Surveys*, 12(4):437–464, December 1980.

[23] A.A.G. Requicha and H.B. Voelcker. Boolean Operations in Solid Modeling: Boundary Evaluation and Boundary Merging. *Proceedings of the IEEE*, 73(1):30–44, January 1985.

[24] B-O Schneider, P. Borrel, J.P. Menon, J. Mittleman, and J. R. Rossignac. BRUSH as a Walk-Through System for Architectural Models. In *Fifth Eurographics Workshop on Rendering*, pages 389–399, Darmstadt, Germany, June 1994.

[25] T.W. Sederberg. Piecewise Algebraic Surface Patches. *Computer Aided Geometric Design*, 2:53–59, 1985.

[26] T.W. Sederberg. Techniques for Cubic Algebraic Surfaces: Tutorial Part Two. *IEEE Computer Graphics and Applications*, 10(5):12–21, September 1990.

[27] V. Shapiro and D.L. Vossler. Construction and Optimization of CSG Representations. *Computer-Aided Design*, 23(1):4–19, Jan/Feb 1991.

[28] V. Shapiro and D.L. Vossler. Separation for Boundary to CSG Conversion. *ACM Transactions on Graphics*, 12(1):35–55, January 1993.

[29] S. Teller and C.H. Sequin. Modeling Implicit Quadrics and Free-form Surfaces with Trimmed Rational Quadratic Bezier Patches. Technical Report UCB/CSD90/577, Computer Science Division, University of California at Berkeley, 1990.

[30] R.B. Tilove. Set Membership Classification:A Unified Approach to Geometric Intersection Problems. *IEEE Transactions on Computers*, 29(20):874–883, October 1980.

[31] H.B. Voelcker. New Directions in Solid Modeling? In *International Conference on Manufacturing Automation*, pages 157–168, eds. N.W.M. Kuo and S.T. Tan, University of Hong Kong, August 1992.

[32] B. Wyvill and K. van Overveld. Constructive “Soft” Geometry: A Unification of CSG and Implicit Surfaces. *Preprint, Department of Computer Science*, 1995. University of Calgary.

Dual Control Polygons for Implicit Splines

Baining Guo
Department of Computer Science, York University

Abstract

We present a shape control scheme for free-form surfaces represented by low-degree algebraic patches. In this scheme, we can create manifold surfaces of arbitrary topology through polyhedron smoothing, and the resulting shapes may be modified by changing control points and/or control weights, with each control point/weight having a local effect. A key ingredient of the scheme is an algorithm that uses Kuhn-Tucker conditions to efficiently compute valid control points.

1 Introduction

Shape control is important in free-form geometric modeling, and is becoming increasingly popular as the performance of graphics workstations continue to improve. In parametric surface design, a common technique is to specify a control polygon that generates an initial shape, which is then refined into the final desired shape through interactive adjustments of control points and/or weights. In principle, this technique works well for a variety of modeling tasks.

We present a shape control scheme for implicit splines, i.e., free-form surfaces represented by low-degree algebraic patches. Our scheme extends the control polygon based techniques for parametric surfaces to implicit surfaces. As in parametric surface design, we create free-form implicit surfaces of arbitrary topology through polyhedron smoothing, and refine the resulting shapes into desired ones by changing either control points or weights. Between implicit surfaces and their control points/weights, we establish a relation somewhat similar to that between NURBS surfaces and their control points/weights [Pie89]. In particular, the effects of control points/weights are guaranteed to be *local*.

As is described in [Guo91, Men94], there is a rudimentary shape control scheme for an implicit spline surface (see slide "early shape control methods" in Appendix B). An implicit spline surface S lies inside a "polyhedral hull", whose vertices include the so-called "apexes". Within the polyhedral hull, it is straightforward to modify the shape of S by changing the control weights associated with the apexes. However, shape modifications with control weights alone are not sufficient for free-form shape design – the situation is similar to designing a NURBS surface without the ability to change its control points [Pie89]. To overcome this limitation, we develop a scheme that uses the vertices of the polyhedral hull as control points, thus allowing shape control with both control points and weights.

A main difficulty with changing control points is that of ensuring a valid polyhedral hull, which is crucial for the smoothness of the implicit splines. Given arbitrary target locations of the control points, we cast the control points computation as a series of *least distance programming* problems of small size [LH74]. With these problems in hand, we use the fundamental Kuhn-Tucker conditions from nonlinear programming [BS93] to move the control points as close to their target positions as possible under polyhedral hull validity constraints. When creating an initial free-form surface via polyhedron smoothing, we also use least distance programming to produce a valid polyhedral hull.

This work is motivated by our research on bilateral Brep/CSG schemes for free-form solids [GM95, MG96]. Recently, Wyvill and Van Overveld [WVO95] present a method for modeling with CSG combinations of virtually unlimited variety of "soft" objects (objects bounded by general implicit surfaces; see also [Duf92]). They demonstrate that implicit surfaces in this constructive "soft" geometry can be efficiently polygonized. In general, primitive soft objects are not localized, as such their combinations can produce a variety of interesting and subtle shape effects. On the down side, modifying a primitive may have a global effect.

The shape control scheme we derive uses a few results from previous research. For creating an initial surface, we use a surface fitting method due to Dahmen and Thamm-Schaar [DT93]. Similar tetrahedron-based surface fitting methods have been explored by others [Sed85, Guo91, BC94]. In particular, we mention the polyhedron smoothing method by Bajaj, Chen, and Xu [BC94] that guarantees "hull validity" [BC94]. As an alternative to tetrahedron-based methods, Middleditch and Dimas have developed a promising heptahedron-based technique [MD94].

The remainder of this note is structured as follows. After introducing the shape control scheme in Section 2, we address the key issue of computing valid control points in Section 3. We then provide examples and discussions in Section 4. Appendix A gives details on polyhedral hull existence, whereas Appendix B includes slides for the course presentation.

2 Meshing Algebraic Patches

To design a free-form shape, we first use polyhedron smoothing to construct an initial surface S, which is then refined into the final shape through a sequence of "sculpting" operations.

Notations: We use lower case boldface for vectors, lower case italics for scalars, capital italics for point sets in $\mathbf{R}^3$, and calligraphics for collections of sets in $\mathbf{R}^3$. For example, $||\mathbf{x}|| = (\mathbf{x} \cdot \mathbf{x})^{1/2}$ is the length of vector $\mathbf{x}$, and ∇f is the gradient of a scalar function. We use $[\mathbf{v}_0 ... \mathbf{v}_k]$ to denote the k-simplex spanned by a set of points $\{\mathbf{v}_0, ..., \mathbf{v}_k\}$, e.g. $[\mathbf{v}_0\mathbf{v}_1]$ for an edge.

2.1 General Concepts

A collection $\mathcal{C}$ of simplices forms a *simplicial complex* if it satisfies the following conditions: (a) for a simplex K of $\mathcal{C}$, the boundary simplices of K are in $\mathcal{C}$, and (b) for two simplices of $\mathcal{C}$, their intersection is also a simplex in $\mathcal{C}$. Two simplices satisfying the condition (b) are referred to as *properly joined* (see [HY61]). A simplicial complex $\mathcal{C}$ can have a *subcomplex* $\mathcal{C}'$, which is a simplicial complex satisfying $\mathcal{C}' \subset \mathcal{C}$. The *underlying space* $|\mathcal{C}|$ of a simplicial complex $\mathcal{C}$ is the union of all of its simplices.

We call a tetrahedron over which a polynomial in Bernstein-Bezier form (BB-form) has been defined *a Bezier tetrahedron.* A simplicial complex whose tetrahedra are Bezier tetrahedra is *a Bezier simplicial complex.* Two tetrahedra of a Bezier simplicial complex are *smoothly joined* if, on their common boundary simplex, their polynomials meet with continuous function value and gradient. A Bezier simplicial complex $\mathcal{B}$ is smoothly joined if every two simplices of $\mathcal{B}$ are smoothly joined. We regard a smoothly joined Bezier simplicial complex $\mathcal{B}$ as a smooth piecewise polynomial function $f(\mathbf{x})$ defined on the underlying space $|\mathcal{B}|$.

2.2 Initial Free-form Surface

We construct the initial surface S from a simplicial polyhedron P (a manifold) whose vertices are $\{\mathbf{x}_1, ..., \mathbf{x}_n\}$, with a normal $\mathbf{n}_k$ prescribed at each vertex $\mathbf{x}_k$ ($k = 1, ..., n$). The construction produces as output a smooth surface S that interpolates both the vertices of P and the prescribed normals. The construction uses a surface fitting method due to Dahmen and Thamm-Schaar [DT93], which takes the following steps: (a) build a polyhedral hull $\mathcal{H}$, and (b) define a trivariate function $f(\mathbf{x})$ whose zero contour is $S \subset |\mathcal{H}|$.

First, consider building the polyhedral hull $\mathcal{H}$, which is a simplicial complex consisting of *face tetrahedra*, *wedges*, and their boundary simplices. The face tetrahedra and wedges are obtained as follows:

face tetrahedra Let $F_i = [\mathbf{x}_{i_1}\mathbf{x}_{i_2}\mathbf{x}_{i_3}]$ be a face of P, with the face normal $\mathbf{n}_{F_i}$ pointing to the outside of P. A pair of face tetrahedra are built on each side of the plane $\langle F_i \rangle$ that contains F_i. One is the *exterior* face tetrahedron $\triangle_i = [\mathbf{x}_{i_1}\mathbf{x}_{i_2}\mathbf{x}_{i_3}\mathbf{v}_i^+]$, which is a pyramid using $\mathbf{v}_i^+$ as its apex and F_i as its base. The term exterior indicates that the apex $\mathbf{v}_i^+$ lies on the side of $\langle F_i \rangle$ pointed to by $\mathbf{n}_{F_i}$. The other tetrahedron is the *interior* $\bigtriangledown_i = [\mathbf{x}_{i_1}\mathbf{x}_{i_2}\mathbf{x}_{i_3}\mathbf{v}_i^-]$, which is similar to $\triangle_i$ except the apex $\mathbf{v}_i^-$ lies on the side of $\langle F_i \rangle$ pointed to by $-\mathbf{n}_{F_i}$. The point $\mathbf{v}_i^+$ ($\mathbf{v}_i^-$) is called the *exterior (interior) apex* of F_i.

wedges A pair of wedges are situated between the exterior face tetrahedra $\triangle_i$ and $\triangle_k$ of two faces $F_i = [\mathbf{x}_{i_1}\mathbf{x}_{i_2}\mathbf{x}_{i_3}]$ and $F_k = [\mathbf{x}_{k_1}\mathbf{x}_{k_2}\mathbf{x}_{k_3}]$ sharing an edge $[\mathbf{x}_{i_2}\mathbf{x}_{i_3}] = [\mathbf{x}_{k_2}\mathbf{x}_{k_3}]$. These wedges are $\vee_{ik} = [\mathbf{m}_{ik}^+\mathbf{x}_{i_2}\mathbf{x}_{i_3}\mathbf{v}_i^+]$ and $\vee_{ki} = [\mathbf{m}_{ki}^+\mathbf{x}_{k_2}\mathbf{x}_{k_3}\mathbf{v}_k^+]$, with the *wedge splitting point* $\mathbf{m}_{ik}^+ = \mathbf{m}_{ki}^+$

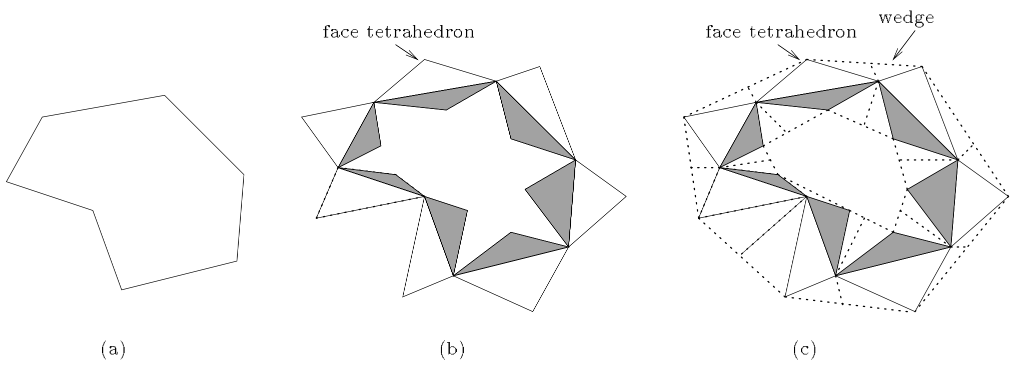


Figure 1: Understanding a polyhedral hull: (a) the polyhedron P, (b) the face tetrahedra – with the interior face tetrahedra shaded, and (c) the polyhedral hull $\mathcal{H}$.

being in the interior of the line segment $[\mathbf{v}_i^+\mathbf{v}_k^+]$. Similarly, a pair of interior wedges $\wedge_{ik}$ and $\wedge_{ki}$ are between the interior face tetrahedra $\bigtriangledown_i$ and $\bigtriangledown_k$.

Figure 1 provides a 2D analogue of the above construction. The two wedges $\vee_{ik}$ and $\vee_{ki}$ may be combined into a single wedge, but doing so complicates the construction of surface S (see [DT93]).

It remains to define the function $f(\mathbf{x})$ whose zero contour is the final surface S. For this, a cubic polynomial is constructed inside each tetrahedron of $\mathcal{H}$. The constructions of cubic polynomials inside the tetrahedra of $\mathcal{H}$ ensure that on the entire underlying space $|\mathcal{H}|$, the function $f(\mathbf{x})$ is not only well defined, but also continuously differentiable. Consequently, the zero contour of $f(\mathbf{x})$, $S = \{\mathbf{x} \in |\mathcal{H}| \mid f(\mathbf{x}) = 0\}$, is a tangent-plane continuous surface. In order for S to interpolate both the vertices of P and the prescribed normals, the function $f(\mathbf{x})$ must satisfy *the interpolation condition*: $f(\mathbf{x}_k) = 0$ and $\nabla f(\mathbf{x}_k) = \mathbf{n}_k$ $(k = 1, ..., n)$.

The interpolation condition and the tangent-continuity of S do not completely determine $f(\mathbf{x})$. For every face F_i of P, both $f(\mathbf{v}_i^+) > 0$ and $f(\mathbf{v}_i^-) < 0$ are left as free *shape parameters* that can be adjusted for shape control (in fact, this leads to the rudimentary shape control scheme in [Men94]). Knowing the shape parameters, we can set the other free parameters in $f(\mathbf{x})$ using the *approximating quadrics* technique: on each tetrahedron of $\mathcal{H}$, $f(\mathbf{x})$ is made to approximate a quadratic polynomial [DT93]. The purpose of approximating quadrics is to ensure that the surface patch inside each tetrahedron of $\mathcal{H}$ is single sheeted and is free of shape defects. While this technique provides no theoretical guarantee (there are methods that do [BC94]), it has been observed to work well in practice. In the following, we will assume that the surface patches inside the tetrahedra of $\mathcal{H}$ are single sheeted.

To summarize, we say that the first step builds a polyhedral hull $\mathcal{H}$, and the second step turns $\mathcal{H}$ into a Bezier simplicial complex $f(\mathbf{x})$ that is smoothly joined and satisfies the interpolation condition. In this note, we concentrate on the problem of building an initial polyhedral hull and maintaining its validity during shape control. We will not discuss the construction of $f(\mathbf{x})$; procedures for finding the initial interpolant and for local updating of $f(\mathbf{x})$ during shape control can be derived from [DT93] quite easily.

2.3 Dual Control Polygons

Now we are ready to describe our scheme for shape control. The set of *control points* of S consists of the vertices of the polyhedron P and the apexes of the polyhedral hull $\mathcal{H}$. The vertices of P are

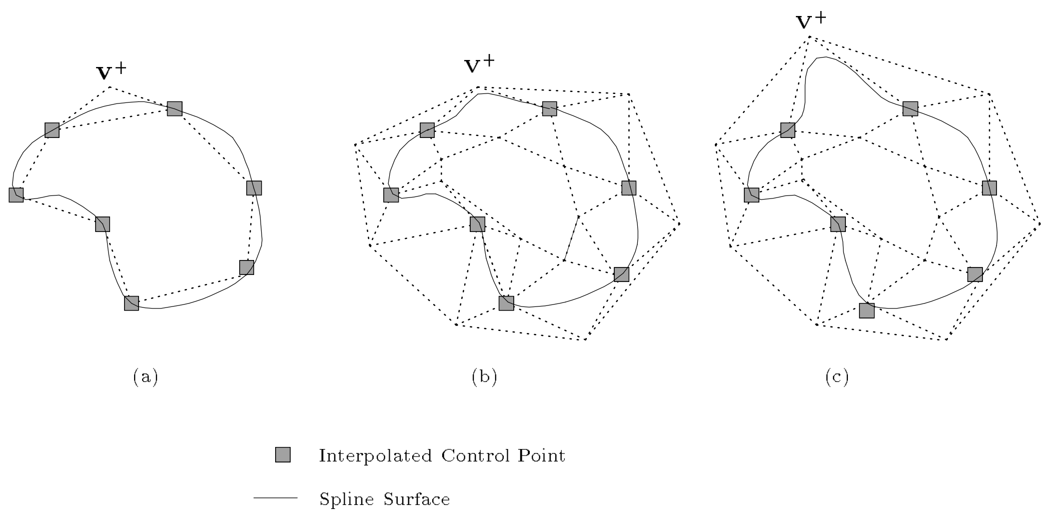


Figure 2: (a) A free-form surface obtained through smoothing a polyhedron. This polyhedron, drawn with dotted lines, has $\mathbf{v}^+$ as one of its exterior apexes. (b) The result of increasing the weight of $\mathbf{v}^+$ while keeping $\mathbf{v}^+$ stationary. (c) The result of moving $\mathbf{v}^+$. In both (b) and (c), we draw the dual control polygons with dotted lines (beware that the 2D drawing has a misleading effect: the interpolated control points appear to be inside the polyhedral hull, even though in 3D they are actually on the polyhedral hull boundary).

interpolated (S passes through these vertices). For each face F_i of the polyhedron P, both apexes $\mathbf{v}_i^+$ and $\mathbf{v}_i^-$ have *control weights*, derived from the shape parameters of S as follows:

$$w_i^+ = \frac{1}{f(\mathbf{v}_i^+)} \text{ and } w_i^- = -\frac{1}{f(\mathbf{v}_i^-)}.$$

The interpolated control points have weights ∞.

The control points determine two related polygons P^+ and P^-, which are the outer and inner boundaries of $|\mathcal{H}|$. The vertices of *exterior* (*interior*) control polygon P^+ (P^-) are the vertices of P plus the exterior (interior) apexes (we ignore the wedge splitting points since they do not influence the shape of $\mathcal{H}$). The polygon pair P^+ and P^- are symmetric with respect to P and meet at the vertices of P. We call this pair the *dual control polygons* of S and S. Figure 2 illustrates the dual control polygons and the effect of control weights/points.

The relation between S and its control points/weights is somewhat similar to that between a NURBS surface and its control points/weights. For example,

- *Control points*: The surface S follows the control points in an intuitive manner (but S does not interpolate a control point unless it has weight of ∞).
- *Control weights*: Increasing a control weight "pulls" the surface S towards the corresponding control point.
- *The polyhedral-hull property*: The surface S always lies inside the polyhedral hull.

While these similarities reflect no connections between the underlying mathematical models, they do allow the user to apply control polygon based techniques to control the shape of S. In Section 4, we will discuss some limitations of our control points when compared to NURBS control points.

The implementation of the dual control polygons raises several complicated issues, among which the most challenging one is that of ensuring a valid polyhedral hull during shape control. We address these issues in the follow sections.

3 Control Points Computation

The most challenging problem in the implementation of the dual control polygons is that of finding the control points. When the user specifies target positions for the control points, these positions may or may not determine a valid polyhedral hull. The objective of the control points computation is to move control points as close as possible to their target positions without violating the polyhedral hull validity constraints.

The constraints on the control points are global, nonlinear inequalities. For smoothing a polyhedron P, any set of apexes satisfying these constraints suffices. To find such apexes, Dahmen and Thamm-Schaar [DT93] first choose apexes satisfying a partial set of the constraints and then traverse the faces of P in some fixed order, making corrections when necessary. Their experiences indicate that after a few traversals, apexes are usually found to satisfy all the constraints.

As our goal is to find control points close to their target positions while respecting the constraints, we handle constraints differently. First, we derive some constraints that involve the interpolated control points only (i.e. the vertices of P). We do so using a linear form of the fundamental Kuhn-Tucker conditions from nonlinear programming [BS93]. The constraints on interpolated control points will be referred to as *the existence conditions of polyhedral hulls*, because the conditions determine, from the polyhedron P and its vertex normals, whether a valid polyhedral hull $\mathcal{H}$ exists.

The existence conditions of polyhedral hulls are weak conditions that can be satisfied for most cases of practical importance. This enables us to move the interpolated control points to their exact target positions. The next step is to find the apexes. We accomplish this step using a quadratic form of Kuhn-Tucker conditions. More specifically, we first break the global, nonlinear inequalities into local, linear inequalities and then we use least distance programming to move the apexes as close as possible to their target positions under the local constraints. As a result, a target position change will affect only the control points and (hence the trunctets) nearby.

Before we detail the control points computation, let us make a simple observation. A polyhedral hull $\mathcal{H}$ has two subcomplexes: (a) the exterior hull $\mathcal{H}_{ext}$ of exterior face tetrahedra, exterior wedges, and their boundary simplices, and (b) the interior hull $\mathcal{H}_{int}$ of interior face tetrahedra, interior wedges, and their boundary simplices. These two subcomplexes form a symmetric partition of $\mathcal{H}$. Because of this symmetry, we can concentrate on the exterior hull $\mathcal{H}_{ext}$ in the following discussion.

3.1 Constraints on Control Points – General

In order to define a valid polyhedral hull $\mathcal{H}$, the control points must satisfy a set of constraints. We will describe the constraints that determine a valid polyhedral hull according to the polyhedral hull model of Dahmen and Thamm-Schaar [DT93], with minor modifications. The purpose of these constraints is to achieve the proper joining of tetrahedra in $\mathcal{H}$ and facilitate surface construction on $|\mathcal{H}|$. We list in the following the constraints on $\mathcal{H}_{ext}$; similar constraints apply to $\mathcal{H}_{int}$.

a) *Tangent containment*: for each vertex $\mathbf{x}_i$ of a face F_k of P, the apex $\mathbf{v}_k^+$ of the exterior face tetrahedron $\triangle_k$ satisfy condition $(\mathbf{v}_k^+ - \mathbf{x}_i) \cdot \mathbf{n}_i > 0$.

b) *Wedge validity*: for each pair of faces F_i and F_k sharing edge $E_{ik} = F_i \cap F_k$, the corresponding exterior face tetrahedra $\triangle_i$ and $\triangle_k$ intersect only along the edge E_{ik}, that is, $\triangle_i \cap \triangle_k = E_{ik}$.

c) *Visibility*: for each pair of faces $F_i = [\mathbf{x}_{i_1}\mathbf{x}_{i_2}\mathbf{x}_{i_3}]$ and $F_k = [\mathbf{x}_{k_1}\mathbf{x}_{k_2}\mathbf{x}_{k_3}]$ sharing a common edge $[\mathbf{x}_{i_2}\mathbf{x}_{i_3}] = [\mathbf{x}_{k_2}\mathbf{x}_{k_3}]$, the corresponding apexes $\mathbf{v}_i^+$ and $\mathbf{v}_k^+$ are mutually visible, that is, the line segment $[\mathbf{v}_i^+\mathbf{v}_k^+]$ lies outside P.

These constraints are illustrated in Figure 3a and b.

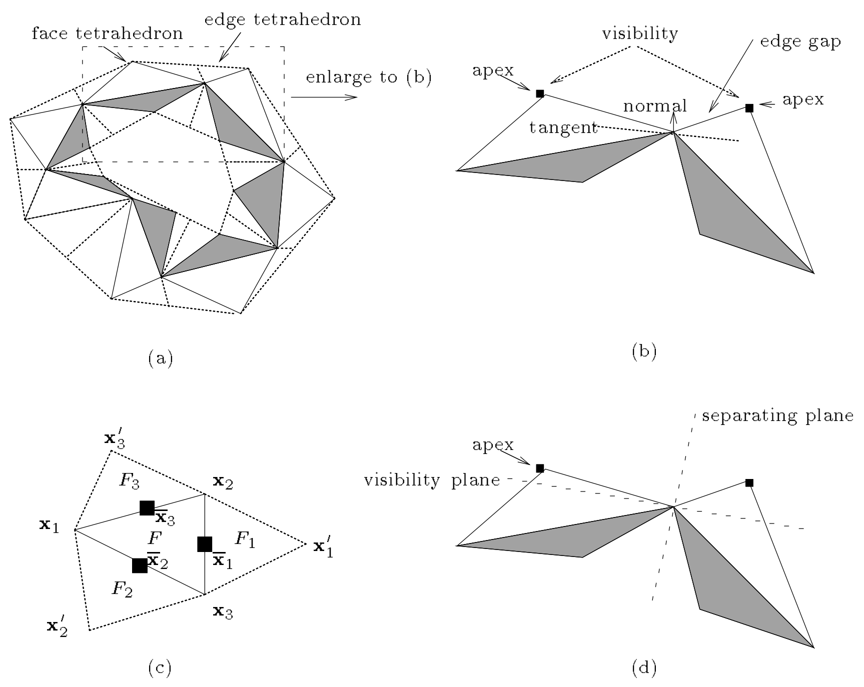


Figure 3: Constructing the polyhedral hull $\mathcal{H}$: (a) face tetrahedra and wedges, (b) the polyhedral hull constraints, (c) the local geometry near face F, and (d) the visibility and dividing planes.

The first two constraints are necessary for building a polyhedral hull $\mathcal{H}$ whose tetrahedra are properly joined and whose underlying space $|\mathcal{H}|$ contains the surface S. Specifically, the purpose of the tangent containment is to ensure that $|\mathcal{H}|$ locally contains the tangent plane defined by the prescribed normal at each vertex. Indeed, together with its counterpart for the interior hull $\mathcal{H}_{int}$, the tangent containment described above guarantees that at each vertex $\mathbf{x}_i$, there is some positive ϵ such that $|\mathcal{H}|$ contains a tangent disk D_ϵ centered at $\mathbf{x}_i$, where $D_\epsilon = \{\mathbf{x} \mid \mathbf{n}_i \cdot (\mathbf{x} - \mathbf{x}_i) = 0,\ ||\mathbf{x} - \mathbf{x}_i|| < \epsilon\}$. As for the wedge validity, it is a prerequisite for constructing the wedges.

The visibility constraint, even though not necessary, provides a simple way for constructing the wedges. More precisely, when the wedge validity and visibility constraints are satisfied for two faces $F_i = [\mathbf{x}_{i_1}\mathbf{x}_{i_2}\mathbf{x}_{i_3}]$ and $F_k = [\mathbf{x}_{k_1}\mathbf{x}_{k_2}\mathbf{x}_{k_3}]$ sharing edge $[\mathbf{x}_{i_2}\mathbf{x}_{i_3}] = [\mathbf{x}_{k_2}\mathbf{x}_{k_3}]$, four tetrahedra $\triangle_i = [\mathbf{x}_{i_1}\mathbf{x}_{i_2}\mathbf{x}_{i_3}\mathbf{v}_i^+]$, $\triangle_k = [\mathbf{x}_{k_1}\mathbf{x}_{k_2}\mathbf{x}_{k_3}\mathbf{v}_k^+]$, $\vee_{ik} = [\mathbf{m}_{ik}^+\mathbf{x}_{i_2}\mathbf{x}_{i_3}\mathbf{v}_i^+]$, and $\vee_{ki} = [\mathbf{m}_{ki}^+\mathbf{x}_{k_2}\mathbf{x}_{k_3}\mathbf{v}_k^+]$ are properly joined for any point $\mathbf{m}_{ik}^+ = \mathbf{m}_{ki}^+$ in the interior of line segment $[\mathbf{v}_i^+\mathbf{v}_k^+]$.

In theory, the above constraints do not guarantee that the exterior hull $\mathcal{H}_{ext}$ is a simplicial complex. Since the constraints only restrict the face tetrahedra of two faces sharing an edge, we may find face tetrahedra $\triangle_i$ and $\triangle_k$ intersect in cases such as: (a) the corresponding faces F_i and F_k share only a vertex, or (b) F_i and F_k are disjointed.

In practical terms, however, the above constraints have been observed to always restrict $\mathcal{H}_{ext}$ to a simplicial complex. The problems with the constraints can also be handled, at the expense of increasing conceptual or computational complexity. Case (a) can be avoided rather easily by replacing the second constraint with a more restrictive one, so that at each vertex of P, all incident face tetrahedra and wedges are properly joined. In fact, this more restrictive constraint has been implemented in our system – even though we find the constraint has little practical impact for all the data that we have experimented with. Case (b) presents a problem that is intrinsically global. A possible solution is to clip, without affecting the relevant surface patches of S, some face tetrahedra and wedges so that the clipped tetrahedra do not overlap.

3.2 Constraints on Interpolated Control Points

From the general constraints on $\mathcal{H}$, we can derive a set of constraints that do not involve the apexes. We call such constraints *the existence conditions* of polyhedral hulls. As is mentioned earlier, these conditions determine, from the polyhedron P and the normals prescribed at its vertices, whether a valid polyhedral hull $\mathcal{H}$ exists.

We study the existence conditions of polyhedral hulls for two reasons. First, these conditions determine to what extent we can move the interpolated control points. Even though the existence conditions are satisfied in most practical cases, identifying these conditions has some theoretical merits. The second reason for studying the existence conditions is that they form the basis of our algorithm for efficient computation of control points.

We start by considering a face $F = [\mathbf{x}_1\mathbf{x}_2\mathbf{x}_3]$ of the polyhedron P and the local geometry near this face. As is illustrated in Figure 3c, each edge E_k $(k = 1, 2, 3)$ opposite to vertex $\mathbf{x}_k$ is shared by F with another face F_k of P. The vertices of F_k are those of E_k plus a new vertex $\mathbf{x}'_k$, and the face F_k is separated from F by the *dividing plane* of E_k, $P_s^k = \{\mathbf{x} \mid s_k(\mathbf{x}) = 0\}$, which passes through E_k and satisfies conditions $s_k(\mathbf{x}_k) > 0$ and $s_k(\mathbf{x}'_k) < 0$. When the internal dihedral angle formed by F and F_k is less than π, the *visibility plane* of E_k, $P_v^k = \{\mathbf{x} \mid v_k(\mathbf{x}) = 0\}$, passes through E_k and induces an open halfspace $H_v^k = \{\mathbf{x} \mid v_k(\mathbf{x}) < 0\}$ whose closure contains both F and F_k; the visibility plane is not necessary if the dihedral angle is greater than π. Denoting the mid-point of edge E_k by $\overline{\mathbf{x}}_k = \frac{1}{2}\sum_{1\le i\le 3,\ i\ne k}\mathbf{x}_i$, we can rewrite the linear functions defining the dividing and visibility planes as $s_k(\mathbf{x}) = \mathbf{s}_k \cdot (\mathbf{x} - \overline{\mathbf{x}}_k)$ and $v_k(\mathbf{x}) = \mathbf{v}_k \cdot (\mathbf{x} - \overline{\mathbf{x}}_k)$, where $\mathbf{s}_k$ and $\mathbf{v}_k$ are the normals of P_s^k and P_v^k respectively.

Now we express the constraints of Section 3.1 in terms of the local geometry near face F. To do this, we construct several cones for F. An open cone with vertex $\mathbf{v}_0$ is an open set C such that $\lambda\ (\mathbf{x} - \mathbf{v}_0) + \mathbf{v}_0 \in C$ for any $\mathbf{x} \in C$ and $\lambda > 0$ [BS93]. We construct the following cones:

a) *Tangent cone*: The tangent cone C_t is formed by the tangent planes: $C_t = \bigcap_{1\le k\le 3} H_t^k$, where $H_t^k = \{\mathbf{x} \mid \mathbf{n}_k \cdot (\mathbf{x} - \mathbf{x}_k) > 0\}$ is an open halfspace bounded by a tangent plane.
b) *Dividing cone*: The dividing cone C_s is formed by the dividing planes: $C_s = \bigcap_{1\le k\le 3} H_s^k$, where $H_s^k = \{\mathbf{x} \mid \mathbf{s}_k \cdot (\mathbf{x} - \overline{\mathbf{x}}_k) > 0\}$ is an open halfspace bounded by a dividing plane.
c) *Visibility cone*: The visibility cone C_v is formed by the visibility planes: $C_v = \bigcap_{1\le k\le 3} H_v^k$, where $H_v^k = \{\mathbf{x} \mid \mathbf{v}_k \cdot (\mathbf{x} - \overline{\mathbf{x}}_k) > 0\}$ is an open halfspace bounded by a visibility plane.

Having constructed these cones, we can restate the constraints of Section 3.1 as follows: for every face F of P, the apex $\mathbf{v}_F^+$ of the exterior face tetrahedron lies inside the *feasible apex set* $A_F = C_t \cap C_s \cap C_v \cap H_F$. Here we use the open halfspace H_F to specify that $\mathbf{v}_F^+$ is an exterior apex. With face normal $\mathbf{f}_F$ and centroid $\mathbf{x}_F$ of F, H_F may be written as $\{\mathbf{x} \mid \mathbf{f}_F \cdot (\mathbf{x} - \mathbf{x}_F) > 0\}$.

Exterior hulls satisfying the constraints of Section 3.1 exist if and only if the feasible apex set is nonempty for every face of P. This leads to the following

Existence conditions of polyhedral hulls: Exterior hulls satisfying tangent containment, wedge validity, and visibility constraints exist if and only if for each face F of P, $a_k = b_k = c_k = d = 0$ $(k = 1, 2, 3)$ are the only non-negative constants satisfying the following constraints: (a) $\sum_{k=1}^{3}(a_k\mathbf{n}_k + b_k\mathbf{v}_k + c_k\mathbf{s}_k) + d\ \mathbf{f}_F = 0$, and (b)

$\sum_{k=1}^{3}(a_k\mathbf{n}_k \cdot \mathbf{x}_k + b_k\mathbf{v}_k \cdot \overline{\mathbf{x}}_k + c_k\mathbf{s}_k \cdot \overline{\mathbf{x}}_k) + d\,\mathbf{f}_F \cdot \mathbf{x}_F \geq 0$. A similar statement can be made about interior hulls.

In the Appendix, we prove the existence condition for exterior hulls using a linear form of Kuhn-Tucker conditions.

The constraint (a) has an intuitive explanation. For a set of vectors $\mathbf{y}_1, ..., \mathbf{y}_k$, the cone spanned by these vectors $\mathbf{y}_1, ..., \mathbf{y}_k$ is the vector set $\{\mathbf{y} \mid \mathbf{y} = \lambda_1\mathbf{y}_1 + \cdots + \lambda_k\mathbf{y}_k,\ \lambda_1, ..., \lambda_k \geq 0\}$ (e.g. see [BS93]). Constraint (a) simply says that the cone spanned by $\mathbf{f}_F$ and $\mathbf{n}_k$ $(k = 1, 2, 3)$ intersects that spanned by $-\mathbf{v}_k$ and $-\mathbf{s}_k$ $(k = 1, 2, 3)$.

Before leaving this section, we note that the above analysis applies to other models of polyhedral hulls as well. For example, we may choose to drop the visibility constraint since it is not a necessary one.

3.3 Initial Control Points and Updates

Now we are ready for the control points computation, which takes two steps: (a) determine the interpolated control points, and (b) find the apexes. Determining the polyhedron P is simple if we assume that P and its vertex normals satisfy the existence condition of polyhedral hulls, which we do. As a vertex $\mathbf{x}_k$ of P is changed during shape control, we only need to update the normal $\mathbf{n}_k$. This is done with a simple averaging scheme: the new $\mathbf{n}_k$ is set to a weighted average of the normals of the incident faces, with each face normal inversely weighted by the area of the face. This averaging scheme is also used when we are given an initial polyhedron P with no prescribed normals at its vertices.

Finding the apexes is more complicated. Moving apexes to their exact target positions can easily create an invalid polyhedral hull. Instead, we move apexes as close as possible to their target positions under the polyhedral hull validity constraints. From Section 3.2, we see that the computation of such apexes requires that we find, for each edge of P, a dividing plane and a visibility plane. The first question is then: should we determine these planes before or during the computation of the apexes?

Determining the dividing and visibility planes during the computation of the apexes presents a fundamental difficulty, namely, that of solving a global system of nonlinear inequalities. With the cones of Section 3.2 for every face, the global system relating the apexes, the dividing and visibility planes can be easily derived. This system is global because the apexes are related through the dividing and visibility planes, each of which is used by two faces of P. See Figure 3d. To verify that the system is nonlinear, recall that in Section 3.2, the apex $\mathbf{v}_F^+$ satisfies constraints of the sort $\mathbf{s}_k \cdot (\mathbf{v}_F^+ - \overline{\mathbf{x}}_k) > 0$ $(k = 1, 2, 3)$. These constraints are nonlinear if both $\mathbf{v}_F^+$ and $\mathbf{s}_k$ are unknowns.

We avoid the above difficulty by choosing dividing and visibility planes before the computation of apexes. We call this a *divide-first* approach. For each edge E_{ik} of P, we set the dividing plane to pass through E_{ik} and bisect the dihedral angle formed by the two faces F_i and F_k sharing E_{ik}. As for the visibility plane, we let it pass through E_{ik} and have normal $\frac{1}{2}(\mathbf{f}_i + \mathbf{f}_k)$, where $\mathbf{f}_i$ and $\mathbf{f}_k$ are the unit normals of the faces F_i and F_k respectively. The dividing and visibility planes break the global, nonlinear system into a series of local, linear systems, one for each face of P. For example, the system for the face $F = [\mathbf{x}_1\mathbf{x}_2\mathbf{x}_3]$ considered in Section 3.2 is $\mathbf{f}_F \cdot (\mathbf{v}_F^+ - \mathbf{x}_F) > 0$, $\mathbf{n}_k \cdot (\mathbf{v}_F^+ - \mathbf{x}_k) > 0$, $\mathbf{s}_k \cdot (\mathbf{v}_F^+ - \overline{\mathbf{x}}_k) > 0$, and $\mathbf{v}_k \cdot (\mathbf{v}_F^+ - \overline{\mathbf{x}}_k) > 0$ $(k = 1, 2, 3)$. This system of constraints determines the feasible apex set A_F. In the following, we use the face F as an example to demonstrate the computation of apexes.

Let $\mathbf{v}_t^+$ be the target position of the apex $\mathbf{v}_F^+$. To move $\mathbf{v}_F^+$ as close as possible to $\mathbf{v}_t^+$ without violating the polyhedral hull validity constraints, we compute the apex $\mathbf{v}_F^+$ by solving the following *least distance programming* (LDP) problem

$$\begin{array}{ll} \text{Minimize} & ||\mathbf{v}_F^+ - \mathbf{v}_t^+||^2 \\ \text{subject to} & \mathbf{s}_k \cdot (\mathbf{v}_F^+ - \overline{\mathbf{x}}_k) \geq \epsilon,\ \mathbf{v}_k \cdot (\mathbf{v}_F^+ - \overline{\mathbf{x}}_k) \geq \epsilon,\ \mathbf{n}_k \cdot (\mathbf{v}_F^+ - \mathbf{x}_k) \geq \epsilon\ (k = 1, 2, 3), \quad \text{and} \\ & \mathbf{f}_F \cdot (\mathbf{v}_F^+ - \mathbf{x}_F) \geq \epsilon_h \end{array}$$

where ϵ is a small positive constant. The constraints in this LDP problem model the feasible apex set A_F within an error tolerance of ϵ, and the constant ϵ_h controls the minimum distance between the apex $\mathbf{v}_F^+$ and the face F. We use an LDP algorithm by Lawson and Hanson, who also give their FORTRAN code [LH74]. Like most algorithms for solving LDP problems, the Lawson-Hanson algorithm uses a quadratic form of Kuhn-Tucker conditions and implicitly verifies the existence condition of Section 3.2 to determine whether the above LDP problem has solutions. For an initial input polyhedron P with no target apex positions, we set $\mathbf{v}_t^+ = \mathbf{x}_F$, which has the effect of avoiding "sharp" wedges.

We note that LDP can be efficiently solved despite the fact that it is a form of quadratic programming (QP), which minimizes a general quadratic function $\mathbf{c} \cdot \mathbf{x} + \mathbf{x}\mathbf{M}\mathbf{x}$ subject to a set of linear inequality constraints. The computational complexity of QP varies according to the symmetric matrix $\mathbf{M}$. QP is NP-complete if $\mathbf{M}$ is indefinite [BS93]. A QP with positive semi-definite $\mathbf{M}$ is referred to as a *convex* problem, and is solved in practice with efficient methods whose worst case behavior is exponential but whose expected running time is polynomial. Finally, LDP has a positive definite matrix $\mathbf{M}$ and can be solved with methods that are efficient in both theoretical and practical senses [LH74]. Of course, our procedure for computing an apex takes constant time because we solve an LDP of ten inequality constraints.

The procedure we have described so far is for computing the exterior apexes. A procedure for computing the interor apexes can be obtained by what we call *normal flipping*. Because of the symmetry between interor hulls and exterior hulls, we compute an interior hull by following the exterior hull procedure with flipped vertex and face normals, $\{-\mathbf{n}_1, ..., -\mathbf{n}_n\}$ and $\{-\mathbf{f}_F \mid F \text{ is a face of } P\}$.

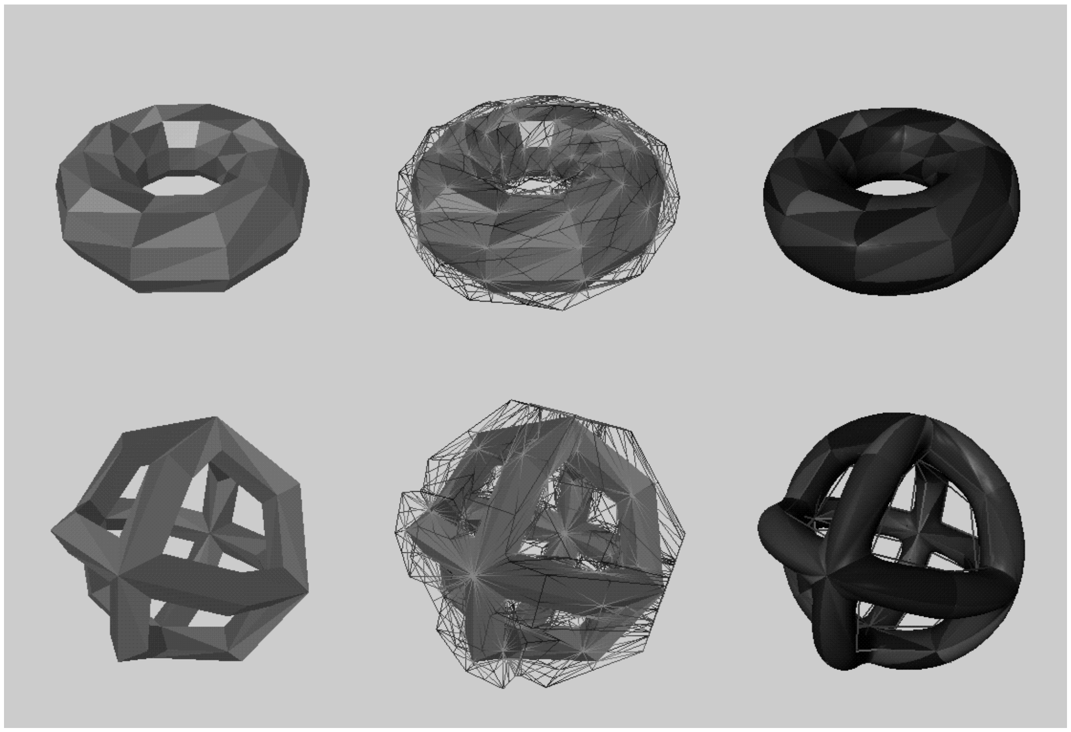


Figure 4: Polyhedron smoothing examples. Left column: The initial polyhedra. Center column: The initial polyhedra with polyhedral hulls. Right Column: Resulting spline surfaces.

4 Examples and discussions

Overall, our experiments indicate that the shape control scheme is effective, both in creating initial shapes from polyhedron smoothing and in shape refinements through adjustments of control points and weights. Let us show some sample results. Figure 4 provides polyhedron smoothing examples, with wireframe polyhedral hulls and Gouroud-shaded trunctets color-coded according to the faces of the initial polyhedra. Note that objects of complicated topology (genus eight is the bottom example) are handled easily. In Figure 5, the torus in Figure 4 is refined into different shapes through adjustments of control points/weights. A similar example is given in Figure 6. In all the examples, the spline surfaces are tangent-plane continuous, but may contain curvature discontinuities.

The shape control scheme we derive does have some limitations when compared to its parametric surfaces counterparts. First, the movements of our apexes (and, to some extent, the vertices of P) are subject to the polyhedral hull validity constraints, whereas the control points of parametric surfaces move freely. Second, the interpolated control points in our system have side effects. When the user moves a vertex $\mathbf{x}_k$ of P, the constraints for the apexes of all faces incident to $\mathbf{x}_k$ (and only these faces) change. If the change of constraints makes an apex violate the polyhedral hull validity constraints, our system will move this apex to the nearest position where the validity constraints are satisfied. The control points of parametric surfaces do not have such side effects.

In the control points computation, we have assumed that the existence conditions of polyhedral hulls are satisfied for the polyhedron P and its (derived or prescribed) vertex normals. This assumption holds in most cases – not only for the initial polyhedron but also for polyhedra in shape

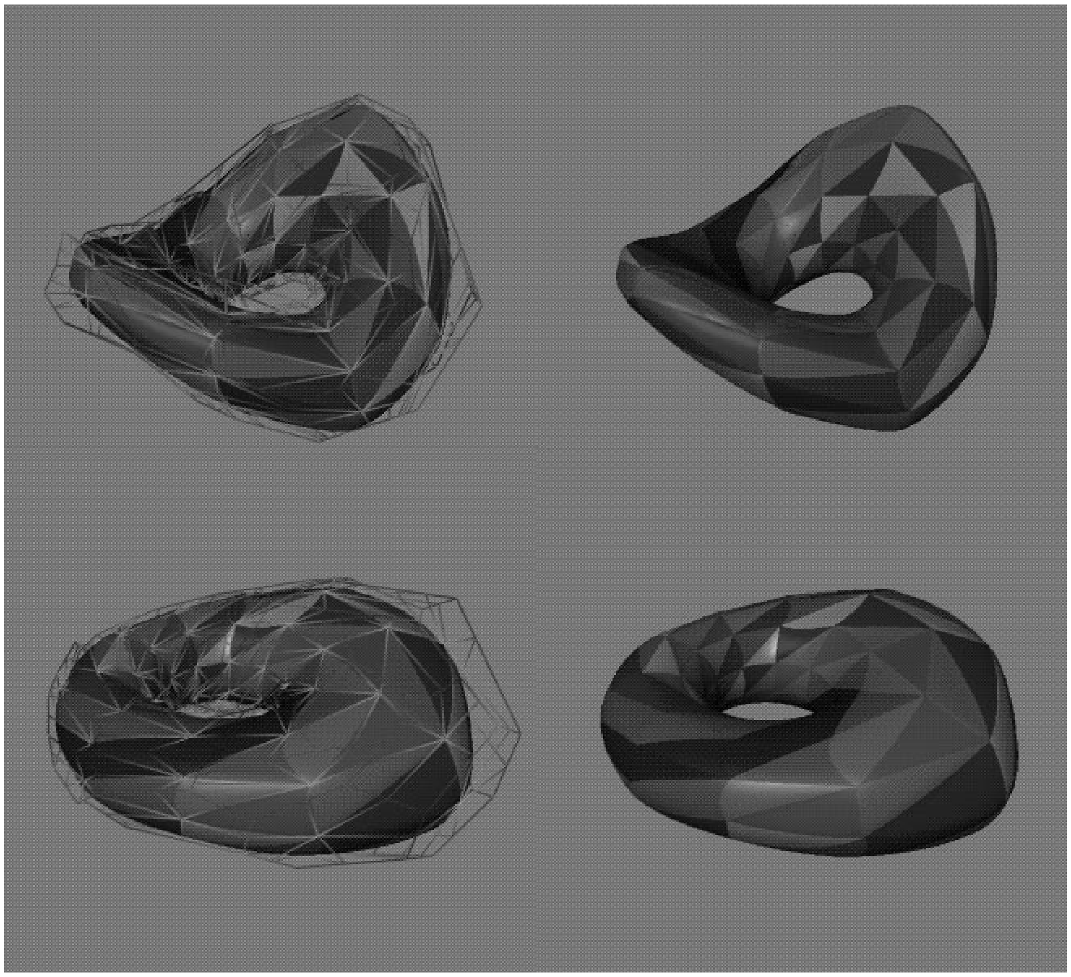


Figure 5: The torus refined into different shapes, with the corresponding control polygons shown in the left column. Many local shape control operations are combined to modify a half of the torus (the bottom row) and the entire torus (the top row).

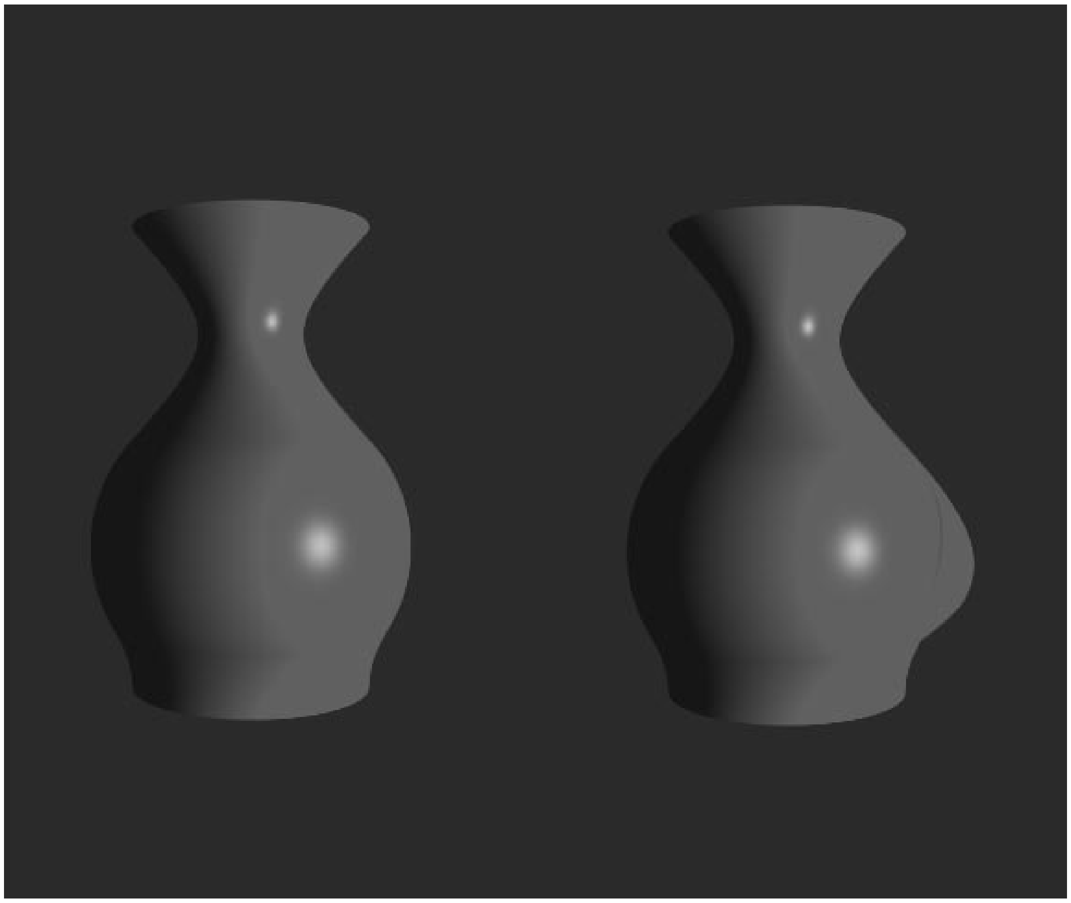


Figure 6: Left: The original shape (obtained via polyhedron smoothing). Right: The result of changing a control point (an apex) and its weight. In contrast to the torus examples, we see a completely local shape effect here.

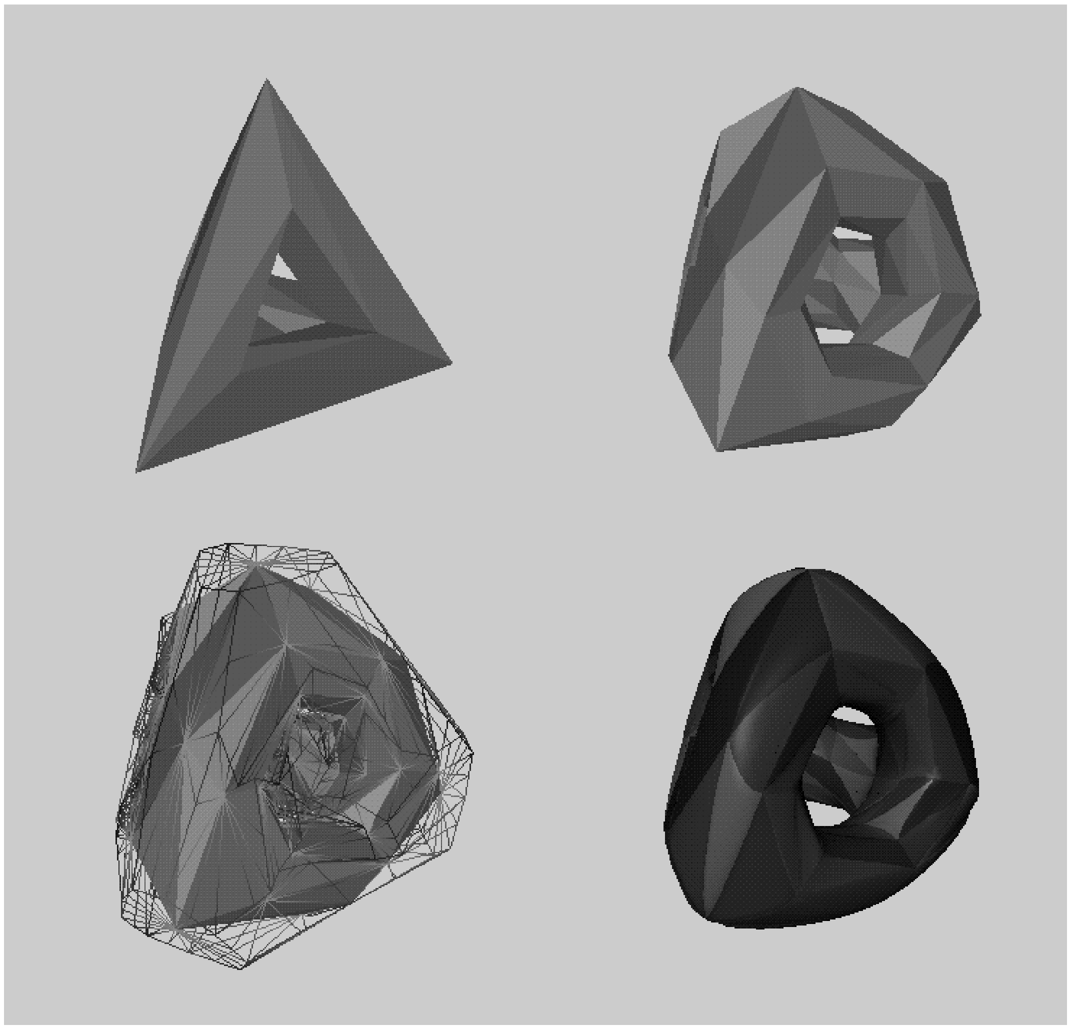


Figure 7: Top left: A polyhedron needing subdivision. Top right: The subdivided polyhedron. Bottom left: The subdivided polyhedron with a polyhedral hull. Bottom right: The resulting spline surface.

refinements, if these polyhedra are obtained from the initial polyhedron through small adjustments of vertices. However, there are cases in which our assumption fails, as is the case with the polyhedron shown in Figure 7. To handle a polyhedron P of this sort, we have developed a successful heuristic method. Because the apexes computation verifies the existence conditions of polyhedral hulls for each face of P (see Section 3.3), we can use the results of the verification to locate the sharp features in P that cause the violation of the existence conditions. We remove these sharp features by locally applying a subdivision surface technique [Loo87]. See Figure 7. Of course, this is a heuristic method and, like any other heuristic method, can be made to fail on pathological examples.

5 Acknowledgements

I would like to thank Jai Menon for being such a competent collaborator. Many thanks to Brian Wyvill for sending me his paper on constructive soft geometry and for useful communications. As always, my work benefits a great deal from the stimulating environment of the Dynamic Graphics Project, directed by Eugene Fiume at the University of Toronto. Colleagues Jörg Peters and Henry Moreton kindly supplied test data for the algorithm presented here. Finally, I am grateful to the National Science Foundation (USA) for a research fellowship.

References

[BC94] C. Bajaj, J. Chen, and G. Xu. Smooth Low Degree Approximations of Polyhedra. Technical Report TR-94-002, Department of Computer Science, Purdue University, 1994.

[BS93] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. *Nonlinear Programming: Theory and Algorithms.* John Wiley & Sons, 1993.

[DT93] W. Dahmen and T.-M. Thamm-Schaar. Cubicoids: modeling and visualization. *Computer Aided Geometric Design*, 10:89-108, 1993.

[Duf92] T. Duff. Interval arithmetic and recursive subdivision for implicit functions and constructive solid geometry. *Computer Graphics*, 26(2):131-138, 1992.

[GM95] B. Guo and J. P. Menon. Local shape control for free-form solids in exact CSG representation. *Computer Aided Design*, to appear, 1995.

[Guo91] B. Guo. *Modeling Arbitrary Smooth Objects with Algebraic Surfaces.* PhD thesis, Cornell University, August 1991.

[HY61] J. G. Hocking and G. S. Young. *Topology.* Addison-Wesley, 1961.

[Loo87] C. Loop. *Smooth Subdivision Surfaces Based on Triangles.* M.S. Thesis, Department of Mathematics, University of Utah, 1987.

[LH74] C. Lawson and R. Hanson. *Solving Least Square Problems.* Prentice-Hall, 1974.

[MD94] A. E. Middleditch and E. Dimas. Solid models with piecewise algebraic free-form faces. In proceedings of *Set-theoretic Solid Modeling: Techniques and Applications – CSG '94*, pages 133-148, Winchester, UK, 1994.

[Pie89] L. Piegl. Modifying the shape of rational B-splines. Part 2: surfaces. *Computer Aided Design*, 21(9):538-546, 1989.

[MG96] J. P. Menon and B. Guo. A framework for sculptured solids in exact CSG representation. In proceedings of *Set-theoretic Solid Modeling: Techniques and Applications – CSG '96*, Winchester, UK, 1996.

[Men94] J.P. Menon. Constructive shell representations for free-form surfaces and solids. *IEEE Computer Graphics & Applications*, 14(2):24–36, March 1994.

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, 1986.

[Sed85] T.W. Sederberg. Piecewise algebraic surface patches. *Computer Aided Geometric Design*, 2:53–59, 1985.

[WVO95] B. Wyvill and K. van Overveld. Constructive "soft" geometry: an unification of CSG and implicit surfaces. Preprint, Department of Computer Science, University of Calgary, 1995.

A Polyhedral Hull Existence

We prove the existence condition of polyhedral hulls stated in Section 4.2. The following theorem is taken from [Sch86], page 95, (33).

Theorem 1 (Carver) *For a matrix* $\mathbf{A}$, $\mathbf{Ax} > \mathbf{b}$ *has a solution, if and only if* $\mathbf{y} = \mathbf{0}$ *is the only solution for* $\mathbf{yA} = 0,\ \mathbf{y} \cdot \mathbf{b} \geq 0,\ \mathbf{y} \geq 0$.

The feasible apex set A_F can be defined as follows:

$$\{\mathbf{x} \mid \mathbf{f}_F \cdot (\mathbf{x} - \mathbf{x}_F) > 0,\ \mathbf{n}_k \cdot (\mathbf{x} - \mathbf{x}_k) > 0,\ \mathbf{v}_k \cdot (\mathbf{x} - \overline{\mathbf{x}}_k) > 0,\ \mathbf{s}_k \cdot (\mathbf{x} - \overline{\mathbf{x}}_k) > 0\ (k = 1, 2, 3)\ \}$$

To see when A_F is nonempty, we construct a 10×3 matrix $\mathbf{A}$ whose row vectors are $\mathbf{n}_k$, $\mathbf{v}_k$, $\mathbf{s}_k$ $(k = 1, 2, 3)$, and $\mathbf{f}_F$. We also construct two ten-component vectors

$$\begin{aligned} \mathbf{b} &= [\mathbf{n}_1 \cdot \mathbf{x}_1,\ \mathbf{n}_2 \cdot \mathbf{x}_2,\ \mathbf{n}_3 \cdot \mathbf{x}_3,\ \mathbf{v}_1 \cdot \overline{\mathbf{x}}_1,\ \mathbf{v}_2 \cdot \overline{\mathbf{x}}_2,\ \mathbf{v}_3 \cdot \overline{\mathbf{x}}_3,\ \mathbf{s}_1 \cdot \overline{\mathbf{x}}_1,\ \mathbf{s}_2 \cdot \overline{\mathbf{x}}_2,\ \mathbf{s}_3 \cdot \overline{\mathbf{x}}_3,\ \mathbf{f}_F \cdot \mathbf{x}_F] \\ \mathbf{y} &= [a_1,\ a_2,\ a_3,\ b_1,\ b_2,\ b_3,\ c_1,\ c_2,\ c_3,\ d] \end{aligned}$$

An application of Carver's theorem gives the existence condition for simplicial hulls.

Carver's theorem is a variant of the duality theorem of linear programming, which is a linear form of Kuhn-Tucker conditions. Even though Kuhn-Tucker conditions only became well known through the historical paper of Kuhn and Tucker in 1951 [BS93], Carver's theorem is part of a much earlier result, namely, *the fundamental theorem of linear inequalities* due to Farkas, Minkowski, Caratheodory, and Weyl. See page 209 of [Sch86] for historical notes. In particular, the connection between the duality theorem of linear programming and Kuhn-Tucker conditions is discussed on page 220.

B Presentation Slides

Dual Control Polygons for Implicit Splines

Baining Guo
Department of Computer Science
York University

Outline

¥ Motivation
¥ Meshing algebraic patches
¥ The dual control polygon scheme
- constraints on control points
- computation of control points
- global and local shape effects

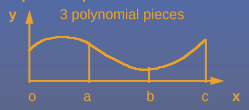
¥ Comparison with NURBS

Motivation

¥ Main goal: shape manipulation with control polygon
¥ Desirable properties
- local shape control
- arbitrary topology
- low-degree patches

Splines as Finite Elements

¥ A simple example



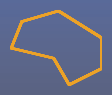
¥ Domain decomposition: domain -> elements
[o c] -> [o a] + [a b] + [b c]
¥ Triangular (2D) and tetrahedral elements (3D)

Meshing Algebraic Patches

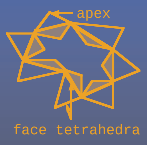
¥ Problem: polyhedron smoothing
given: a polyhedron **P** with vertex normals
find: an interpolative C1 mesh of algebraic patches, each bounded by a tetrahedron
¥ **Solution:** construct finite elements on a poly hull

Finite Element Construction

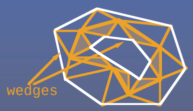
¥ Domain synthesis: tetrahedra -> poly hull
¥ **Idea:** build a Òthick shellÓ around **P**
(illustrated with a 2D analogue)



¥ First, build face tetrahedra



¥ Then, add wedges

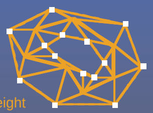


¥ Now, finite elements can be adapted from popular schemes (e. g. Clough-Tocher)

Examples of Meshing

Early Shape Control Methods

¥ Mainly based on ***apex weight***



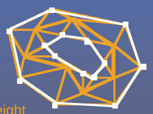
■ apex weight
- origin: from free parameters in FE
- effect: patch push/pull

Shape Handles in Poly Hull

¥ Apex weights
¥ Vertices of polyhedral hull
¥ Apex weights only => limited shape effects
¥ Want a more comprehensive scheme

Dual Control Polygon Scheme

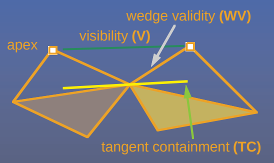
¥ Control polygons Òwrapped aroundÓpoly hull



■ apex weight

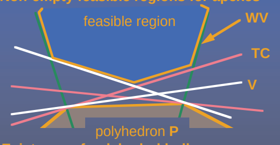
¥ A weight assigned to each control point
- interpolated vertices: weight = infinity
- changing a weight pulls/pushes a patch

Constraints on Control Points



Hard Constraints

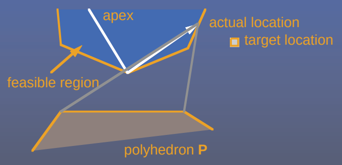
¥ Non-empty feasible regions for apexes



¥ Existence of polyhedral hull
¥ Kuhn-Tucker conditions (for LP)

Soft Constraints

¥ Within hard constraints, apexes should best approximate target locations



Apex Computation

¥ Linear constraints, quadratic objective function => least distance program (LDP)
¥ LDP is efficiently computable, based on Kuhn-Tucker conditions (for QP)
¥ Default target locations

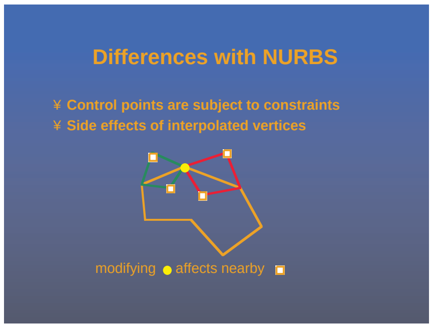
Global and Local Shape Effects

¥ Individual control point/weight (local effect)
¥ Multiple control point/weights (global effect)
- space warping
- free-form deformation

¥ Examples

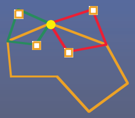
Similarity with NURBS

¥ Effects of control points/weights
¥ The polyhedral hull property (compared to the convex hull property)

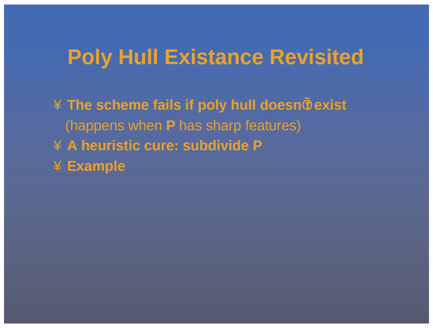


Differences with NURBS

¥ **Control points are subject to constraints**
¥ **Side effects of interpolated vertices**

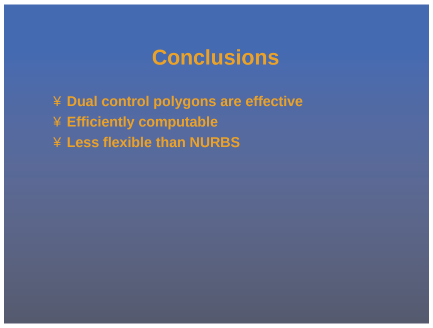


modifying ● affects nearby ▣



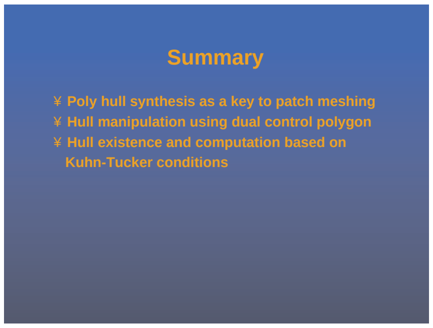
Poly Hull Existance Revisited

¥ **The scheme fails if poly hull doesnÕt exist**
(happens when **P** has sharp features)
¥ **A heuristic cure: subdivide P**
¥ **Example**



Conclusions

¥ **Dual control polygons are effective**
¥ **Efficiently computable**
¥ **Less flexible than NURBS**



Summary

¥ **Poly hull synthesis as a key to patch meshing**
¥ **Hull manipulation using dual control polygon**
¥ **Hull existence and computation based on Kuhn-Tucker conditions**

SECTION C
Beyond Low Degree Algebraics

Abstract

Progressing beyond implicit polynomial forms, the third section of the course will cover several aspects of implicit skeletal methods, including tiling techniques, creation of smooth volumes, bulge avoidance, blending volumes with surfaces, direct CSG ray tracing, animation design including space warping and metamorphosis, and surface and solid texturing. This section will then show how common fractal representations of rough surfaces can be reformulated into an implicit definition. A new geometric rendering technique called sphere tracing, that properly renders rough implicit surfaces, will be described. Given the rough implicit surface model, the course will show how many of the geometry processing operations applied to smooth surfaces for computer-aided geometric design now extend to rough surfaces for modeling natural phenomena, with an example of grafting a stem on a leaf and merging a tree bark texture at a branch point. Switching the focus to interactive rendering, a new technique based on gradient dynamical systems, that maintains correct topology of implicit surfaces during direct manipulation, will be discussed for fast computation of a polygonal representation of the implicit surface.

◊

Tiling Techniques for Implicit Skeletal Models

Brian Wyvill and Kees van Overveld*

Abstract

An overview of implicit surface polygonization (tiling) techniques is presented and details of two algorithms; uniform space subdivision and *Shrinkwrap*. The former performs well on generalised skeletal data although it is non-adaptive. The latter is a fast adaptive surface following technique which is limited to single sheet surfaces. A description is given of an extension to the uniform algorithm which includes CSG operations between groups of blended primitives.

1 Introduction

Visualizing implicit surfaces is not straightforward. Traditional modelling methods such as parametric surfaces lend themselves to visualization since it is easy to iterate over points on the surface which can be found directly from the defining equations. Implicit surface models (ISM's) are defined by black box functions from which a value can be calculated for any point in space a scalar field. The model is defined by an iso-surface in the field. Two main visualization methods are available, direct visualization (i.e. ray tracing) and polygonization in which the surface is approximated by a mesh of planar polygons. These methods involve sampling space at some chosen points, evaluating a function to find a scalar value at that point and comparing the value to the iso-value to determine if the point is inside or outside the surface.

There are two main approaches to solving the space search problem:

1. *Space Partioning.* Partioning space into manageable units such as cubes.

2. *Non-space Partioning.* The Shrinkwrap Algorithm.

In practice a designer wants to visualize an Implicit Surface Model (ISM) quickly, sacrificing quality for speed for interaction purposes. Algorithms for prototyping ISM's have been concerned with producing a polygon mesh which can be rendered in real time on modern workstations. Finding the polygonal mesh which best approximates the desired surface is referred to as *polygonization* or *surface tiling.* For animation or for a final visualization where quality can be traded for speed ray tracing implicit surfaces directly without first polygonizing produces excellent results. (See chapter on ray tracing.)

For a detailed review of these techniques see [15] and [9].

2 Space Partioning

2.1 Exhaustive Search

A first approach to finding the implicit surface might be to uniformaly subdivide space into a regular lattice of cubic cells and calculate a value for every vertex (see [11]). Each cell is replaced with a set of polygons that best approximates the part of the surface contained within that cell.

The problem with this algorithm is that many of the cells will be completely outside or completely inside the volume, thus many cells that contain no part of the surface are processed. For large grids of data this can be very time consuming.

3 A Practical Method

The method described here is based on the data structures used in [22], with the addition of table driven polygonization and tetrahedral decomposition. Working software was published in Graphics Gems IV [2]. The algorithm is based on *numerical continuation*; it starts with a seed cube which intersects part of the surface and builds neighboring cubes as necessary to follow the surface.

The algorithm has two parts. In the first, cubic cells are found that contain the surface, and, in the second, each of these cells are replaced by triangles. The first part of the algorithm is driven by a queue of cubic cells, each of which contains part of the surface; the second part of the algorithm is table driven.

3.1 Continuation Algorithm

This algorithm subdivides space into a cubic lattice. Cubes that are examined by the algorithm are stored in a hash table; initially the table is empty. The algorithm starts from a seed cube that is found to contain part of the surface. The neighbors of the seed cube are examined. If any edge has vertex values which are opposite in sign, then the surface cuts that edge. The cube containing that edge will then be processed and all its neighbors and so on until the entire surface has been covered. See figure 1.

Each cube is indexed by an *identifying vertex*, which we define to be the lower, left, far corner (i.e. the vertex with the least x,y,z coordinate values; see Figure 2). The identifying vertex is addressed by an integer i, j, k , computed from the x, y, z coordinate location of the cube such that $x = side * i$ etc. where *side* is the size of the cube. The identifying vertex of each cube may appear in as many as eight other cubes, and it would be inefficient to store these vertices multiple times. Thus, the vertices are stored uniquely in a chained hash table.

Since most of the space does not contain any part of the surface, only those cubes which are visited will be stored. The implicit function value is found for each vertex as it is stored in the hash table.

Since the surfaces we deal with are closed, at some point a cube will be re-visited. It is necessary to keep track of which cubes have already been processed. This could be done by keeping a separate cube table, however, for space efficiency a Boolean is stored in the hash table to indicate that the cube indexed by the inentifying vertex has been visited.

To process a cube we examine each face. If any of the bounding edges have oppositely signed vertices the surface will pass through that face and the face neighbour must be processed. When this process has been competed for all the faces, the second phase of the algorithm is applied to the cube.

3.1.1 Data Structures

The Hash Table entry holds five values:

- i, j, k lattice indices of the identifying vertex.
- f implicit function value of the identifying vertex.

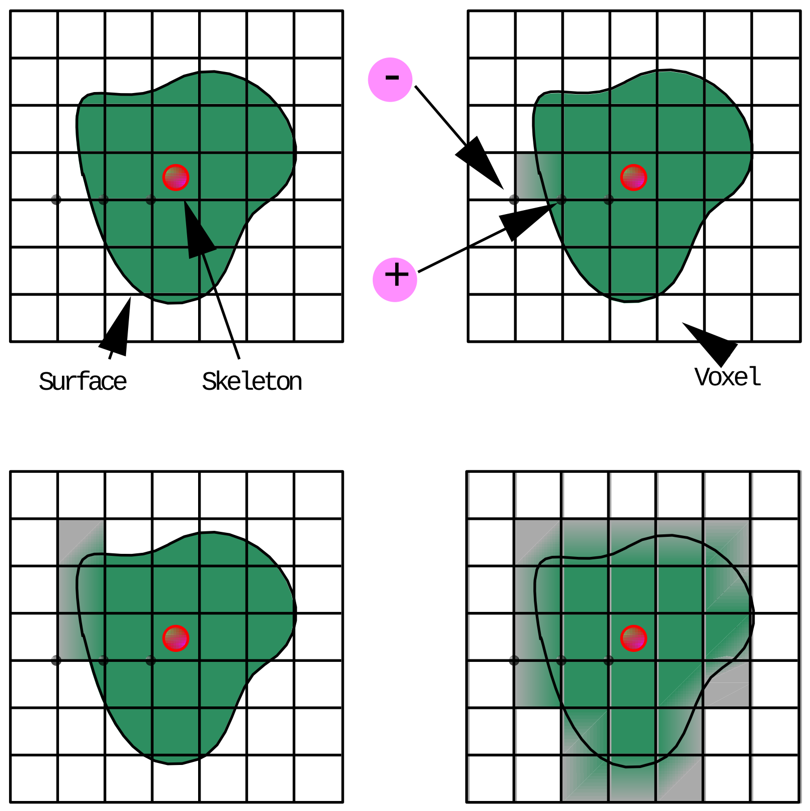
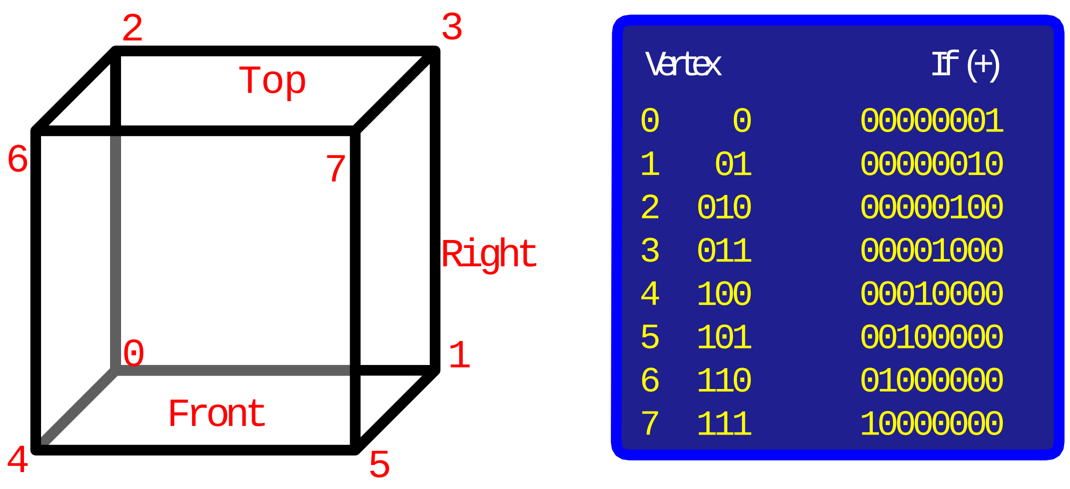


Figure 1: A section through the cubic lattice



Vertex		If (+)
0	0	00000001
1	01	00000010
2	010	00000100
3	011	00001000
4	100	00010000
5	101	00100000
6	110	01000000
7	111	10000000

Figure 2: Cubic Cell Vertex Numbering

```
begin
   Set seed cube’s done flag to true
   Add seed cube to the queue.
   while queue is not empty do
   begin
      remove one cube from the queue
      for each face of cube do
      begin if surface intesects face then
          begin select neighbour cube for that face
             if neighbours done flag is not true then
             begin set neighbours done flag to true
                add neighbour to queue
             end
          end
      end
      Pass cube to second stage
   end
end
```

Figure 3: Continuation Method Algorithm

- *Boolean* to indicate whether this cube has been visited.

The hash function computes an address in the hash table by selecting a few bits out of each of i, j, k and ORing them together. For example, the 5 least significant bits to produce a 15 bit address for a table which must have a length of 2^{15}. (Specific details are given in Graphics Gems IV [2] and [22]).

The Queue ($FIFO$ list) is used as temporary storage to identify the neighbors for processing (others have used a stack ($LIFO$ list) although there is some evidence that the queue processes the cubes in a more memory efficient order). The algorithm begins with a seed cube that is marked as visited and placed on the queue. The first cube on the queue is dequeued and all its unvisited neighbors added to the queue. Each cube is processed and if it contains part of the surface output to the second phase of the algorithm. The queue is then processed until empty. The *continuation* algorithm proceeds as indicated in pseudo code see 3.

3.2 Polygonization Algorithm

The second phase of the algorithm treats each cubic cell independently. The cell is replaced by a set of triangles that best matches the shape of the part of the surface that passes through the cell. The algorithm must decide how to polygonise the cell given the implicit function values at each vertex. These values will be positive or negative (ie less than or greater than the iso-value) giving 256 combinations of positive or negative vertices for the eight vertices of the cube. Earlier methods employed a table of 256 entries which provided the right vertices to use in each triangle figure 4. However this leads to ambiguities when opposite corners of a face (or the cube) have the same sign and the other pair of vertices on the face the opposite sign (see Figure 5). This problem is avoided by subdividing the cubic cell into tetrahedra. The tetrahedra can then be polygonized unambiguously. Since there are four vertices in each tetrahedron, a table of sixteen entries will provide the correct triangle information.

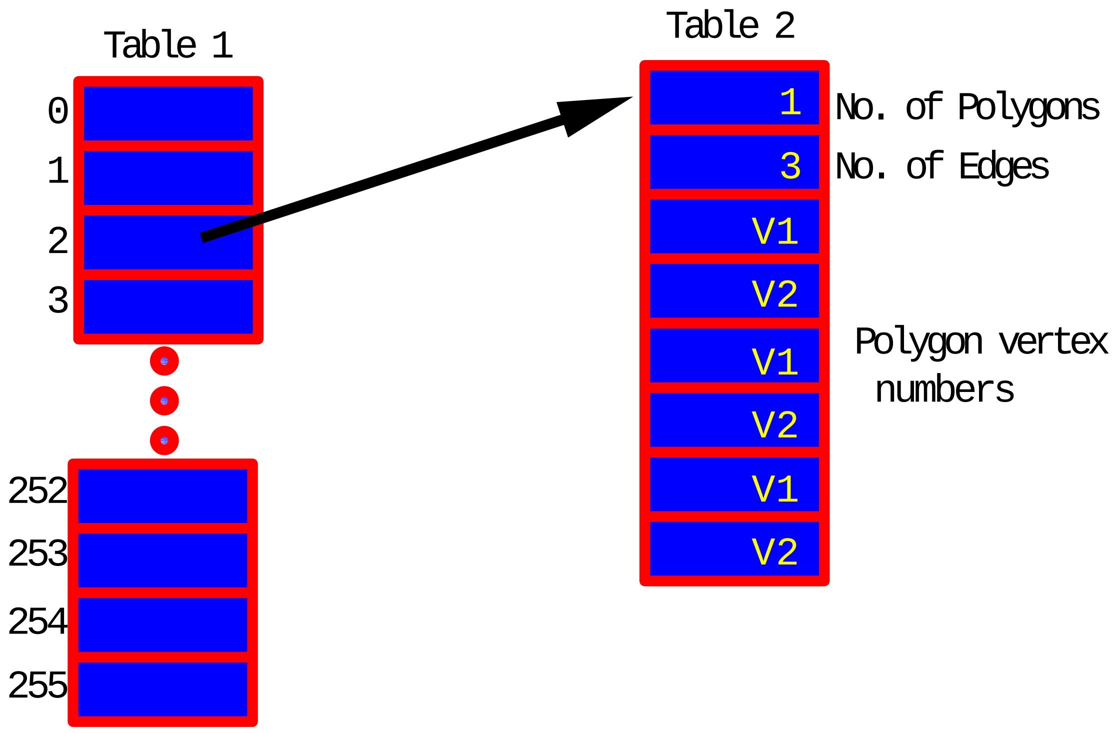


Figure 4: Continuation Method Tables

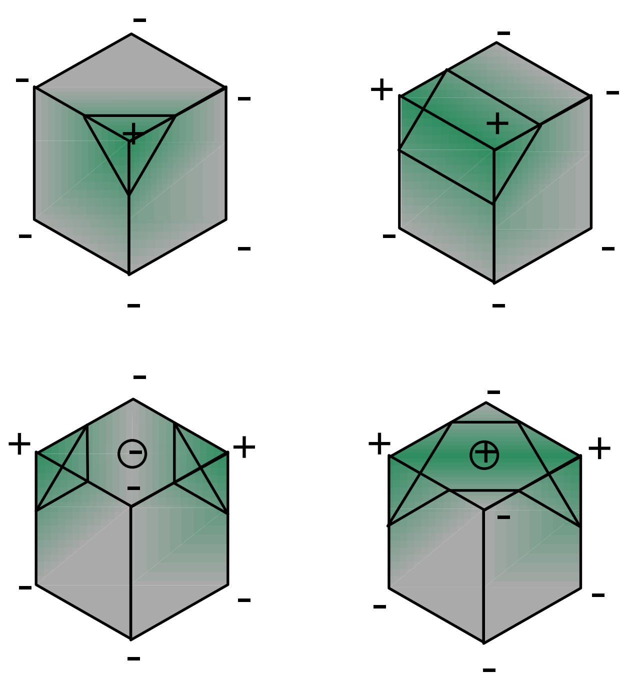


Figure 5: Polygonization Given Cube Vertices

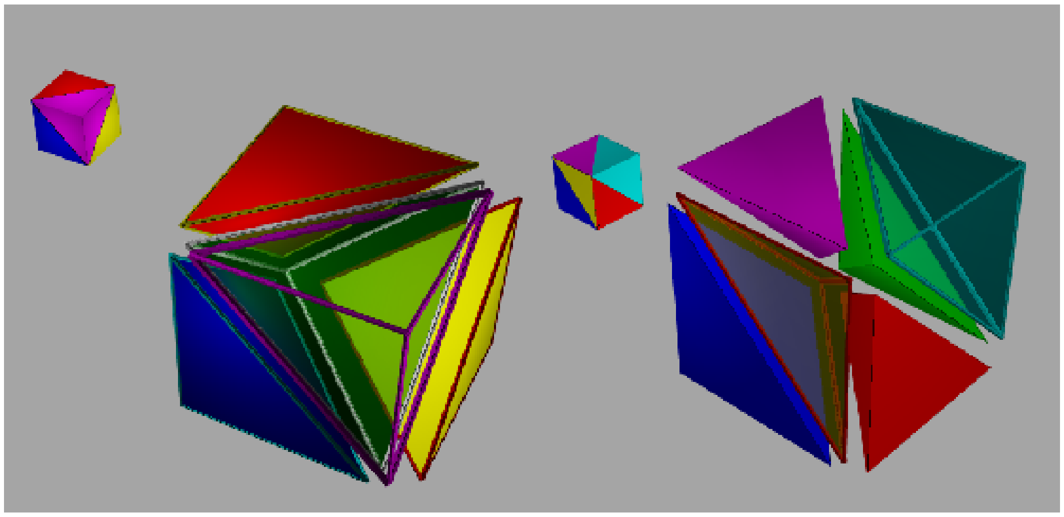


Figure 6: Decomposing a cube into 5 or 6 tetrahedra

3.2.1 Subdividing a Cube

Without requiring additional cell vertices a cube may be decomposed into five [16] or six [7] tetrahedra as shown in Figure 6. These decompositions introduce diagonals on the cube faces, thus determining the resulting face contours. The introduction of diaganol edges produces a higher resolution surface compared with replacing each cube directly with triangles. However, the decomposition into tetrahedra and the replacement of the tetrahedra with triangles are fast, table driven algorithms which produce topologically consistent meshes.

3.3 Cell Polygonization

Two obvious problems emerge from the use of uniform space subdivision. The size of triangles output by this algorithm do not adapt to the curvature of the surface and there are ambiguities in the second part of the algorithm where cubic cells are replaced by polygons. A space sudivision algorithm based on an octree was published in [1] which does adapt to the curvature of the surface. Cells are subdivided into eight octants and cracks are avoided by using a restricted octree scheme, i.e. neighbouring cells cannot differe by more than one level of subdivision. This indeed reduces the amount of polygons generated, but full advantage of large cells can only be taken if the flat regions of the surface happen to fall entirely within the appropriate octants. The algorithm proves in practice to be considerably slower than the uniform voxel algorithm [22], and is very complicated to implement. In [10] a method of decomposing cubic cells into five tetrahedra is described which avoids some of the ambiguity problems. It should be made clear that a spatial grid stores a sample of the implicit function at every vertex. If the function happens to vary considerably within a cell the polygonal representation will not show such variations.

4 Shrinkwrap

Most currently existing techniques for the polygonisation of iso-surfaces are based on data structures that allow spatial indexing: either a voxel-based structure ([1]) or the hash-table structure of ([22]) may be used.

Some inherent disadvantages of these data structures exist:

First, the data structure comprises a partitioning of the space rather a tesselation of the surfaces to be polygonalised. Especially in the case of animation (e.g. in the computer animation "The great train rubbery", [19]), this is likely to cause geometric artefacts that are fixed with respect to space, thus moving in an incoherent way over every moving surface.

Second, there is an apparent mismatch between the number of triangles that is generated by these algorithms and the complexity of the surface that is approximated: even relatively smooth and flat segments of an iso-surface usually result in large amounts of facets.

Third the ambiguious cases mentioned earlier in the case of cubic lattice methods.

In this work, a different approach to the tesselation of a class of iso-surfaces is taken. This class is defined in 4.2. The proposed approach is adaptive to the local behavior of the surface rather than being imposed by an octtree with a priori defined cutting planes; this causes the tesselation to move along with the surface in the case of (smooth) animations.

4.1 An algorithm to arrive at an adaptive triangulation for an iso-surface

First we define the class of iso-surfaces for which our algorithm should produce triangulations. Next the algorithm, together with an intuitive motivation are given, and some aspects are discussed in further detail .

4.2 The definition of the iso-surface

An iso-surface is the collection of points $r \in \mathbf{R}^3$ such

that a given function $V(r) = V_0$. We consider the class of functions V where $V(r) = \sum_i \frac{\rho_i}{|r-R_i|}$. The summation over i indicates that the function is composed of a number of components with relative strength (weight) ρ_i. Components can be thought to be generated by different types of geometric primitives: points, line segments or convex polygons.

If a component is generated by a point, then R_i is the location of this point, and the set $\frac{\rho_i}{|r-R_i|} = V_0$ designates a sphere with radius ρ_i/V_0 round R_i.

The element can also be a line segment, say ab. In that case R_i depends on r; R_i is the projection of r onto ab and the set $\frac{\rho_i}{|r-R_i|} = V_0$ designates a cylinder with hemi-spherical caps with radius ρ_i/V_0 and ab as the axis.

Similarly, elements can be triangles or other convex polygons in which case a projection of r onto that polygon has to take place in order to obtain R_i. In these cases the designated surface is an offset surface of the original polygon with cylindrically rounded edges. Another way to envisage the designated surface is as the Minkowsky sum of a sphere with radius ρ_i/V_0 and the geometric object (point, line segment or convex polygon).

The collection of points, line segments and convex polygons that define $V(r)$ is called the skeleton ; each geometric object in the skeleton is called a skeleton element.

The addition of the several components results in an iso-surface which is a smoothly blended union of the several iso-surfaces associated with the individual skeleton elements.

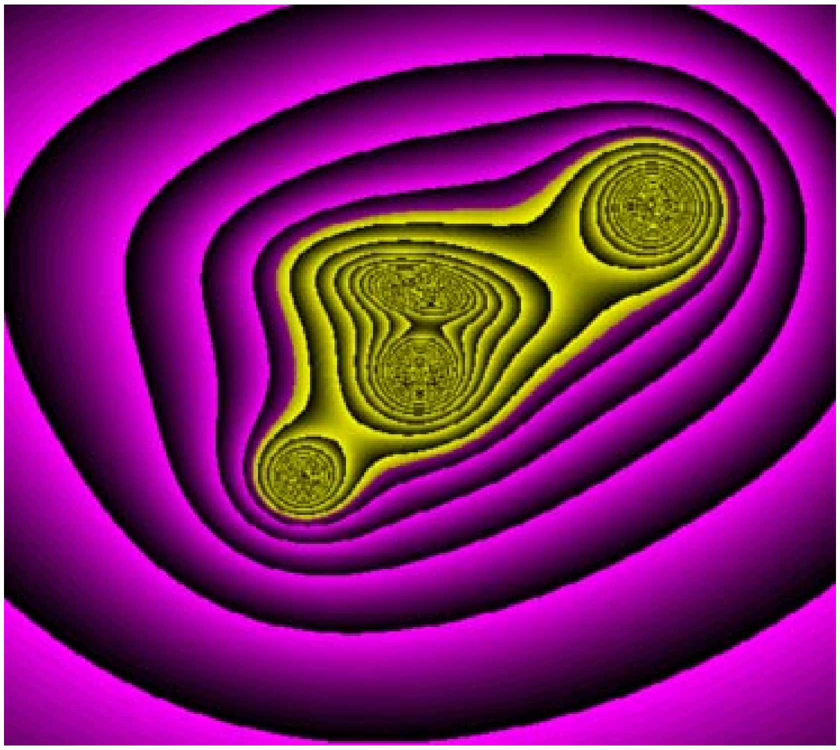


Figure 7: Implicit Contours

4.3 Intuitive introduction to the shrinkwrap algorithm

The basic idea that underlies the algorithm is that the iso-surface is to be sampled with sufficient density in order to capture all shape detail; further, that these samples are connected by edges to form a triangular mesh. In voxel-based methods, such edges result from intersecting the surface with voxel boundaries, and hence their directions all lay in one of three orthogonal planes (the XY, YZ or ZX planes of some world coordinate system). These directions have no apparent relations with the iso-surface, and therefore many unnecessarily short edges, and consequently, small triangles, result. In the algorithm proposed here, the edges should adjust themselves to the shape of the surface. This means that e.g. in a cylindrical part of the surface, there should be relatively long edges directed more or less along the axis of the cylinder and relatively short edges in directions perpendicular to the axis. This allows for an adaptive triangulation. In order to let edges gradually adjust themselves to the shape of the surface an iterative approach is adopted where the surface develops in several steps from a sphere to the final shape. The original shape (a sphere) has uniform curvature, and hence adaptivity plays no role there; the triangulation of a sphere in order to achieve an initial estimate for the mesh topology is simple. There are two natural ways to have an iso-surface develop itself from a sphere: One method operates by first collapsing the entire skeleton into a point and gradually expanding back ("inflating") to the desired skeleton geometry. The second method ("shrinking"), which allows a slightly simpler mathematical analysis, and which will be chosen therefore for our algorithm, is based on the following observation.

Figure 7 depicts a 2-dimensional cross section through a distribution of some point skeleton elements; colors indicate the function value $V(r)$ in a point. The iso-contour we are interested in is at the boundary between the yellow and the magenta regions. We observe that iso-surfaces

$$\{r \in \mathbf{R}^3 | \sum_i \frac{\rho_i}{|r - R_i|} = V_0\}$$

with $V_0 < 1$ have a shape which is less involved, whereas iso-surfaces with $V_0 > 1$ are more involved. An extreme case of the first example is $V_0 = 0$, which produces a sphere with an infinitely large radius.

So an algorithm could start by setting V_0 to a value close to 0, and providing a triangulation for the resulting sphere. This triangulation should consist of approximately equilateral triangles, since the curvature is the same anywhere.

Next the value of V_0 should be increased, in a number of steps, and with each step the surface shrinks a bit towards its final shape and size. With every shrink step, the vertices of the triangulation should move towards the new surface. This is discussed in section 4.4. Also, in order to meet with accuracy requirements it might be necessary to split edges and triangles. But we note that edges and triangles are only split when necessary, so there should be fewer unnecessarily small triangles at the end of the porcess than with voxel-based methods.

Of course, this gradually shrinking of the triangulation will only work as long as the topological structure of the surface stays equivalent (*homeomorphic*) to the sphere that we start with. Details of the proposed technique to deal with topological changes (ruptures and holes) is currently being prepared.

4.4 getting the vertices onto the surface

Using the stepwise approach, and assuming the difference in iso-values V_0 from one step to the next sufficiently small[1], a Newton-Raphson method is used to displace vertices. So we make use of a first order Taylor expansion to compute a first estimate for the new location of a vertex when increasing V_0 to $V_0 + \Delta V$, and, when necessary, iterate. Assume r is on the $V = V_0$ surface: $V(r) = V_0$. Next we look for a new location, $r + \delta$, such that

$$V(r + \delta) = V_0 + \Delta V.$$

Taylor expansion round $\delta = 0$ gives:

$$V(r + \delta) = V(r) + (\delta \cdot \nabla V(r)) + O(|\delta|^2) = V_0 + \Delta V,$$

or

$$\Delta V \approx (\delta \cdot \nabla V(r)).$$

Of course, this does not tell us in which direction the step δ should be taken. A reasonable choice seems to be to set

$$\delta = \lambda \nabla V(r)$$

which gives

$$\lambda = \frac{\Delta V}{(\nabla V(r) \cdot \nabla V(r))}.$$

So the new location is

$$r + \frac{\Delta V \nabla V(r)}{(\nabla V(r) \cdot \nabla V(r))} \tag{1}$$

[1]and assuming that no topological changes occur between this step and the next

4.5 The shrinkwrap algorithm

We are now able to write down the total shrinkwrap algorithm. Vertices are defined as tuples $(r, E, V, d) \in \mathbf{R}^3 \times \mathbf{R}^3 \times \mathbf{R} \times \mathbf{R}^3$; the r-attribute is the location; E is the gradient $\nabla V(r)$; V is the value of the function $V(r)$, and d is the displacement vector that should apply to this vertex in order to get it to the surface with next higher iso-value.

Edges contain two references e1 and e2 to the vertices in the extremes, and two references to the two adjacent triangles. Furthermore, an edge has a boolean n to indicate if it is non-acceptable (the acceptability of an edge is discussed in [18]) for now it suffices to observe that edges should be in some way close to the underlying surface in order to make quantitative statements about the accuracy of the triangulation; this can be obtained by splitting edges that are non-acceptable).

The difference ΔV is an entire fraction of V_0, say $\Delta V = V_0/N_{steps}$, and for V_0 the value $V_0 = 1$ will be used. When using this convention, the interpretation of

the value ρ_i is simply "the radius of the offset surface if component i was the only one skeleton element".

The issue of the number of steps N_{steps} in relation to the robustness of the algorithm is discussed in a report which may be obtained from the authors.

Initially, the set of vertices consists of the vertices in the initial object (a triangulated sphere with more or less equilateral triangles); this object is assumed to be sufficiently large to be outside the entire iso-surface even for iso-value $V_0 = 1/N_{steps}$. All vertices v have $v.V = 0$; $v.E$ is pointing radially outwards, and $v.d$ is proportional to $v.E/(v.E \cdot v.E)$ (as derived in 4.4).

4.6 How to split edges

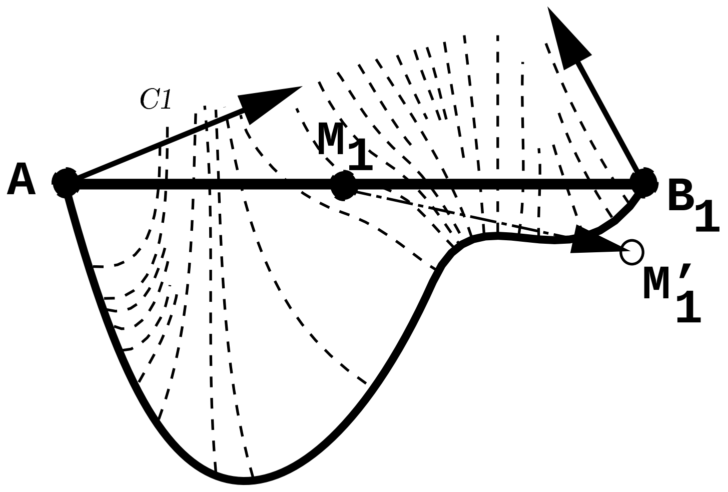
We have not defined yet what an acceptable edge is. For now it suffices to state that an acceptable edge should be short enough so that the surface cannot bend away too much between the two extremes of the edge. Conversely, an unacceptable edge is an edge which is too long. So we see that the remedy to an unacceptable edge is to split it, and to make sure that the new midpoint is again on the iso-surface. (for details see [18]).

A naive way to do so is depicted in the left column of figure 8 (a-d). The original edge is $A - B_1$ in figure 8-a; $M_1 = (A + B_1)/2$.

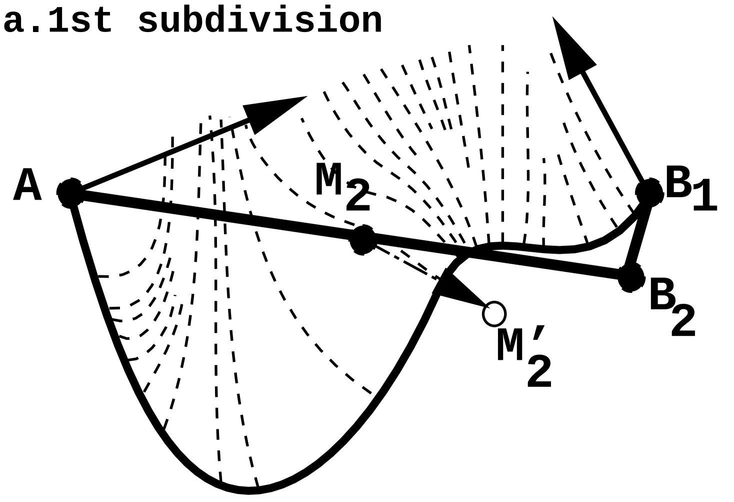
The array of dotted curves indicate the direction of the gradient of the function $V(r)$ in the neighbourhood of the iso-surface; the thick curve represents the iso-surface proper. If we move the point M_1 in accordance with the local gradient, we arrive in M_1', as indicated by the dash-dotted

arrow. Note that although M_1' will be close to the surface, since we only use a linear approximation for $V = V(r)$, it will not lie on the surface in general. In order to get it closer to the surface, we have to iterate. Now M_1' will be a new vertex. The edge $M_1' - B_1$ is very likely acceptable, but it may be much shorter than needed. On the other hand, $A - M_1'$ is probably still unacceptable. As shown in figure 8-b, we therefore have to repeat the process on edge $A - B_2$ (B_2 is the M_1' from the previous phase), which yields M_2'. As a result, we end up with a series of unnecessary short edges, as depicted in figure 8-d. The main cause for this unfortunate behaviour is that we use information about the geometry of the function $V(r)$ and its gradients in the points M_1, M_2, M_3, ..., which are possibly far from the surface. Evaluating the gradients in these points may yield misleading information on the geometry of the iso-surface, causing a slow convergence and many unnecessary short edges.

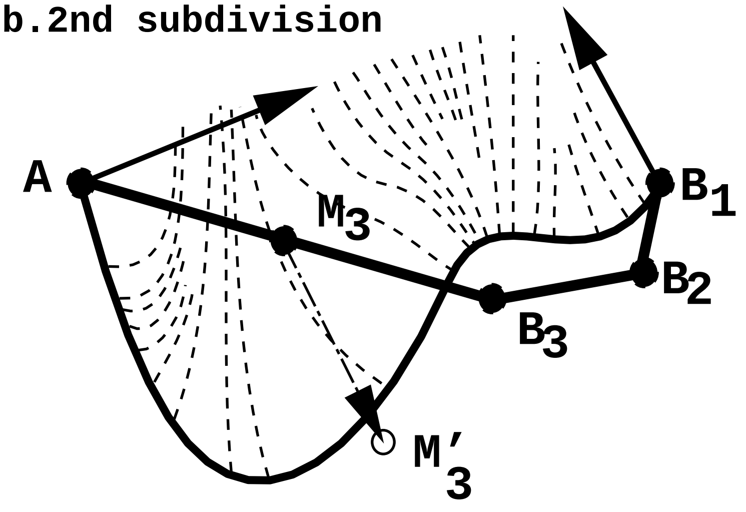
A more efficient splitting strategy therefore should make use of reliable information only, that is information in points that are already on the surface. In figure 8-e, the same configuration is shown as in figure 8-a. The dashed thick curve is a curve which passes through the extremes A and B



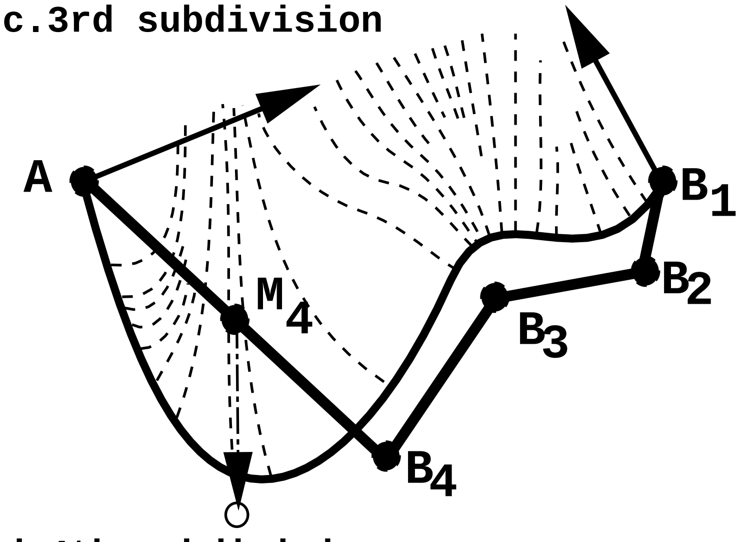
a.1st subdivision



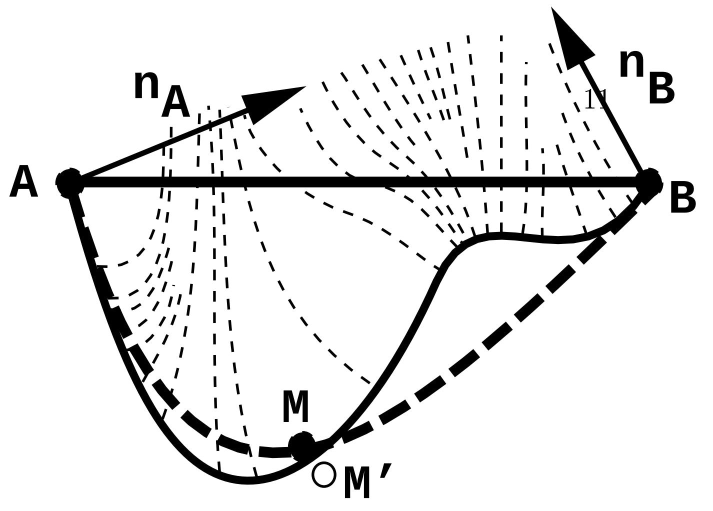
b.2nd subdivision



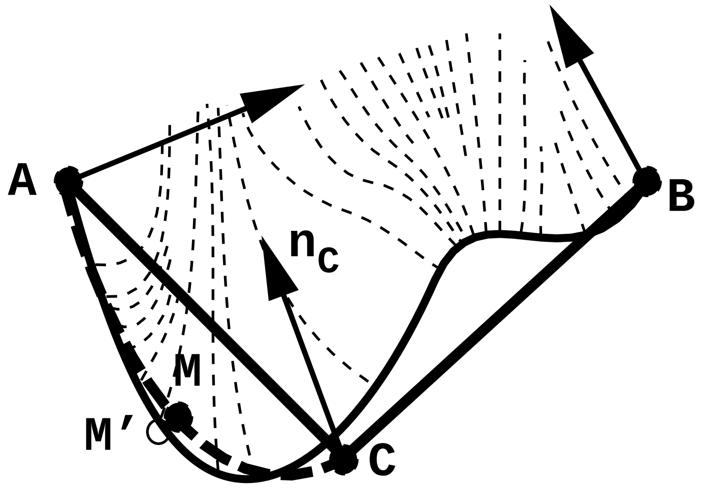
c.3rd subdivision



d.4th subdivision



e.subdivision using curve AMB



f. subdivision using curve AMC

and is perpendicular to the normal vectors n_A and n_B, respectively. Moreover, it is the smoothest curve with these properties. This curve serves to approximate a curve that lies in the iso-surface $V(r) = V_0$ through A and B, i.e. the thick solid curve in the figure. Based on the dashed thick curve, we propose point M, i.e. its parametric midpoint as a next point to evaluate the function's gradient. Point M in figure 8-e is likely to be closer to the iso-surface than M_1 in figure 8-a, so the gradient computed in M is likely to be more adequate to get acceptable edges than the gradient in M_1. In this case, $M - B$ already might be acceptable (the edge $C - B$ in figure 8-f); $A - M$ (the edge $A - C$ in figure 8-f) might need one more subdivision as depicted in figure 8-f.

4.7 How to split triangles

In case one or more edges are unacceptable, they have to be split. Figure 9 shows a splitting scheme which illustrates how a triangle can be subdivided into smaller triangles. In the top row one of the edges is subdivided; the bottom row shows the case of three subdivided edges. In the case of two subdivided edges, two possible schemes exist; in case $|M_{AC} - B| < |M_{BC} - A|$ we choose the first alternative; otherwise we choose the second one.

4.8 Remarks

The described algorithm has been implemented. In Figure 10, its qualitative behaviour is depicted: while increasing V_0 in 9 steps from 0.1 to 1.0, we see a penguin emerge from what starts of as a feature-less large spherical shape. Note how gradually the details become visible, first the most protruded ones (for this reason we equipped the penguin with an oversized bill), later on the more subtle ones. Figure 11 shows the final object together with the triangle mesh. Here, the adaptiveness of the algorithm is clearly visible, e.g. the bill consists of mostly very slender triangles, whereas in the spherical part of the head we find more obtuse ones. Also, in the concave regions, having a relatively high curvature, the triangle mesh has a much higher resolution than elsewhere.

The algorithm is adaptive and performs well compared with the uniform voxel algorithm. A disadvantage of this algorithm, is that it can only cope with a single closed surface without holes. However, an extension of this approach has been devised to solve this problem and will be available shortly. Further details concerning the algorithm and the robustness of this approach are available from the authors.

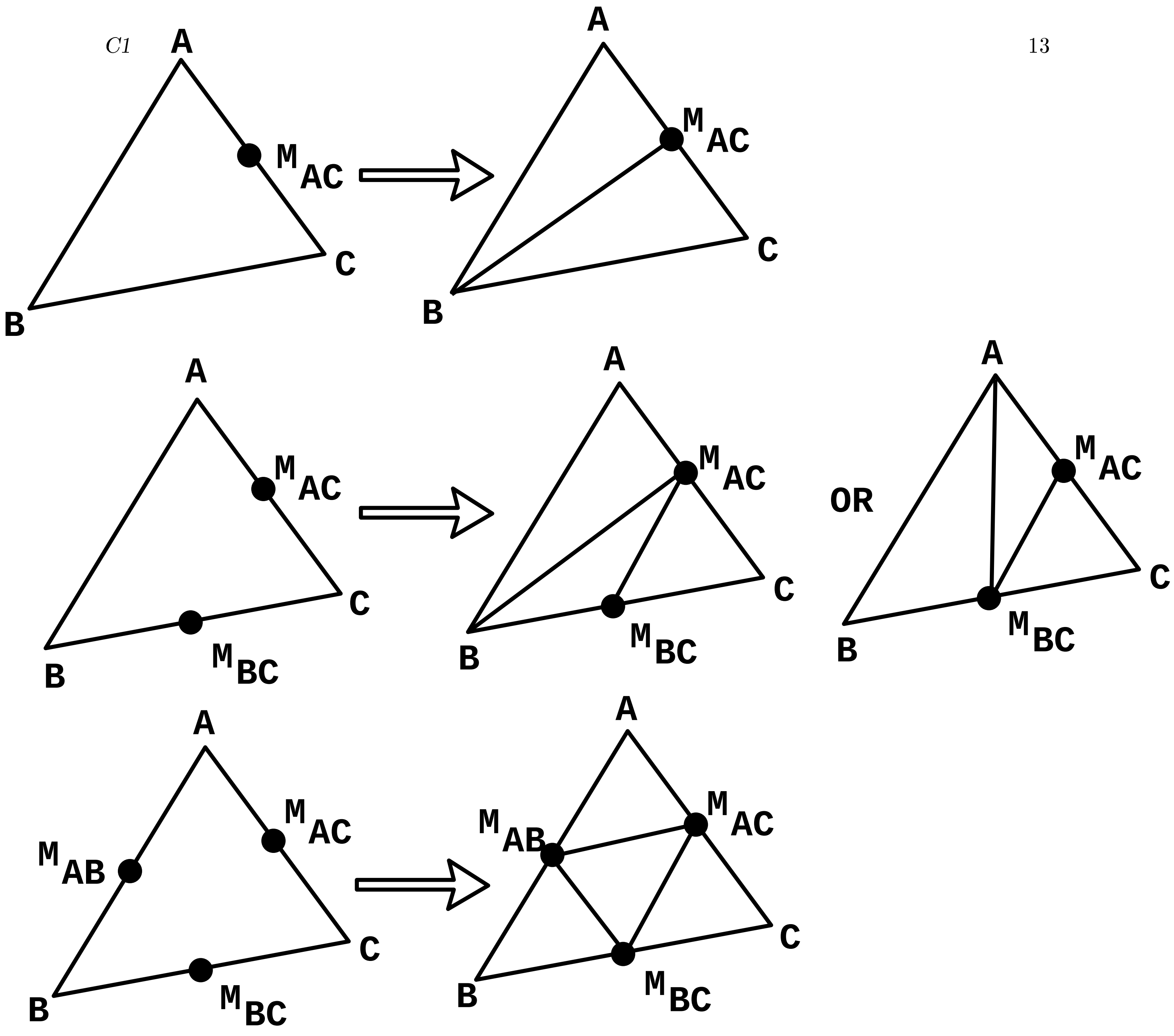


Figure 9: The triangle subdivision scheme

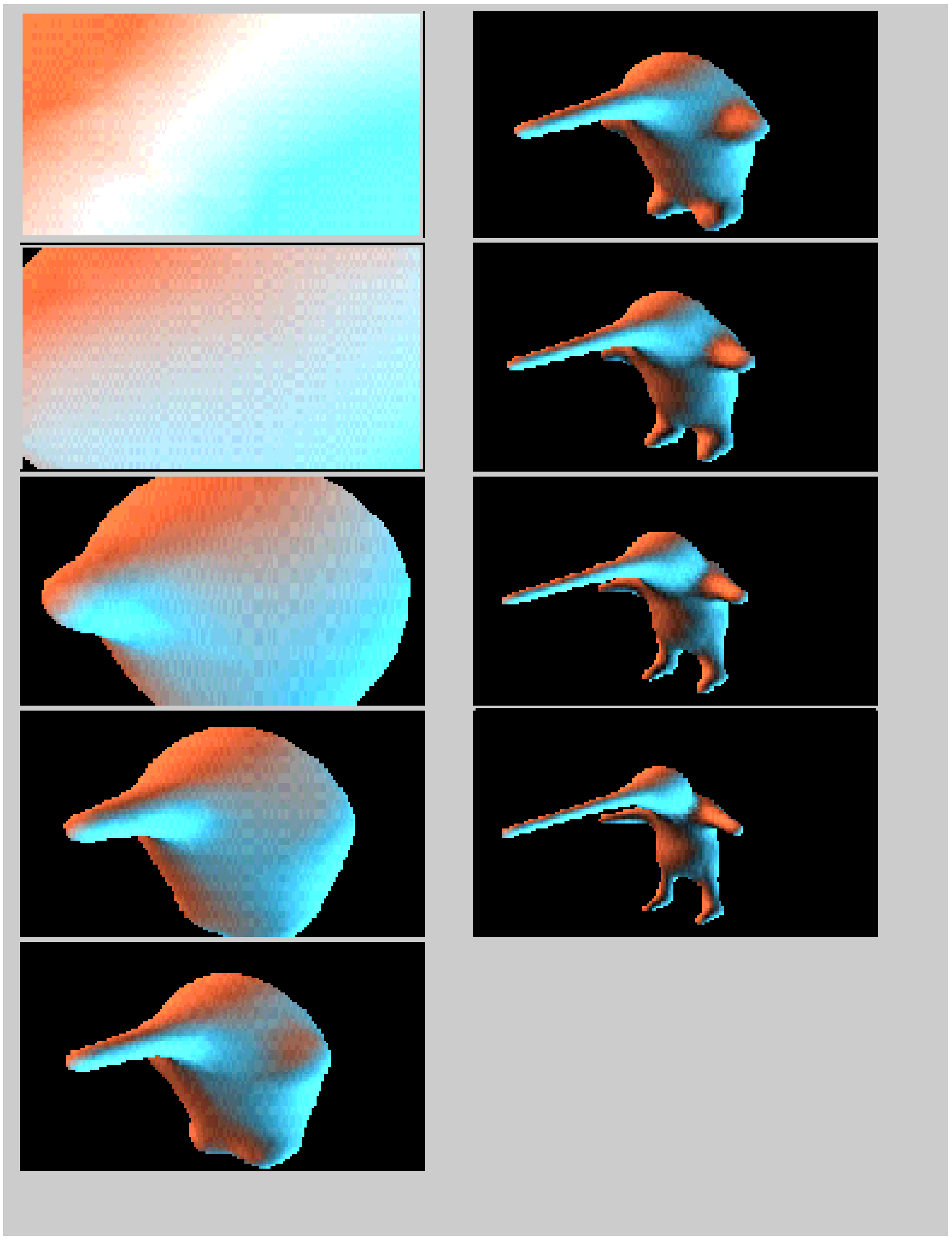


Figure 10: Shrinkwrapping a penguin

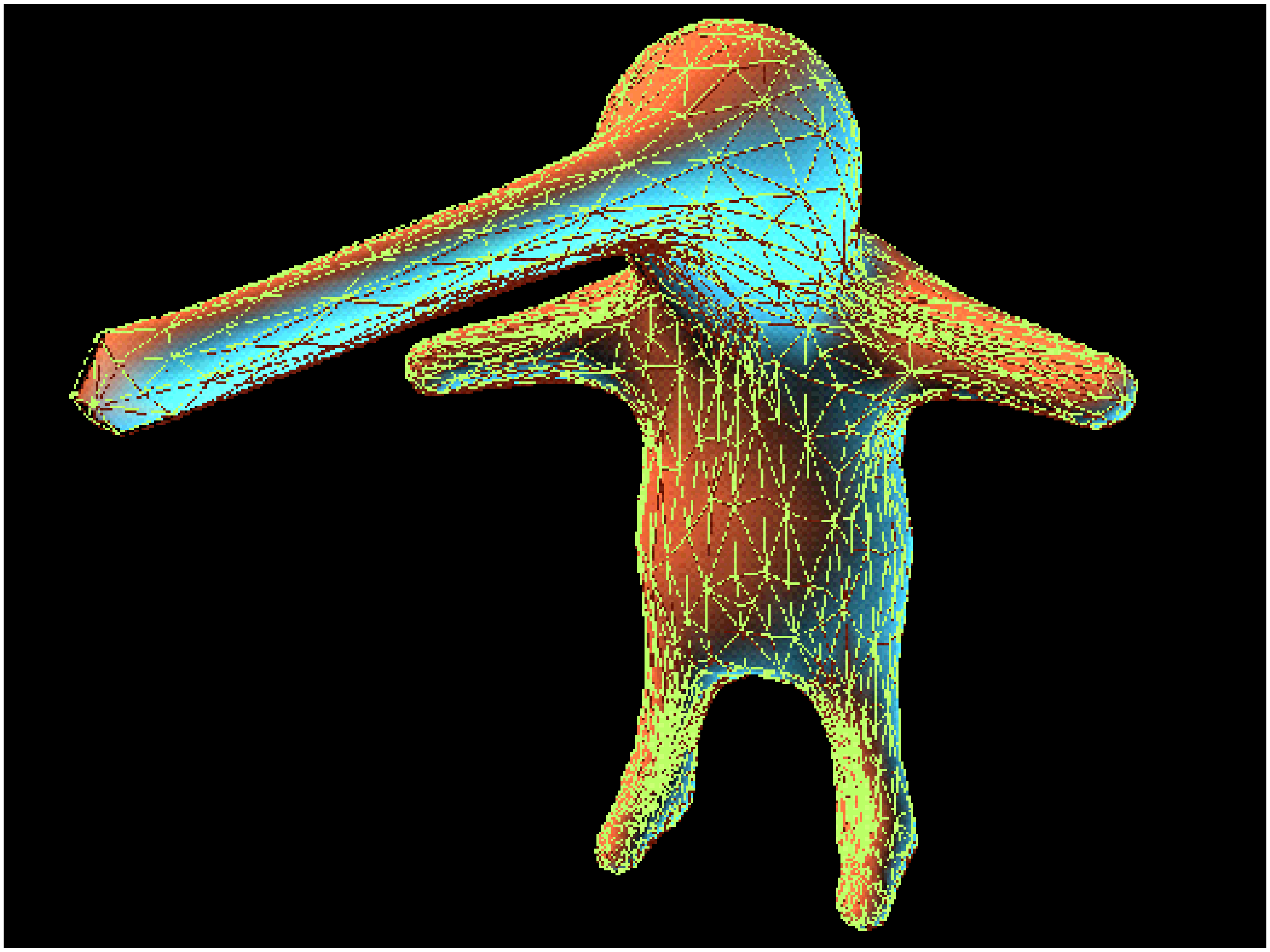


Figure 11: Penguin showing adaptive mesh

5 Tiling CSG

Constructive Solid Geometry (CSG; [17], [12]) is a techniques that has been used to construct a wide variety of non-trivial geometric objects. Due to the difference in nature from implicit surface modelling, the application fields of the two methods appears to be quite different. An an algorithm that combines these approaches. An earlier system that also combines CSG and implicit surface objects was developed by Geoff Wyvill [21], using ray tracing to both traverse the CSG tree and render the objects. In this work our approach is quite different in that we do not use ray tracing, instead we have developed a polygonizing algorithm to facilitate fast rendering and improve the design cycle of such objects. The primitives in standard CSG, are a limited set of closed geometric objects such as the sphere, cone, torus etc. Extended versions can also support primitives bounded by free form surfaces, sweep surfaces, or other deformed primitives ([5], [8]).

Implicit surface systems also employ geometric primitives, known as skeletal elements, ([3]). Skeletal elements are point sets that allow easy computation of a distance to a given point in 3-space.

Junctions in the boundary between surface fragments of different CSG primitives are generally not C^1. It requires special primitives to obtain smooth blends ([13]). Alternatively, filleting and rounding operations may apply to the boundary representation of the CSG-object ([4]).

The implicit functions used in ISM that give rise to the resultant iso-surface are in general differentiable everywhere in 3-space, so the surface is smooth everywhere. Since there is no notion of explicitly represented junctions in ISM, it is not possible to get non-C^1 boundaries anywhere. (See however [6]).

CSG and implicit surface systems are similar in that the underlying model description has to be visualized. There are two basic approaches for each type of representation:

- For CSG systems, find a boundary representation (b-rep) and render the model as boundary fragments (mostly converted to polygon meshes), or alternatively, the object may be ray traced while the CSG-expression evaluation takes place for each ray while being traced ([8]).

- An implicit iso-surface, once polygonized, is just a polygon mesh that can be rendered as it stands. Alternatively, it may be rendered directly via ray tracing. Although this is often too computationally expensive for many applications, it has been demonstrated that CSG-type boolean combinations of several iso-surfaces can be obtained by evaluating the boolean expressions along with the ray intersections in a manner similar to standard CSG [21].

Geometric primitives are complete ISM's, so a virtually unlimited variety of primitives is available. Next these primitives are assembled via the usual CSG-type boolean constructors. The skeletal elements within one ISM primitive blend smoothly, so there are no visible non-C^1 junctions within those elements. On the other hand, two ISM primitives are combined in the CSG-sense, and hence a visible junction arises there. So both types of junctions are supported within one surface representation scheme. When ray tracing is employed for ISM's, CSG-type operations can be berformed on-the-fly where the operands are individual ISM's [21], but ray tracing is computationally expensive. On the other hand, when all ISM's are polygonized first, then CSG-type operations can be performed afterwards on the resulting polygon meshes ([14]), since they are closed manifolds, but this has a high complexity in terms of the number of triangles in the meshes involved: the fully triangulated meshes of all input ISM's have to be available, even if a given ISM only contributes for a small fraction of its surface. Also, this strategy cannot be used if one the participating CSG-primitives (ISM's) is unbounded, as for instance when intersecting with a planar half space in order to 'cut an object in half'. In the algorithm described we perform the CSG-operations on-the-fly

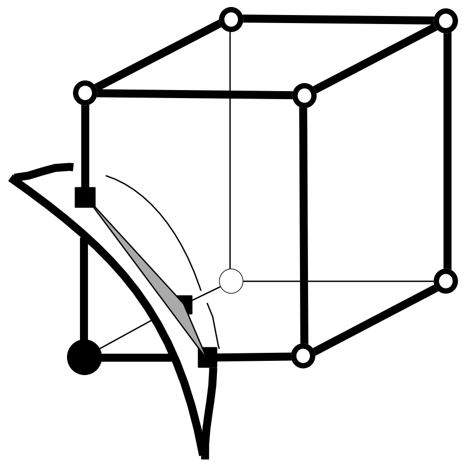


Figure 12: A cubic voxel intersected by an iso-surface.

while polygonizing the *resulting* surface. This means that the complexity is linear in the number of triangles of the resulting surface only, even if some of the contributing ISM's would have given rise to much larger triangular meshes.

An example of combining an implicit surface model (ISM) and a CSG model is shown in fig. 12 spheres smoothly blend together difference operations are used to remove three blended cylinders, a fourth cylinder is joined by union and the result is intersected with a plane. The ISM cylinder primitive has hemispherical ends.

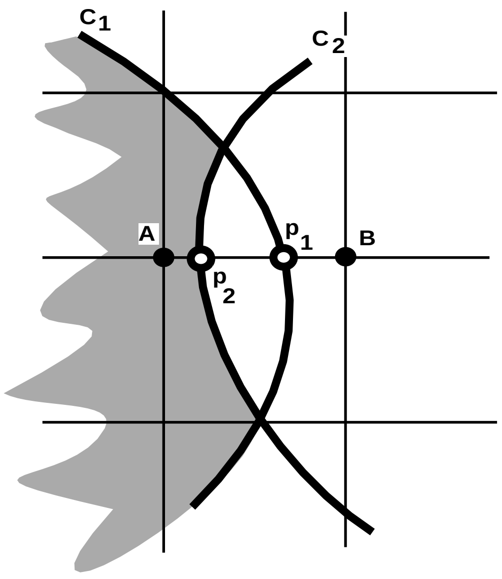
5.1 Voxel-based CSG-operations

Given a scalar field function $f = f(x,y,z)$, the Uniform voxel Subdivision Algorithm of [22] estimates intersections of the iso-surface $\{(x,y,z)|f(x,y,z) = 0\}$, to be polygonized, with the 12 edges of a cubic voxel, on the basis of the $f(x_c,y_c,z_c)$ values, $c = 0,\cdots,7$, in the 8 corner vertices (x_c,y_c,z_c) of that voxel. See figure 12

Here, the front lower left corner vertex (solid circle) has $f > 0$ whereas the other corner vertices (open circles) have $f < 0$. A vertex with $f > 0$ classifies 'in' with respect to the iso-surface and a vertex with $f < 0$ classifies 'out'. In the case where an intersection of that edge with the iso-surface exists, the extreme vertices of a voxel edge are classified differently. Cases where an edge contains one intersection are indistinguishable from cases where there are any odd number of intersections. Similarly, the occurrence of an even number of intersections goes unnoticed.

In order to generalize towards CSG-expressions in iso-surfaces, we assume that instead of a scalar function $f(x,y,z)$, we have an $n-$ component vector function $f_j(x,y,z), j = 0,\cdots,n-1$. Each of the components f_j gives rise to its own iso-surface, each iso-surface can be seen as the boundary of one ISM primitive. The resulting BCSO surface has to be constructed such as to bound the appropriate CSG-expression in each of the ISM primitives. In the sequel, the CSG-operations are denoted as $DIFF$, $UNION$, and $INTSCT$, for difference, union, and intersection, respectively. The arguments of these operators will be either numbers of ISM primitives (the above j) or other CSG-operations.

In order to see how this works out, we study a 2-D version first (see figure 13). Here C_1 and C_2 are two iso-value contours that both intersect voxel edge A-B. They give rise to (estimated) intersections p_1 and p_2, respectively. Suppose C_1 is the boundary of ISM primitive 1 whereas C_2



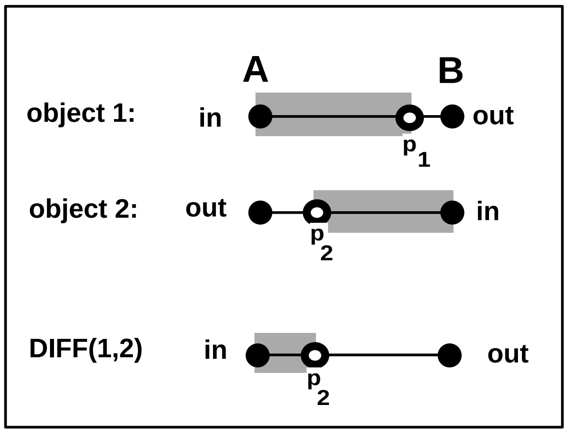


Figure 13: Using vector functions to represent several ISM objects at once.

bounds ISM primitive 2, and we want to polygonize the boundary of the object $DIFF(1,2)$. It can be seen from fig. 13 that p_2 is the relevant intersection of the two.

In general, we observe that depending on the in- or out-classifications of each of the components f_j in A and B we can determine, for each of the operators $DIFF$, $UNION$, $INTSCT$, which of the intersections is the relevant one. Note that there does not always have to be a relevant intersection: if the resulting BCSO iso-surface does not pass through the edge AB, the 'relevant intersection' is not defined. This information can be stored in a sixteen entry table providing an exhaustive list of all possible in- and out-cases for two participating objects i and j (see [20])

Based on the table and on a straightforward binary tree-representation of the boolean expression, an algorithm to compute the intersection point and in/out classification of the resulting surface, given the intersection points and in/out-classifications of the n ISM primitives, is readily obtained.

In ANSI-C, it reads as follows:

```
typedef struct B_EXP {
  char kind;  /* the kind of this operator node u,i,d or n for union,
               * intersection, difference or primitive number */
  int n;      /* if kind=='n', the value of the primitive number */
  struct B_EXP *se1, *se2;
              /* if kind!='n', the first and second sub-expressions */
} B_EXP;

typedef struct {
  int A_in,B_in;/* two booleans that indicate if the two
                 * extremes are inside */
  float p;      /* a number between 0 and 1, defined if (A_in!=B_in)
                 * indicating the relative position of
                 * the surface intersection */
}SEGMENT;
```

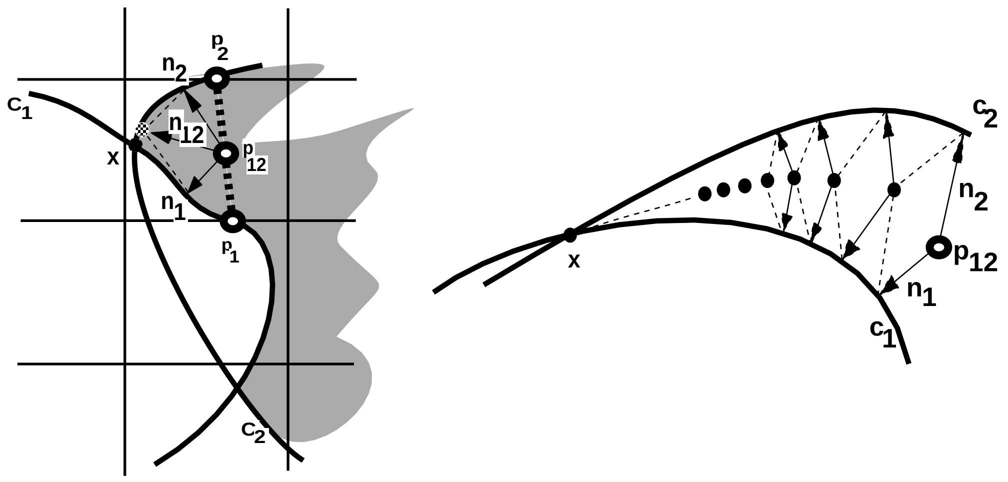


Figure 14: Approximating the intersection of two implicit contours

```
SEGMENT combine(SEGMENT s[],B_EXP *e) {
SEGMENT rs,ss1,ss2; /* if the expression is an operator,
                     * ss1(2) are its operands */
  if(e->kind=='n') {
    rs=s[e->n];
    return rs;
  }
  ss1=combine(s,e->se1);
  ss2=combine(s,e->se2);
  switch(e->kind) {
    case 'u':rs=form_union(&ss1,&ss2);break;
    case 'i':rs=form_intersection(&ss1,&ss2);break;
    case 'd':rs=form_difference(&ss1,&ss2);break;
  }
  return rs;
}
```

The functions *form_union()*, *form_intersection()*, and *form_difference()* implement the instructions in the table. Before a call to combine is made, the caller has to set up the array $s[]$ of segments, one segment for each of the ISM primitives. This means that for each of the primitives the intersection point with the current voxel edge has to be computed, as well as the in/out classification for that primitive in both extremes of the voxel edge. The segment that is returned by *combine()* contains the intersection of the resulting boundary surface of the BCSO-object with the current voxel edge (if it exists), as well as the in/out classification in both extremes of this edge.

So an existing implementation of the uniform voxel space-subdivision algorithm can be easily extended by replacing the computation of the intersection by a loop that computes the intersections for all ISM surfaces, and next perform a call to *combine()* to have the intersection with the resulting surface computed.

5.2 Arriving at non-smooth edges

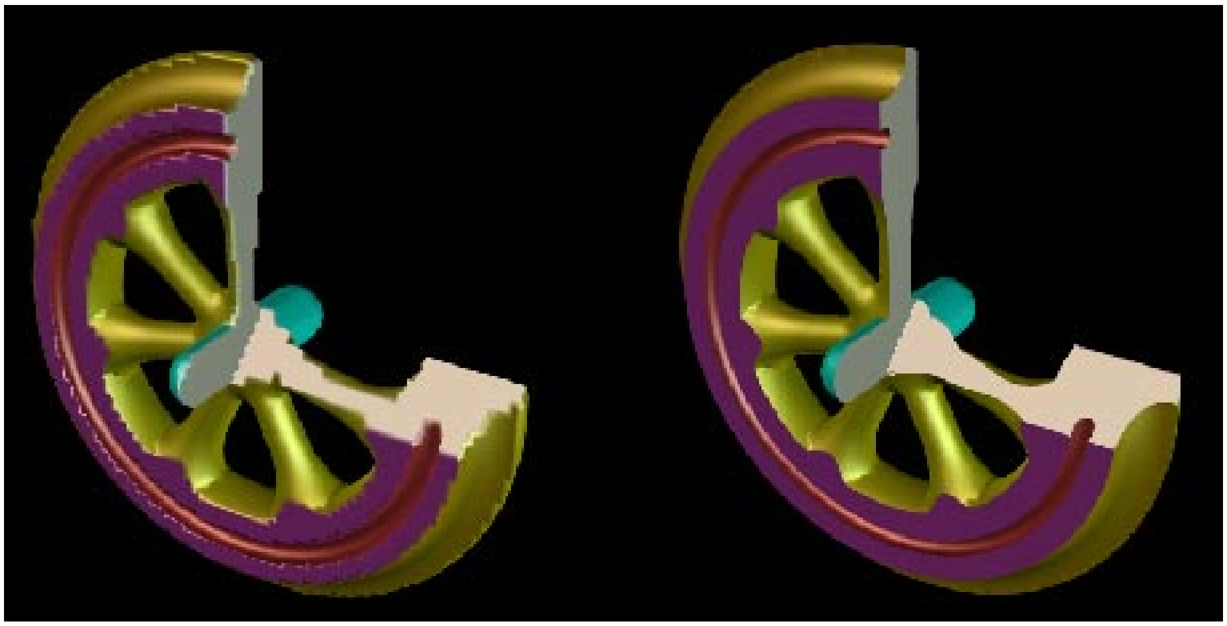


Figure 15: Wheel before and after postprocessing.

Unlike implicit blends, CSG operations should result in sharp contours between primitives. Again we first study the problem in 2-D. Consider fig. 14. Here we have again the configuration that two iso-value contours, C_1 and C_2 intersect. The intersection point is $\mathbf{x} = (x, y, z)$, but this is of course a priori unknown and we should try to find an approximation to it. Suppose that the CSG-expression is $DIFF(2, 1)$ where the interior region associated with curve C_1 is on the left of C_1 and the interior region associated with C_2 is on the right of C_2. Then the *combine()*-function from Section 2 results in the two relevant intersections between the resulting contour and the voxel edges $\mathbf{p}_1$ and $\mathbf{p}_2$, respectively. Following the uniform algorithm for inferring the contour from intersection points, we would obtain the dashed line $\mathbf{p}_1\mathbf{p}_2$ as a segment of the contour. Since this is quite far from the actual intersection point, the non-C^1 junction is not very well reproduced. Instead we observe that we should find an estimate for $\mathbf{x}$ such that $f_1(\mathbf{x}) = f_2(\mathbf{x}) = 0$ where f_1 and f_2 are the scalar field functions for the two ISM primitives with contours C_1 and C_2, respectively. Assuming we have a starting point which is not too far off, say $\mathbf{p}_{12} = \frac{\mathbf{p}_1+\mathbf{p}_2}{2}$, we can apply first order Taylor expansion to the difference $\mathbf{n}_{12} = \mathbf{x} - \mathbf{p}_{12}$. (see [20]for details) Figure 15 shows a wheel model before and after the application of the edge enhancement described above. This image clearly shows the improvement to the edges of the model with the additional polygons.

The image shows a wheel built from 14 BCSO primitives. These are combined to form seven ISM's. For example, the seven spokes are blended together with the outer torus to form a single ISM. Intersecting half planes are used to flatten the front and back and a difference operation was used to subtract a torus from the flattened surface to form the decorative groove. The colours have been chosen to illustrate the separate ISM's.

The coffee grinder was constructed from 15 ISM's, composed of some 25 primitives. The table and mirror frame were also modelled using BCSO's. Our polygonizer took about 1 minute on an SGI Indy to convert the primitive description and boolean expression to about 1.5 million triangles. There are 37 BCSO primitives in the coffee grinder (14 blending groups), 17 primitives in the table (10 blending groups) and 19 primitives in the mirror (16 blending groups). The polygonal scene was ray traced.

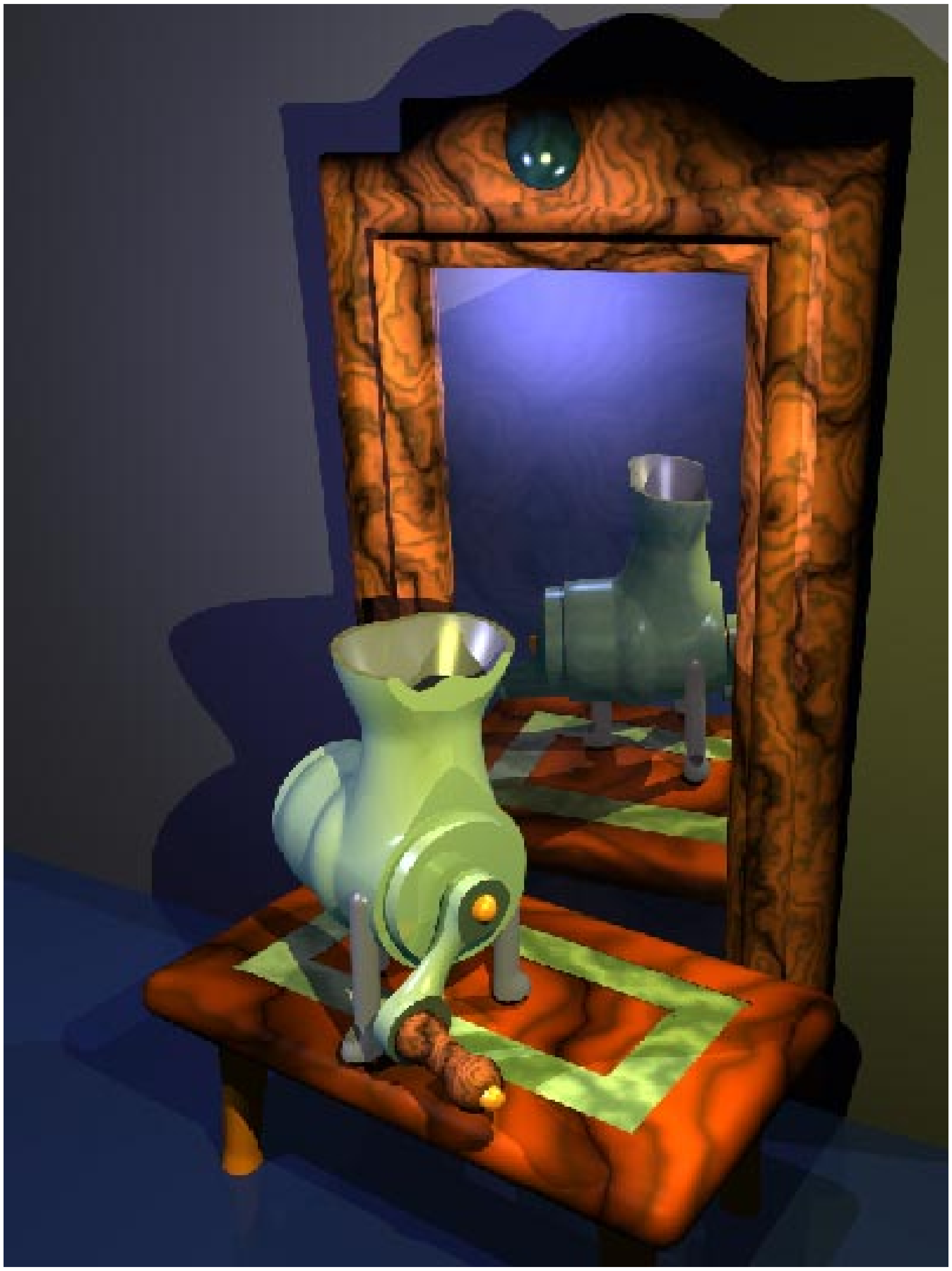


Figure 16: The Canmore coffee grinder and friends.

5.3 Building the Piano

An example of a more sophisticated BCSO object is shown in Figure 17 The piano body is made from two cylinders, with a half space parametric curve subtracted from it. A piecewise planar parametric cubic curve is defined along with an *up* direction and a normal in the plane of the curve which points towards the positive side of the primitive. The curve is extruded in the *up* direction, for example to form the curved side of the piano case. To extend the extruded piecewise curve so that it forms a half space, a planar extension is added at each end. Each plane has the same *up* as the curve and is defined by the last or first two control points along the curve. The piano was designed using the polygoniser to prototype the model, it is a scale model of a 9ft. Steinway Concert Grand. Figure 18 shows a diagramatic representation of a 2D slice through the primitives that make up the piano body including the parametric curve. Two large cylinders form the basis for the piano, the curve is subtracted to form the body. A smaller version of the body was also built and subtracted from the outer body to give the side walls. The piano demonstrates blended primitives as in the piano legs and pedals, and non-blended as in the keyboard. The plant in figure 17 is not a BCSO model but is defined by an L-system designed by Dr. Przemek Prusinkiwicz and Mark Hammel at the University of Calgary.

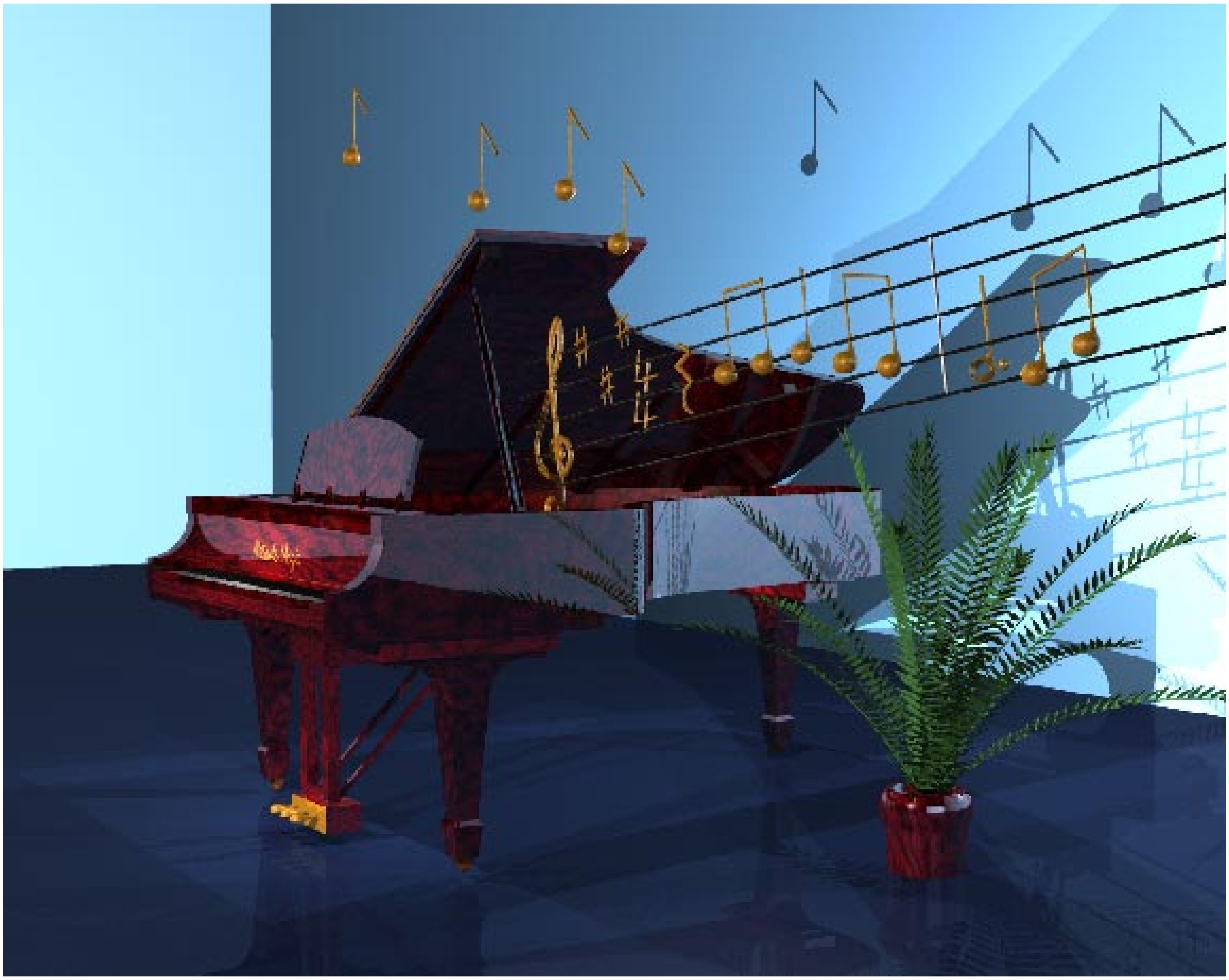


Figure 17: Model of *Camille* a Steinway Concert Grand

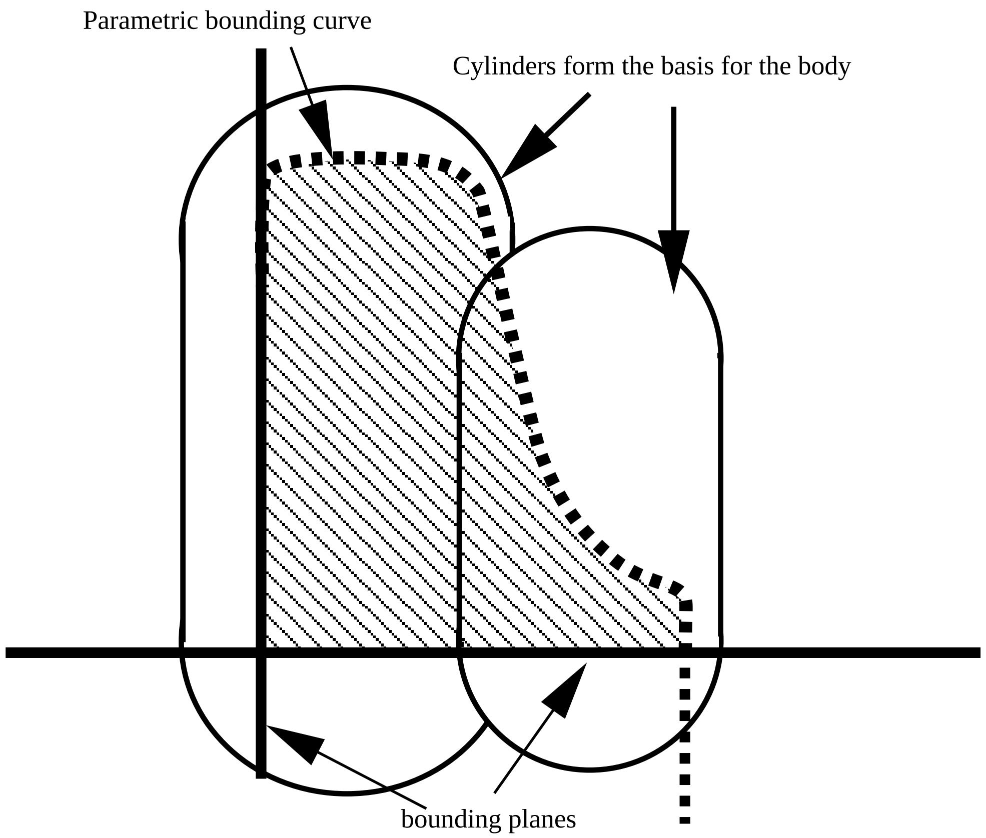

Parametric bounding curve
Cylinders form the basis for the body
bounding planes

Figure 18: Building the piano

6 Acknowledgements

We would also like to thank Jules Bloomenthal for his encouragement and ideas over the years and for some of the material which is part of a collaborative effort which will result in a book. We are particularly in debt to Geoff Wyvill, who started the whole thing off by solving a problem in scientific visualization, the solution to which turned out to be so useful for building models. Thanks also to the Banff Centre for Fine Arts for letting Brian take accurate measurements of *Camille*, a Hamburg Steinway concert grand piano. We would also like to thank the many students who have contributed so greatly to this research over the years.

This work is partially supported by the Natural Sciences and Engineering Research Council of Canada in the form of a research grant and equipment grants.

References

[1] Jules Bloomenthal. Polygonisation of Implicit Surfaces. *Computer Aided Geometric Design*, 4(5):341–355, 1988.

[2] Jules Bloomenthal. An Implicit Surface Polygonizer. *Graphics Gems IV*, pages 324–349, 1994. Edited by paul Heckbert.

[3] Jules Bloomenthal and Brian Wyvill. Interactive Techniques for Implicit Modeling. *Computer Graphics*, 24(2):109–116, 1990.

[4] H. Chiokura and F. Kimura. Design of Solids with Free Form Surfaces. *Computer Graphics (Proc. SIGGRAPH 83)*, 17(3):289–296, July 1983.

[5] G.A. Crocker and W.F. Rainke. Boundary evaluation of non-convex primitives to produce parametric trimmed surfaces. *Computer Graphics (Proc. SIGGRAPH 87)*, 21(4):129–136, July 1987.

[6] Marie-Paule Gascuel. An Implicit Formulation for Precise Contact Modeling Between Flexible Solids. *Computer Graphics (Proc. SIGGRAPH 93)*, pages 313–320, August 1993.

[7] G.M.Nielson, T.A.Foley, B.Hammann, and D.Lane. Visualizing and Modeling Scattered Multivariate Data. *IEEE Computer Graphcis and Applications*, 11(3):47–55, May 1991.

[8] F.W Jansen. *Solid Modeling with Faceted Primitives.* PhD thesis, Delft University of Technology, Netherlands, September 1987.

[9] A.D. Kalvin. A Survey of Algorithms for Constructing Surfaces from 3D Volume Data. *Research Report*, January 1992. RC 17600 (#77606).

[10] A. Koide and K. Kajioka. Polyhedral approximation approach to molecular orbital graphics. *Journal of Molecular Graphics*, 4(3), September 1986.

[11] W. Lorensen and H. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics (Proc. SIGGRAPH 87)*, 21(4):163–169, 1987.

[12] Martti Mantyla. *An Introduction to Solid Modeling.* Computer Science Press, Rockville, Maryland 20850, 1988.

[13] A. Middleditch and K. Sears. Blend Surfaces for Set Theoretic Volume Modelling Systems. *Computer Graphics (Proc. SIGGRAPH 85)*, 19(3):161–170, 1985.

[14] B. Naylor, J. Amantides, and J. Thibault. Merging BSP trees yields polyhedral set operations. *Computer Graphics (Proc. SIGGRAPH 90)*, 224(4):115–124, August 1990.

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

[16] B.A. Payne and A.W. Tioga. Surface Mapping Brain Function on 3D Models. *IEEE Computer Graphics and Applications*, 10(5):33–41, Sep. 1990.

[17] A.A.G. Requicha. Representations for Rigid Solids: Theory, Methods, and Systems. *ACM computing surveys*, 12(4):437–464, December 1980.

[18] Kees van Overveld and Brian Wyvill. Potentials, Polygons and Penguins. An efficient adaptive algorithm for triangulating an equi-potential surface . pages 31–62, 1993.

[19] Brian Wyvill. The Great Train Rubbery. *SIGGRAPH 88 Electronic Theatre and Video Review*, Issue 26, 1988.

[20] Brian Wyvill and Kees van Overveld. Constructive *Soft* Geometry: The unification of CSG and Implicit Surfaces. Technical report, University of Calgary, Dept. of Computer Science, 1995.

[21] G. Wyvill and A. Trotman. Ray tracing soft objects. *Proc. CG International 90*, 1990.

[22] Geoff Wyvill, Craig McPheeters, and Brian Wyvill. Data Structure for Soft Objects. *The Visual Computer*, 2(4):227–234, February 1986.

Animation and Design Systems for Skeletal Models

Brian Wyvill

Abstract

A variety of techniques have been developed for animating implicit surfaces. In this chapter we look at deformation, matamorphosis, blending, warping and making objects glow.

1 Introduction

In this chapter we look at some techniques for animating implicit surface models. In particular we are interested in methods which show how a distinct advantage may be gained from using skeletal implicit surface models. Skeletal implicit surfaces were initially developed in the early to mid 1980's, in the *flying logos era*, too many commercial animations consisted of a 3D model of someone's logo spinning through space. By simply moving the individual skeletal elements an animation could be created which showed a model changing shape over time. Since then a number of other techniques have been developed which are particular to these models, some examples described in this chapter are:

- Path Following
- Negative Primitives
- Metamorphosis
- Warping
- Glowing Objects

Traditional animators often criticize 3D computer generated animation for the stilted way in which objects move. Computer generated objects or *characters* tend to lack the subtleties in motion seen in traditional animation. Characters do not have to be humanoid, objects can be given character such as the brooms in the Sorcerer's apprentice sequence from Disney's Fantasia. In other words they are anthropomorphic. The motion of such objects is controlled to a fine degree to characterize their movement. Characters tend to bend as they move. At times, they must conform to their surroundings: a figure sitting in a chair for example, or flowing water. In computer animation, popular modeling techniques such as polygon meshes or spline patches do not lend themselves to the manufacture of models which can perform this type of motion. However implicit surface models do lend themselves to certain kinds of shape change. The intention is to let the animator design the skeleton of the character or object and then the system automatically *clothes* this skeleton with a surface. If the skeleton moves then the surface changes its shape smoothly to conform. If the skeleton undergoes metamorphosis to a totally different skeleton or inbetweens to a skeleton in a new position then a new surface is calculated at every frame. The surface is represented in such a way that it maintains nearly constant volume as the skeleton moves, thus providing convincing *character coherence*. This property of coherence is very important. Intuitively,

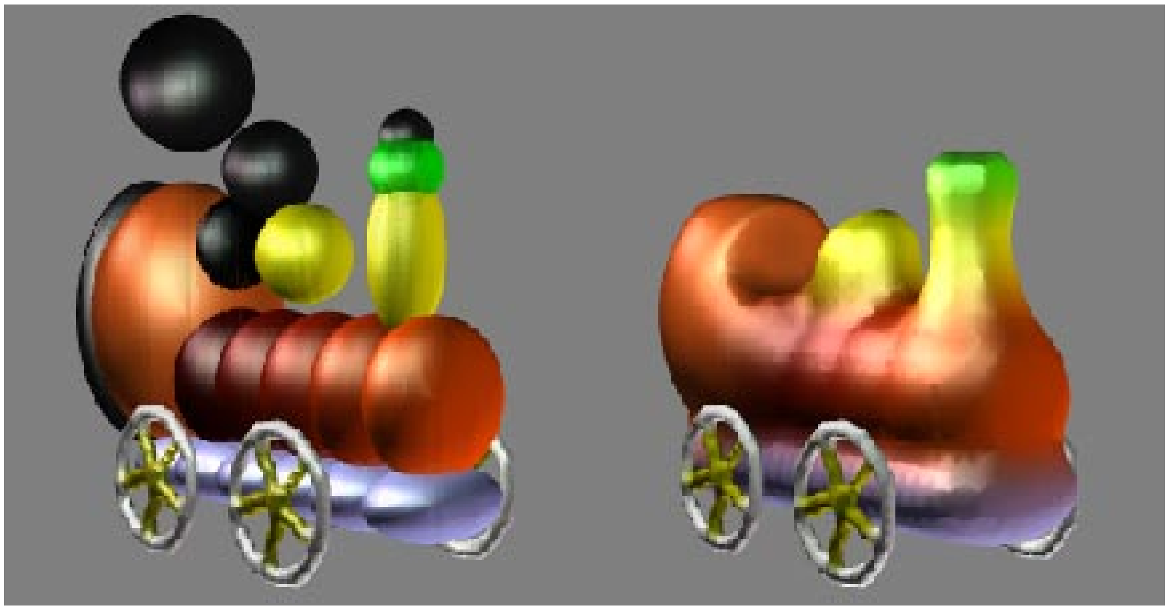


Figure 1: Train Skeleton

it means that throughout metamorphosis, the model remains recognizable as the same character. A model built from skeletal elements can be moved without shape change by maintaining the relative spatial relationship between the elements, in the absence of any influencing field. Achieving shape change without distorting the model beyond recognition requires that certain constraints be placed on the animation system. The following techniques can be used to animate a model built from a skeleton to achieve shape change:

- Geometric Transformation: motion of skeletal elements relative to each other.
- Altering the Field: influence of global or local field and warping.
- Altering parameters describing a skeletal element.

The following sections describe how these methods can be used by high level motion specifications to create the appropriate blended surfaces. It should be noted that this method does not manufacture accurate human models. However it does provide a very fast way of producing blended surfaces that can approximate a humanoid character. The methods described here are low level, they affect individual skeletal elements or groups of skeletal elements. It is intended that they can be assimilated into a higher level animation system.

1.1 The Implicit Function

The skeleton is surrounded by a scalar field $F_{total}(P)$ (equation 1). The intensity of the field being the highest on the skeleton, and decreasing with distance from the skeleton. The function $F_{total}(P)$ relates the field value (intensity) to distance from the skeleton, it has an impact on the shape of the surface, and determines how separate surfaces blend together (see [2]). The surface is defined by the set of points in space for which the intensity of the field has some chosen constant value (or iso-value thus the name *iso-surface*). Fields from the individual elements of the skeleton are added to find the potential at some chosen point. (Values can be negative or positive). The value at some point in space is calculated as follows:

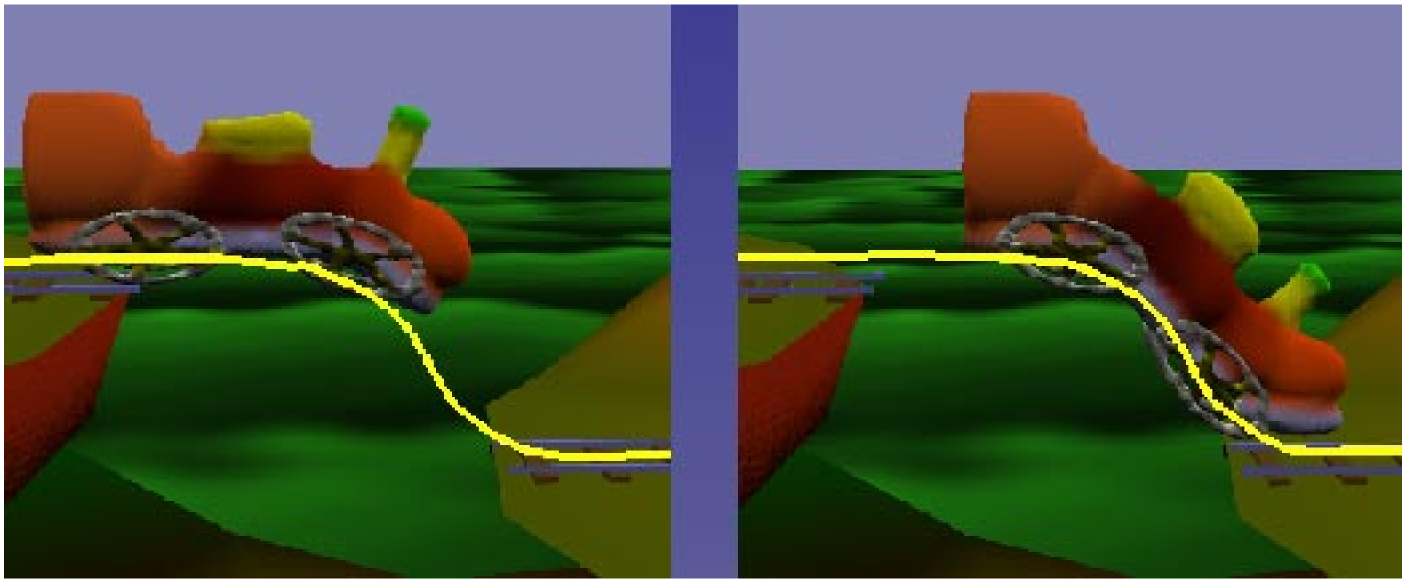


Figure 2: Bending the train

$$F_{total}(P) = \sum_{i=1}^{i=n} c_i F_i(r_i) \tag{1}$$

where P is a point in space
$F_{total}(P)$ is the value of the field at P
n is the number of skeletal elements
c_i is a scalar value (used for positive or negative elements)
F_i is the blending function of the i_{th} element
r_i is the distance from P to the nearest point Q_i on the i_{th} element.

The parameters listed above can be used to alter the shape of the surface. By changing one of these over time animation can be achieved which involves shape change difficult to achieve with other modelling techniques.

2 Geometric Motion

By simply altering the relative spatial relationship between skeletal elements a surface can be made to alter its shape. Figure 2 shows the train model with the position of the skeletal elements marked. By moving the skeletal elements the surface blends smoothly and the train appears to bend.

2.1 Path Deformation

Motion paths are extremely useful when applied to the skeletal elements of an implicit surface model. A cartoon-like character can be made to conform to its surroundings by using the shape of an object to define the path which a group of skeletal elements must follow. This effect was demonstrated in the movie SOFT (see [6]). The character in this case was a group of letters spelling the word “SOFT”.

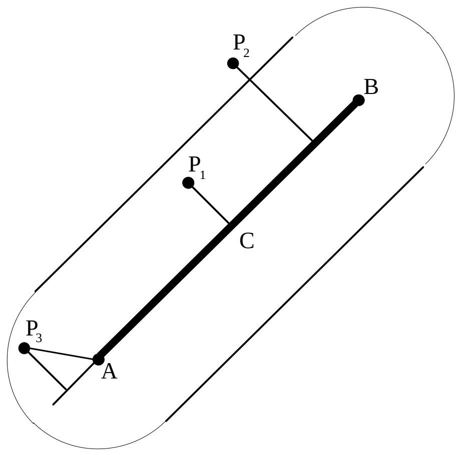


Figure 3: Line Primitive

Each letter was made from about 20 skeletal elements, the path was drawn up a flight of stairs by marking various positions and passing a spline through these points using a spline technique similar to that described in [3]. Each skeletal element in the character is moved to the interpolated path position at each frame. Each point along the path represents a position at a particular instance in time. Rather than define a separate path for each key, the skeletal elements are grouped so that each group is moved together to the next point in the path. Normally a group is chosen according to some spatial relationship, for example all skeletal elements within a specific range of z values. To maintain "character coherence" the relative positions of the skeletal elements within a group must be maintained within certain constraints. In this case the amount each group is allowed to vary its position is constrained by the change along the path. Each skeletal element maintains its position relative to the group. It is the origin of the group that follows the path. The positions of the groups change relative to the other groups by an amount specified in the path. Thus the model undergoes the desired deformation but maintains the overall shape and retains the character coherence property which distinguishes shape distortion from metamorphosis. When the letters are moved up the stairs (see figure 4) in the movie SOFT the direction of motion was along the (-)ve z axis so the groups were chosen as skeletal elements with same z value. As each group advances along the path the y value of the group was altered according to the path specification. Figure 2 is another example of path animation. The train model is comprised of a number of elements arranged in vertical columns (see figure 1), each column is moved as a single unit to the next point along the path.

2.2 Altering the Field

Implicit surface objects may also be animated by altering the field in which the objects exist. Local deformation can be achieved by moving other skeletal elements relative to these objects, or by placing some global influence into the field. Global deformation is more difficult to specify in

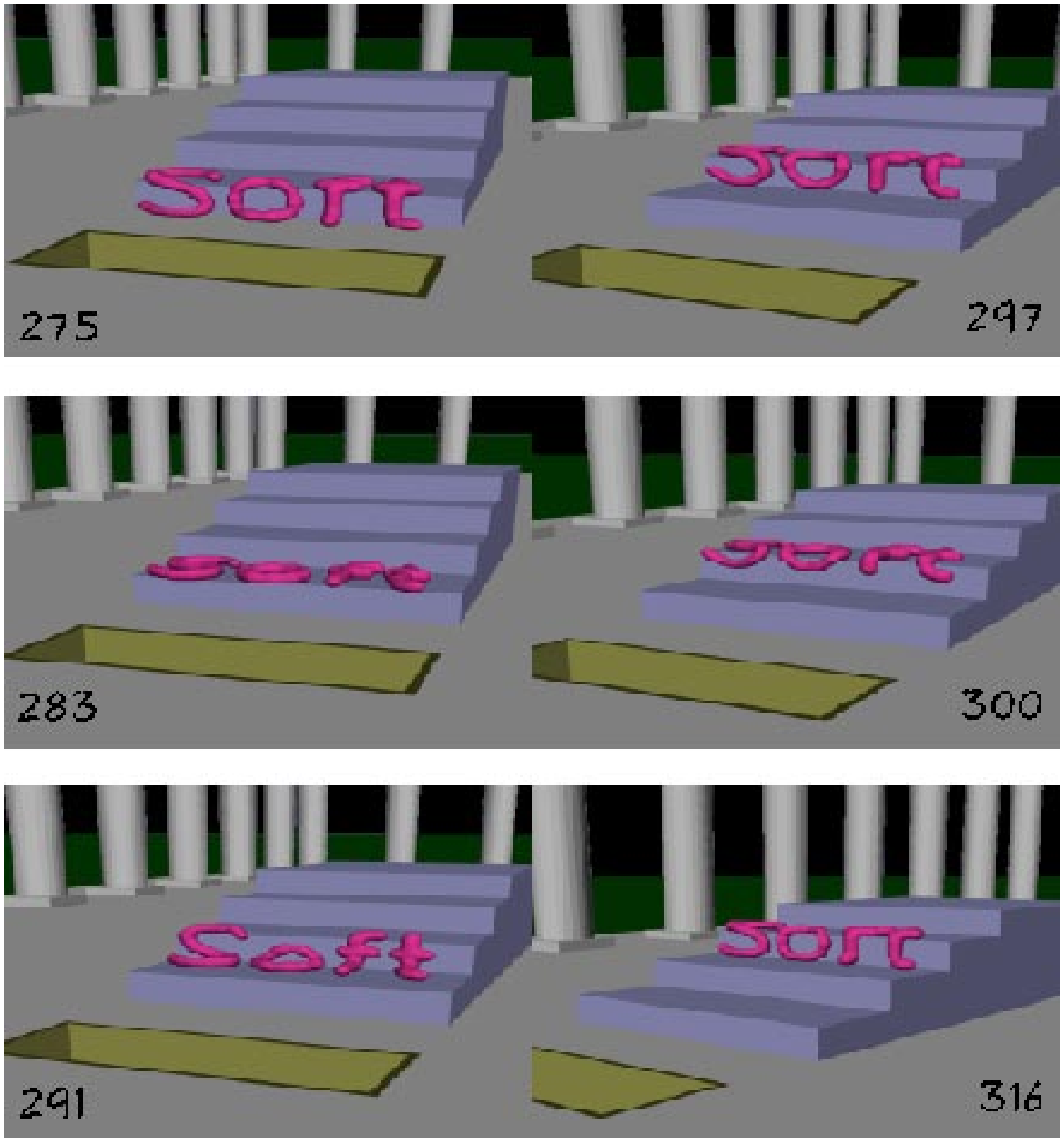


Figure 4: Frames from the animation SOFT

a completely general way. The field itself could have some external influence such as a plane of constant value. Objects approaching that plane will deform according to the value chosen. This technique has been adopted along with warping, see section 4.

3 Metamorphosis

One area in which implicit surface models are particularly useful is in metamorphosis. A method suitable for two-dimensional (cartoon) models, would have the character drawn on a display in two positions, and the inbetween frames interpolated by the system. The simplest way to do this is to interpolate each point of the first (*source*) object to a corresponding point on the second (*destination*) object. Difficulties arise if the number of points is different on source and destination objects. New points have to be created or several points have to collapse onto a single point. Even if the number of points is the same on the two objects, if they are distributed in a different way the image will be scrambled as each point is interpolated. Implicit surface models always guarantee a closed surface, so this problem does not arise. However, it is still easy to lose character coherence in the inbetween versions.

This technique can also be used to show metamorphosis from one character to another. If the characters are very different in shape and number of skeletal elements, then the scrambling problem is difficult to avoid. Peter Foldes uses software by Burtnyk and Wein [1] in the film, “La Faim” and exploits this technique to good advantage. However to avoid *scrambling* is a difficult and tedious task which requires very careful design of the skeletal element positions for inbetweening.

Burtnyk and Wein developed a computer version of this technique using skeletons. These skeletons defined a conformal mapping from one key frame to the next. The space itself was distorted, thus any line within the space was similarly distorted according to the mapping function. In contrast, 3D computer generated characters are often moved by applying geometric transforms to the different parts, which changes their relative positions but do not necessarily give the character a smooth change of shape as can be achieved using the 2D techniques. An effective way of producing shape change in 3D animation is to use the inbetween technique. However extending the 2D technique to 3D introduces new problems. Reeves points out in [5] that it is difficult to identify corresponding points (and polygons) on different characters. Even with functional representations, the parameters from which a surface is manufactured must be chosen so that the source model *matches* the destination model. Each of the parameters defining the *source* model must be changed to one of the parameters defining the destination model. The matching process chooses the appropriate *destination parameters* corresponding to the *source parameters*. At each intermediate stage during the inbetween, a model will be manufactured from an interpolated set of parameters.

In the following paragraphs several different heuristics for matching the models are presented. The shape of the intermediate models vary according to the chosen method, based on one or more of the heuristics.

3.1 Heuristics for Point Matching in Metamorphosis

In this section four approaches are described that we have found useful for defining the matching process. Although the implicit surface modeling system has been used to illustrate how these heuristics may be applied, the methods are general and can be extended to other modeling techniques. In practice an animator will want to experiment with different combinations of these techniques to arrive at the desired effect.

We start with two models; the source model and the destination model. The source model must be made to change into the destination model. The models are defined as a set of skeletal elements as described above. Although point primitives are used in the discussion on metamorphosis, these techniques do have application using other geometric primitives. Each ellipsoid skeletal element has the following properties:

Axes Vectors	$\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3$

Position	x, y, z
Force	F

Each of these methods assumes that the objects have been pre-processed so that there are the same number of skeletal elements defining each object. This may involve creating zero weighted skeletal elements. A skeletal element can be weighted using the force, F, which scales the contribution to the field, or by scaling the axes. A zero weighted element has its axis vectors set to zero or $F = 0$. When a source element is inbetweened to a destination element, at each frame a new element is chosen which has an interpolated value for position, axes vectors and force. The start or finish position of the new skeletal elements is chosen by the appropriate method.

3.1.1 Hand Matched

The simplest method of establishing which skeletal elements are to be interpolated is to to order the source and destination skeletal elements by hand and to process each pair in turn. Since the number of skeletal elements is small compared to the number of polygons in an equivalent model, this method is feasible for some objects. However computer animation is generally moving towards higher levels of control so this method is considered a last resort.

3.1.2 Hierarchical Matching

In this heuristic it is assumed that each model is represented by a hierarchy of skeletal elements. Each node in the hierarchy has an arbitrary number of sibling nodes and one or zero child nodes. Nodes are matched at the same level in the hierarchy. It is assumed that the hierarchies are designed that each level of the source object has an equivalent level in the destination object. If a man is to change into a rabbit the heads will be matched, the arms of the man can match the front legs of the rabbit and so on The main problem with this approach is that the hierarchies have to be constructed carefully. Not only do the levels have to match but within a level the nodes must either be ordered or labeled to match. Despite these drawbacks this method is still preferable to ordering all the skeletal elements by hand and for small sets of skeletal elements, with suitable interactive tools quite acceptable.

3.1.3 Cellular Inbetweening

In this technique the models are matched corresponding to the space they occupy. The world is first divided into a 3D grid of cells. This is done by finding the extents of each model and manufacturing the corresponding rectangular box. Each box is then divided along the x,y,z axes by some user defined amount. The two boxes may be different shapes but they are divided into an equal number of cells. The skeletal elements are then sorted into the cellular grid and the skeletal elements in each source grid cell are then interpolated to the skeletal elements in the corresponding grid cell of the destination model. The objects can have different sizes but the method tries to maintain some sort of position coherence between source and destination objects. Figure 5 shows a 2D version of how the skeletal elements are matched. Circles with similar shading patterns are matched between source and destination. In the top diagram the points match exactly, for every element in every cell in the destination object there is a corresponding element in the corresponding cell in the source object. In the lower diagram there are some cells containing skeletal elements in the destination object for which there are no skeletal elements in the corresponding cell in the source object. In

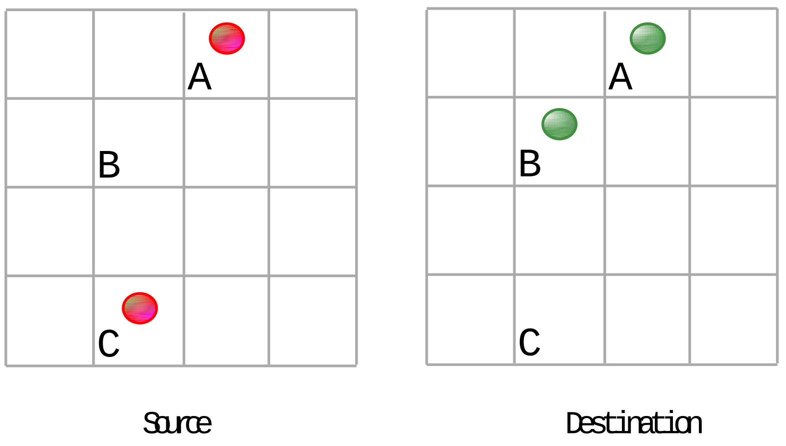


Figure 5: Cellular Inbetweening

this case a zero weighted element (indicated as a small circle) will be manufactured in the source object. Similarly skeletal elements which exist in the source object are grown in the destination.

3.1.4 Surface Inbetweening

In this method there is no matching necessary. All the skeletal elements from both source and destination objects define each intermediate model. However the force property of each source element is weighted. The weighting is gradually changed from one to zero as the inbetween progresses. Also dependent on time is a second weighting applied to the force property of the destination skeletal elements. This value changes from zero to one. The shape of the weight value vs. time curve controls the shape of the intermediate model. This is shown in figure 6. A simple linear interpolation means that both source and destination objects are reduced to half the weight half way through the simulation. In practice this gives poor results as the surfaces around each element no longer merge. If the source is weighted by a cosine function and the destination weighted by a sine function each object is never weighted by less than $\frac{1}{\sqrt{2}}$. The objects can still be matched by one of the sorting techniques (hand, hierarchy, cellular), but the inbetweening process is different using surface inbetweening. An example is shown in [7].

4 Warping

A useful tool in our system is the ability to distort the shape of a surface, by warping space around it. A warp is a continuous function, w , from $\mathbb{R}^3$ into $\mathbb{R}^3$. In the following section we suggest some specific warp functions that are useful for producing some unusual animations.

The warped surface is defined from equation 1 above:

$$F_{total}(P) = \sum_{i=1}^{i=n} c_i F_i(r_i)$$
$$r_i = f_i(P) = dist(w_i(P), Q_i)$$

where $w_i(P)$ is the position of the point P in warped space. In fact each skeletal element may reside in a different warped space. So when evaluating the contribution from the i^{th} skeletal element, P

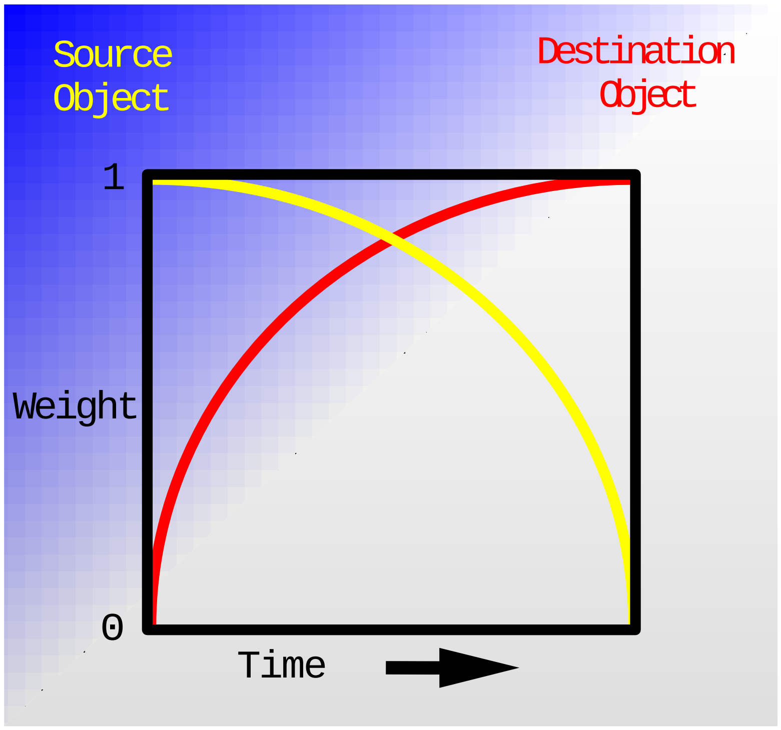


Figure 6: Surface Inbetweening

is first warped to the appropriate position before evaluating the distance function. As a first example, we study a warp function $w_i(P)$, which warps a point P to a point Q along a given vector v; it may be given by the vector equation:

$$w_i(P) = P - \hat{\boldsymbol{v}}(\boldsymbol{v}.\boldsymbol{p})$$

where p is $P - S_{0,i}$.
$S_{0,i}$ is the origin of the i_{th} skeleton
$\hat{\boldsymbol{v}} = \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|}$

To understand how this affects the iso-surface consider P to be a point some distance from the surface of a sphere, such that the point is warped in the direction of the center of the sphere, to a position Q which is on the iso-surface. The value returned for P by the implicit function is the value that would have been returned for Q if warping were not in effect. In this case that value is the iso-value. Thus P becomes a point on the surface and the sphere is warped to an ellipsoid. (See Figure 7)

Many kinds of warp are possible, by simply writing a function that transforms a point from ordinary space into warped space. Each skeletal element contributes in a local way to the warped space, since each has its own local warp function associated with it. John Lasseter notes [4] the importance of squash and stretch in traditional animation. The example he gives is of a ball traveling along a parabolic path and bouncing. The shape of the ball distorts into an ellipsoid to give the feeling of speed, when it bounces the distortion changes so that the long axis of the ellipse is parallel to the ground, in other words a squash effect.

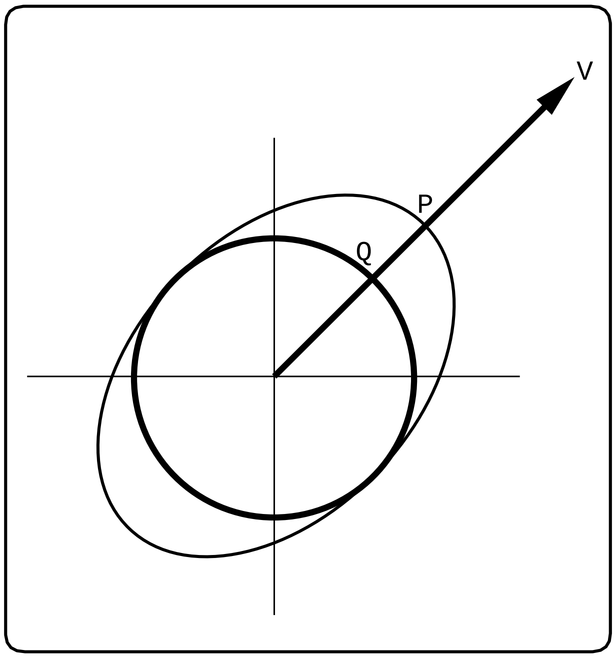


Figure 7: P is warped to Q. The original sphere is warped to an ellipsoid.

To simulate the distortion of the ball to the ellipsoid, the usual computer graphics approach would be to use a scale operation over time. Since the scale is in the direction of the velocity vector v, two rotations are necessary, to first align the object with one of the major axes, then to rotate back again. With a complex 3D object consisting of many skeletal elements, shape distortion using a scaling operation may not be exactly what the animator requires. For instance on the impact plane, the ball should flatten out and the distortion is different from the deformation when the ball is further away, an effect which cannot be achieved with linear transformations. By exaggerating the non-linearity the ball could appear to be made of putty. Such a non-linear operation can easily be achieved by a warp operation, as shown in the example below.

Figure 8 shows some frames selected from an animation showing the putty like ball bouncing. This is implemented in the warp function in the following manner:

Let P be a point in space at which the implicit function is to be evaluated. For simplicity, assume the collision plane to be the plane $y = 0$.

$$w(P) = \begin{cases} P - \beta\hat{\boldsymbol{v}}(\boldsymbol{v}.\boldsymbol{p}) - \boldsymbol{p}\downarrow\gamma S & \text{if } P.y > 0 \\ (P.x, \infty, P.z) & \text{otherwise} \end{cases}$$

Here

$p = P - S_0$

$S_0 =$ the origin of the skeleton

$p\downarrow = (P.x, 0, P.z)$

$\gamma = h\left(\frac{S_0.y}{y_0}\right) h\left(\frac{P.y}{y_0}\right)$

$h(t) =$ a decreasing differentiable function (e.g.a cubic polynomial) such that:

$$\begin{aligned} h(t) &= \begin{cases} 1 & \text{for } t \leq 0 \\ 0 & \text{for } t \geq 1 \end{cases} \\ \beta &= \mathit{clamp}(1 - 2.0\gamma) \end{aligned}$$

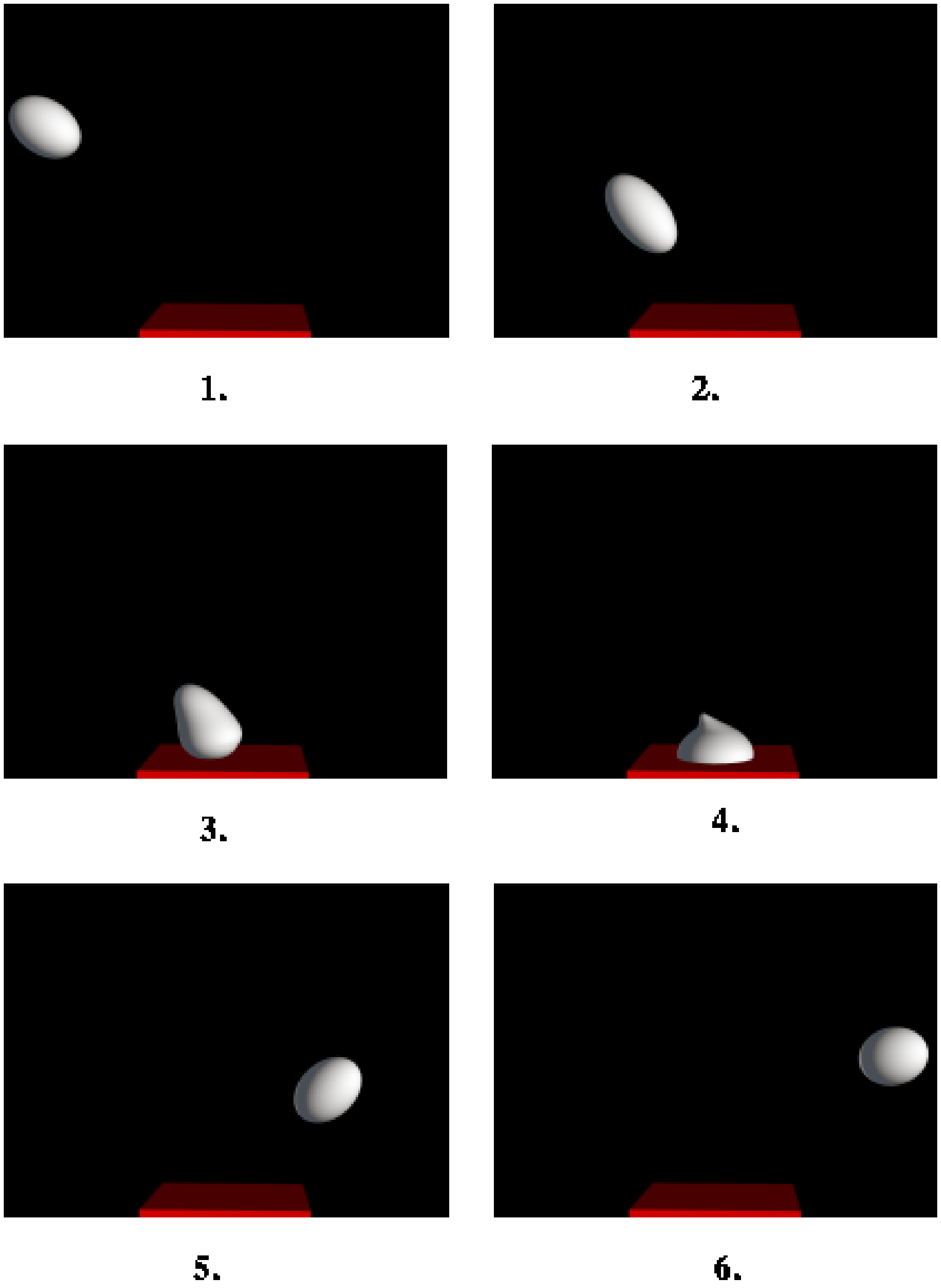


Figure 8: Frames from the bouncing ball animation.

$$S \;=\; a \text{ parameter, typically around } 0.5$$

The interpretation of the terms is as follows:

$(P.x, \infty, P.z)$ guarantees that no part of the object protrudes below the collision plane.

$-\boldsymbol{p}\downarrow \gamma S$ accounts for spreading out the lower part of the object (squashing). The vector $\boldsymbol{p}\downarrow$ is parallel to the collision plane indicating that squashing should be a horizontally directed effect. The factor γ assures that squashing increases near the collision plane. No squash is applied if the center of the objects is higher than y_0 or if P is higher than y_0. The parameter S controls the amount of squashing.

$-\beta\hat{\boldsymbol{v}}(\boldsymbol{v}.\boldsymbol{p})$ is a modified version of the simple linear velocity warping as discussed in section 4. The factor β is introduced to quench the velocity warping near the impact point, to avoid discontinuities in the warp function.
At the point of impact, v undergoes a discontinuous change,
$(v.x, v.y, v.z) \rightarrow (v.x, -v.y, v.z)$.

This would cause a discontinuous warp function if it was not compensated. The factor 2.0 in the expression for β has been found experimentally to be a reasonable value. The function *clamp(..)* clamps its argument value between 0 and 1.

This approach initially aligns the warp vector with the velocity vector of the ball in the bouncing ball example. When the impact occurs the ball will start to deform in a non-linear oblate fashion according to the $-p\downarrow \gamma S$ term. At the same time the linear prolate deformation (due to $-\beta\hat{v}(.p)$) subsides. Our implicit surface models can be a collection of skeletal elements, rather than a single spherical element, such as the ball. Thus we can easily apply this non-linear warp to a complex shape. In figure 8 some frames are shown from an animation which applies this method to a bouncing ball (one soft primitive $S = 0.25$). The ball is distorted into an ellipsoid (Frame 1) whose long axis grows as the vertical velocity increases (Frame 2). On impact the ball warps (Frame 3) into a bulging shape. As the ball bounces it regains its ellipsoid shape and as its vertical velocity decreases the ball remains stretched along the horizontal axis by an amount corresponding to its horizontal velocity. The technique easily extends to skeletons consisting of many primitives.

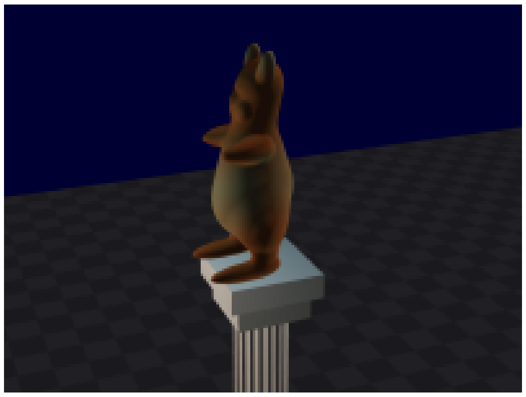
In figure 9 the same method is shown to work for Nelson, the jumping bear (consisting of 25 soft primitives). The slug in Figure 10 is in fact a warp applied to three ellipsoid primitives. Warping can be applied in space or time and may be non-linear, for example, a warp can be applied:

- To the space in which a model exists, then move the model.
- To the space over time, the model will change with time.

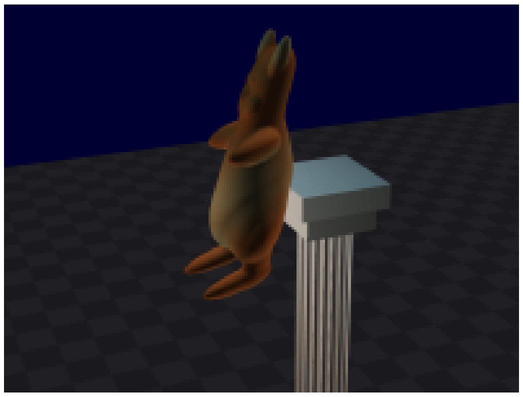
We have tried several different types of warp and the outstanding problem is to present warping to the animator with a consistent user interface, so that custom warps may be designed.

5 Ray Tracing Glowing Objects

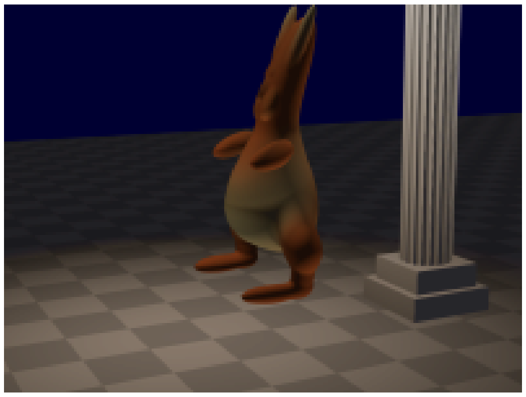
Making an object glow is a useful effect in computer animation. This can be done relatively easily with implicit surface models. A glow should be seen surrounding an object and fade to zero at some distance away. A value for the brightness of the glow can be obtained as a function of the field in which the model is defined.



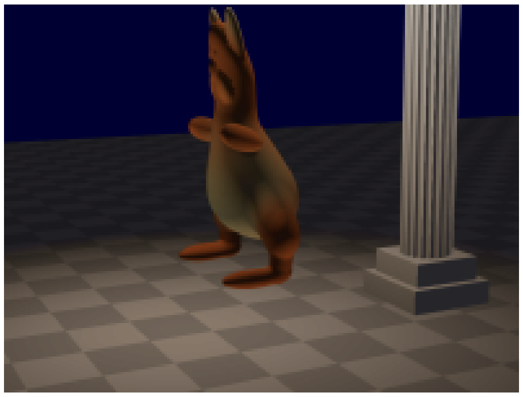
1.



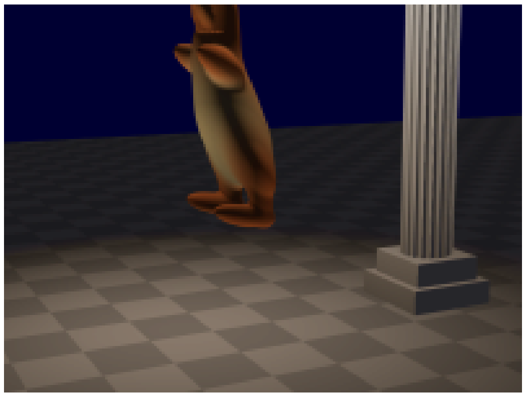
2.



3.

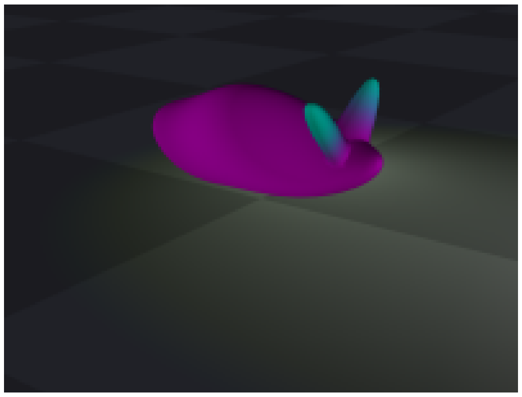


4.

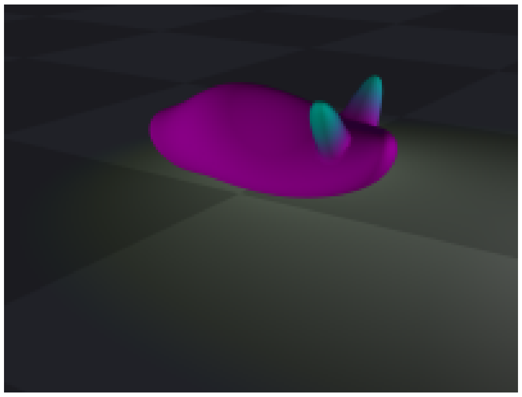


5.

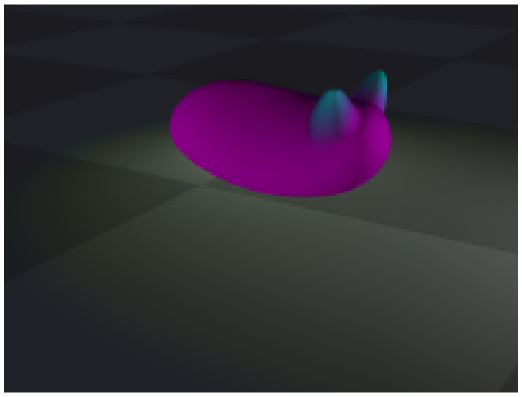
Figure 9: Frames from Nelson the bouncing bear animation.



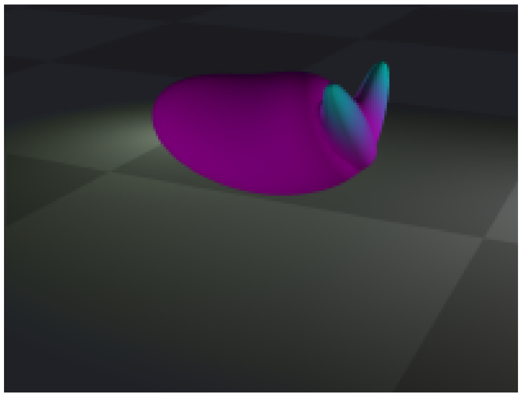
1.



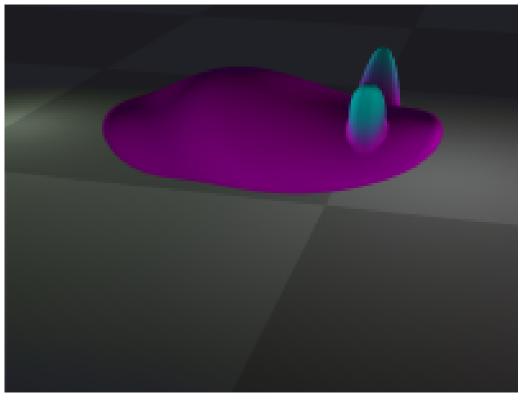
2.



3.



4.



5.

Figure 10: Frames from the slug animation.

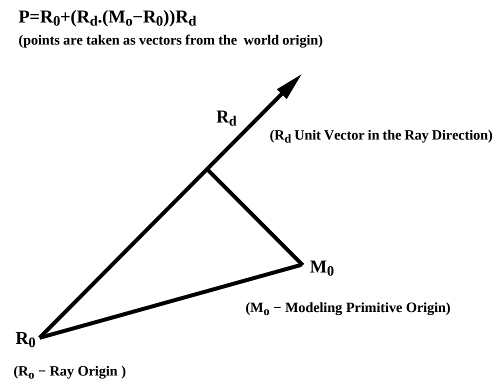

$P=R_0+(R_d.(M_o-R_0))R_d$
(points are taken as vectors from the world origin)
R_d
(R_d Unit Vector in the Ray Direction)
M_0
(M_o – Modeling Primitive Origin)
R_0
(R_o – Ray Origin)

Figure 11: Finding the nearest distance from a ray to an implicit surface primitive

This has been implemented as a part of a ray tracer. The nearest distance between each ray and each primitive ellipsoid is calculated. The point on the ray is shown as P in figure 11. One way to calculate the glow is to take the value of P corresponding to the shortest of these distances and pass it to equation 1 to obtain the implicit value, v. This is used to calculate the brightness of the glow as follows:

$$G_\lambda = m_\lambda v' \tag{2}$$

where

$$v' = \begin{cases} 0 & \text{if } v < 0 \\ 0.5 & \text{if } v > 0.5 \\ v & \text{otherwise} \end{cases} \tag{3}$$

and $m_\lambda \epsilon [0, 1]$ are scalars controlling the intensity of independent wavelengths.

There is a problem with this procedure. Consider two neighbouring rays, R_a and R_b. The nearest primitive to R_a may be different from the nearest primitive to R_b, thus returning two different points with correspondingly different implicit values. Thus causing a discontinuity in the glow. An obvious method around this problem would be to find the closest distance for each primitive and sum the implicit values contributed by each primitive for each ray. Since each primitive contributes a value between 0 and 1, the sum can be normalized by simply dividing by the number of primitives. This procedure causes the glow to be very dense near to areas where there are lots of primitives grouped together, but far too sparse in areas where there are few primitives. For example figure 12 shows a dinosaur model next to its glowing counterpart. The tail section received no glow by this method. To achieve the even glow in the figure the points along the ray representing the closest point to each primitive are selected as above. The implicit values are calculated as before but only the N highest values are used for the glow calculation. It has been found empirically for our models that $N = 3$ is a reasonable number to choose. With $N < 3$, discontinuities appear, with $N > 3$, no glow is seen where there are few primitives.

Figure 13 shows two rows of merging sphere primitives. The top row has been raytraced and a texture applied. The bottom row is the same set of textured primitives with a glow added. The advantage of using implicit surface models is that the glow can be calculated directly from the field as shown above. The shape of the glow follows the zero contour. Values of the field greater than zero are brighter than the background.

It is also possible to change the shape of the glow, by altering the blending function used in the expression for $F_{total}(P)$. For the dinosaurs we use the cubic (introduced in [8]). The glow field can

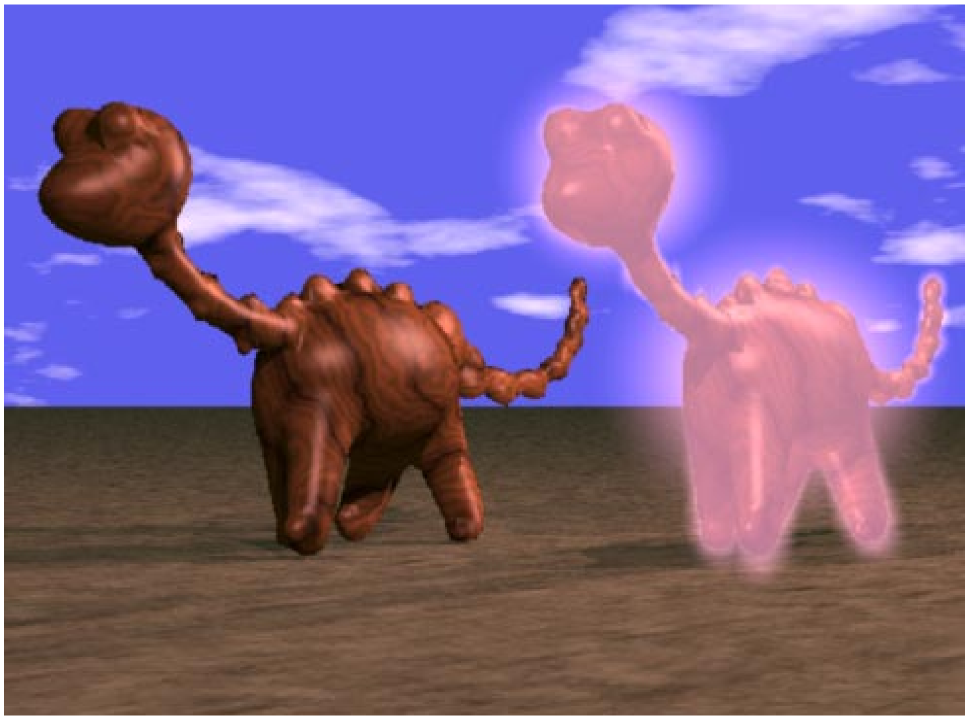


Figure 12: When Dinosaurs Glowed!

be increased by using a function that falls to zero more slowly than the blending function for the train model itself.

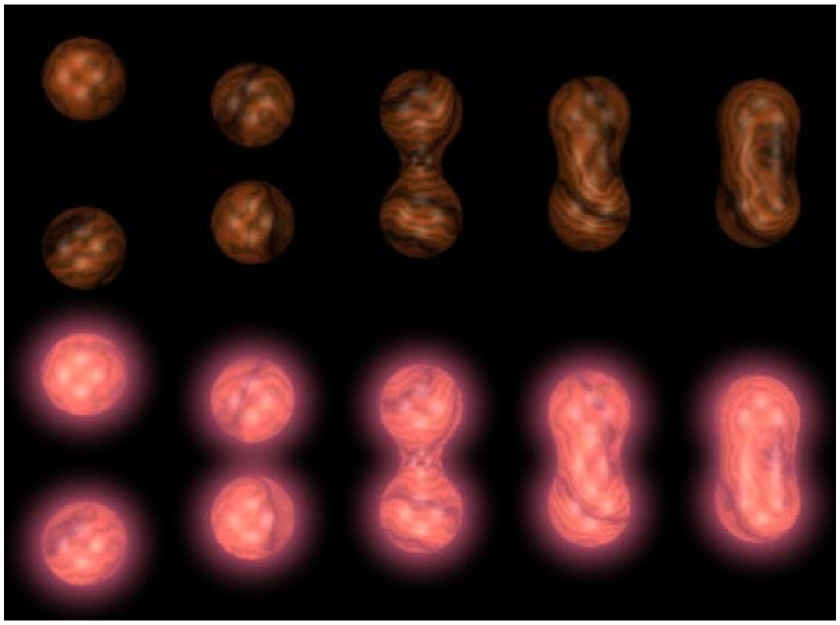


Figure 13: Spheres Blending using the cubic blending function and demonstrating the glow.

6 Conclusions

In this chapter some techniques for animating implicit surfaces have been presented. Skeletal Implicit Surface techniques have proved useful for designing models but still await widespread use in the design community.

7 Acknowledgements

I would like to thank the many students who have contributed so greatly to this research. In these notes Andrew Guy helped with the proof reading and made the fist and Radomir Mech made the glass dinosaur from one of my original models. I would also like to thank Jules Bloomenthal for his encouragement and ideas over the years and to Kees van Overveld. I am particularly in debt to my brother, friend and colleague, Geoff Wyvill, who started the whole thing off by solving a problem in scientific visualization, the solution to which turned out to be so useful for building models.

This work is partially supported by the Natural Sciences and Engineering Research Council of Canada in the form of a research grant and equipment grants.

References

[1] N. Burtnyk and M. Wein. Interactive Skeleton Techniques for Enhancing Motion Dynamics in key Frame Animation. *CACM*, 19(10):564, Oct 1976.

[2] Z. Kacic-Alesic and B. Wyvill. Controlled Blending of Procedural Implicit Surfaces. Technical Report 90/415/39, University of Calgary, Dept. of Computer Science, 1990.

[3] D. Kochanek. Interpolating Splines with local Tension, Continuity and Bias Control. *Computer Graphics (Proc. SIGGRAPH 84)*, 18(3):33–41, 1984.

[4] John Lasseter. Principles of Traditional Animation Applied to 3D Computer Animation. *Computer Graphics (Proc. SIGGRAPH 87)*, 21(4):35–44, July 1987.

[5] W. Reeves. Inbetweening for Computer Animation Utilizing Moving Point Constraints. *Computer Graphics (Proc. SIGGRAPH 81)*, 2:263–269, 1981.

[6] Brian Wyvill. SOFT. *SIGGRAPH 86 Electronic Theatre and Video Review*, Issue 24, 1986.

[7] Brian Wyvill, Jules Bloomenthal, Geoff Wyvill, Jim Blinn, John Hart, Chandrajit Bajaj, and Thad Bier. Course Notes. *SIGGRAPH '93, Course #25, Modeling and Animating with Implicit Surfaces*, 1993.

[8] Geoff Wyvill, Craig McPheeters, and Brian Wyvill. Data Structure for Soft Objects. *The Visual Computer*, 2(4):227–234, February 1986.

Implicit Blends and Skeletal Methods

Jules Bloomenthal

In this course notes chapter we relate review several implicit blend methods, particularly those related to a defining skeleton. We also examine in depth the related problem of surface bulge, and methods for its elimination.

smooth

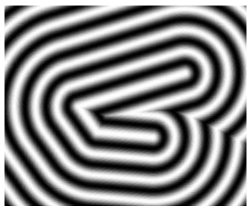
blend

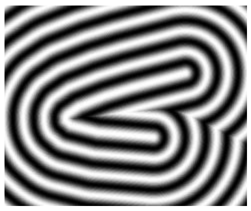
fillet

round

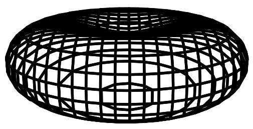
blobby molecule

Distance to a Curve

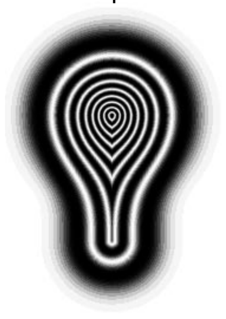

3 linear segments

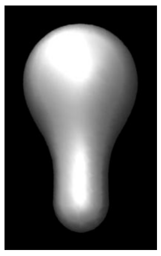

9 linear segments

Blend of Primitives

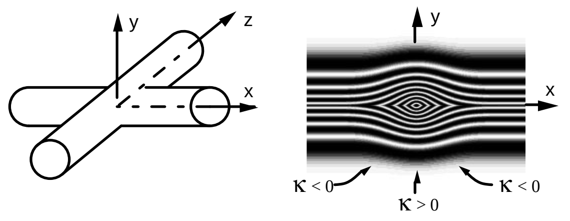

sphere and torus

Super-Elliptic Blend of
Sphere and Cylinder

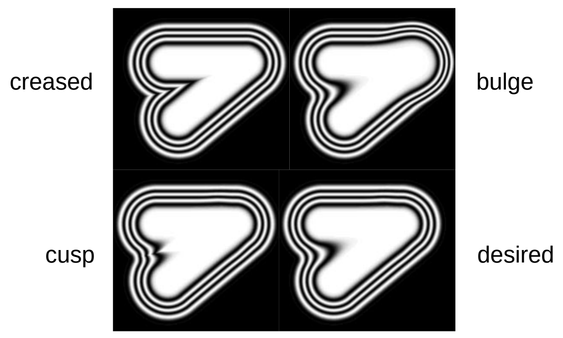




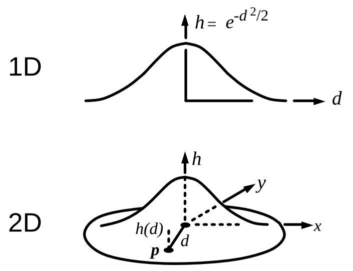
Super-Elliptic Blend of Two Cylinders



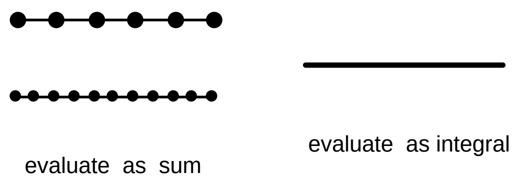
Four Blends of Two Segments



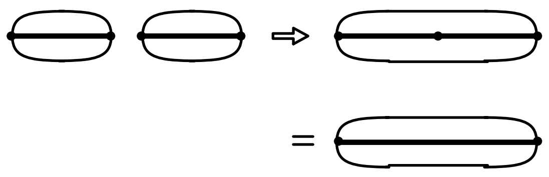
Gaussian Kernels



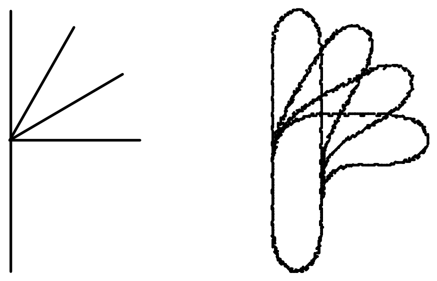
Line Segment Skeleton as a Set of Points



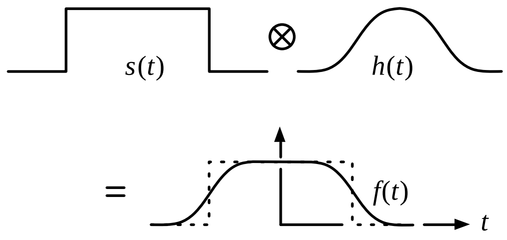
The Superposition Property of Convolution



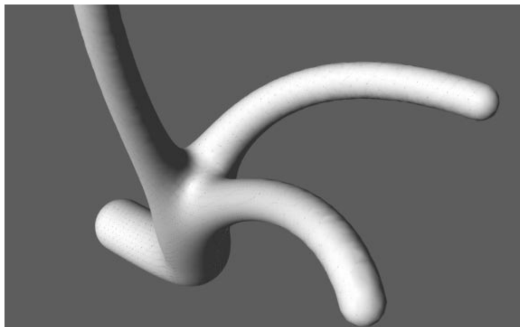
Two Segments and their Convolution



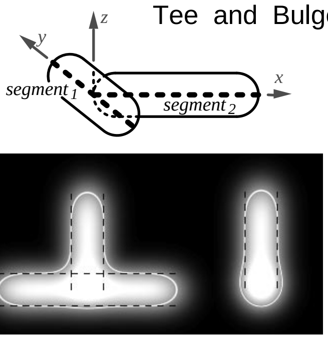
Convolution of a Box Function



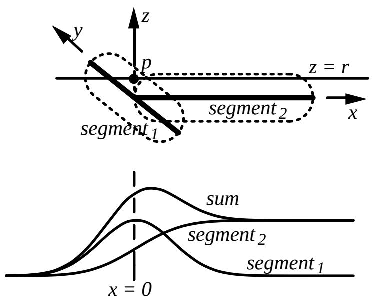
Trifurcated Ramiform with Bulge



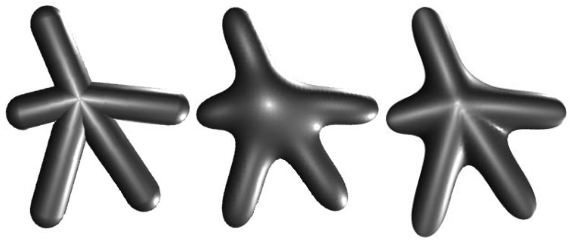
Tee and Bulge



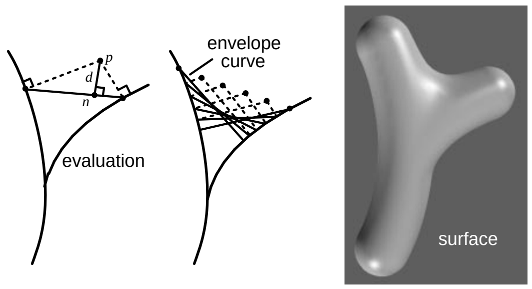
Analysis of Tee Bulge



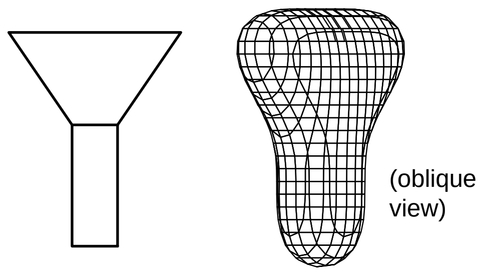
Union Surface, Convolution Surface, and Combination



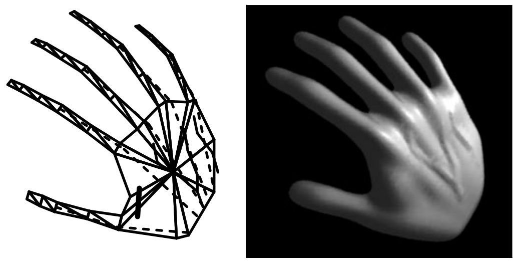
Branching Cylinder with Equal Radii



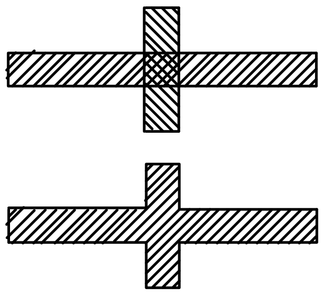
Polygonal Skeleton and Convolution Surface



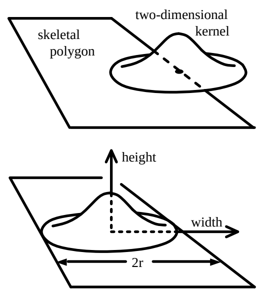
Skeleton and Hand



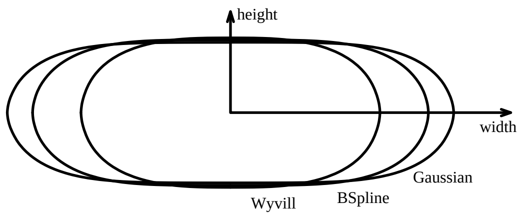
Overlapped and Contiguous Skeletons



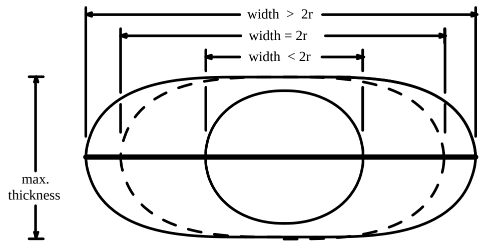
Kernel Coverage of Polygon



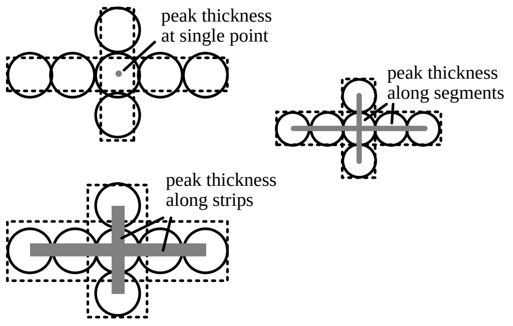
Object Cross-Sections for Various Kernels



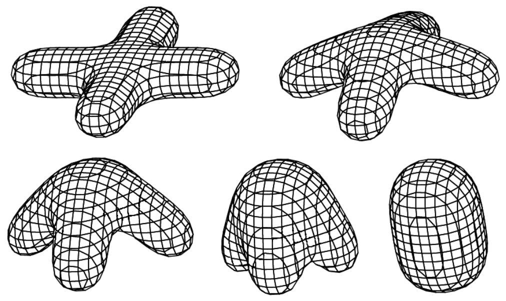
Wide and Narrow Polygons



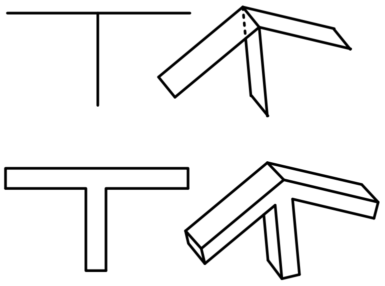
Effect of Polygon Width on Integration



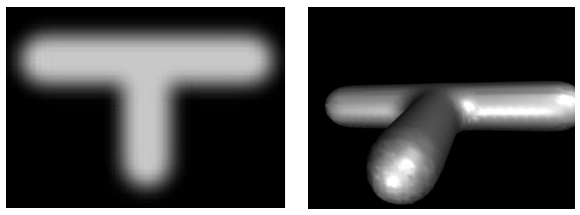
A Smoothly Folding, Bulge-Free Form



1D, 2D, and 3D Skeletons



Bulge-Free, Circular Voxel Blend



References

Much of the above material may be found in my own works, the most relevant of which are listed below. In these works are found numerous references to related material.

A detailed review of skeletal methods:

Skeletal Design of Natural Forms by J. Bloomenthal
Ph.D. Dissertation, University of Calgary, 1995.

Implementation details for implicit surface polygonizers (manifold and non-manifold):

Polygonization of Non-Manifold Implicit Surfaces
by Jules Bloomenthal and Keith Ferguson
SIGGRAPH'95. In Computer Graphics 29, 4.

An Implicit Surface Polygonizer, by J. Bloomenthal
in Graphics Gems IV (Paul Heckbert, editor)
Academic Press, New York, 1994.

An Evaluation of Implicit Surface Tilers
by Paul Ning and Jules Bloomenthal
Computer Graphics and Applications, Nov., 1993.

Polygonization of Implicit Surfaces
by Jules Bloomenthal
Computer Aided Geometric Design 5, 4, Nov., 1988.

Detailed discussions of implicit surface blends:

Convolution Surfaces
by Jules Bloomenthal and Ken Shoemake
Proc. SIGGRAPH'91. In Computer Graphics 25, 4.

Bulge Elimination in Implicit Surface Blends
by Jules Bloomenthal
Computer Graphics Forum (to appear, 1996).

Some useful techniques for spline based skeletons:

Calculation of Reference Frames along a Space Curve
by Jules Bloomenthal
in Graphics Gems (Andrew Glassner, editor)
Academic Press, New York, 1990.

An early example (the tree trunk) of implicit and skeletal techniques:

Modeling the Mighty Maple
by Jules Bloomenthal
Proc. of SIGGRAPH'85. In Computer Graphics 19, 3.

Implicit Skeletal Models and CSG

Geoff Wyvill
University of Otago

Abstract

Implicit equations define the set of points inside an object. These sets can be added and subtracted to form complicated objects and we can make pictures of these objects by ray tracing. Blended implicit skeletal models can be combined with simple implicit shapes like spheres to make a great variety of models.

1 Introduction

Constructive Solid Geometry (CSG) is a way to define and represent models as a hierarchy of combined implicit equations. The method has mostly been used for engineering models where it is useful to include components like a cylinder with precisely known diameter. However, it also offers a very general tool from which we can construct whole scenes and animation sequences.

1.1 Hierarchy

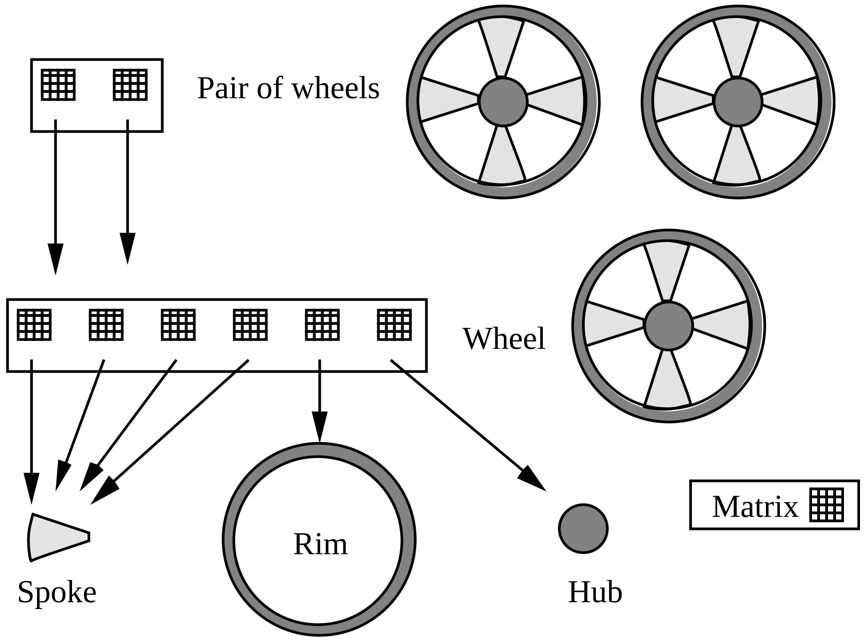


Figure 1: Hierarchy.

Fig. 1 shows a pair of wheels with the associated data structure. Where is the top spoke of the left hand wheel? The pair definition contains a matrix, L, to position the left hand wheel in the pair.

The wheel definition contains a matrix, T, to position the top spoke. The position of the spoke in the pair is defined by:

$$LT \tag{1}$$

In a hierarchical modeling system, we can represent objects of arbitrary complexity. Everything is defined in terms of something else until, at the lowest level, we must use some more basic object that does not need the hierarchy to describe it. In hierarchical systems we call these bottom level objects *primitives.* In most conventional systems the primitives are polygons or parametric patches. In CSG systems, they can be implicit equations defining spheres, cylinders and other mathematical shapes. The matrices at each level provide a translation from the coordinate system of the component objects to that of the current object. We can process the structure in a top-down traversal to find the coordinates of the basic points:

```
procedure generate(matrix, graph):
    for each component of graph do
      if component is a primitive then output(matrix, component)        (2)
      else generate(matrix * component.matrix, component.graph)
```

1.2 Inverse operations

The procedure 'generate' answers the question "Where is the top spoke of the left wheel?" In interactive systems, we sometimes select a point in an image and would like the system to find out which logical component we are selecting. This amounts to finding the point within the definition of wheel that translates to the given point. We do this using an inverse matrix. In some systems, it is useful to create the inverse matrix for every primitive in the model. We can do this without actually inverting any matrices, using the identity:

$$B^{-1}A^{-1}AB = I, \quad \text{the identity matrix} \tag{3}$$

This tells us that the inverse of AB is $B^{-1}A^{-1}$. If we store each matrix and its inverse with each definition in the hierarchy, we can accumulate the inverse matrices in the same way and at the same time as the original. Whenever we describe one of the basic transformations, the inverse matrix is easy to find. If, for example, a matrix rotates through an angle θ. Its inverse rotates through $-\theta$.

This process of finding all the inverse matrices is very efficient. The cost is the time to construct the inverse of each basic transformation plus one extra matrix multiplication for every node in the traversal. When we traverse the graph using (2) we are, in effect, generating a tree. We are making actual copies of the shared nodes to produce every primitive in our final model. The primitives form the leaves of the tree. Now every tree with n leaves has fewer than $2n$ nodes. So instead of inverting n matrices, we perform fewer than $2n$ extra matrix multiplications.

2 CSG Graphs

Implicit representation means that we have a procedure that tells us whether a given point is inside an object or not. Such a procedure, represents the solid; not just the bounding surface, but every point within the solid too.

Since our solid objects are definitions of sets of points, it seems reasonable that we should be able to perform the operations of set theory on them. This means that we can create holes in objects by subtracting other objects from them. Fig. 2 shows how we can build a mug from elementary cylinders, a block and a torus.

The data structure used to represent this kind of model is a directed graph. Like the structure of Fig. 1, each node of the graph contains matrices of transformation. It also contains information to say how the sub-graphs are to be combined. In Fig. 2, each node specifies addition or subtraction but other operations like set intersection can also be used.

The whole graph is an implicit model. It is possible to traverse the graph and determine whether a given point lies inside the object defined. We can make a picture of the model by ray tracing the graph directly.

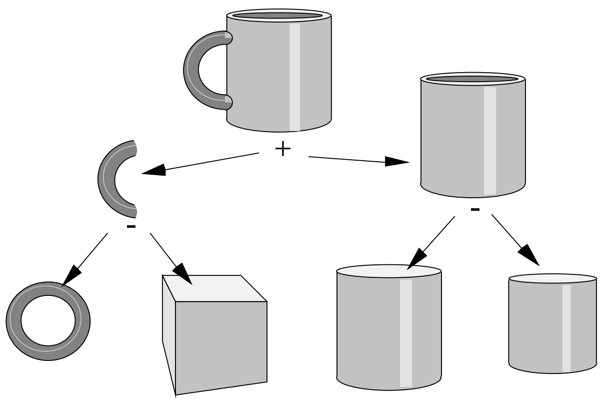


Figure 2: CSG operations.

2.1 Half spaces

We tend to think of a block or a cylinder with closed ends as a fundamental shape suitable to be labeled *primitive*. But even these are not ideal as the most fundamental shapes. For example, we can define a cylinder,

$$x^2 + y^2 \leq r^2 \tag{4}$$

This cylinder is infinite in extent along the z-axis. It is just as convenient a primitive as the finite cylinder and the procedures needed to represent it are simpler because it has such a simple, mathematical definition. Similarly, we can define a *plane* primitive,

$$z \leq 0 \tag{5}$$

The cylinder with closed ends can be built from the infinite cylinder by subtracting two planes. So, given that the mechanism for subtraction is in the system anyway, these pure mathematical primitives are sufficient.

In some systems, the idea of a primitive block and closed cylinder is preserved because it is more intuitive than handling infinite objects. The word *primitive* is then used to describe standard objects available to the user and the term *half-space* is used to describe the true primitives. The term half-space reflects their nature. They divide all of space into two regions: inside and outside.

CSG is a very attractive alternative to polygon or patch modeling because the models are truly three dimensional. Volume calculations and interference tests are also possible with this structure. In the next section we show how to extract high quality images by ray tracing.

3 Ray tracing

What is the simplest way to produce an image of high quality from a computer model? The answer is "use a ray tracer." We didn't use ray tracing in the early years of computer graphics because the computers were too slow and most of us couldn't afford the expensive color displays anyway. Because of this, ray tracing tends to get treated as an *advanced technique*. Nothing could be further from the truth. A ray tracer is the simplest rendering program.

3.1 Just rays

The basic principle is shown in Fig. 3. We choose an eye point and erect a view plane in the space of the 3D model. This viewplane represents our physical screen so we set up a grid on it dividing it into little squares, one for each pixel on our intended output device. To make a picture we have to color each pixel. We find the position for that pixel in the grid and generate a ray from the eye, through the pixel position and into the 3D scene.

The rays, of course, represent light rays traveling backwards. We trace them backwards because it would be difficult to predict which rays leaving a light source would eventually find the eye via a particular pixel. We follow the ray until it hits something and then try to work out what color the hit point is.

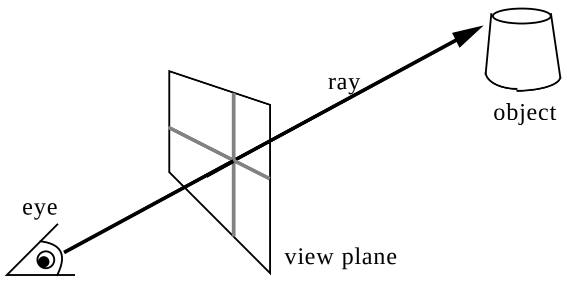


Figure 3: Principle of ray tracing.

To find the color we apply a lighting model. This determines the brightness of the suface in terms of the material properties and the incident light. Additional rays are cast at each light source to see if the point is in shadow and more rays can be generated to produce reflections and other optical effects. To write a ray tracer, we must be able to determine the points of intersection of rays with the primitive objects and also find the surface normals at the points of intersection.

3.2 Normals and smoothing

One way to ray trace CSG trees and other implicit models is first to convert to a polygon mesh. But there are some very good reasons for not wanting to do this. An approximate model of a sphere needs at least 1000 triangles before it looks anything like round and a simple model of a human face needs over 15000. One popular way to improve the appearance of a polygon model is to smooth the shading across each facet. To do this, we usually store a false surface normal at each vertex. These vertex normals are then interpolated across the polygon to create the illusion of a smoothly curving surface. When a ray intersects the polygon the illumination is calculated using this interpolated normal. This works quite well most of the time but it leads to some odd effects, particularly where the light meets the surface at a low angle.

In Fig. 4, the smoothed normal actually points away from the light source. This means that careless application of Lambertian or Phong shading can produce negative illumination causing

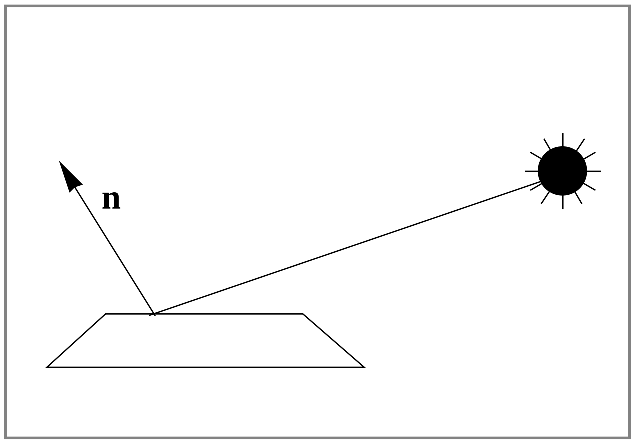


Figure 4: Smooth normal points away from light.

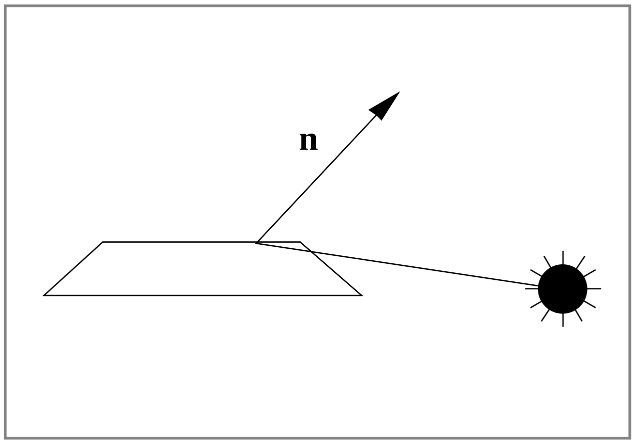


Figure 5: Surface is in shadow even though normal suggests otherwise.

colors to overflow. In Fig. 5, the surface shadows itself. The point of interest is not illuminated although the curved surface suggested by the normal would be. Some systems attempt to fix these problems by using the true normal when the interpolated one gives trouble. But there is no proper fix. The surface shape and normals are inconsistent and will give problems at some angles. Reflected rays can also be created that enter the surface they are reflected from and refracted rays may not lie inside the surface that they should have passed through. The commonest solution is to adjust the position of lights or objects to eliminate the "bug".

3.3 Intersection tests

In a CSG system, we can find the intersection of a ray with a primitive object, directly. This does mean that each type of primitive has to have its own intersection routine, but there are not so many primitive types and the routines are straightforward. Here, for example, we deal with the sphere.

For all intersection tests we represent the ray in vector form as:

$$\mathbf{p} = \mathbf{u} + \mathbf{v}t \tag{6}$$

u is the starting point and **v** is the direction of the ray. The variable t indicates how far down the ray the point **p** is.

The equation of a sphere of unit radius centered at the origin of coordinates is:

$$x^2 + y^2 + z^2 = 1 \tag{7}$$

We can also express this in vector form:

$$\mathbf{p}^2 = 1 \tag{8}$$

Where the ray meets the sphere $\mathbf{p} = \mathbf{u} + \mathbf{v}t$ satisfies (8) and we have:

$$(\mathbf{u} + \mathbf{v}t)^2 = 1 \tag{9}$$

or

$$\mathbf{u}^2 + 2\mathbf{u}.\mathbf{v}t + \mathbf{v}^2t^2 = 1 \tag{10}$$

The dot products, $\mathbf{u}^2$, $\mathbf{u}.\mathbf{v}$, $\mathbf{v}^2$ are all scalars so (10) is an ordinary quadratic equation and we can solve for t.

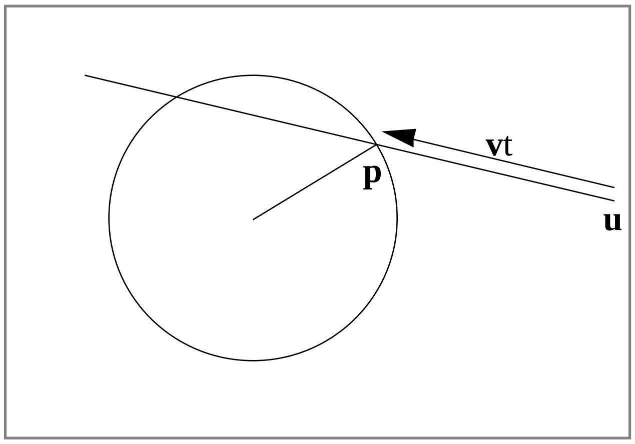


Figure 6: Intersection with sphere.

3.4 Intersections with transformed objects

The transformation matrices are capable of expressing rotation, stretching and shifting operations. How then do we make an intersection test with a shifted, stretched and rotated sphere? By far, the simplest way is to apply an inverse transformation to the ray and then perform the calculation in the original space. Suppose we start with a unit sphere at the origin as in section 3.3. If the sphere is a primitive at the bottom level of a hierarchy, its final position rotation and stretch will all be described by some matrix, S which is made by multiplying, in order, the matrices encountered as the hierarchy is traversed. The matrix, S, describes the operation of transforming an arbitrary point from the 'primitive' space in which the sphere has been defined into the 'world' space in which the ray tracing takes place. The inverse matrix S^{-1} transforms points in the world space into points in primitive space.

First find

$$\mathbf{u_p} = \mathbf{S^{-1}u} \textbf{ and } \mathbf{v_p} = \mathbf{S^{-1}v} \tag{11}$$

Then find t_1, t_2 as described in section3.3 using $\mathbf{u}_p + \mathbf{v}_p t$ as the ray. The values of t_1, t_2 are the same in world space as in primitive space, so the intersection points can then be calculated as before, directly in world space.

Calculating the surface normal is a little more tricky because although the transformations preserve straight lines and planes, they do not preserve angles. The equation of a plane passing through the origin can be written:

$$\mathbf{n}.\mathbf{p} = 0 \tag{12}$$

where $\mathbf{n}$ is the normal to the plane. In matrix form:

$$\mathbf{n}^t\mathbf{p} = 0 \tag{13}$$

Suppose this is an equation in primitive space. Let $\mathbf{q}$ be the point corresponding to $\mathbf{p}$ in world space.

$$\mathbf{q} = S\mathbf{p} \text{ and } \mathbf{p} = S^{-1}\mathbf{q} \tag{14}$$

so

$$\mathbf{n}^t S^{-1}\mathbf{q} = 0 \tag{15}$$

This describes a plane in world space whose normal vector is $\mathbf{n}^t S^{-1}$ but the normal of the transformed plane is the correct transformation of the normal, $\mathbf{n}_w$.

$$\mathbf{n}_w = \mathbf{n}^t S^{-1} = S^{-1t}\mathbf{n} \tag{16}$$

Thus the normal is put into world space using the transpose of the inverse matrix.

This result makes it easy to ray trace any transformation of a model for which we can ray trace the original. Fig. 7 shows a collection of ladders arranged in a logarithmic spiral. Each ladder is a different transformation of a single original ladder. Indeed each ladder consists of differently transformed cylinders. The ray tracer that produced this image works exactly as described above. The cylinder routine works only for a standard cylinder in its primitive space.

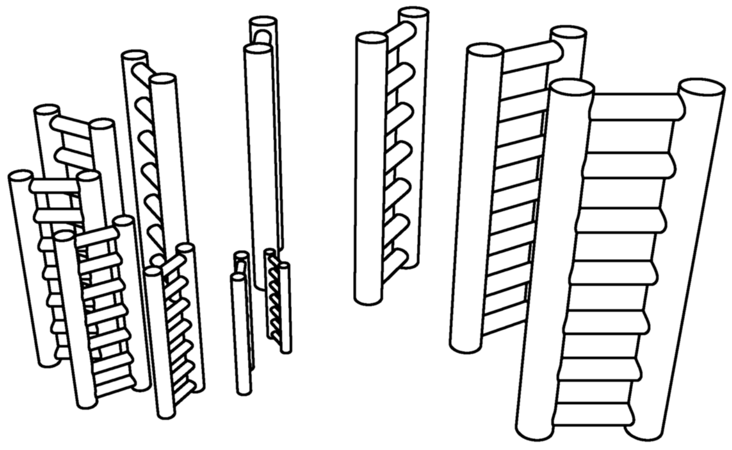


Figure 7: Hierarchical example: Ladders.

3.5 Ray tracing CSG models

In principle, we have to find the ray intersections with every primitive in the CSG model. They are ordered according to their distance from the eye. How can we use the CSG tree to determine the correct intersection point? This was first described by Roth [2] but the method described here is taken from [1].

This is illustrated in Fig. 8. The ray from **p** to **q** finds intersections with the various primitives at **a**, **b**, **c**, **d**, **e**. The CSG graph is shown on the right. Which is the correct intersection point? Points **a** and **b** are inside the subtracted plane primitive and **c** is inside the smaller, subtracted sphere. Point **d** is where the ray enters the solid and **e** where it leaves. The nearest valid intersection is, therefore, **d**. To determine this, we must traverse the CSG structure applying a set of logical rules at each node. The rules are easy to understand. They are applied recursively so if they work for primitives and they handle the joining of sub-graphs correctly, they will work in all cases.

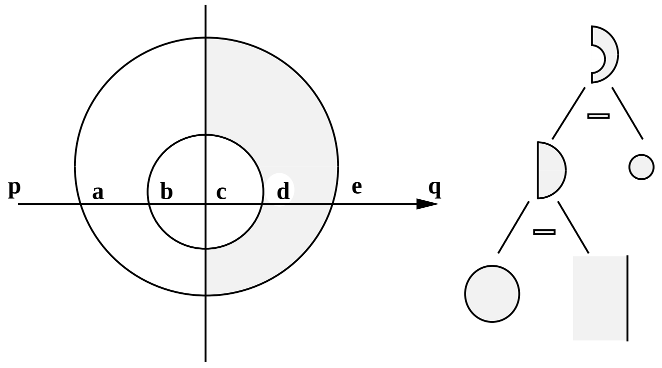


Figure 8: Unraveling the CSG structure.

PLUS Node

	Right branch		
Left branch	IN	OUT	BORDER
IN	IN	IN	IN
OUT	IN	OUT	BORDER
BORDER	IN	BORDER	

MINUS Node

	Right branch		
Left branch	IN	OUT	BORDER
IN	OUT	IN	BORDER
OUT	OUT	OUT	OUT
BORDER	OUT	BORDER	

Figure 9: Combining rules.

The combining rules expressed in Fig. 9, work in a system of three value logic. Each primitive has associated with it an inside/outside test which tells us whether a point is inside its volume or not. The special value, border, is not returned by any of these routines. It is given only to the particular instance of the primitive with which a particular intersection test has been done.

The algorithm is simple. If the current node represents a primitive, then apply the primitive test. Otherwise get the values, recursively, from the sub-nodes and combine according to the table. Here we deal only with "+" and "-" nodes, but any set operation is possible.

3.6 Implicit skeletal models

Implicit skeletal models can be included as primitives in a CSG system. Here we look at the simplest kind made from a collection of key points where each has a surrounding field of influence expressed by a cubic function of distance from the key point, $C(r)$. We assume that the value of $C(r)$ and its derivative vanish at a distance $r = R$, the radius of influence:

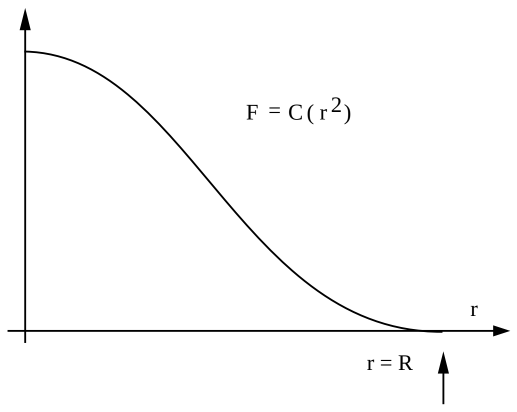


Figure 10: A suitable soft field function.

Suppose $\mathbf{p} = \mathbf{u} + t\mathbf{v}$ is a point on a ray. Then the line from $\mathbf{p}$ to a given key point, $\mathbf{k}$ is

$$\mathbf{r} = \mathbf{u} + t\mathbf{v} - \mathbf{k} \tag{17}$$

and the field at p is given by:

$$F = C(\mathbf{r}.\mathbf{r}) \tag{18}$$

This is easily seen to be a sixth order polynomial in t, Fig. 11.

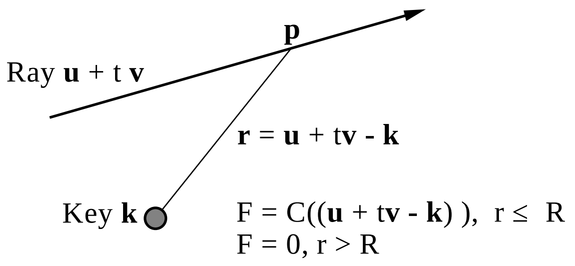

p
Ray **u** + t **v**
r = **u** + t**v** - **k**
Key **k**
F = C((**u** + t**v** - **k**)), r ≤ R
F = 0, r > R

Figure 11: Field along a ray.

The field due to each individual key point has no influence beyond the distance, R, so the field at any point on the ray is the sum of these sixth order polynomials taken for each key point that is close enough. For each key point, we first find the intersections of the ray with a sphere of radius R that surrounds the key point. Between these two intersection points, the key point is active and we must include its effect in constructing our polynomial. Elsewhere the key has no influence and we can ignore it. Fig. 12 shows a ray passing three key points with areas of influence A, B and C. There are five separate regions where, in turn, the polynomial is affected by A, A and C, C, C and B, then B. We solve the polynomial in each region.

Fig. 13 shows the distinction between CSG subtraction and the subtraction of field values. On the left is a simple dumbell shape made from two key points. In the center, a raised copy of this

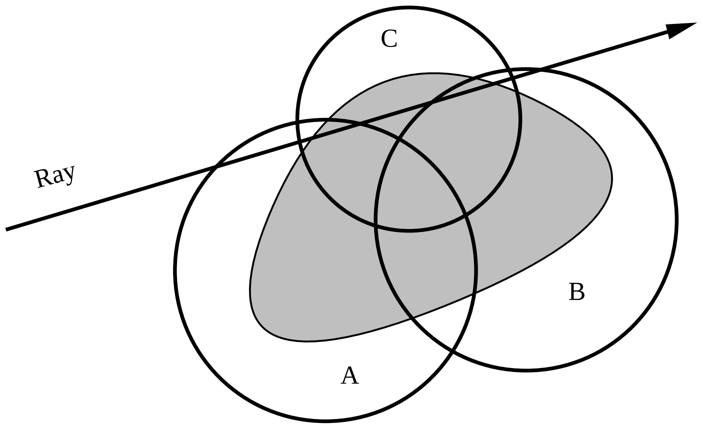


Figure 12: Regions of influence.

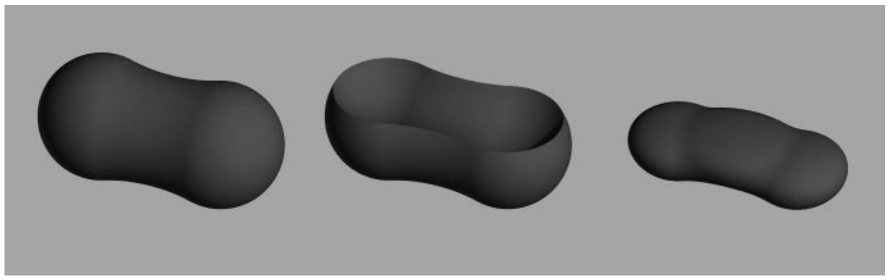


Figure 13: CSG subtraction and field subtraction.

shape has been subtracted from the original (CSG). The right hand picture uses the same four key points, but the upper two have negative fields, creating a single blended object.

4 Efficiency of ray tracing

Making pictures of high quality is inherently a slow process, and ray tracing has acquired a reputation for being an expensive solution. Yet the comparisons made with other methods are not usually very fair. Scan-line algorithms do not produce shadows or reflections, so they don't even do the same job. If you have a very simple model, then ray tracing is likely to be slow but, if your algorithms are good, a ray tracer can prove faster than other methods when a large number of objects is present.

4.1 Space division

The basic key to fast ray tracing is that you must not test every object against every ray. Some form of sorting is needed so that objects that are well away from the path of a ray do not require intersection tests.

One method of doing this is to divide the space of the model into a 3D grid of cubes and follow the ray through the grid. If the amount of work done in following the ray is less than for the intersection tests avoided, we get a speed up. Fig. 14 illustrates this effect dramatically. The ray shown, gets intersected with only two of the objects. This is because the ray tracer only checks for intersection with objects in the current space division cell. The line of the ray passes many other objects but it doesn't enter a cell where these are present until after the intersection has been found.

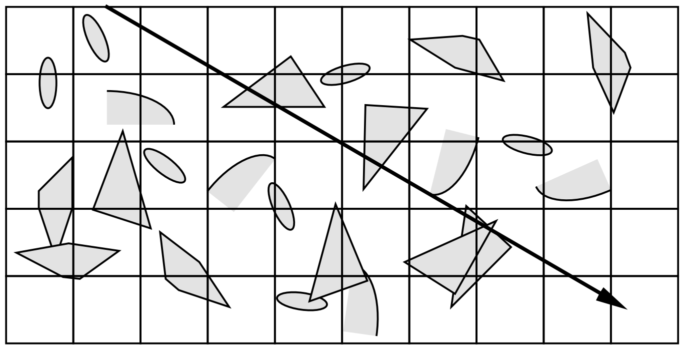


Figure 14: Space division.

To skip cells quickly requires a little cunning. Fig. 15 shows the geometry. A ray starts from $\mathbf{p}$ and makes an angle θ with the x-axis. The distances dx and dy, measured along the ray tell us how far away is the next intersection with the y-axis and x-axis. If $dx < dy$ then our next move into a new cell will be in the x-direction. As soon as we make this move, we can find a new value for dx:

$$dx := dx + \frac{s}{cos(\theta)} \tag{19}$$

where s is the size of the grid cell. This means that we can again compare dx with dy to find whether our next move is horizontal or vertical. The important thing to notice is that $\frac{s}{cos(\theta)}$ is a constant calculated only once for each ray.

This idea readily transfers to the 3D case where we define dx, dy, dz in the same way. Measured along the ray, dz is the distance from $\mathbf{p}$ to the next intersection with the x/y-plane. With a few refinements, we can turn this into an algorithm that uses only half a dozen integer operations to identify the next cell [1].

We set up a data structure that represents a 3D array of cells where each array entry contains a list of objects that need to be tested in that cell. With the very high speed of the cell-skipping algorithm it is possible to use a fine grid, say $100 \times 100 \times 100$ cells. This can reduce the average number of intersection tests per ray to two or three even if there are thousands of objects.

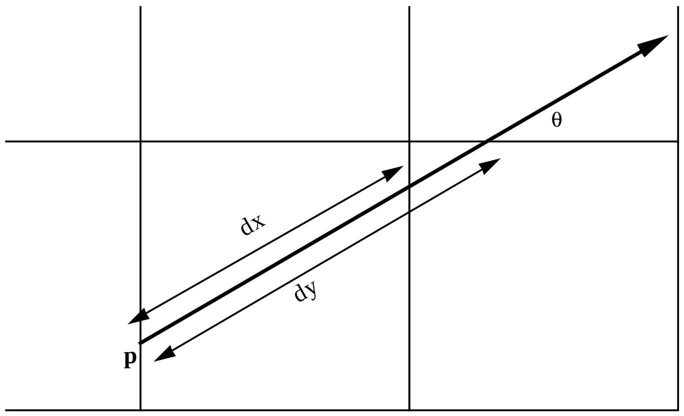


Figure 15: Cell skipping algorithm.

4.2 Creating the space division

Separating objects into a grid is straightforward for explicit objects like triangles, but how do you divide up a scene described by a single CSG tree? The single CSG tree can be replaced by a set of reduced CSG trees, RCSGs, each of which describes only the subset of primitives present in each cell.

There are many ways to do this, here we describe a simple, recursive algorithm based on the idea of a octree. In our octrees each node represents a volume of space that:

- is full,
- is empty,
- contains one primitive surface,
- contains an RCSG or
- is divided into eight sub-volumes.

In Fig. 16, cell 4 contains an RCSG, cells 5 and 6 contain single cylinders and the other cells are empty.

The algorithm works as follows:

1. If the CSG tree is a single primitive then do a primitive test to see if this volume is full, empty or contains a surface. If it contains a surface then result is the single primitive.
2. Otherwise, recursively construct the RCSGs, left and right, for the left and right branches of the CSG tree.
3. If right is empty then the result is left.

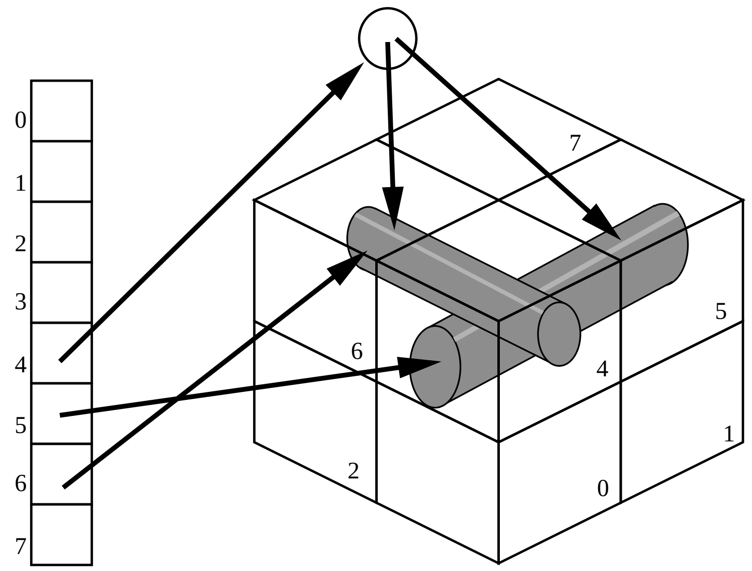


Figure 16: Octree structure.

4. If the CSG operation at this node is "+" and left is empty, the result is right.

5. If the CSG operation at this node is "-" and right is full, the result is empty.

6. If the current cell volume is equal to the smallest allowed then combine right and left to make an RCSG.

7. Otherwise divide the volume into eight sub-volumes and find the result in each recursively.

This algorithm is described in detail in [3].

4.3 Cylinder voxel test

The algorithm, above depends on having a test for each primitive to determine if a given cell is full, empty or contains part of the surface. As an example, we outline this calculation for the cylinder primitive.

As we traverse the CSG graph, we accumulate the matrices of transformation in exactly the same way as in other hierarchies. Ultimately, we will be dealing with a stretched, rotated and shifted cylinder whose defining equation is rather complicated. But we can apply the inverse matrix to the vertices of the cube rather than cope with a transformed cylinder.

The cube is then out of shape but it is still a six-faced polyhedron with parallel faces. Project this polyhedron and the cylinder onto the x/y-plane. The cylinder becomes a circle and the polyhedron faces become parallelograms. We can distinguish five cases and test for each in turn (Fig. 17).

1. If all eight vertices of the polyhedron are inside the cylinder then the cube is full.

2. If some are inside the cylinder and some are outside, the state of the cube is border because some edges must cross the surface.

3. When all the vertices are outside, even if there is no intersection of an edge with the circle, the cylinder can still pass through one of the parallelogram faces. In this case, the center of

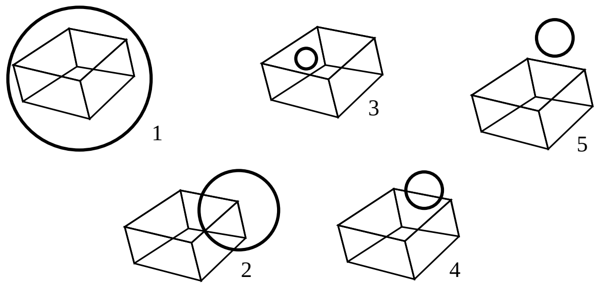


Figure 17: Five cases of polyhedron and cylinder.

the cylinder passes through the face. So we perform an inside/outside test. If its center is inside any face the result is border.

4. All vertices are outside but the circle is intersected by one or more edges. Result is border.

5. If none of the preceding four tests has established a result full or border, the result is empty.

4.4 Signatures and mailboxes

A minor problem with space division schemes is shown in Fig. 18 The rays shown may be tested for intersection with the same primitive more than once. To avoid this we count the rays and label every new ray with its number. This number can then be used as a unique signature for that ray. When an intersection test is performed, the ray signature is stored in a 'mailbox' location associated with the CSG node for that primitive. Before an intersection test is performed the mailbox is checked to see if the value matches the ray signature. If it matches, then there is no need to repeat the test.

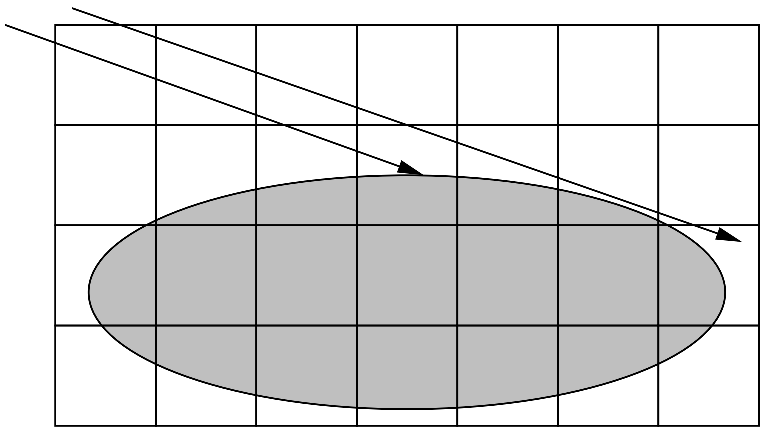


Figure 18: Multiple intersection tests.

Unfortunately this doesn't work too well with the implicit skeletal models. Each blended object needs a different mailbox for each subset of keypoints, Fig. 12 So far, no one has suggested a good solution to this problem and this slows down the ray tracing of these models quite a bit.

4.5 Examples

Fig. 19 shows a full combination of blended and normal CSG objects. The X-wing, Fig. 20, shows the high level of detail that can be achieved with CSG. The model has just 513 primitives.

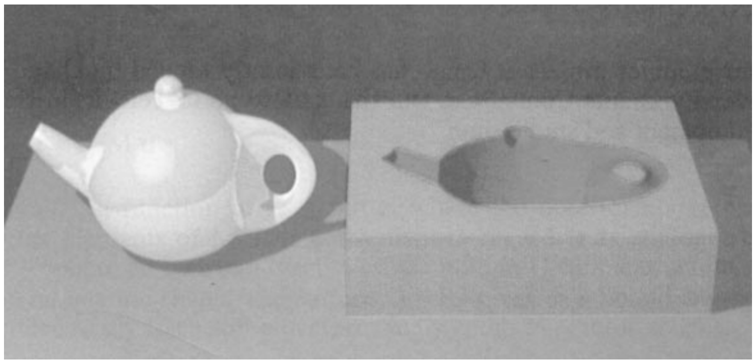


Figure 19: Blended teapot and CSG mold.

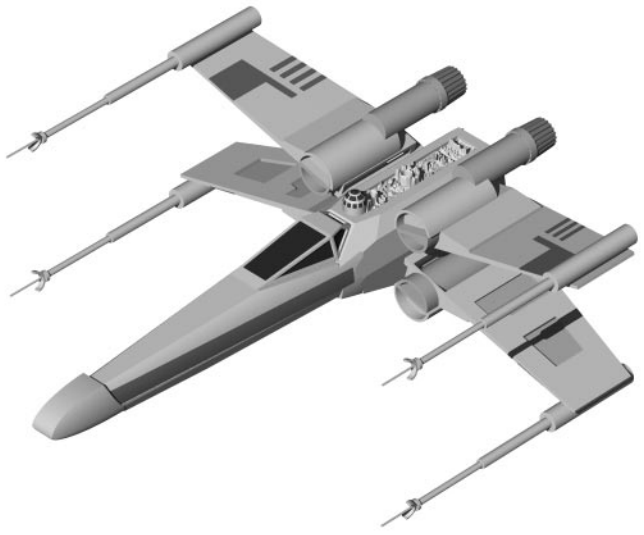


Figure 20: X-wing model, 513 CSG primitives.

5 Acknowledgements

I wish to thank George Sealy for the X-wing model. X-wing is a registered trademark of Lucasfilm Ltd.

References

[1] J. Cleary and G. Wyvill. Analysis of an algorithm for fast ray tracing using uniform space subdivision. *The Visual Computer*, 4(2):65–83, July 1988.

[2] S. Roth. Ray casting for modeling solids. *Computer Graphics and Image Processing*, 18:109–144, 1982.

[3] G. Wyvill, T. Kunii, and Y. Shirai. Space division for ray tracing in csg. *IEEE Computer Graphics and applications*, 6(4):28–34, April 1986.

Texturing Implicit Models

Geoff Wyvill
University of Otago

Abstract

Texture mapping enables us to represent surface detail cheaply. But the very power of implicit surface modeling creates extra problems. How can we find texture coordinates on a deformable object whose topology can change? How do we find the correct color of a surface produced by subtraction? How do we take advantage of the implicit form to allow texturing to support the modeling process?

1 Introduction

The fundamental notion behind texture mapping is that we can separate the description of an object into two parts. The first describes the major aspects of the shape and the second describes a refinement in terms of surface detail.

The reasons for the success of this separation are variously physical, perceptual and technical. The physical justification is that many objects are literally constructed in this way. A painted object has its surface appearance uniformly modified by the layer of paint; an otherwise flat water surface is disturbed by surface ripples and so on. The perceptual aspect is connected with the way in which we deduce surface detail from the visual appearance. At a distance, a slightly wavy or bumpy surface is visible only because of the perceived color or shade differences produced by the bumps. The technical reason is that we already have efficient ways to model and render objects of smooth, simple shapes and texture mapping allows us to extend this range of objects at low computational cost.

Because CSG systems use sets of points in 3D space, the representation of objects is truly volumetric. In principle, CSG gives us a full representation of a shape and not just its bounding surfaces.

But this completeness of description is of no use unless it is exploited to provide information not available from other kinds of model. We are particularly interested in cases where an object, made of several pieces glued together, is cut or shaped so that the inner structure is revealed at the surface of the cut. Fig. 1 shows such a model. It has been made by turning a layered wooden billet. The layers are made by boring holes with a slight taper and filling them with accurately fitted plugs made of a contrasting wood. How do we represent models like this with CSG?

Most of the material in this chapter has been taken from [6] and [7], where a little more detail can be found.

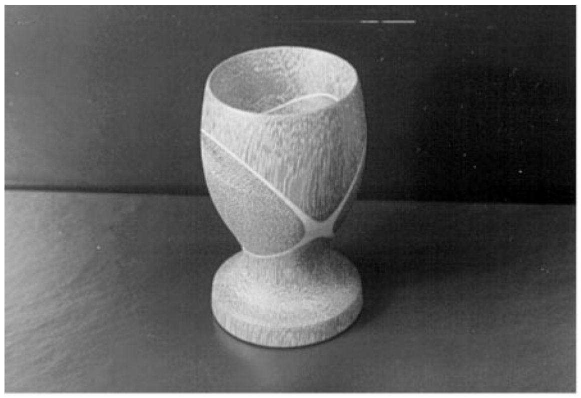


Figure 1: The wooden eggcup.

2 Concept of texture mapping

The description of surface detail is called a texture map and it can be represented by mathematical functions or tables. In the process of texture mapping we must relate each visible point on the surface of an object to a point in the texture map. A "point" for this purpose, will correspond to a pixel in the final image. The general method of doing this requires that we set up a co-ordinate system on the object's surface. The method used to construct such a co-ordinate system depends on the way the object is defined. If parametric patches have been used then each surface element is already defined as a function of some s, t and the same s, t make convenient coordinates for texture mapping. If the surface is defined by polygons, the simplest approach is to divide the polygons into rectangles or triangles and then treat these as a special case of patches. Continuity at patch borders is achieved by careful definition of the texture map, using complementary coordinates $(1-s, 1-t)$ in adjacent patches and other methods.

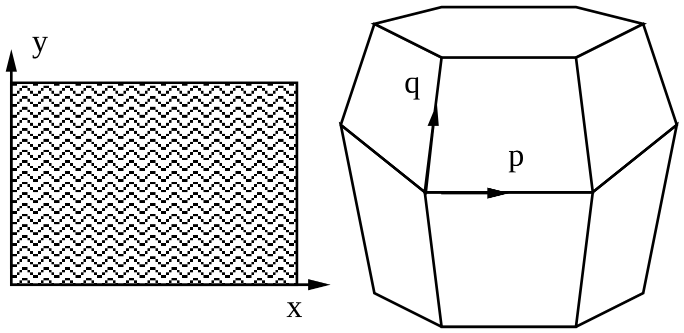


Figure 2: Texture mapping.

We can map a variety of information onto a surface. If, for example, we use our texture to modify the surface normals, we produce the illusion of a bumpy surface [1].

Peachey [4] introduced the idea of using 3D coordinates of suface points for texture mapping. This is equivalent to carving the object out of a non-uniform substance represented by the 3D texture map. This is an excellent approach for modeling materials like wood and marble where the surface is indeed the result of carving from a material patterned in 3D. Where this approach can be used, there are advantages over the 2D method. Consistency of texture is achieved regardless of the topology of the object. For most rendering techniques, the 3D coordinate values which correspond

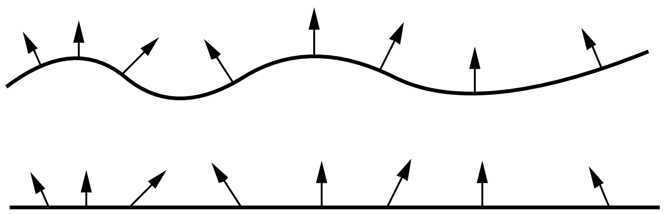


Figure 3: Bump mapping.

to a particular screen pixel have to be calculated anyway, so the coordinates for texture mapping are available at no extra computational cost.

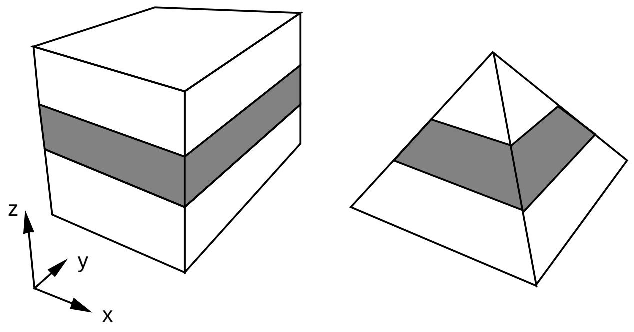


Figure 4: Solid texturing.

There are, however, cases where 3D texture mapping is unsuitable: surface marks on machined objects, patterns on textiles and applied patterns like paintwork. In these examples the texture of the object we are modeling is not a surface manifestation of a three dimensional structure so we should not expect to be able to model it that way. Implicit skeletal models do not lend themselves to solid texturing for different reasons, explained below.

An excellent survey of texture mapping can be found in Heckbert [2].

3 Texturing implicit skeletal models

Implicit skeletal models present a special problem for texture mapping. If we use a conventional, two-dimensional map, we have to provide some sort of co-ordinate system on the surface of our object. But this surface not only changes shape, it can change topologically. A torus can change (nearly) smoothly into a sphere and as the hole closes up, we may not want an obvious discontinuity in the texture.

It would seem that our only choice is to use a three-dimensional (solid) texture, but this, too, is unsatisfactory for the following reason. Suppose we use a three-dimensional texture map which is fixed in world space, then the texture of an object moving through space will change as it moves, Fig. 5. One way to avoid this, is to tie the texture space to the object. That is, we use a co-ordinate system in which the object is defined to be stationary and define our texture map there. During the rendering process, we must translate the coordinates of each surface component back into the object's system before we look up the texture. This is a satisfactory solution for rigid objects, but it fails in many cases for deformable objects.

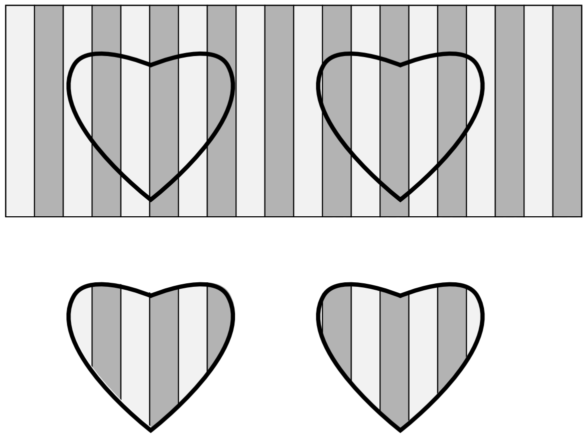


Figure 5: Object moving through texture space.

Figure 6 shows such a case. A large droplet splits into two smaller ones. If we regard the two droplets as a single object, then the origin of the texture space will be at the center of gravity of the large droplet. In the final frame, it will be mid-way between two separate objects traveling in opposite directions. Each of these objects will be traveling through a texture space just as if it were tied to the world coordinates.

Similar problems arise with rotation. If an object rotates without distortion, then the textures on its surface should not change. This implies that the solid texture space must rotate with the object. But with our skeletal models we do not refer to objects as such. We control the position of key points and allow the surface topology to change as it will. It seems that whatever method we choose, we can find an example where we do not get the effect we are seeking.

We need a method of setting up a texture map which refers only to key points and never to objects. Each key point, therefore, is embedded in its own abstract texture space. This space is used only as a device for assigning a value to every part of the surface. It need have no geometrical significance. There are many ways in which we can set up this abstract texture space, so let us first demonstrate it by means of a simple example. Suppose a key point is considered to be at the origin of an ordinary co-ordinate system. Then $< x, y, z >$ is the name of a point in space defined

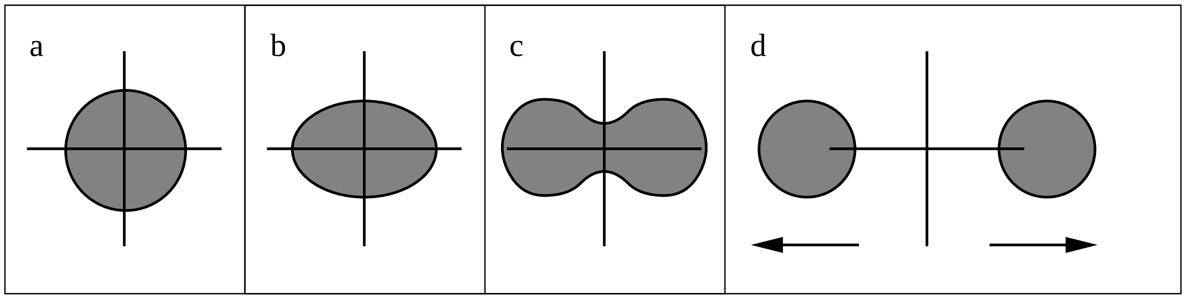


Figure 6: Nowhere to put the texture space.

with respect to this key point. In creating the coordinate system, we have also given the key point a new property. It is no longer symmetrical. We can assign properties to the space surrounding it and we have given meaning to the idea that the point itself can rotate: Rotation of a key point implies rotation of these local coordinates. We define the abstract texture space as follows:

$$f = F(x, y, z) \tag{1}$$

$$h = H(x, y, z) \tag{2}$$

$$c = C(x, y, z) \tag{3}$$

$< f, h, c >$ is the name of a point in abstract texture space and the functions F, H, C can be chosen at will, depending on the application. In most of what follows, we simply set $f = x$, $h = y$, $c = z$ so that $< f, h, c >$ can be regarded as a point in space in a system whose origin is a particular key point.

For our purpose, we need to assign an $< f, h, c >$ triplet to every point P, on the surface of a soft object. We do this by means of a simple, weighted sum. Assume each key point i, contributes an amount Qi to the field at P. If $< f_i, h_i, c_i >$ are the values of the abstract coordinates of P in the system of i, then:

$$f = \frac{\Sigma f_i Q_i}{\Sigma Q_i} \tag{4}$$

$$h = \frac{\Sigma h_i Q_i}{\Sigma Q_i} \tag{5}$$

$$c = \frac{\Sigma c_i Q_i}{\Sigma Q_i} \tag{6}$$

Of course, ΣQ_i is equal to the field value at the surface.

Once we have defined $< f, h, c >$ for a point on the surface, we can look up values in tables, or apply functions to describe the texture and color at that point. Our textures become functions of the three dimensional f-h-c space and we can perform solid texturing just as we would for a conventional object in world space.

Fig. 7 shows the classical case of a sphere breaking into two droplets. The texture is a 3D bump map. During animation, this texture changes smoothly as the droplets merge. In this case, the f, h, c coordinates are the local coordinates of the two points. Note that the two spheres are identically textured after they separate.

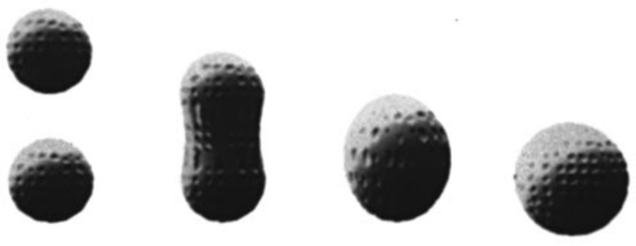


Figure 7: Textured droplets.

Fig. 8 shows stages in animation as a sphere changes into an object with a hole in it. The texture is also changing from 'tartan' to radial stripes.

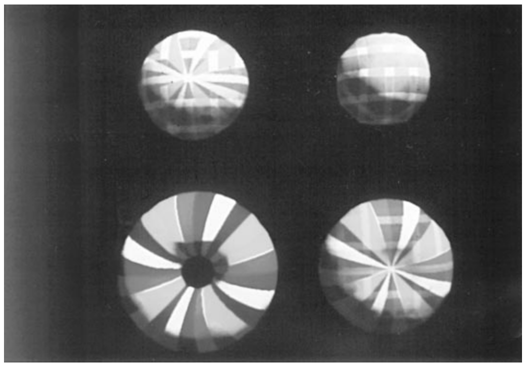


Figure 8: Sphere to torus inbetweening.

The way the texture behaves in animation, depends on what we suppose the motion means. Fig. 9 shows two ways to rotate a complicated object. The original object is on the left. In the center, the object is shown rotated while the abstract space attached to each key point maintains its orientation. The object on the right has been rotated by the same amount, but this time the abstract space coordinates have been rotated as well. Because the object has not changed its shape, we see this right hand object as the correct one. If we want to regard the motion of a collection of key points as a rotation, then we must rotate the $< f, h, c >$ space. Otherwise the texture behaves as if the object has merely transformed to the new shape by distortion. There is nothing wrong with the central object in Fig. 9. The animation represents a blob of soft material whose surface is changing (perhaps because of some internal flow of material). Only if we want to regard the motion as a rotation without distortion need we deliberately rotate the abstract texture space.

We can, indeed, represent objects which both rotate and distort. In doing so, we make a conscious choice about how to handle our abstract space. We are made to specify just how much of the motion of key points is attributable to the rotation. What appeared to be an impasse, turns out to be a need for more detailed specification. The idea of an abstract texture space gives us a simple tool with which to express that specification.

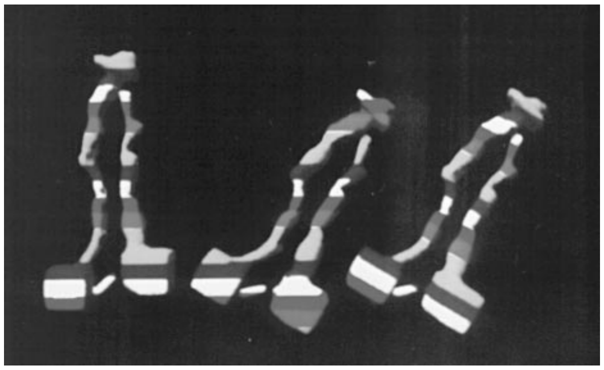


Figure 9: Two ways to rotate a textured object.

3.1 Choice of f-h-c space

Fig. 10 shows the droplets again, textured with stripes. In this case, identical F, H, C functions are used for each of two key points, so when they are sufficiently separated, they appear as two identical spheres. As the key points approach each other, parts of the surface acquire $< f, h, c >$ values interpolated by the weighted sum and the texture changes accordingly. Notice that at no stage does the texture exhibit a sudden break. Even at the moment the two objects become one, the texture retains its continuity. This is because the values of f, h and c change continuously both as functions of time and as functions of position on the surface.

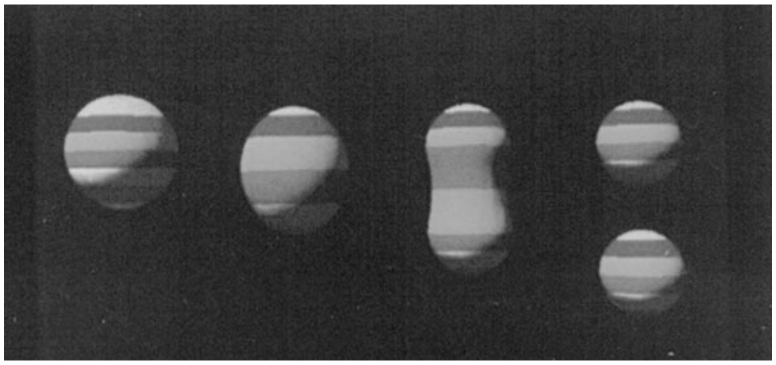


Figure 10: Texture stretching during animation.

There is, of course, distortion. In Fig. 10, the texture is defined as a pattern of colored slices in f-h-c space. Because we are using $F(x) = x$, $H(y) = y$, $C(z) = z$, every $< f, h, c >$ in the region of an isolated key point is equivalent to a simple position in space and the isolated sphere appears to have been carved from a solid mass with this texture. In the region where the droplets begin to merge, $< f, h, c >$ is interpolated between two values. One is characteristic of the top of the lower sphere and the other is characteristic of the bottom of the upper sphere. Thus in a small region of physical space there will be a rapid, continuous change in f-h-c space. This causes the texture to appear stretched.

Such distortion is inevitable when we try to map a texture onto a surface which can stretch and even change its topology, but if we know, in advance, what kind of relative motion of key points to expect, we can take steps to minimise this. For example, we can define:

$$F(x) = x, \quad H(y) = y, \quad C(z) = z \tag{7}$$

as before, for the top key point and:

$$F(x) = x, \quad H(y) = y - R, \quad C(z) = z \tag{8}$$

for the lower point, where R is the radius of influence associated with these key points. The effect of this is that the two spheres appear to have been carved out of different parts of f-h-c space, but when they merge, the parts of the surface most violently affected by the interpolation are not too far apart in f-h-c space. This is illustrated in Fig. 11.

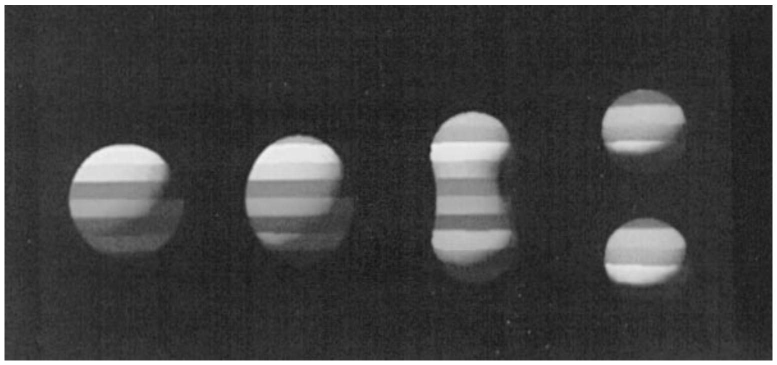


Figure 11: Stretching reduced by careful choice of f-h-c.

4 Color and texture in CSG

In CSG systems, the shape of a surface is very often created by a subtraction. In Fig. 12, two holes have been made in a rectangular block by subtracting cylinders. The surface of the hole on the left is lighter. It has taken its color from the subtracted cylinder, not from the material of the block.

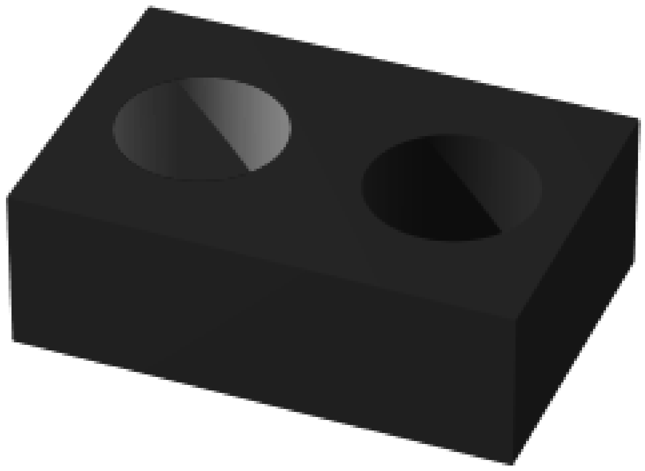


Figure 12: Two ways to drill a hole.

We expect the inside of the hole to have the color of the block because we think of color as a property of the material of the block. If the color is merely painted on, we would expect the inside of the hole to be different. Some properties are logically inherited from surfaces. We could, for example, texture the cylinder to represent surface scratches made by a drill. We would think it logical for the surface to acquire such a texture from the cylinder. In our application, we want to treat color as a property of volumes rather than surfaces.

To get the desired color and texture on a surface we have to build the concepts of volume and surface properties into our data structure. In the CSG tree every leaf node carries a separate entry for surface and volume properties. When the ray tracer finds an intersection it applies the volume properties of the left node and the surface properties from the right (subtracted node).

4.1 Space division

This has an interesting consequence for the space division algorithm [5]. Full cells have to carry a pointer to say of what material they are full. In practice this is a pointer to the leaf node for that primitive.

4.2 Asymmetric addition

Taking account of volume properties introduces another problem. When we add two overlapping objects with different material properties what should be the properties of the common volume? Our solution is to regard the PLUS operator as asymmetrical, Fig. 13. The right hand operand overwrites the left. This convention enables the user to specify exactly which parts of the model have which properties. Using this feature of the CSG system, the eggcup design, Fig. 14 was created. Including glue setting time, the real wooden eggcup took about four days to make. Without the CSG design, there is no obvious way to predict the surface pattern.

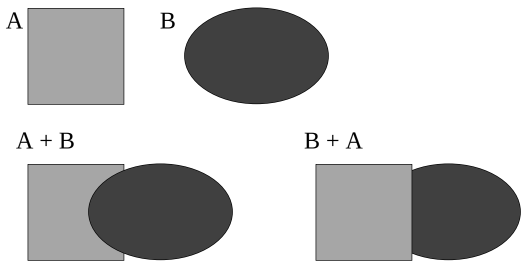


Figure 13: Asymmetric addition operator.

Given the correct inheritance of volume properties, the eggcup design, Fig. 14 is straight forward. We build the billet by adding differently colored cylinders and subtract away the material around the eggcup. The surface design appears automatically.

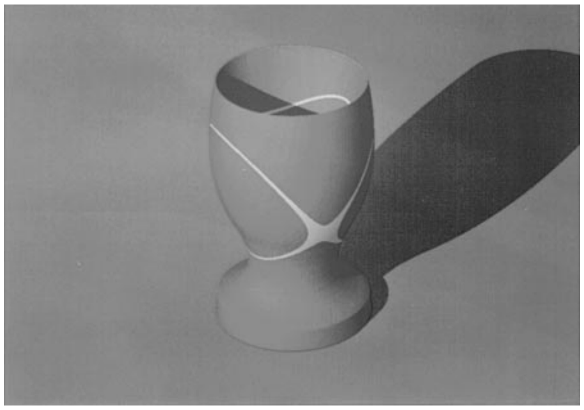


Figure 14: CSG design for the eggcup.

4.3 Using texture to modify shape

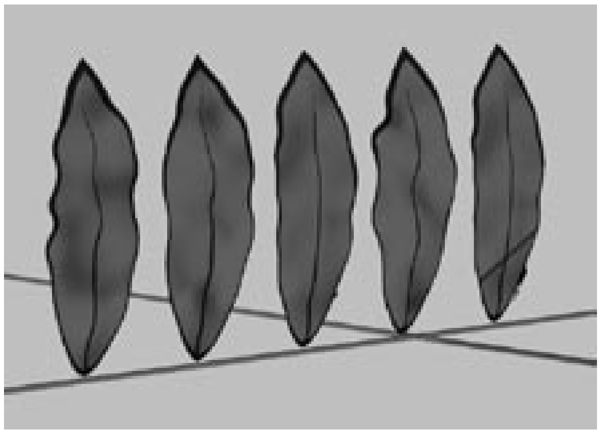


Figure 15: Leaves.

When objects are represented as an iso-surface of a field, we can cause actual change in shape by applying a 3D texture function to modify the field value. This rather complicates the intersection calculations, but there are some special cases where this can be avoided.

The leaves in Fig. 15 were designed for a commercial animation. We were asked to make leaves in a similar style to some exising art work except that ours had to be 3D objects. The basic leaf shape is made by taking the intersection of a bounding ellipsoid with a thin cylindrical surface. The ellipsoid is defined by:

$$ax^2 + by^2 = k \tag{9}$$

where x, y are local coordinates and k is a constant. After finding the intersection with the cylinder, a noise texture function is added to k. The effect of this is to change the shape of the ellipsoid and give the leaf a wavy edge. Of couse this technique could be used with non-implicit models but in the CSG system all the mechanism is there already.

5 Examples

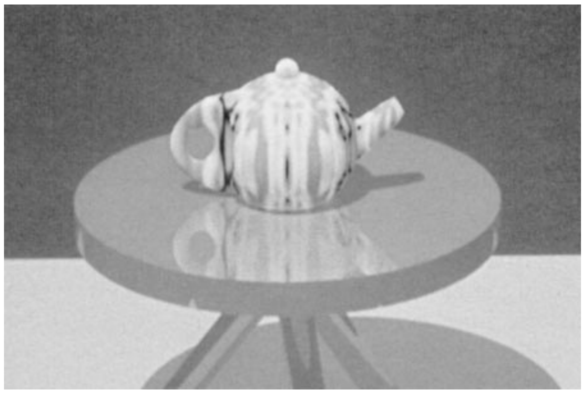


Figure 16: Soft CSG teapot, textured.

For the coral, Fig. 17, the portion of each polyp is taken from the nearest point in a 3D grid. A local coordinate system is set up around this point and the polyp is "grown" to meet its neighbours [3].

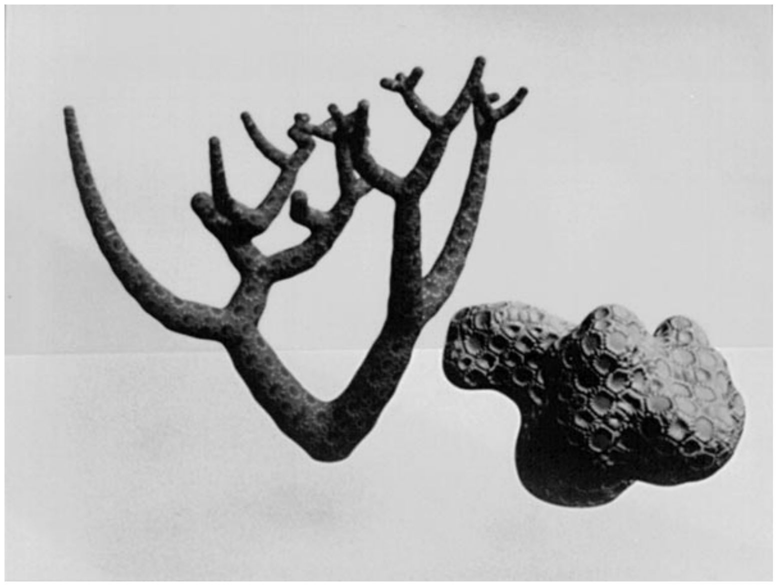


Figure 17: Coral.

6 Acknowledgements

I wish to thank Nelson Max for the coral image, and Andrew Trotman for the teapot image.

References

[1] J. Blinn. Simulation of wrinkled surfaces. *Computer Graphics (Proceedings SIGGRAPH '78)*, 12(3):282–292, 1978.

[2] P. Heckbert. Survey of texture mapping. *IEEE Computer Graphics & Applications*, 6(11):56–67, 1986.

[3] N. L. Max and W. Geoff. Shapes and textures for rendering coral. *Scientific Visualization of Physical Phenomena [Proceedings of CG International '91]*, pages 333–343, 1991.

[4] D. Peachey. Solid texturing of complex surfaces. *Computer Graphics (Proceedings SIGGRAPH '85)*, 19(3):279–286, 1985.

[5] G. Wyvill. Implicit skeletal models and csg. In J. Menon, editor, *Implicit Surfaces for Geometric Modeling and Computer Graphics*, Siggraph '96 Course Notes, chapter C4, pages 13–14. 1996.

[6] G. Wyvill, W. Brian, and M. Craig. Solid texturing of soft objects. *IEEE Computer Graphics and applications*, 7(12):20–26, December 1987.

[7] G. Wyvill and P. Sharp. Volume and surface properties in csg. *New Trends in Computer Graphics: Proceedings of CG International '88*, pages 257–266, 1988.

Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces

To appear: The Visual Computer

John C. Hart
School of EECS
Washington State University
Pullman, WA 99164-2752
(509) 335-2343
(509) 335-3818 (fax)
hart@eecs.wsu.edu

May 12, 1996

Abstract

Sphere tracing is a new technique for rendering implicit surfaces using geometric distance. Distance-based models are common in computer-aided geometric design and in the modeling of articulated figures. Given a function returning the distance to an object, sphere tracing marches along the ray toward its first intersection in steps guaranteed not to penetrate the implicit surface.

Sphere tracing is particularly adept at rendering pathological surfaces. Creased and rough implicit surfaces are defined by functions with discontinuous or undefined derivatives. Current root finding techniques such as L-G surfaces and interval analysis require periodic evaluation of the derivative, and their behavior is dependent on the behavior of the derivative. Sphere tracing requires only a bound on the magnitude of the derivative, robustly avoiding problems where the derivative jumps or vanishes. This robustness and scope support sphere tracing as an efficient direct visualization system for the design and investigation of new implicit models.

Furthermore, sphere tracing efficiently approximates cone tracing, supporting symbolic-prefiltered antialiasing. Signed distance functions for a variety of primitives and operations are derived and appear independently as appendices, specifically the natural quadrics and torus, superquadrics, Bezier-based generalized cylinders and offset surfaces, constructive solid geometry, pseudonorm and Gaussian blends, taper, twist and hypertexture.

Keywords: area sampling, blending, deformation, distance, implicit surface, Lipschitz condition, numerical methods, ray tracing, solid modeling.

1 Introduction

Whereas a parametric surface is defined by a function which, given a tuple of parameters, indicates a corresponding location in space, an implicit surface is defined by a function which, given a point in space, indicates whether the point is inside, on or outside the surface.

The most commonly studied form of implicit surfaces are algebraic surfaces, defined implicitly by a polynomial function. For example, the unit sphere is defined by the second degree algebraic implicit equation

$$x^2 + y^2 + z^2 - 1 = 0 \tag{1}$$

as the locus of coordinates whose hypotenuse (squared) is unity.

Alternatively, using a distance metric, one can represent the unit sphere geometrically by the implicit equation

$$||\boldsymbol{x}|| - 1 = 0 \tag{2}$$

as the locus of points of unit distance from the origin. Here $\boldsymbol{x} = (x, y, z)$ and $||(x, y, z)||$ denotes the Euclidean magnitude $\sqrt{x^2 + y^2 + z^2}$. The implicit surface of (2) agrees with that of (1), though their values differ at almost every other point in $\mathbb{R}^3$. Specifically, (1) returns *algebraic distance* [Rockwood & Owen, 1987] whereas (2) returns *geometric distance.*

A comparison of geometric versus algebraic representations of quadric surfaces preferred the geometric representation [Goldman, 1983]. The parameters of a geometric representation are coordinate-independent, and are more robust and intuitive than algebraic coefficients. Distance-based functions like (2) are one method for representing implicit surfaces geometrically.

Distance-based models can be found in a variety of areas. *Offset* surfaces have become valuable in computer-aided geometric design for their use of distance to model the physical capabilities of machine cutting tools [Barnhill *et al.*, 1992]. *Skeletal* models, which in computer graphics simulate articulated figures such as hands and dinosaurs, are equivalent to offset surfaces. Computer vision's medial-axis transform converts a given shape to its skeletal representation [Ballard & Brown, 1982]. *Generalized cylinders* began as a geometric representation in computer vision [Agin & Binford, 1976] but have also matured into a standard modeling primitive in computer graphics [Bloomenthal, 1989] — special ray tracing algorithms were developed for their rendering in [van Wijk, 1984; Bronsvoort & Klok, 1985].

1.1 Previous Work

Several methods exist for rendering implicit surfaces. Indirect methods polygonize the implicit surface to a given tolerance, allowing the use of existing polygon rendering techniques and hardware for interactive inspection [Wyvill *et al.*, 1986; Bloomenthal, 1988]. Although polygonization transforms implicit surfaces into a representation easily rendered and incorporated into graphics systems, polygonizations are typically not guaranteed and may not accurately detect disconnected or detailed sections of the implicit surface. Production rendering systems tend to polygonize surfaces, resulting in large time and memory overhead to accurately represent an otherwise simple implicit model.

In an effort to combine speed and accuracy, [Sederberg & Zundel, 1989] developed a direct scan-line method to more accurately render algebraic implicit surfaces at interactive speeds. Although slower, ray tracing provides a direct, accurate and elegant method for investigating a much larger variety of implicit surfaces.

Let

$$\boldsymbol{r}(t) = \boldsymbol{r}_o + t\boldsymbol{r}_d \tag{3}$$

parametrically define a ray anchored at $\boldsymbol{r}_o$ in the direction of the unit vector $\boldsymbol{r}_d$. Plugging the ray equation $\boldsymbol{r} : \mathbb{R} \to \mathbb{R}^3$ into the function $f : \mathbb{R}^3 \to \mathbb{R}$ that defines the implicit surface produces the composite real function $F : \mathbb{R} \to \mathbb{R}$ where $F = f \circ \boldsymbol{r}$ such that the solutions to

$$F(t) = 0 \tag{4}$$

correspond to ray intersections with the implicit surface. Implicit surface ray-tracing algorithms simply apply one of the multitude of numerical root finding methods to solve (4).

When $f(\boldsymbol{x}) = 0$ implicitly defines an algebraic surface, (4) is a polynomial equation, and can be solved by DesCartes' rule of signs [Hanrahan, 1983], Sturm sequences [van Wijk, 1984], and Laguerre's method [Wyvill & Trotman, 1990].

Ideally, the root-finding procedure should only need the ability to evaluate the function at any point. However, one can always construct a pathological function that will cause such a "blind" technique to miss one or more roots, by inserting an arbitrarily thin region between samples where the function zips off to zero and back (a point reiterated from [Kalra & Barr, 1989; Von Herzen *et al.*, 1990]). Hence, any robust root finder needs more information than simple function evaluation.

The "Hypertexture" system used a brute-force blind ray-marching scheme, using only function evaluation [Perlin & Hoffert, 1989]. This supported the design of implicit surfaces without regard to the analytic properties of the defining functions. Freed from such constraints, fractal and hairy surfaces were modeled by implicit surfaces whose functions contained procedural elements. The high frequencies produced by these geometric textures required fine sampling along the ray, resulting in a rendering speed so slow as to necessitate parallel implementation.

Guaranted ray intersection requires extra information, which in most cases is produced by the derivative of the function. Interval analysis finds ray intersections by defining the function and its derivative on intervals instead of single values [Mitchell, 1990].

The LG-surfaces method imposed the Lipschitz condition on f to create an efficient octree partitioning guaranteed to contain the implicit surface, and imposed the Lipschitz condition on F' to find ray intersections within each octree cell [Kalra & Barr, 1989].

1.2 Overview

Sphere tracing is a guaranteed technique for ray tracing implicit surfaces. Unlike LG-surfaces or interval analysis, it does not require the ability to evaluate the derivative of the function. Instead, it requires only a bound on the magnitude of the derivative — that the function be continuous and Lipschitz. Thus, the derivative of the function need not be continuous, nor even defined.

Sphere tracing benefits from this relaxation by using the continuous but non-differentiable minimum and maximum operations for constructive solid geometry instead of the commonly used Roth diagrams [Roth, 1982]. Unlike typical ray tracers, Sphere tracing finds the first ray intersection, the least positive solution t to (4). Typically, all ray intersections must be determined for constructive solid geometry [Roth, 1982]. Sphere tracing overcomes this requirement by using maximum and minimum operations to model the entire scene with a single, non-differentiable function. This also supports the blending of non-differentiable CSG results.

Sphere tracing allows the efficient visualization a wider range of implicit surfaces than before possible, including creased, rough and fractal surfaces. Like the slower brute-force rendering approach of the "Hypertexture" system [Perlin & Hoffert, 1989], sphere tracing frees the implicit surface designer from many concerns regarding the analytic behavior of the defining function, fostering more diverse implicit formulations. Moreover, structures in mathematics are often specified as the locus of points that satisfy a particular condition. Sphere tracing visualizes such structures, regardless of smoothness, extent and connectedness, given only a bound on the rate of the condition's continuous changes over space. Sphere tracing provides a direct and flexible visualization tool for the development of new implicit models.

Sphere tracing approximates cone tracing [Amanatides, 1984] to eliminate aliasing artifacts and simulate soft shadows.

2 Sphere Tracing

Sphere tracing capitalizes on functions that return the distance to their implicit surfaces (Section 2.1) to define a sequence of points (Section 2.2) that converges linearly to the first ray-surface intersection (Section 2.3). Section 2.4 incorporates constructive solid geometry into sphere tracing at the model level. Section 2.5 describes several enhancements to sphere tracing to hasten convergence.

2.1 Distance Surfaces

This section defines and discusses functions that measure or bound the geometric distance to their implicit surfaces. Such functions implicitly define *distance surfaces,* as mentioned in [Bloomenthal & Shoemake, 1991]. The appendices derive functions that measure or bound distances for a variety primitives and operations.

Let the function f be a continuous mapping $f : \mathbb{R}^n \rightarrow \mathbb{R}$ that implicitly describes the set $A \subset \mathbb{R}^n$ as the locus of points

$$A = \{\boldsymbol{x} : f(\boldsymbol{x}) \leq 0\}. \tag{5}$$

The continuity of f implies that it return zero on the boundary ∂A which forms the *implicit surface* of f. If f is strictly negative[1] over the interior $\overset{\circ}{A}$, then the multivalued function image $f^{-1}(0)$ concisely represents the implicit surface of f.

Definition 1 The point-to-set distance defines the distance from a point $\boldsymbol{x} \in \mathbb{R}^3$ to a set $A \subset \mathbb{R}^3$ as the distance from $\boldsymbol{x}$ to the closest point in A,

$$d(\boldsymbol{x}, A) = \min_{y \in A} ||\boldsymbol{x} - \boldsymbol{y}||. \tag{6}$$

[1]Even if f is continuous, it need not be strictly negative over the interior. For example, the set A may be validly represented by a continuous function that returns zero for every point in $\overset{\circ}{A}$.

Given a set A, the point-to-set distance $d(\boldsymbol{x}, A)$ implicitly defines A (from the outside) [Kaplansky, 1977]. Here, we are interested in the converse: *Given an implicit function, what is the point-to-set distance to its surface?*

Definition 2 A function $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ is a *signed distance bound* of its implicit surface $f^{-1}(0)$ if and only if

$$|f(\boldsymbol{x})| \leq d(\boldsymbol{x}, f^{-1}(0)). \tag{7}$$

If equality holds for (7), then f is a *signed distance function.*

Table 1 lists the primitives and operations for which the appendices contain signed distance functions and bounds.

Primitive/Operation	Signed Distance Function	Signed Distance Bound
plane	Appendix A	
sphere	Appendix A	
ellipsoid	[Hart, 1994]	Appendix A & E
cylinder	Appendix A	
cone	Appendix A	
torus	Appendix A	
superquadrics		Appendix B
generalized cylinder	Appendix C	
union	Section 2.4	
intersection		Section 2.4
complement	Section 2.4	
soft objects		Section D
pseudonorm blend	[Rockwood, 1989]	Appendix D
isometry	Appendix E	
uniform scale	Appendix E	
linear transformation		Appendix E
taper		Appendix E
twist		Appendix E
hypertexture		Appendix F
fractals		Appendix F

Table 1: Directory of signed distance functions and bounds.

The Lipschitz constant is a useful quantity for deriving signed distance bounds to complex shapes. Lipschitz constants have been used in computer graphics for collision detection [Von Herzen & Barr, 1987] and rendering implicit functions [Kalra & Barr, 1989].

Definition 3 A function $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ is *Lipschitz* over a domain D if and only if for all $\boldsymbol{x}, \boldsymbol{y} \in D$, there exists a positive finite constant λ such that

$$|f(\boldsymbol{x}) - f(\boldsymbol{y})| \leq \lambda ||\boldsymbol{x} - \boldsymbol{y}||. \tag{8}$$

The value λ is called the *Lipschitz constant.* The function Lip f, returns the minimum Lipschitz constant λ satisfying (8).

A Lipschitz constant of the sum of two functions results from the sum of the functions' Lipschitz constants. By the chain rule, a Lipschitz constant of the composition of functions results from the product of the component functions' Lipschitz constants.

The following theorem shows how to turn a Lipschitz function into a signed distance bound, allowing sphere tracing to render any implicit surface defined by a Lipschitz function.

Theorem 1 *Let f be Lipschitz with Lipschitz constant λ. Then the function f/λ is a signed distance bound of its implicit surface.*

Proof: Given a point $\boldsymbol{x}$, let $\boldsymbol{y} \in f^{-1}(0)$ be one of the points such that

$$||\boldsymbol{x} - \boldsymbol{y}|| = d(\boldsymbol{x}, f^{-1}(0)). \tag{9}$$

Then by (8) and $f(\boldsymbol{y}) = 0$ it follows that

$$|f(\boldsymbol{x})| \leq \lambda \; d(\boldsymbol{x}, f^{-1}(0)). \tag{10}$$

Hence, $\lambda^{-1} f(\boldsymbol{x})$ is a signed distance bound for any Lipschitz function f. (Compare Eq. (8) of [Kalra & Barr, 1989]). □

If $\lambda = \text{Lip}\, f$ then an optimal signed distance bound results.

2.2 Ray Intersection

One intersects a ray $\boldsymbol{r}(t)$ with the implicit surface defined by the signed distance bound $f(\boldsymbol{x})$ by finding its least positive root (the *first* root) of $F(t)$. This root is the limit point of the sequence defined by the recurrence equation

$$t_{i+1} = t_i + F(t_i) \tag{11}$$

and the initial point $t_0 = 0$. The sequence converges if and only if the ray intersects the implicit surface. This sequence forms the kernel of the geometric implicit surface rendering algorithm in Figure 1.

The convergence test ϵ is set to the desired precision. The maximum distance D corresponds to the radius of a viewer-centered yonder clipping sphere, and is necessary to detect non-convergent sequences.

The absolute value of the signed distance function can be considered the radius of a sphere guaranteed not to penetrate any of the implicit surface. This sphere was called an *unbounding* sphere in [Hart *et al.*, 1989] (which used a distance bound to implicitly define and visualize 3-D deterministic fractals) because the implicit surface is contained in the closed complement of this sphere. Unlike a bounding volume which surrounds an object, an unbounding volume surrounds an area of space not containing the object. The name "sphere tracing" arose from the property that ray intersections are determined by sequences of unbounding spheres.

As did [Ricci, 1974], sphere tracing uses the minimum and maximum functions for constructive solid geometry. These operations crease the implicit surface locally, such that the defining function

```
Given signed distance bound $f$, ray $\boldsymbol{r}(t)$ and maximum ray traversal distance $D$.
Initialize $t = 0$ and $d = 0$
While $t < D$
    Let $d = f(\boldsymbol{r}(t))$
    If $d < \epsilon$ then return $t$ — intersection
    Increment $t = t + d$
return $\emptyset$ — no intersection
```

Figure 1: Pseudocode of the geometric implicit surface rendering algorithm.

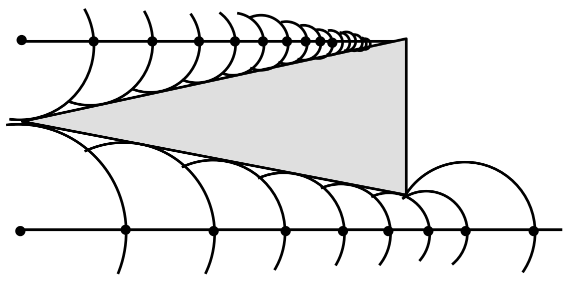


Figure 2: A hit and a miss.

remains continuous in value, but not in derivative. Derivative discontinuity can cause problems with root finders, which must find all roots of the function and resolve the CSG operation using a Roth diagram [Roth, 1982]. Sphere tracing operates independent of the derivative, given its bound, and need converge only to the first root, even for CSG models.

2.3 Analysis

Root refinement methods, such as Newton's method, converge quadratically to simple roots (where the ray penetrates the surface), and linearly to multiple roots (where the ray grazes the surface) [Gerald & Wheatley, 1989]. Root isolation methods which divide and conquer, such as LG-surfaces [Kalra & Barr, 1989] and interval analysis [Mitchell, 1990], converge linearly since the width of the intervals are reduced by a factor of one-half at each iteration. Root isolation methods are allowed to converge only in the event of a multiple root, otherwise they pass control to a faster root refinement method the moment they find a monotonic region straddling the t-axis.

Theorem 2 *Given a function $F : \mathbb{R} \to \mathbb{R}$ with Lipschitz bound $\lambda \geq \text{Lip}\, F$, and an initial point t_0, sphere tracing converges linearly to the smallest root greater than t_0.*

The sphere-tracing sequence can be written

$$t_{i+1} = g(t_i) = t_i + \frac{|F(t_i)|}{\lambda}. \tag{12}$$

In this form, the similarities of (12) to Newton's method are more visible. Let r be the smallest root greater than the initial point t_0. Since $F(r) = 0$ then $g(r) = r$, and at any non-root $|F|/\lambda$ is positive. Hence (12) converges to the first root.

Without loss of generality, F is assumed to be non-negative in the region of interest, which eliminates the need for the absolute value. The Taylor expansion of $F(t_i)$ about the root r is

$$g(t_i) = g(r) + (t_i - r)g'(r) + \frac{(t_i - r)^2}{2} g''(\tau) \tag{13}$$

for some $\tau \in [t_i, r]$ and $g'(r) = 1 + F'(y)/\lambda$. The error term becomes

$$e_{i+1} = t_{i+1} - r = g(t_i) - g(r) = g'(r)e_i + \text{higher order terms} \tag{14}$$

Since $g'(r)$ is constant in the iteration, (12) converges linearly to y. □

Corollary 2.1 *Sphere tracing converges quadratically if and only if the function is steepest at its first root.*

In the event $F'(r) = -\lambda$, the linear term of the error (13) drops out, leaving the quadratic and higher order terms. □

2.4 Constructive Solid Geometry

Following [Ricci, 1974], the minimum and maximum operations on functions results in union and intersection operations on their implicit surfaces. In the following equations, let f_A, f_B be signed distance functions of sets A and B respectively. If f_A or f_B is a signed distance bound, then the resulting CSG implicit function will be also be a bound.

The distance to the union of A and B is the distance to the closer of the two

$$d(\boldsymbol{x}, A \cup B) = \min f_A(\boldsymbol{x}), f_B(\boldsymbol{x}). \tag{15}$$

Similarly, the distance to a list of objects is the smallest of the distances to each of the component objects.

The distance to the complement of A takes advantage of the signed nature of the distance function

$$d(\boldsymbol{x}, \mathbb{R}^3 \setminus A) = -f_A(\boldsymbol{x}). \tag{16}$$

Although DeMorgan's theorem defines intersection as the complement of the union of complements, the minimum operators used in the union are not complemented properly. Instead, the distance to the intersection is bound by the distance to the farthest component.

Theorem 3 *The distance from a point $\boldsymbol{x}$ to the intersection of two implicit surfaces $A = f_A^{-1}(0)$ and $B = f_B^{-1}(0)$ defined by signed distance bounds f_A, f_B is bounded by*

$$d(\boldsymbol{x}, A \cap B) \geq \max f_A(\boldsymbol{x}), f_B(\boldsymbol{x}). \tag{17}$$

Proof: By parts, as illustrated on a sample intersection in Figure 3.

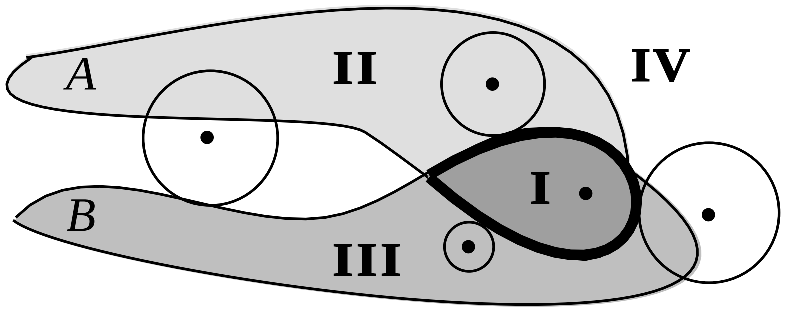


Figure 3: Sample points illustrated a bound on the distance to the intersection between two sets.

Case I: $\boldsymbol{x} \in A \cap B$. Both f_A and f_B are negative, and the larger of the two indicates the (negative) distance to the closest edge of the intersection.

Case II: $\boldsymbol{x} \in A, \boldsymbol{x} \notin B$. The function f_A is negative whereas f_B is positive, hence the greater of the two. The closest point on B to $\boldsymbol{x}$ may not be in the intersection, but there cannot be any point in the intersection closer.

Case III: $\boldsymbol{x} \notin A, \boldsymbol{x} \in B$. Symmetric with Case II.

Case IV: $\boldsymbol{x} \notin A \cup B$. As before, the closest point in the intersection $A \cap B$ can be no closer than the farther of the closest point in A and the closest point in B. $\square$

From its definition, set subtraction $A - B$ may be simulated as $A \cap (\mathbb{R}^3 \setminus B)$, though yielding only a signed distance bound due to the intersection operator.

The union and intersection operators are demonstrated in Figure 9 in Section 4.2.

2.5 Enhancements

The following enhancements increase the efficiency of sphere tracing by reducing unnecessary distance computations, which can be quite expensive and even iterative in some cases. The enhancements are evaluated and analyzed empirically in Section 4.3.

2.5.1 Image Coherence

An algorithm similar to sphere tracing has been developed for rendering discrete volumetric data using the 3-D *distance transform* [Zuiderveld *et al.*, 1992]. The distance transform takes a binary "filled/unfilled" voxel array to a numerical voxel array such that each voxel contains the distance to the closest "filled" voxel, under a given metric. We have also extended the concept of Lipschitz constants to volume rendering [Stander & Hart, 1994], trading the distance transform for an octree of local Lipschitz constants as in [Kalra & Barr, 1989], allowing distance-based accelerated volume rendering of arbitrary isovalued surfaces while eliminating the need to recompute the preprocessed data structure for each change in the threshold.

One enhancement in [Zuiderveld *et al.*, 1992] kept track of the smallest distance encountered by a ray that misses the object. Under an orthogonal projection, this smallest distance defines the radius of a disk of guaranteed empty pixels surrounding the sample point. Under a perspective projection, the minimum *projected* distance must be computed (requiring ray-sphere intersection), and this enhancement becomes less efficient. Initial tests have shown this enhancement to degrade performance in the perspective case for typical implicit surfaces.

2.5.2 Bounding Volumes

Bounding volumes are a useful mechanism to cull processing of intricate geometries which are irrelevant to the current task. Beyond their typical benefit of avoiding the casting of rays that miss an object, they also help sphere tracing avoid distance computations for objects farther away than others. The overhead of quick bounding-volume distance checks is, in most cases, a small price to pay for the benefit of avoiding many expensive but useless distance computations.

First, the distances to each bounding volume in a union or collection of objects is computed. Then in order of increasing bounding volume distance, the distance to the contents of each bounding volume is computed until a content's distance is less than the smallest bounding volume distance. This distance is then the point-to-set distance to the collection of objects. This process is sketched in Figure 4.

A Lagrange multiplier method for finding the bounding parallelepiped of an implicit surface appears in [Kay & Kajiya, 1986]. The signed distance bound has properties which might yield an alternative implicit surface bounding volume algorithm, but this topic is left for further research.

2.5.3 The Triangle Inequality

When computing the shortest distance between a point and a collection of objects, one need not compute the distance to objects whose last distance evaluation minus the distance traversed along the ray since that last evaluation is still larger than the distance to the currently closest object. This triangle inequality enhancement is implemented in Figure 5.

```
Make a heap $D$ of bounding volume distances to each object.
Initialize $d = \infty$.
Repeat
    Let $d$ be the lesser of $d$ or the distance to the contents of the bounding volume
    at the top of the heap.
    Remove the top of the heap and re-heap.
    Let $d_h$ be the distance to the bounding volume now at the top of the heap.
Until $d < d_h$ or the heap is empty.
return $d$.
```

Figure 4: An efficient algorithm for finding the closest object of a collection using bounding volumes.

```
Given ray $\boldsymbol{r}(t)$, maximum distance $D$ and a collection of objects $O$.
Initialize $d_{\text{last}} = 0$ and $t = 0$.
For each object $o \in O$ initialize $o_d = 0$.
Until $d_{\min} < \epsilon$ or $t > D$.
    For each object $o \in O$.
        If $o_d - d_{\text{last}} > d_{\min}$ then
            Update $o_d = o_d - d_{\text{last}}$.
        Otherwise
            Let $d = d(\boldsymbol{r}(t), o)$.
            Reset $o_d = d$.
            Update $d_{\min} = \min(d_{\min}, d)$.
        End if.
        Let $d_{\text{last}} = d_{\min}$.
    End for.
    Update $t = t + d_{\min}$.
End until.
```

Figure 5: Triangle inequality algorithm for avoiding unnecessary distance computations.

2.5.4 Octree Partitioning

Eliminating empty space certainly aids rendering efficiency, but the major benefit of partitioning is that it allows the imposition of local bounds on the Lipschitz constants yielding more accurate signed distance bounds. Octree partitioning has been used in the polygonization [Bloomenthal, 1989] and ray tracing [Kalra & Barr, 1989] of implicit surfaces. Sphere tracing reaps the same benefits from spatial partitioning as did the root finding method in [Kalra & Barr, 1989], which used the Lipschitz constant to cull octree nodes guaranteed not to intersect the implicit surface.

Ray intersection with an implicit surface defined by a signed distance bound is penalized by the section of the domain where the gradient magnitude is greatest. Chopping an object into the union of smaller chunks allows each chunk to be treated individually, penalized only by the largest gradient within its bounds. Since the partitioning algorithm in [Kalra & Barr, 1989] required only a bound on the Lipschitz constant of the function, the use of this octree in no way restricts the domain of functions available for sphere tracing.

Octree partitioning further enhances sphere tracing of unions and lists by optionally storing an index to the object closest to the cell. An object is closest to an octree cell if and only if it is the closest object to every point in the cell. Under this definition, some cells may not have a closest object. By the triangle inequality an object is closest to a cell if the distance from the cell's centroid to the object, plus the distance from the centroid to the cell corner, is still less than the distance from the centroid to any other object.

2.5.5 Convexity

Knowing that an object is convex can make sphere tracing more efficient by increasing the step size along the ray.

Theorem 4 *Let $A \subset \mathbb{R}^3$ be a convex set defined implicitly by the signed distance function f. Then given a unit vector $\boldsymbol{v} \in \mathbb{R}^3$ the line segment*

$$[\boldsymbol{x}, \frac{f(\boldsymbol{x})}{-\boldsymbol{v} \cdot \nabla f(\boldsymbol{x})}\boldsymbol{v}] \tag{18}$$

does not intersect A except possibly at its second endpoint.

Proof: The gradient of a signed distance function ∇f has the following properties on the complement of a convex set $\mathbb{R}^3 \setminus A$: (1) it is continuous; (2) its magnitude is one (the change in the function equals the change in the distance); and (3) its direction points directly away from the closest point on the implicit surface. Hence, for any $\boldsymbol{x} \in \mathbb{R}^3 \setminus A$ we know the closest point in A, and its surface normal points toward $\boldsymbol{x}$. Since A is convex, it cannot penetrate the tangent plane to $\boldsymbol{x}$.

The intersection of a ray anchored at $\boldsymbol{x}$ and direction $\boldsymbol{v}$ with the tangent plane normal to the vector $\nabla f(\boldsymbol{x})$ a distance of $f(\boldsymbol{x})$ from $\boldsymbol{x}$ is given by the second endpoint of (18). □

Corollary 4.1 *If $\nabla f(\boldsymbol{x}) \cdot \boldsymbol{v} \geq 0$ then the ray anchored at $\boldsymbol{x}$ and direction $\boldsymbol{v}$ does not intersect the implicit surface of f.*

Theorem 4 allows sphere tracing to make larger steps toward convex objects, and Corollary 4.1 allows sphere tracing to avoid computing the distance to convex objects it has stepped beyond. The convexity enhancement likely causes sphere tracing to converge quadratically, because of its similarity to Newton's method, which also converges quadratically.

Bounding volumes are usually convex, and combining these two techniques can further reduce the computation of unnecessary distances.

Knowledge of convexity becomes a necessity for rendering scenes with a horizon line. Consider a ground plane and a ray parallel to it. Sphere tracing will step along this ray at fixed intervals looking for an intersection that never happens. Corollary 4.1 avoids this situation whereas Theorem 4 hastens convergence of rays nearly parallel to the ground plane.

3 Antialiasing

Tracing cones instead of rays resulted in an area-sampling antialiasing method in [Amanatides, 1984]. Cone tracing computed the intersection of cones with spheres, planes and polygons to symbolically prefilter an image, eliminating the aliasing artifacts that result from point sampling. Sphere tracing can detect and approximate cone intersections with any implicit surface defined by a signed distance function. One must still implement the details of the cone tracing algorithm to determine the shape of the cones as they bounce around a scene, but may rely on unbounding spheres to increase the efficiency of computing cone intersections.

At some point along a grazing ray, the sequence of unbounding spheres shrinks, falling within the bounds of the cone, then enlarges, escaping the bounds of the cone. This poses the problem of "choosing a representative" [Amanatides, 1984] — a location to take a sample to approximate the shading of the cone's intersection with the surface.

A cover is a pixel-radius offset bounding an implicit surface on the inside and outside such that a ray-cover intersection indicates a cone-object intersection [Thomas *et al.*, 1989]. Given an implicit surface defined by the signed distance function $f(\boldsymbol{x})$, its outer cover is the global offset surface implicitly defined by $f(\boldsymbol{x}) - r_p$ and its inner cover is the global offset surface implicitly defined by $f(\boldsymbol{x}) + r_p$, where r_p is the radius of a pixel (one-half of the diameter of a pixel [Hart & DeFanti, 1991]). In other words, the outer cover is the surface $f^{-1}(r_p)$ and the inner cover is the surface $f^{-1}(r_p)$. Instead of sphere tracing the implicit surface of $f(\boldsymbol{x})$, the antialiasing algorithm sphere traces the inner cover — the implicit surface of $f(\boldsymbol{x}) + r_p$.

The development of covers proposes the most representative choice for silhouette antialiasing would be the point along the section of the ray closest to the surface. Hence, of the unbounding spheres inside the cone, the center of the smallest sphere (with respect to pixel size) becomes the representative sample. Though this sample is off the implicit surface, one assumes a reasonable level of continuity in the gradient of the distance function to define a usable surface normal. The sequence along the ray of unbounding spheres are related to a cone as shown in Figure 6.

For smooth implicit surfaces, one may assume local planarity. Hence the implicit surface is assumed to cover the cross section of the cone with a straight edge of the given distance from the cone's center. The amount of influence this shaded point has, with respect to the points the ray intersects further on, depends on the signed distance function evaluated at the representative $f(\boldsymbol{x})$ (the radius of the closest unbounding sphere) to the implicit surface. The fraction of coverage of a

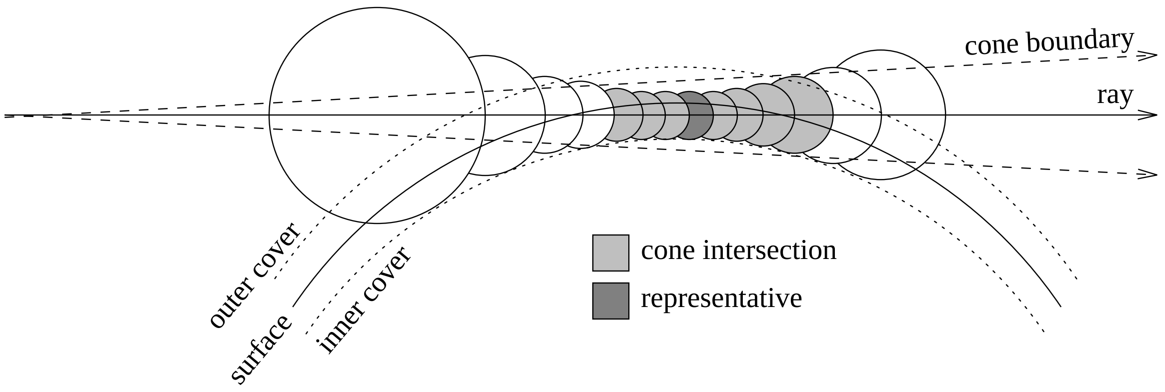


Figure 6: Sphere tracing approximates cone intersection. The ray intersects the original surface but misses its inner cover. This cone intersection will account for more than half of the pixel's illumination.

disk of radius r_p by an intersecting half-plane of signed distance $f(\boldsymbol{x})$ from its center is given by

$$\alpha = \frac{1}{2} - \frac{f(\boldsymbol{x})\sqrt{r_p^2 - f(\boldsymbol{x})^2}}{\pi r_p^2} - \frac{1}{\pi}\arcsin\frac{f(\boldsymbol{x})}{r_p} \tag{19}$$

and is derived in [Thompson, 1990].

Ray traversal proceeds in steps of $f(\boldsymbol{x}) + r_p$ (which may take it through the surface). The percentage of coverage α represents the cone intersection of the grazing ray. It is treated as an opacity and is accumulated and used to blend the shading of the current representative $\boldsymbol{x}$ with the shading resulting from further near misses and intersections, using the standard rules of image compositing [Porter & Duff, 1984].

For intersection edges, one must keep track of all signed distance functions whose unbounding spheres fit within the bounds of the cone. Upon ray intersection approximation, the signed distance functions of each of the intersecting surfaces provide the proportions for the proper combination of their shading properties. The representative for intersection is the last point of the ray traversal sequence, the point that satisfies the convergence test.

Often the signed distance function is too expensive to compute efficiently and a signed distance bound is used. A bound may return unbounding spheres whose radii prematurely shrink below the radius of a pixel, resulting in incorrect cone intersections. In this case, a separate distance approximation may be useful. For example, [Pratt, 1987; Taubin, 1994] estimate the distance to the implicit surface of f with the first order approximation $f/||\nabla f||$. In general, this approximation is not necessarily a distance bound. Lemma 1 of [Taubin, 1994] asserts that this approximation is asymptotic to geometric distance as one approaches the surface. Cone intersections can hence be more accurately determined by this approximation than by the signed distance bound.

Cone tracing inhibits texture aliasing by filtering the texture based on the radius of the cone at intersection, and extends directly to the sphere tracing method.

4 Results

Sphere tracing simplifies the implementation of an implicit surface ray tracer, and runs at speeds comparable to other implicit surface rendering algorithms.

4.1 Implementation

Sphere tracing has been implemented in a rendering system called zeno. Inclusion of an implicit surface into zeno requires the definition of two functions: a signed distance function for ray intersection, and a surface normal function for shading.

A new primitive or operation can be incorporated into zeno with no more than a distance bound. The negative part of the signed distance bound is only necessary for some constructive solid geometry and blending operations, and is not needed for the visualization of functions that are zero-valued inside the implicit surface. The surface normal function can be avoided by using a general six-sample numerical gradient approximation of the distance bound gradient. Since most of the time is spent on ray intersection, the inefficient numerical gradient approximation has a negligible impact on rendering performance.

The simplicity with which implicit surfaces are incorporated in zeno makes it useful for visualization of mathematical tasks and investigation of new implicit surfaces. For example, a homotopy that removes a 720° twist from a ribbon without moving either end formed the basis for the animated short “Air on the Dirac Strings” [Sandin *et al.*, 1993], for which zeno rendered a segment. This homotopy is based heavily on interpolated quaternion rotations and was easily incorporated into zeno as a domain transformation after a quick search and analysis of the most extreme deformation in the homotopy [Hart *et al.*, 1993].

4.2 Exhibition

The three tori in Figure 7 are combined using the superelliptic blend described in Appendix D.2. The tori all are of major radius one, and minor radius one-tenth. The blue-green blend is quadratic extending along the tori a radius of 0.5 from their intersection. The red-green blend also has radius 0.5 but is degree eight. The red-blue blend is also degree eight but has a radius of only 0.2.

Sphere tracing rendered Figure 7 (left) in 12:47 at a resolution of only 256×256 using prefiltering to avoid the severe aliasing that ordinarily accompany such low sampling rates. Experiments on the difference of execution using point sampling and area sampling show that the increased execution time due to area sampling is negligible.

Although the superelliptic blend is implemented in zeno as a signed distance bound, it returns an underestimated distance of no less than 70% of the actual distance which adequately indicated cone intersections, as the enlargement demonstrates in Figure 7 (upper right).

The work image in Figure 7 (lower right) shows that sphere tracing concentrates on silhouette edges. Blue areas converge from 10 iterations, green around 50 and red over 100.

Figure 8 demonstrates a generalized cylinder, from Appendix C, whose skeleton consists of a space curve modeled with 14 Bezier control polygons. Sphere tracing can render this scene in as fast as 5:30 using bounding spheres to eliminate unnecessary distance computations. The curved horizon is an artifact of the yonder clipping sphere of radius $1,000$ used to terminate ray stepping.

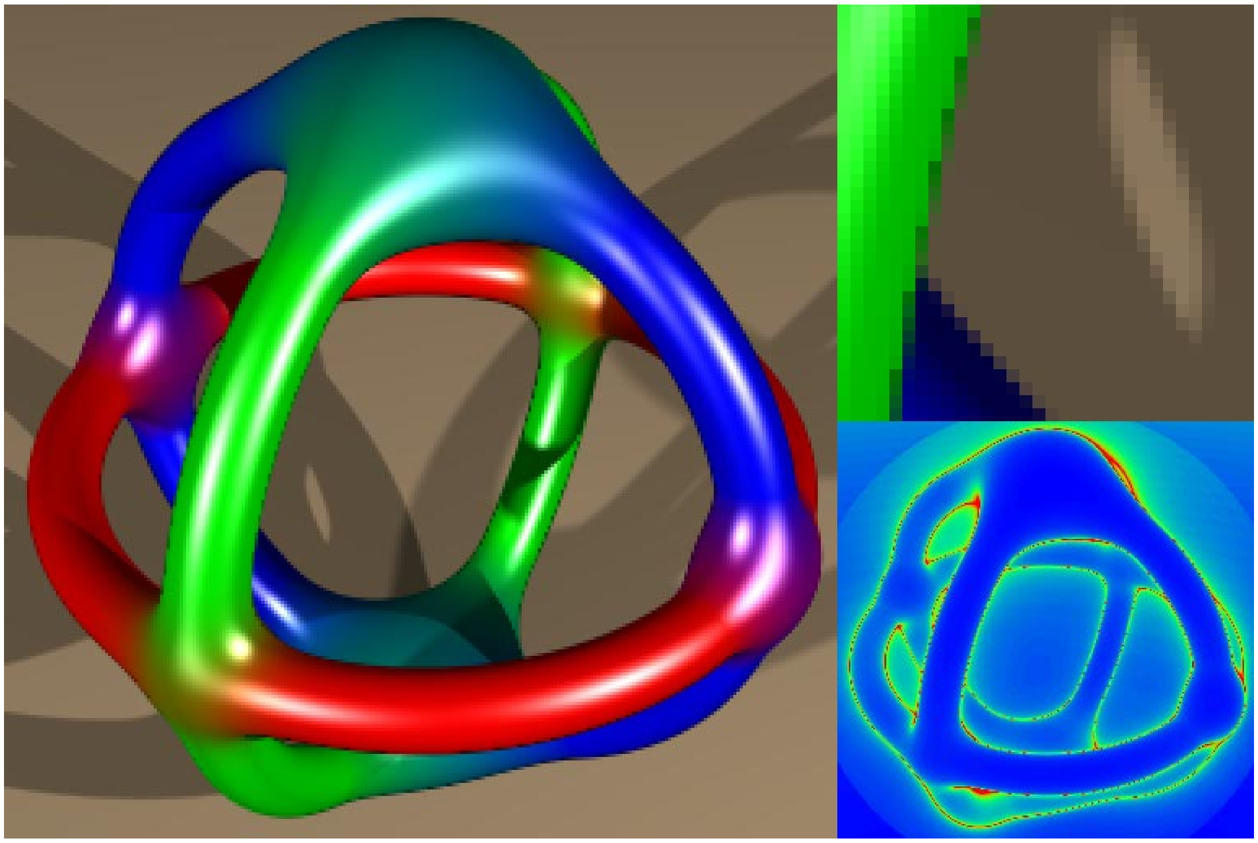


Figure 7: Three blends of tori (left), blowup (upper right) and work image (lower right).

Figure 8: A logo for zeno.

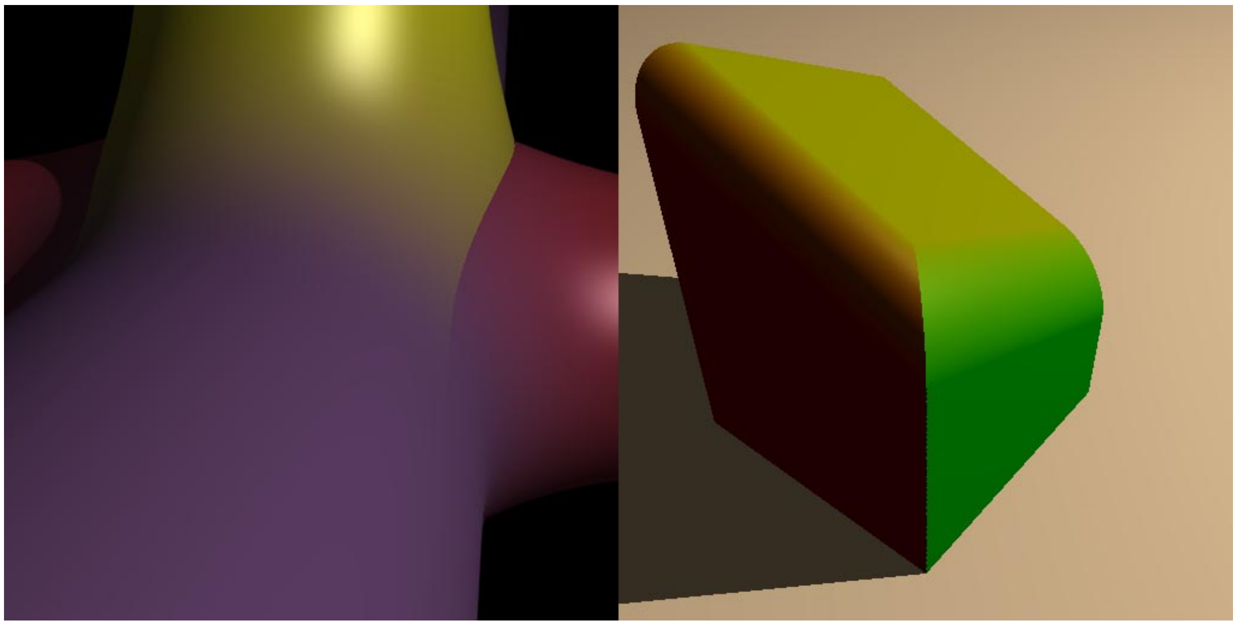


Figure 9: Creases created by blended edges.

Figure 9 demonstrates the robustness of sphere tracing on creased surfaces. Both images were rendered with prefiltering at a resolution of 512×512, and in 16:48 for the cylinders, 12:36 for the cube.

The creases were created as CSG unions and intersections, defined implicitly by the continuous but non-differentiable minimum and maximum operations from Section 2.4. The resulting edge was then merged into a third object using the pseudonorm blend from Appendix D.2. Such creased surfaces appear periodically in a variety of shapes, particularly in the modeling of biological forms.

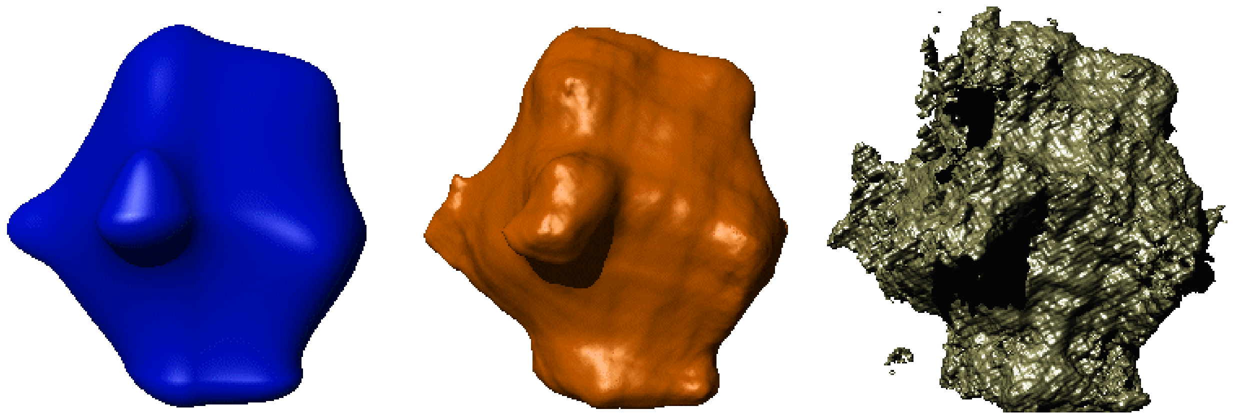


Figure 10: “Lava” (left) modeled as a sphere deformed by the noise function. “Muscle” (center) modeled with $\beta = 2$ noise. “Rock” (right) modeled with $\beta = 1$ noise.

Figure 10 illustrates the “noise” range deformation described in Appendix F. The left image uses a single octave of noise, whereas the next two use six octaves, whose amplitude was scaled by $1/f^2$ and $1/f$, respectively, yielding a muscle texture and a rocky surface. The three images were each rendered at a resolution of 256×256 in (from left to right) approximately five minutes, half-an-hour, and two hours. The high variation of distance estimates prohibited prefiltering the results of the noise function.

4.3 Analysis

Sphere tracing convergence is entirely linear whereas other general root finders, such as interval analysis, have a linearly-convergent root isolation phase followed by a quadratically-convergent root refinement stage. Work images, such as Figure 7 (lower right), show that ray intersection is most costly at silhouette edges. When sphere tracing these edges, the distance to the surface is only a fraction of the distance to the ray intersection which slows convergence. For other methods like interval analysis, silhouettes are double roots (that prevent root refinement) and their neighborhoods consist of closely-spaced pairs of roots. Such root pairs are costly for midpoint subdivision root refinement methods to separate since the distance between the two roots can be several orders of magnitude smaller than the initial interval.

scene	execution time	relative time	enhancement
single sphere	2:00	100%	none
	1:23	69%	convexity
9 spheres/plane	2:53	100%	none
	1:42	59%	convexity
	1:19	46%	triangle inequality
	1:10	40%	both
zeno logo	26:29	100%	none
	19:23	73%	triangle inequality
	5:28	21%	bounding spheres
“Lava”	4:37	1	(single noise)
“Muscle”	33:52	7.3	($1/f^2$ noise)
“Rock”	2:06:56	27.5	($1/f$ noise)

Table 2: Comparison of execution times for enhanced sphere tracing of various scenes.

The convexity enhancement hastened convergence by 31% as shown in Table 2. With more primitives, this same table shows the triangle inequality enhancement to more than double the convergence rate, and when combined with convexity, enhances ordinary sphere tracing by 60%.

Table 2 also compares various enhanced rendering times for the zeno logo. The fact that all 14 Bezier curves were nearly equidistant from the eye prevented the triangle inequality from significantly reducing unnecessary distance evaluations until sphere tracing had traversed much of each ray.

Figure 11 reveals the distribution of step sizes used in sphere tracing a ball. This histogram counted only the distance evaluations used to intersect primary (eye) rays.

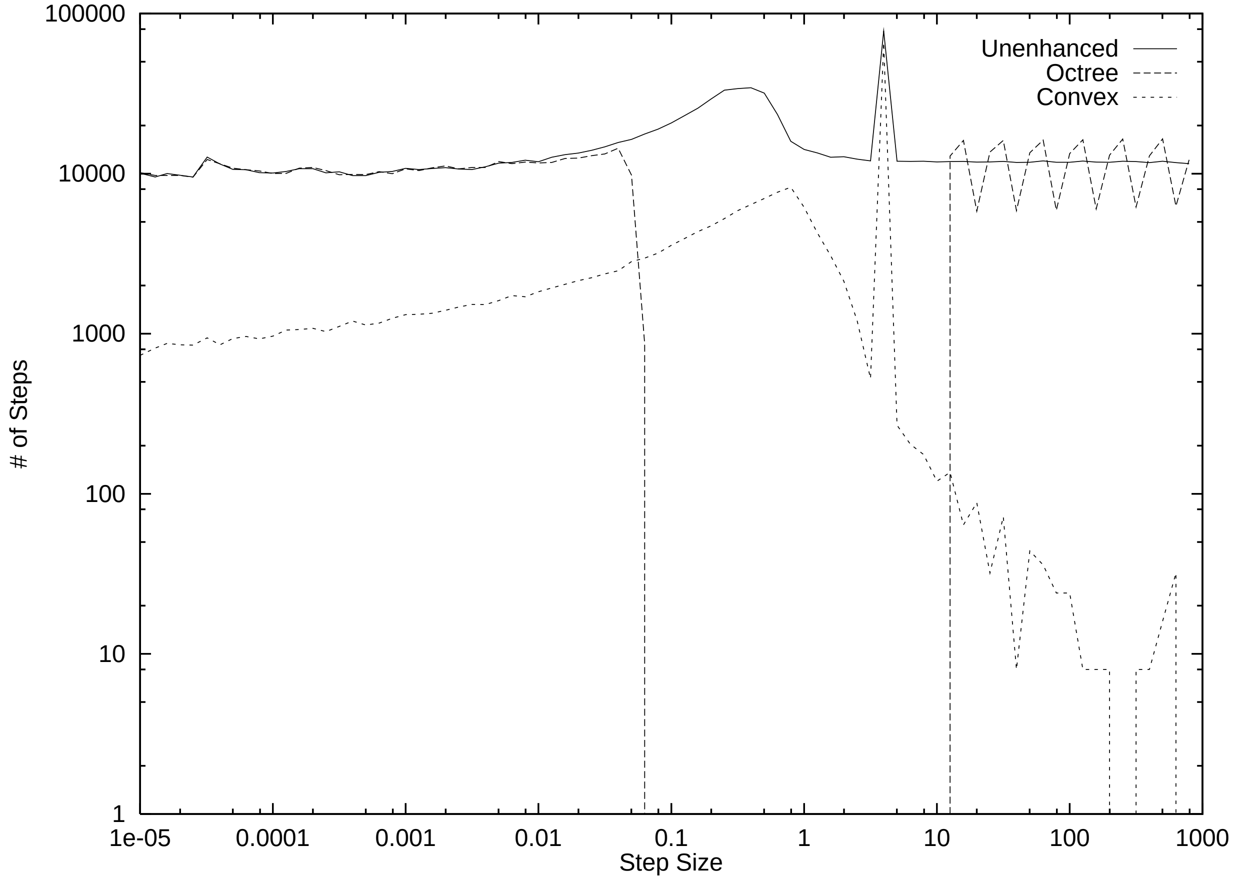


Figure 11: Histogram of step sizes for sphere tracing a ball.

Unimproved sphere tracing is evenly distributed, with a small hump in the middle. An octree replaces the increased distance computation in this humped area with octree parsing overhead, (which this histogram does not measure). Echoes of the octree bounds cause the oscillations at the high end of its spectrum, whereas the low end adheres to the unenhanced performance. Experiments on simple scenes failed to demonstrate any increased performance from the octree enhancement, although more complicated scenes are likely to benefit from its use.

The convex histogram demonstrates the power of this enhancement. Its slope on the left confirms the expectation from Section 2.5.5 that it provides sphere tracing a faster order of convergence. The right side of this histogram is significantly reduced, due to the cessation of stepping after moving beyond the sphere.

The spike in the unenhanced and convex graphs indicates the distance from the eye to the ball, which is the first step taken by every ray emanating from the eye-point. One can remove these spikes from the graph by measuring this distance once and refer to it as the first step for rays emanating from the eye-point, and likewise for the light sources. This “head start” barely improved performance in experiments.

Similar histograms in [Zuiderveld *et al.*, 1992] measured performance logarithmically in the number of steps but linearly in step size. As a result, their graphs were more logarithmically shaped than Figure 11.

The accuracy of the distance estimate is directly proportionate to the rate of convergence. Experiments on a sphere show that half the distance doubles the number of steps. The step-size histograms in Figure 12 reveals the effects of distance underestimation.

The relationship between distance accuracy and sphere tracing performance suggests that in certain cases a slower signed distance function may perform better than a fast distance underestimate. For example, consider the distance to an ellipsoid with major axes of radius 100, 100 and 1 modeled as a non-uniform scale transformation of the unit sphere. Section E yields a signed distance bound which returns at best the distance to the ellipsoid, and at worst 1% of the distance, in closed form, whereas [Hart, 1994] yields a signed distance function which returns the exact distance at the expense of several Newton iterations. In this case, the signed distance function would likely result in better performance.

Finally, the Lipschitz constants of the noise functions are 3 for single noise, 6 for $1/f^2$ noise and 18 for $1/f$ noise (six octaves). The timings in Table 2 corresponding to the images in Figure 10 show that the $1/f^2$-noise rendering time was actually 7.3 times (instead of the expected value of twice) the single noise time. The likely reason is that the $1/f^2$ noise invokes the noise function six times more than the single noise function (yielding an expected value of 12 times). The $1/f$-noise rendering time was 27.5 times longer than that of single noise (less than the expected 36 times), and 3.75 times longer than the $1/f^2$ noise (slightly larger than the expected value of 3).

5 Conclusion

Sphere tracing provides a tool for investigating a larger variety of implicit surfaces than before possible.

With its enhancements and prefiltering, sphere tracing becomes a competitive presentation-quality implicit surface renderer. In particular, the convexity enhancement greatly increases render-

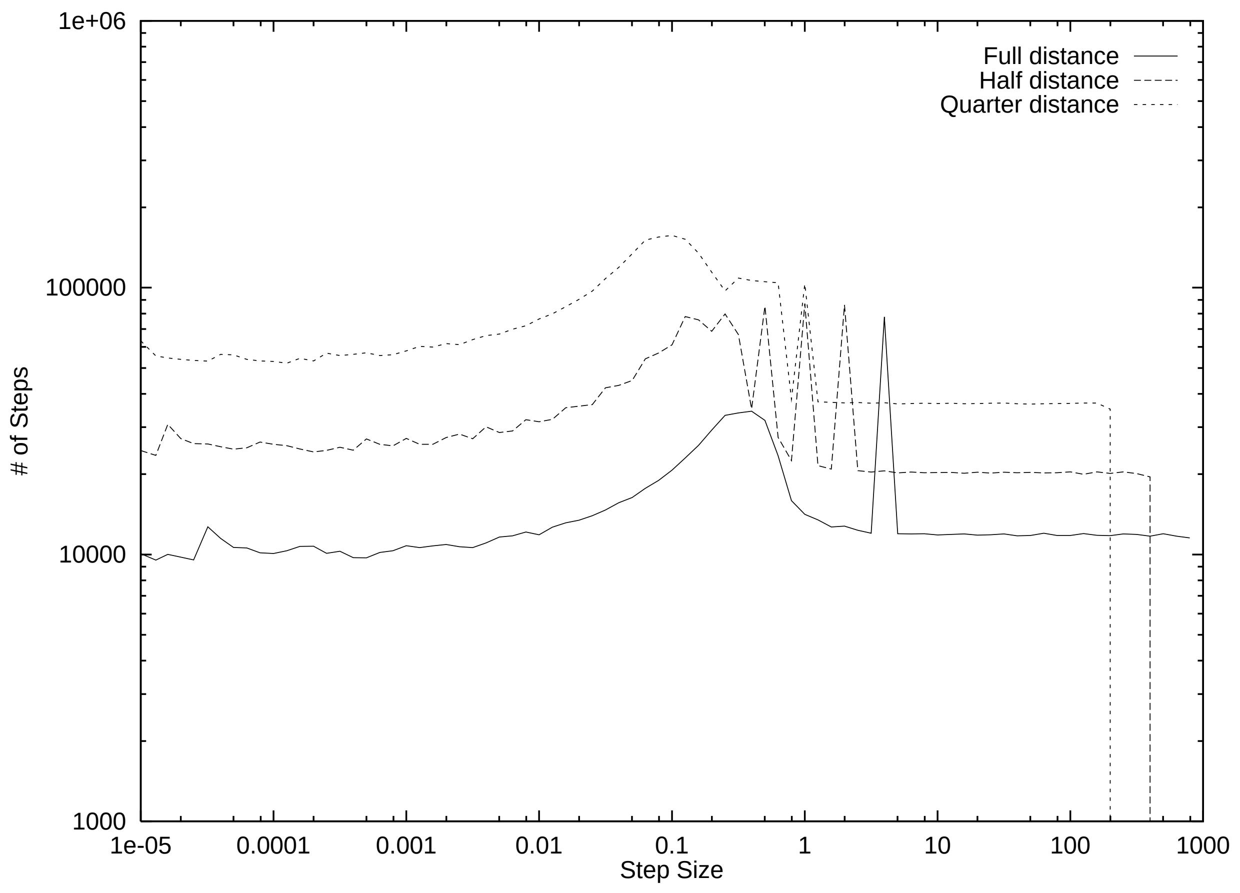


Figure 12: Halving step sizes doubles convergence time.

ing speeds, and the triangle inequality is quite effective for large assortments of objects. Bounding volumes also increase rendering performance as expected. However, techniques based on image coherence and space coherence (octree) did not perform as well.

Whereas sphere tracing performed significantly slower than standard ray tracing on simple objects consisting of quadrics and polygons, it excelled at rendering the results of sophisticated geometric modeling operations.

The geometric nature of sphere tracing adapts it to symbolic prefiltering, supporting antialiasing at a nominal overhead.

In lieu of direct experimental comparison, several theoretical arguments show sphere tracing as a viable alternative to interval analysis and L-G surfaces.

5.1 Further Research

Sphere tracing demonstrates the utility of signed distance functions in the task of rendering geometric implicit surfaces. We expect these functions will similarly enhance other applications, particularly in the area of geometric processing. As geometric distance becomes more important in computer-aided geometric design and other areas of modeling, the demand for more efficient geometric distance algorithms will increase.

In retrospect, the use of the Euclidean distance metric seems an arbitrary choice for sphere tracing. The linear nature of the chessboard and Manhatten metrics may result in more efficiently computed distances and ray intersection. “Cube-tracing” and “octahedron-tracing” algorithms are left as further research.

5.2 Acknowledgments

Thanks to Tom DeFanti and Larry Smarr for procuring the support of the first year of this research. This research was subsequently supported by the National Science Foundation under the Research Initiation Award #CCR-9309210, and its implementation and distribution supported by a Research Experience for Undergraduates extension. The research was performed at the Imaging Research Laboratory, which is supported in part by NSF grants #CDA-9121675 and #CDA-9422044.

Special thanks to Alan Norton who, in 1989, encouraged me to apply sphere tracing to non-fractal models. The task of tracking down and deriving the distances in the appendix was greatly helped by conversations with Al Barr, Chandrajit Bajaj, Charlie Gunn, Pat Hanrahan, Jim Kajiya, Don Mitchell and Alyn Rockwood. I would also like to thank the anonymous reviewers, especially Jules Bloomenthal, for many insightful comments and recommendations. Brian Wyvill and Jules Bloomenthal deserve special mention for inspiring this research by putting together an excellent course on implicit surfaces at SIGGRAPH ’90.

References

[Agin & Binford, 1976] Agin, G. J. and Binford, T. O. Computer description of curved objects. *IEEE Transactions on Computers* C-25(4), Apr. 1976, pp. 439–449.

[Amanatides, 1984] Amanatides, J. Ray tracing with cones. *Computer Graphics* 18(3), July 1984, pp. 129–135.

[Ballard & Brown, 1982] Ballard, D. H. and Brown, C. M. *Computer Vision*. Prentice-Hall, Englewood Cliffs, NJ, 1982.

[Barnhill *et al.*, 1992] Barnhill, R. E., Frost, T. M., and Kersey, S. N. Self-intersections and offset surfaces. In Barnhill, R. E., ed., *Geometry Processing for Design and Manufacture,* pp. 35–44. SIAM, 1992.

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

[Barr, 1984] Barr, A. H. Global and local deformations of solid primitives. *Computer Graphics* 18(3), July 1984, pp. 21–30.

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

[Bloomenthal & Shoemake, 1991] Bloomenthal, J. and Shoemake, K. Convolution surfaces. *Computer Graphics* 25(4), July 1991, pp. 251–256.

[Bloomenthal, 1988] Bloomenthal, J. Polygonization of implicit surfaces. *Computer Aided Geometric Design* 5(4), Nov. 1988, pp. 341–355.

[Bloomenthal, 1989] Bloomenthal, J. Techniques for implicit modeling. Technical Report P89-00106, Xerox PARC, 1989. Appears in SIGGRAPH '93 Course Notes #25 "Design, Visualization and Animation of Implicit Surfaces".

[Bronsvoort & Klok, 1985] Bronsvoort, W. F. and Klok, F. Ray tracing generalized cylinders. *ACM Transactions on Graphics* 4(4), Oct. 1985, pp. 291–303.

[Gerald & Wheatley, 1989] Gerald, C. F. and Wheatley, P. O. *Applied Numerical Analysis*. Addison-Wesley, Reading, MA, 1989.

[Goldman, 1983] Goldman, R. N. Two approaches to a computer model for quadric surfaces. *IEEE Computer Graphics and Applications* 3(5), Sept. 1983, pp. 21–24.

[Hanrahan, 1983] Hanrahan, P. Ray tracing algebraic surfaces. *Computer Graphics* 17(3), 1983, pp. 83–90.

[Hart & DeFanti, 1991] Hart, J. C. and DeFanti, T. A. Efficient antialiased rendering of 3-D linear fractals. *Computer Graphics* 25(3), 1991.

[Hart *et al.*, 1989] Hart, J. C., Sandin, D. J., and Kauffman, L. H. Ray tracing deterministic 3-D fractals. *Computer Graphics* 23(3), 1989, pp. 289–296.

[Hart *et al.*, 1993] Hart, J. C., Francis, G. K., and Kauffman, L. H. Visualizing quaternion rotation. Manuscript, in review, 1993.

[Hart, 1994] Hart, J. C. Distance to an ellipsoid. In Heckbert, P., ed., *Graphics Gems IV*, pp. 113–119. Academic Press, 1994.

[Hoffman, 1989] Hoffman, C. M. *Geometric and Solid Modeling*. Morgan Kaufmann, 1989.

[Kalra & Barr, 1989] Kalra, D. and Barr, A. H. Guaranteed ray intersections with implicit surfaces. *Computer Graphics* 23(3), July 1989, pp. 297–306.

[Kaplansky, 1977] Kaplansky, I. *Set Theory and Metric Spaces*. Chelsea, New York, 1977.

[Kay & Kajiya, 1986] Kay, T. L. and Kajiya, J. T. Ray tracing complex scenes. *Computer Graphics* 20(4), 1986, pp. 269–278.

[Lewis, 1989] Lewis, J. P. Algorithms for solid noise synthesis. *Computer Graphics* 23(3), July 1989, pp. 263–270.

[Mitchell, 1990] Mitchell, D. P. Robust ray intersection with interval arithmetic. In Proc. of *Graphics Interface '90*. Morgan Kauffman, 1990, pp. 68–74.

[Nishimura *et al.*, 1985] Nishimura, H., Hirai, M., Kawai, T., Kawata, T., Shirakawa, I., and Omura, K. Object modeling by distribution function and a method of image generation. In Proc. of *Electronics Communication Conference '85*, 1985, pp. 718–725. (Japanese).

[Perlin & Hoffert, 1989] Perlin, K. and Hoffert, E. M. Hypertexture. *Computer Graphics* 23(3), July 1989, pp. 253–262.

[Porter & Duff, 1984] Porter, T. and Duff, T. Compositing digital images. *Computer Graphics* 18(3), 1984, pp. 253–259.

[Pratt, 1987] Pratt, V. Direct least-squares fitting of algebraic surfaces. *Computer Graphics* 21(4), July 1987, pp. 145–152.

[Ricci, 1974] Ricci, A. A constructive geometry for computer graphics. *Computer Journal* 16(2), May 1974, pp. 157–160.

[Rockwood & Owen, 1987] Rockwood, A. P. and Owen, J. C. Blending surfaces in solid modeling. In Farin, G., ed., *Geometric Modelling*, pp. 367–383. SIAM, 1987.

[Rockwood *et al.*, 1989] Rockwood, A., Heaton, K., and Davis, T. Real-time rendering of trimmed surfaces. *Computer Graphics* 23(3), July 1989, pp. 107–116.

[Rockwood, 1989] Rockwood, A. P. The displacement method for implicit blending surfaces in solid models. *ACM Transactions on Graphics* 8(4), Oct. 1989, pp. 279–297.

[Roth, 1982] Roth, S. D. Ray casting for modeling solids. *Computer Graphics and Image Processing* 18(2), February 1982, pp. 109–144.

[Sandin *et al.*, 1993] Sandin, D. J., Kauffman, L. H., and Francis, G. K. Air on the Dirac strings. *SIGGRAPH Video Review* 93, 1993. (Animation).

[Schneider, 1990] Schneider, P. J. Solving the nearest-point-on-curve problem. In Glassner, A. S., ed., *Graphics Gems (I),* pp. 607–611. Academic Press, Boston, 1990.

[Sederberg & Zundel, 1989] Sederberg, T. W. and Zundel, A. K. Scan line display of algebraic surfaces. *Computer Graphics* 23(3), July 1989, pp. 147–156.

[Stander & Hart, 1994] Stander, B. T. and Hart, J. C. A Lipschitz method for accelerated volume rendering. In Proc. of *Volume Visualization Symposium '94*, Oct. 1994. To appear.

[Taubin, 1994] Taubin, G. Distance approximations for rasterizing implicit curves. *ACM Transactions on Graphics* 13(1), Jan. 1994, pp. 3–42.

[Thomas *et al.*, 1989] Thomas, D., Netravali, A. N., and Fox, D. S. Antialiased ray tracing with covers. *Computer Graphics Forum* 8(4), December 1989, pp. 325–336.

[Thompson, 1990] Thompson, K. Area of intersection: Circle and a half-plane. In Glassner, A. S., ed., *Graphics Gems,* pp. 38–39. Academic Press, Boston, 1990.

[van Wijk, 1984] van Wijk, J. Ray tracing objects defined by sweeping a sphere. In Proc. of *Eurographics '84*. Elsevier, 1984, pp. 73–82.

[Von Herzen & Barr, 1987] Von Herzen, B. and Barr, A. H. Accurate triangulations of deformed, intersecting surfaces. *Computer Graphics* 21(4), July 1987, pp. 103–110.

[Von Herzen *et al.*, 1990] Von Herzen, B., Barr, A. H., and Zatz, H. R. Geometric collisions for time-dependent parameteric surfaces. *Computer Graphics* 24(4), Aug. 1990, pp. 39–48.

[Wyvill & Trotman, 1990] Wyvill, G. and Trotman, A. Ray tracing soft objects. In Proc. of *Computer Graphics International '90*. Springer Verlag, 1990.

[Wyvill *et al.*, 1986] Wyvill, G., McPheeters, C., and Wyvill, B. Data structure for soft objects. *Visual Computer* 2(4), 1986, pp. 227–234.

[Zuiderveld *et al.*, 1992] Zuiderveld, K. J., Koning, A. H. J., and Viergever, M. A. Acceleration of ray-casting using 3-D distance transforms. In Proc. of *Visualization in Biomedical Computing 1992,* vol. 1808, Oct. 1992, pp. 324–335.

A Distance to Natural Quadrics and Torus

These appendices derive signed distance functions, bounds and Lipschitz constants and bounds for a variety of primitives and operations in the hope that they will aid in the implementation of sphere tracing, while also serving as a tutorial in developing signed distance functions, bounds and Lipschitz constants and bounds for other primitives and operations.

Distances to the standard solid modeling primitives are listed below. The geometric rendering algorithm is not as efficient compared to the standard closed-form solutions. Instead, these distances are useful when the primitives are used in higher-order constructions such as blends and deformations.

Plane The signed distance to a plane P with unit normal $\boldsymbol{n}$ intersecting the point $r\boldsymbol{n}$ is

$$d(\boldsymbol{x}, P) = \boldsymbol{x} \cdot \boldsymbol{n} - r. \tag{20}$$

Sphere A sphere is defined as the locus of points a fixed distance from given point. The distance to the unit sphere S about at the origin hence given by

$$d(\boldsymbol{x}, S) = ||\boldsymbol{x}|| - 1. \tag{21}$$

Through domain transformations (Section E, the radius and location of the sphere may be changed. The sphere may even become an ellipsoid, though this reformulates the signed distance function into one requiring the solution to a sixth-degree polynomial [Hart, 1994]. Through alternate distance metrics (Section B), the sphere can become a superellipsoid. These techniques also generalize the rest of the basic primitives as well.

Cylinder The distance to a unit-radius cylinder centered about the z-axis is found by projecting into the xy-plane and measuring the distance to the unit circle

$$d(\boldsymbol{x}, Cyl) = ||(x, y)|| - 1. \tag{22}$$

Note that in (22), and throughout the rest of the appendix, $\boldsymbol{x} = (x, y, z)$.

Cone The distance to a cone centered at the origin oriented along the z-axis is

$$d(\boldsymbol{x}, Cone) = ||(x, y)|| \cos\theta - |z| \sin\theta, \tag{23}$$

where θ is the angle of divergence from the z-axis. The trigonometry behind its derivation is illustrated by Figure 13.

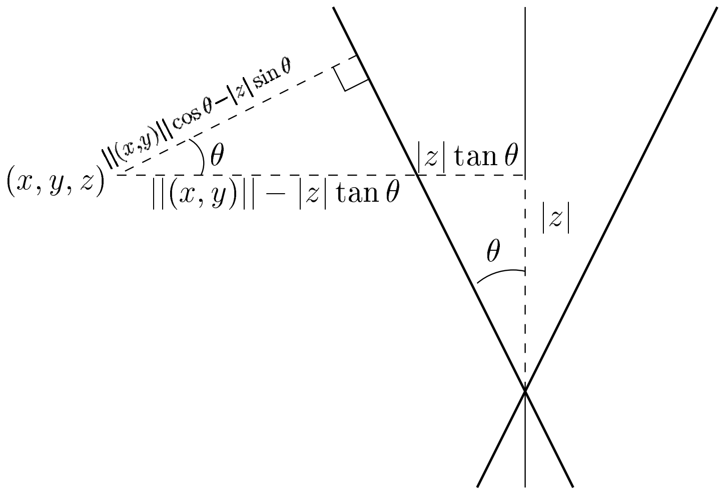

$||(x,y)|| \cos\theta - |z| \sin\theta$
θ
(x, y, z)
$||(x, y)|| - |z| \tan\theta$
$|z| \tan\theta$
$|z|$
θ

Figure 13: Geometry for distance to a cone.

Torus The torus is the product of two circles, and its distance is evaluated as such

$$d(\boldsymbol{x}, T) = ||(||(x, y)|| - R, z)|| - r \tag{24}$$

for a torus of major radius R and minor radius r, centered at the origin and spun about the z-axis.

B Distance to Superquadrics

Superquadrics [Barr, 1981] result from the generalization of distance metrics. Distance to the basic primitives all used the $||\cdot||$ operator. In two dimensions, this operator generalizes to the p-norm

$$||(x, y)||^p = (|x|^p + |y|^p)^{\frac{1}{p}} \tag{25}$$

which, when $p = 2$, becomes the familiar Euclidean metric whose *circle* is a round circle. The Manhattan metric $(p = 1)$ has a diamond for its *circle.* Taking the limit as $p \to \infty$ results in the chessboard metric

$$||(x, y)||^\infty = \max x, y \tag{26}$$

where a square forms its *circle.* The other intervening values for p produce rounded variations on these basic shapes, and setting $0 < p < 1$ produces pinched versions. Generalized spheres, so-called *superellipsoids,* are produced by a pq-norm as

$$||(x, y, z)||^{pq} = ||(\ ||(x, y)||^p,\ z)||^q. \tag{27}$$

The natural quadrics now generalize to superquadrics, and tori likewise become supertori, whose distances are measured in the appropriate metric. One unifying metric space must be used for the distances to be comparable. Hence, pq-norm distances must be converted into Euclidean distances.

Let $f(\boldsymbol{x})$ return a pq-norm distance to its implicit surface. This distance defines the radius of an unbounding superellipsoid. The radius of the largest Euclidean sphere r_e inscribed within the pq-norm superellipsoid of radius r_s (in the pq-norm metric) is given by

$$r_e = \begin{cases} r_s/||(\frac{\sqrt{3}}{3}, \frac{\sqrt{3}}{3}, \frac{\sqrt{3}}{3})||^{pq} & \text{if } p < 2 \\ r_s & \text{otherwise} \end{cases} \tag{28}$$

C Distance to Offset Surfaces

Given some closed skeleton geometry $S \subset \mathbb{R}^3$, then the *global* offset surface is defined geometrically by the implicit equation

$$d(\boldsymbol{x}, S) - r = 0. \tag{29}$$

Local offsets are defined parametrically using the normal of the skeleton geometry. Global offsets are the more desirable representation [Hoffman, 1989], and in particular avoid interior surfaces which can cause problems in ray-tracing and CSG [van Wijk, 1984].

The offset of an algebraic implicit surface is algebraic, though of higher degree in general. Several techniques have been developed to approximate offset surfaces with lower-degree representations. Treating offset surfaces geometrically overcomes the problems of dealing with high-degree algebraic representations and the loss of precision of low-degree approximations.

One useful skeletal model is the generalized cylinder, such as the fixed-radius global offset surface of a Bezier curve. Define the space curve parametrically as the image of the function $\boldsymbol{p} : \mathbb{R} \to \mathbb{R}^3$. Without loss of generality, assume the point from which we want to find the distance to the space curve is the origin.

Let $\boldsymbol{p}(u)$ define a cubic Bezier space curve. The point on the space curve closest to a given point $\boldsymbol{x}$ occurs either at one of the endpoints, or at point $\boldsymbol{p}(u)$ on the space curve such that

$$(\boldsymbol{x} - \boldsymbol{p}(u)) \cdot \boldsymbol{p}_u(u) = 0. \tag{30}$$

Equation (30) can be converted into a degree-five 1-D Bezier curve [Schneider, 1990], and can be solved efficiently using a technique described in [Rockwood *et al.*, 1989]. Such a generalized cylinder is demonstrated in Figure 8 in Section 4.2.

D Distance to Blended Objects

Blends smoothly join nearby objects, and have found applications in image synthesis and computer aided geometric design.

D.1 Soft Metablobbies

[Blinn, 1982] used a Gaussian distribution function to produce a blending function which has come to be known as the "blobby" model. "Soft" objects approximate Gaussian distribution with a sixth-degree polynomial to avoid exponentiation and localize the blends [Wyvill *et al.*, 1986]. "Metaballs" approximate Gaussian distributions with piecewise quadratics to avoid exponentiation and iterative root-finding [Nishimura *et al.*, 1985].

Following [Wyvill *et al.*, 1986], the following piecewise cubic in distance r

$$C_R(r) = \begin{cases} 2\frac{r^3}{R^3} - 3\frac{r^2}{R^2} + 1 & \text{if } r < R, \\ 0 & \text{otherwise.} \end{cases} \tag{31}$$

approximates a Gaussian distribution.

Reformulating this function to accommodate the implicit surface definitions in this paper, (31) forms the basis for a *soft* implicit surface consisting of n key points $\boldsymbol{p}_i$ with radii R_i, and threshold T, defined by the function

$$f(\boldsymbol{x}) = T - \sum_{i=1}^{n} C_{R_i}(||\boldsymbol{x} - \boldsymbol{p}_i||) \tag{32}$$

Negative keypoints are incorporated into the model by negating the value returned by $C_{R_i}()$.

Theorem 5 *The distance to the implicit blend B defined by (32) is bounded by*

$$d(\boldsymbol{x}, B) \geq \frac{2}{3} f(\boldsymbol{x}) \sum_{i=1}^{n} R_i. \tag{33}$$

Proof: Repeated differentiation of (31) produces

$$\begin{aligned} C'(r) &= 6\frac{r^2}{R^3} - 6\frac{r}{R^2} && (34) \\ C''(r) &= 12\frac{r}{R^3} - \frac{6}{R^2}. && (35) \end{aligned}$$

Solving $C''(r) = 0$ yields the maximum slope, which occurs at the midpoint $r = R/2.$ Its Lipschitz constant is given by

$$\operatorname{Lip} C(r) = |C'(R/2)| = \frac{3}{2R}. \quad (36)$$

The Lipschitz constant of a sum is bounded by the sum of the Lipschitz constants, which gives the above result. □

In practice, local Lipschitz bounds may be used for tighter distance bounds by taking the first summation in (33) over keypoints i with non-zero contributions. Additional efficiency results from the use of bounding volumes of radius R_i surrounding the keypoints $\boldsymbol{p}_i,$ as detailed in [Wyvill & Trotman, 1990].

D.2 Superelliptic Blends

The pseudonorm blend of [Rockwood & Owen, 1987] returns the p-norm distance to the blended union of implicit surfaces of signed distance functions. Hence, using the techniques from Appendix B, sphere tracing can render pseudonorm-blended surfaces, as demonstrated in Figures 7 and 9 in Section 4.2.

The pseudonorm blend creases the space surrounding the blend [Rockwood & Owen, 1987]. Such gradient discontinuities can be disastrous for some root finders, but do not impact sphere tracing.

E Distances to Transformed Objects

Implicit surfaces are transformed by applying the inverse transformation to the space before applying the function. Let $\boldsymbol{T}(\boldsymbol{x})$ be a transformation and let $f(\boldsymbol{x})$ define the implicit surface. Then the transformed implicit surface is defined as the implicit surface of

$$f(\boldsymbol{T}^{-1}(\boldsymbol{x})) = 0. \quad (37)$$

The Lipschitz constant of the composition is no greater than the product of the component Lipschitz constants. We are concerned with the Lipschitz constant of the transformation inverse, which is not necessarily the inverse of the Lipschitz constant of the transformation.

Isometry Isometries are transformations that preserve distances. If $\boldsymbol{I}$ is an isometry, the distance returned by f needs no adjustment

$$d(\boldsymbol{x}, \boldsymbol{I} \circ f^{-1}(0)) = d(\boldsymbol{I}^{-1}(\boldsymbol{x}), f^{-1}(0)). \quad (38)$$

Isometries include rotations, translations and reflections.

Uniform Scale A uniform scale is a transformation $\boldsymbol{S}(\boldsymbol{x})$ of the form

$$\boldsymbol{S}(\boldsymbol{x}) = s\boldsymbol{x} \tag{39}$$

where s is the scale factor. The inverse $\boldsymbol{S}^{-1}$ is a scale by $1/s$. Hence, the distance to a scaled implicit surface is

$$d(\boldsymbol{x}, \boldsymbol{S}(f^{-1}(0))) = sd(\boldsymbol{S}^{-1}(\boldsymbol{x}), f^{-1}(0)) \tag{40}$$

and the Lipschitz constant of the inverse scale is $1/s$.

Linear Deformation The distance to the linear image of an implicit surface is found by determining the Lipschitz constant of the linear transformation's inverse, which is also a linear transformation.

The Lipschitz constant of an arbitrary linear transformation is found by the power method, which iteratively finds the largest eigenvalue of a matrix [Gerald & Wheatley, 1989].

Taper The taper deformation scales two axes by a function $r(\cdot)$ of the third axis [Barr, 1984]. The taper is defined

$$\text{taper}\ (\boldsymbol{x}) = (r(z)x,\ r(z)y,\ z) \tag{41}$$

whereas its inverse differs only by using $r^{-1}(\cdot)$ instead of $r(\cdot)$. The Lipschitz constant of the inverse deformation is

$$\text{Lip taper}\ = \min_{z\in\mathbb{R}} r^{-1}(z). \tag{42}$$

In other words, the Lipschitz constant of the inverse taper is the amount of its "tightest" tapering.

Twist The twisting deformation rotates two axes by a linear function $a(\cdot)$ of the third axis. Twisting is defined

$$\text{twist}\ (\boldsymbol{x}) = \begin{pmatrix} x\cos a(z) - y\sin a(z), \\ x\sin a(z) + y\cos a(z), \\ z \end{pmatrix} \tag{43}$$

whereas its inverse differs only by using $a^{-1}(\cdot)$ instead of $a(\cdot)$. Twisting is not Lipschitz on $\mathbb{R}^n$ since for any Lipschitz bound λ one can find two points $\mathbb{R}^n$ at a great distance from the twisting axis that are transformed farther apart by a ratio greater than λ. Thus, twisting must be constrained to a domain where it satisfies the Lipschitz criterion. One such domain is the unit cylinder oriented along the twisting axis. The Lipschitz constant of the twist is computed from the worst case scenario within the bounds of the unit cylinder as illustrated in Figure 14,

$$\text{Lip twist}\ = \sqrt{4 + \left(\frac{\pi}{a'}\right)^2}. \tag{44}$$

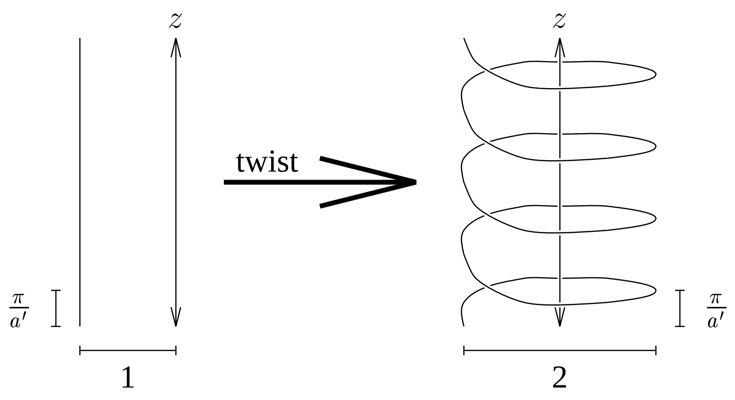


Figure 14: Geometric calculation of the Lipschitz constant of the bounded twist deformation.

F Distance to Hypertextures

The use of sophisticated noise functions has greatly increased the power of procedural models for making existing geometric representations more realistic. The recent work has applied stochastic textures directly to the geometry instead of altering the shading [Perlin & Hoffert, 1989; Lewis, 1989].

The original "Hypertexture" system formulated implicit models for a variety of surface phenomena, including hair and fire. This appendix focuses on incorporating hypertexture's model of noise into sphere tracing, though the same techniques can be used to adapt the other hypertexture models as well.

"Hypertexture" treats solid procedural noise as a deformation, and was designed for use with implicit surfaces. Its original ray-tracing algorithm stepped along the ray in fixed intervals. Determining a distance bound on a "hypertextured" shape allows sphere tracing to more efficiently render its result.

Band-limited solid noise results from the smooth interpolation of a lattice of random unit vectors. Condensing [Perlin & Hoffert, 1989], the noise function is given by

$$\text{noise}(x,y,z) = \sum_{k=\lfloor z\rfloor}^{\lfloor z\rfloor+1} \sum_{j=\lfloor y\rfloor}^{\lfloor y\rfloor+1} \sum_{i=\lfloor x\rfloor}^{\lfloor x\rfloor+1} C_1(|x-i|)C_1(|y-j|)C_1(|z-k|)\Gamma(i,j,k)\cdot(x-i,y-j,z-k) \quad (45)$$

where C_R is the cubic Gaussian approximation (31) used for soft objects, and Γ is an array of random unit vectors. From Theorem 5, we know that $\operatorname{Lip} C_1 = 3/2$. Two opposing vectors can be neighbors in Γ, so $\operatorname{Lip}\Gamma = 2$. Hence, their composition results in $\operatorname{Lip}\text{noise} = 3$.

Fractal noise is formed by summing scaled versions of the noise function

$$\text{noise}_\beta(\boldsymbol{x}) = \sum_{i=0}^{n-1} \frac{\text{noise}(2^i\boldsymbol{x})}{2^{\beta i}}. \quad (46)$$

over n octaves [Perlin & Hoffert, 1989].

For $\beta = 1$, the amplitude decreases proportionately to the increase in frequency, so its Lipschitz constant equals the sum of the individual noise functions,

$$\mathrm{Lip}\,\mathrm{noise}_{\beta=1} = 3n. \tag{47}$$

Thus $\beta = 1$ noise is not Lipschitz, but its band-limited form for finite n is.

For $\beta = 2$ noise, the amplitude decreases geometrically as the frequency increases, resulting in

$$\mathrm{Lip}\,\mathrm{noise}_{\beta=2} = 3(2 - \frac{1}{2^{n-1}}) \leq 6. \tag{48}$$

Hence, Brownian motion is Lipschitz (which can also be derived from the definition of Brownian motion as the integral of white noise).

Sphere tracings of noise-textured spheres appear in Figure 10 in Section 4.2.

Guaranteeing the Topology of an Implicit Surface Polygonization

Barton T. Stander John C. Hart
School of EECS
Washington State University
{bstander,hart}@eecs.wsu.edu

May 12, 1996

Abstract

An algorithm for guaranteeing topological correctness of the polygonization of an implicit surface is presented, using the critical points of a function. The methodology is extended to allow real-time guaranteed polygonized implicit surface modeling for simple scenes. For complicated scenes, some topological changes can still be detected and repolygonized in real time, but the guarantee is only re-established after a pause in user activity. The implemented system currently handles blobby ellipsoids, but could be extended to include any smooth, bounded implicit surface.

Keywords: implicit surfaces, polygonization, topology, critical points, interval analysis, interactive modeling, particle systems.

1 Introduction

Implicit surfaces have some modeling advantages over their parametric counterparts including the ability to easily be combined and blended in various fashions. Implicit surfaces are commonly polygonized to simplify their display and interrogation. Much effort has been made to ensure the consistency of the polygonization, to avoid surface holes, dangling polygons and other non-manifold anomalies.

In each of the two-dimensional examples in Fig. 1, the connectedness of the surface is undiscernable from the point samples. Numerous techniques have been developed to resolve ambiguities when the surface samples lack the necessary information to consistently select one configuration over another [Ning & Bloomenthal, 1993]. These disambiguation techniques correctly discern the configurations for only very low degree (linear, quadratic) surfaces. Interval analysis is capable of detecting and correctly resolving such situations (to a given degree of accuracy), but only after numerous subdivision steps over the entire surface [Snyder, 1992].

The *topology* of an implicit surface refers to the number of disjoint components together with the *genus* (or number of holes) of each component. Thus, a topologically correct polygonization to

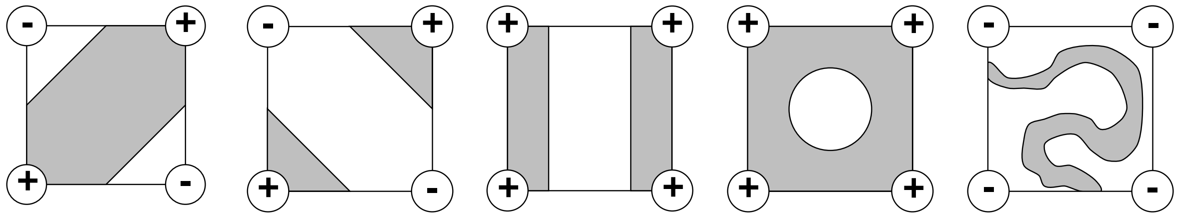


Figure 1: Cases where point samples provide insufficient information for correct topological identification.

an implicit surface must share the same number of components, each component having the same genus as its corresponding component in the true surface.

Catastrophe theory deals with cases where a small change in the domain space causes a significant alteration in the resulting range space. Similarly, polygonization must accomodate the minute surface details that completely change connectedness when perturbed.

The *Crowbar Principle* of catastrophe theory [Gilmore, 1981] states that to pry the information out of a problem, one needs to look at the "cracks" — the exceptions. To guarantee the topology of an implicit surface polygonization, we look at the points where the gradient of the implicit surface vanishes, the *critical points*.

The analysis of topology using critical points is not entirely new to computer graphics. Critical points of vector and tensor fields are used to delineate topologically-distinct regions in the visualization of flow [Helman & Hesselink, 1991; Delmarcelle & Hesselink, 1994].

Section 3 describes a technique that guarantees the topology of the polygonization matches the topology of the implicit surface. An interval-analysis search finds the critical points of the implicit surface. These critical points dictate the topology of the surface. During polygonization, these critical points are used to correctly resolve topologically ambiguous cases.

Implicit surfaces are difficult to model in an interactive environment due to real-time display problems and the indirect relationship between the control parameters and the resulting shape. Witkin and Heckbert ameliorated these problems by representing the implicit surface with a collection of oriented surface particles [Witkin & Heckbert, 1994]. Oriented surface particles circumvent the need for a topological guarantee. One infers a surface from their appearance, even though the particles themselves are totally disconnected. The natural next step to this research is the real-time polygonization of these particles, but this requires a topological guarantee.

Section 4 describes a technique for repolygonization of an implicit surface. Topological correctness requires not only the tracking of surface particles but also the tracking of critical points. In fact, during manipulation, critical points are created and destroyed in pairs. The techniques in this paper can track critical points and detect their destruction in real-time. The process of detecting newly created critical points is numerically troublesome, and its searching step may require a pause in user interaction to obtain a topological guarantee for complicated functions.

1.1 Implicit Surfaces

An implicit surface is the set of points $\mathbf{x} \in \mathbf{R}^3$ that satisfy $F(\mathbf{x}) = 0$. We define implicit surfaces such that $F(x) > 0$ indicates $\mathbf{x}$ is inside the surface, and $F(x) < 0$ means $\mathbf{x}$ is outside. This convention agrees with popular implicit surface applications in image-synthesis, but is opposite of the convention used in computer-aided geometric design.

This research focuses on the so-called blobby model as the basis of its implicit surface, although it also applies to any continuously differentiable and bounded surface. The blobby model was developed from a model of the electron density maps in molecular structures but has evolved into a general modeling tool [Blinn, 1982; Wyvill *et al.*, 1986]. It defines the implicit surface as the solution of

$$F(\mathbf{x}) = -1 + \sum_{i=1}^{n} e^{kf(\mathbf{q}_i,\mathbf{x})} = 0. \tag{1}$$

where the $f(\mathbf{q}_i, \mathbf{x})$ are the n blobby primitives and $\mathbf{q}_i$ is a vector of the primitives parameters. The constant k is a negative number which determines the sharpness of the blend. Spherical primitives are used for the sake of simplifying the discussion, but the current implementation readily handles blobby ellipsoids. Spherical primitives are defined implicitly by the function

$$f_i(\mathbf{q}_i, \mathbf{x}) = (\mathbf{x} - \mathbf{o}_i)^2 - r_i^2, \tag{2}$$

where the parameter vector $\mathbf{q}_i = (\mathbf{o}_i, r_i)$ holds the sphere's center $\mathbf{o}_i$ and radius r_i.

Several variations on blobby surfaces use much faster piecewise polynomial approximations of the Gaussian distribution [Nishimura *et al.*, 1985; Wyvill *et al.*, 1986]. This research uses the exponential form instead of the piecewise polynomial forms to better understand the analytical properties of the critical points. The following methods require a function that is once differentiable with respect to its control parameters and twice differentiable with respect to its spatial variable. With careful tracking of the piecewise domains, the following techniques could be extended to the polynomial approximations. Nonetheless, for the sake of simplicity and proof-of-concept, this work focuses on the infinitely-differentiable exponential form and postpones its implementation on piecewise domains for future work.

2 Previous Work

Our interactive repolygonization method for implicit surface manipulation combines tools from several fields of research: polygonization, particle systems, and mesh optimization. We also review other surface modeling strategies.

2.1 Voxel-Based Polygonization

Whereas parametric surfaces lend themselves to forward visible-surface algorithms, such as the z-buffering, implicit surfaces are better suited for backward algorithms, such as ray tracing. As many graphics workstations implement z-buffering in hardware but few similarly support ray tracing, real-time manipulation demands forward rendering.

The implicit formulation enjoys many benefits, though its main drawback is that its surface is difficult to interrogate. Hence fast "forward" rendering typically requires a polygonization step.

Polygonization algorithms typically interrogate implicit surfaces through spatial sampling. There are several spatial sampling techniques for interrogating implicit surfaces, and all divide space into cells and search for only those cells that intersect the implicit surface. Ning and Bloomenthal review several variations on this theme [Ning & Bloomenthal, 1993], focusing on topological consistency and correctness, although none of their surveyed algorithms guarantees topological correctness.

Surface tracking techniques [Norton, 1982; Wyvill *et al.*, 1986] partition space into small cells and begin with a cell known to straddle the implicit surface (some vertices in, some out). The technique then recursively (or iteratively) finds straddling cells among its neighbors until all neighbors have been checked, yielding a collection of cells enclosing the geometry of the implicit surface.

Spatial subdivision techniques [Bloomenthal, 1988; Kalra & Barr, 1989; Snyder, 1992] begin with one large cell known to bound the implicit surface, then repeatedly subdivide cells intersecting the implicit surface. Lipschitz bounds or interval analysis can guarantee that the implicit surface is bounded by the resulting cells, hence yielding a guarantee on surface topology.

Spatial interrogation techniques, particularly in guaranteed form, remain too costly for real-time use.

2.2 Particle-Based Polygonization

Reeves invented particle systems as a tool for modeling various natural phenomena such as fire, clouds, and water [Reeves, 1983], and they have since been used in many other applications. When modeling implicit surfaces, an alternative to spatial interrogation constrains a particle system to the implicit surface. Such physically-based approaches tend to be much faster at interrogating an implicit surface. The technique scatters particles randomly throughout space and then forces them to migrate to the implicit surface using its defining function's sign and gradient direction. Once the particles reach the surface, they repel each other in order to achieve a more uniform sampling distribution [Turk, 1991; Bloomenthal & Wyvill, 1990; de Figueiredo *et al.*, 1992].

Szeliski and Tonnesen model free-form surfaces using oriented particles [Szeliski & Tonnesen, 1992]. Whereas particles in other systems stick to a mathematically-defined surface or to a polygonal surface, their particles form surfaces in 3D space by aligning their orientations with that of their neighbors. They also provide a toolkit to split, join, and otherwise manipulate their free-form surfaces.

In an interactive environment, such sample points may "fall off" when the user quickly alters the surface, and otherwise find it difficult to maintain a balanced distribution. Witkin and Heckbert [Witkin & Heckbert, 1994] overcame these problems by solving for particle velocities in terms of surface parameter velocities q. This resulted from the key observation that $\partial F/\partial q$ is continuous. This kept the particles on the surface, nearly eliminating the dependency on "feedback" terms. They balance the distribution of sample points by introducing an intricate particle "birth and death" scheme.

Witkin and Heckbert also provide a direct manipulation interface by allowing the user to choose one or more "control particles" which also reside directly on the surface. The user drags or nails these control particles, which in turn alter the actual surface parameters. They keep the surface on the control particles by solving the reverse problem from above. That is, they solve for surface parameter velocities in terms of control point velocities. In both cases, a differential equation solver and a feedback term keep the particles and the actual surface exactly in sync.

The rendering technique used by Witkin and Heckbert's algorithm consists of drawing each particle as a disk oriented tangent to the surface. This gives a somewhat useful representation of the underlying surface, but it lacks the important depth cues of hidden surface removal, and topological information is not always discernible. Furthermore, a topologically-interesting component such as a disjoint piece or a hole inside of a known piece may be missed altogether if the initial set of randomly placed particles never found it, or if it came into existence later at a place disjoint from the known components.

Particle system approaches swiftly interrogate the surface with sample points, but "connecting the dots" can easily kill an otherwise interactive environment. Figueiredo, for example, uses Delaunay triangulation to build a polygonal mesh [de Figueiredo *et al.*, 1992]. The result is not interactive.

2.3 Other Surface Interrogation Techniques

Sederberg and Zundel used silhouette edges and surface intersection points to quickly render algebraic surfaces of any degree using a scanline approach [Sederberg & Zundel, 1989]. Their algorithm guaranteed correct topology, even where singularities occur.

Desbrun, Tsingos, and Gascuel interactively displayed surfaces generated by skeletons [Desbrun *et al.*, 1995]. Each skeleton sends out an ordered group of "seeds" which migrate along predefined vectors until they intersect the surface. Where two skeletons intersect, some seeds temporarily become invalid. The display can be either scales on the surface or a piecewise polygonization. Their approach also lends itself to avoiding unwanted blending, and to partitioning the surface into local bounding boxes. This method handles topology changes, though not necessarily in a guaranteed fashion due to finite sampling.

3 Guaranteeing Topology

The critical points of a function completely determine the function's qualitative properties [Gilmore, 1981]. This section introduces critical points and demonstrates how they are used to determine the validity of a given polygonization of some function. The next section describes how critical points are used to detect topology changes in an interactive environment.

3.1 Critical Points

The critical points of a function occur where its gradient

$$\nabla F(\mathbf{x}) = (F_x(\mathbf{x}), F_y(\mathbf{x}), F_z(\mathbf{x})) \tag{3}$$

vanishes. The stability matrix V is the Jacobian of the gradient, defined

$$V(\mathbf{x}) = J(\nabla F(\mathbf{x})) = \begin{bmatrix} F_{xx}(\mathbf{x}) & F_{xy}(\mathbf{x}) & F_{xz}(\mathbf{x}) \\ F_{yx}(\mathbf{x}) & F_{yy}(\mathbf{x}) & F_{yz}(\mathbf{x}) \\ F_{zx}(\mathbf{x}) & F_{zy}(\mathbf{x}) & F_{zz}(\mathbf{x}) \end{bmatrix}. \tag{4}$$

Since $F_{xy} = F_{yx}$, etc., the stability matrix is symmetric.

Each critical point is either a maximum, minimum or a saddle point. Furthermore, In three dimensions, saddle points come in two varieties, which we call type I and type II saddle points. The critical points are classified based on the sign of the eigenvalues of the stability matrix $V(\mathbf{x})$ [Taylor, 1955].

Let l_1, l_2 and l_3 be the three eigenvalues of V evaluated at a critical point $\mathbf{x}$. Then

- $\mathbf{x}$ is a maximum point iff all three eigenvalues are negative.
- $\mathbf{x}$ is a type I saddle point iff only two of the eigenvalues are negative.
- $\mathbf{x}$ is a type II saddle point iff only one eigenvalue is negative.
- $\mathbf{x}$ is a minimum point iff all three eigenvalues are positive,
- If any of l_1, l_2 and l_3 are zero then a critical point pair is created or destroyed, the critical point is degenerate, and the results of this classification scheme are undetermined.

If any of the eigenvalues are zero, the critical point is degenerate. Degenerate critical points signal a change in the number of critical points.

3.2 Finding Critical Points

An interval analysis search finds all of the critical points. Interval arithmetic operates on interval numbers rather than on single values [Moore, 1966; Ratschek & Rokne, 1984]. In this manner, large portions of space can be determined to have certain properties. An interval search can be guaranteed to find all critical points in a given bounded domain.

The interval search for critical points starts with an initial box bounding the space of interest. Blobbies are designed to have a limited range and then rapidly fall off to near zero, and hence their area of interest is easily bounded (e.g. to three standard deviations [Witkin & Heckbert, 1994]).

A simple interval search for critical points eliminates large portions of space that cannot contain a critical point. Given a box $X = [x_0, x_1] \times [y_0, y_1] \times [z_0, z_1]$ the algorithm checks whether the intervals returned by any of the partial derivatives $F_x(X), F_y(X), F_z(X)$ might contain zero. If not, then X contains no critical points. If so, then the algorithm halves X in its widest direction, and tests each half separately.

Simple subdivision performs remarkably well, discarding large portions of space known not to contain critical points. This technique eventually finds all critical points to any degree of accuracy within a given bounding box. However, it is only linearly convergent.

When the box size is sufficiently small, a quadratically-convergent interval Newton method refines or further subdivides the box to the desired numerical precision. Each iteration of the scalar Newton's method adjusts the previous guess by evaluating the function and following its slope back to the x-axis. The three-dimensional interval version of Newton's method determines "slope" by inverting a 3×3 interval Jacobian matrix [Snyder, 1991; Hansen & Greenberg, 1983].

Given initial box X, the algorithm successively seeks for smaller boxes $X' \subset X$ such that any roots of ∇F contained in X are also contained in X'. Taylor's Theorem asserts that given 2 points $\mathbf{x}, \mathbf{y}$ there exist points $\mathbf{z}$ between $\mathbf{x}$ and $\mathbf{y}$ such that

$$\nabla F(\mathbf{x}) + V(\mathbf{z})(\mathbf{y} - \mathbf{x}) = \nabla F(\mathbf{y}), \tag{5}$$

where V (the stability matrix) is the Jacobian of ∇F. Let $\mathbf{x}$ be the midpoint of box X,. The algorithm seeks $\mathbf{y} \in X$ such that $\nabla F(\mathbf{y}) = 0$. Since $\mathbf{x}$ and $\mathbf{y}$ are both in X, the $\mathbf{z}$ satisfying (5) must be in X as well. Thus, solve

$$\nabla F(\mathbf{x}) + V(X)(\mathbf{y} - \mathbf{x}) = \nabla F(\mathbf{y}) = 0 \tag{6}$$

for $\mathbf{y}$. Since box X is an interval vector, the solution set Y containing $\mathbf{y}$ is also an interval vector, so the actual equation to be solved is

$$\nabla F(\mathbf{x}) + V(X)(Y - \mathbf{x}) = 0. \tag{7}$$

for Y.

Any roots in X must also be in Y, and the algorithm refines X into $X' = X \cap Y$. If $X' = \emptyset$, then there were no solutions in X. If $X' = X$ then subdivide X and continue recursively[1]. This iteration continues until X reaches the desired precision.

Solving for Y in (7) can be troublesome for two reasons. First, the diagonal elements of V(X) might contain zero, resulting in the necessity of extending the interval arithmetic division operation to correctly perform a division by an interval containing zero [Snyder, 1991; Hansen, 1978].

The second difficulty arises from the large numbers of interval operations involved, which can lead to an “interval explosion” – a gross overestimate of the range of the result. The following algorithm [Hansen & Greenberg, 1983] appears to work well, except for their “inner iteration optimization,” which increased the execution time for blobby functions.

1. Let the real matrix V_c, be comprised of the midpoints of interval matrix $V(X)$.
2. Compute $B = V_c^{-1}$. (If V_c has no inverse, subdivide X and start over.)
3. Multiply both sides of (7) by B, obtaining

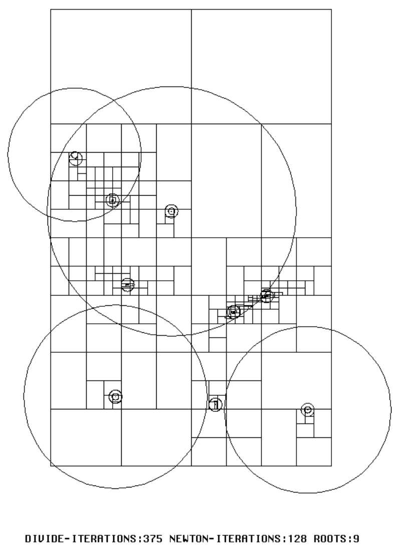
$$\mathbf{b} = M(Y - \mathbf{x}) \tag{8}$$

   where $M = BV(X)$ (which approximates the identity matrix), and $\mathbf{b} = -B\nabla F(\mathbf{x})$.
4. Solve (8) for Y using an interval version of the Gauss-Seidel method[2]

The second derivatives comprising the Jacobian of an exponential function can get quite large, making the quadratically-convergent Newton’s method much slower than its performance on other functions, such as polynomials. In fact, it performs worse than simple subdivision when the intervals are not very small. For this reason, the simple subdivision method executes first until the intervals subdivide down to a predefined width, at which point the interval Newton’s method takes over the search.

[1]Snyder proves that if Y is a proper subset of X then X contains a unique solution to ∇F [Snyder, 1991]. This property can terminate the search early if X contains a known critical point, but the experimental gain is minimal, reducing the number of operations only by about 1%. Nonetheless, this result can prove that there are not two distinct but extremely close critical points within an interval.

[2]It is recommended [Hansen & Greenberg, 1983] to solve the rows whose diagonal elements (intervals) do not contain zero first to reduce the occurrence of semi-infinite intervals. When such an interval inevitably arises, its intersection with X can yield two new boxes, both of which must be processed individually.



```
DIVIDE-ITERATIONS:375 NEWTON-ITERATIONS:128 ROOTS:9
```

Figure 2: Example of the interval critical point search convergence in two dimensions.

Interval arithmetic finds all critical points interactively for simple configurations. In the 2-D example in Figure 2, the algorithm finds all 9 critical points in 1/18th of a second.

A common problem occurs when the desired point is exactly on an edge of the box. This can result in the loss of quadratic convergence. Extending the intervals outward by small, random amounts avoids this problem.

3.3 Determining Topology

The sign of F at the critical points dictate the topology of the implicit surface. Assuming that the system already has a consistent polygonization of one or more connected components, and that each point of the polygonization is constrained to lie on the surface, the problem reduces to determining if the components are topologically accurate.

The Jordan curve theorem determines if the critical points are inside or outside with respect to the polygonization whereas simple function evaluation determines if they are inside or outside with respect to the implicit surface. If a critical point's status with respect to the polygonization disagrees with its status with respect to the implicit surface, then the polygonization must be corrected in the neighborhood of the critical point.

Table 1 enumerates all of the possible critical-point/sign combinations and their corresponding implications on the implicit surface topology.

4 Interactive Repolygonization

The use of critical points simplifies topologically guaranteed, direct manipulation of implicit surfaces through a polygonal representation. The key to solving the topology problem is that a change

Critical Point	Sign	Indication
Maximum	-	no component
Maximum	+	a component
Saddle (Type I)	-	disconnection of two components
Saddle (Type I)	+	connection of two components
Saddle (Type II)	-	a hole
Saddle (Type II)	+	no hole
Minimum	-	a hollow pocket
Minimum	+	no hollow pocket

Table 1: The affect of critical point sign on topology.

in the topology of a surface is always accompanied by a change in the sign of the function's value at one of the critical points (i.e., points where the gradient vector vanishes). By watching the critical points, the heavy burden of detecting topological change is greatly simplified.

The interaction algorithm consists of an initialization stage followed by an interactive loop of user input, model update, and model display. The system is initialized with several particles distributed on each component of the true surface, correctly connected to each other to create closed polyhedral components. The problem then becomes one of maintaining this correct picture during user interaction.

For each time step, the interaction algorithm performs the following steps:

1. Allow the user to modify the surface with the mouse, and alter the implicit surface's parameters accordingly [Witkin & Heckbert, 1994].

2. Solve for the updated positions of the particles so that they stay exactly on the updated surface [Witkin & Heckbert, 1994].

3. Solve for the updated positions of the critical points (Section 4.1).

4. Delete pairs of critical point that collide.

5. Detect any topology changes using the list of critical points. That is, if the functional value of any of them change sign over a time step then the topology also changes in the time step about that point.

6. Correct any topology changes (Section 4.2).

7. Perform an interval analysis search for newly created critical points if time permits (Section 3.2).

8. Maintain a consistent polygonization (Section 4.3).

9. Render the polygonized components.

During user interaction, the critical points move and change sign. Furthermore, one or more of the eigenvalues of the stability matrix can change sign at some degenerate critical point $\mathbf{x}$, resulting in the creation or annihilation of a pair of critical points. Critical points annihilation is easily detected as the collision of two tracked critical point particles. Critical point creation can happen anywhere, and relies on the interval critical-point search for detection.

The interval critical-point search operates at interactive rates (10 frames per second) for simple scenes containing three or four blobby ellipsoids. For more complex scenes, this process is performed only at initialization and when user activity subsides. The user is informed of the topological "honesty" of the polygonization by a "streetlight," where a yellow light indicates caution and a green light guarantees the topology of the displayed polygonization.

4.1 Tracking Critical Points

Altering the implicit surface parameters changes the positions of some or all of the critical points. Local searching approaches to updating the critical point positions using gradient descent or Newton's method fail to converge to the correct critical point when the step size is too large.

Instead, an extension of the techniques in [Witkin & Heckbert, 1994] solve for critical point velocities in terms of parameter velocities. Let $\mathbf{x}$ be a particle constrained to follow one of the critical points of the function F from (1). Then its partial derivative with respect to x

$$F_x(\mathbf{x}) = \sum_{i=1}^{n} e^{kf(\mathbf{q}_i,\mathbf{x})} k f_x(\mathbf{q}_i, \mathbf{x}), \tag{9}$$

must be zero. Likewise $F_y(\mathbf{x}) = 0$ and $F_z(\mathbf{x}) = 0$. (For spherical primitives $f_x(\mathbf{q}_i, \mathbf{x}) = 2(x - o_{ix}.)$

As time t increases from t_0 to t_1, some or all of the parameters $\mathbf{q}$ of F may change. In order to constrain the particle $\mathbf{x}$ to track the critical point, its change $\dot{\mathbf{x}}$ must be expressed in terms of the changes in the parameters $\dot{\mathbf{q}}$.

In order to ensure that (9) remains constant (at zero), its derivative with respect to time

$$\dot{F}_x(\mathbf{x}) = \sum_{i=1}^{n} k e^{kf(\mathbf{q}_i,\mathbf{x})} \left(\dot{f}_x(\mathbf{q}_i, \mathbf{x}) + k f_x(\mathbf{q}_i, \mathbf{x}) \dot{f}(\mathbf{q}_i, \mathbf{x}) \right), \tag{10}$$

must also be zero, where

$$\begin{aligned} \dot{f}(\mathbf{q}_i, \mathbf{x})) &= 2(\mathbf{x} - \mathbf{o}_{ix})(\dot{\mathbf{x}} - \dot{\mathbf{o}}_{ix}) - 2r\dot{r}, & (11) \\ \dot{f}_x(\mathbf{q}_i, \mathbf{x})) &= 2(\dot{x} - \dot{o}_{ix}). & (12) \end{aligned}$$

Likewise $\dot{F}_y(\mathbf{x}) = 0$ and $\dot{F}_z(\mathbf{x}) = 0$. These three equations are linear in the three unknowns of $\dot{\mathbf{x}}$, and their solution yields the change in the critical points position.

Using these velocities, a differential equation solver (such as fourth-order Runge-Kutta) approximates the new location of the critical point, and Newton's method refines the solution.

4.2 Correcting Topology

If it is determined that a critical point's value changed sign, then the topology of the true surface is also altered, and the polygonal approximation must be updated to reflect that change.

In two dimensions, whenever two components merge, or when one component separates, the alteration always occurs about the saddle point. In both cases, exactly two segments become invalid (i.e., they attempt to approximate pieces of true surface which are no longer there). The two offending segments are detected by examining all segments in the vicinity of the saddle point. The four points involved are reconnected in the other direction (Figure 3).

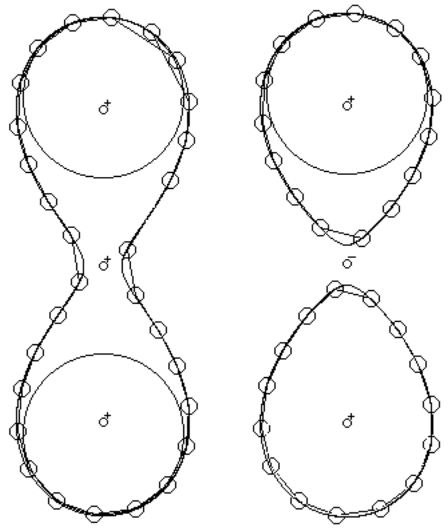


Figure 3: In two dimensions, when the functional value of a saddle point changes sign from positive (left) to negative (right), one component separates into two. The 4 nearby points are reconnected to contain valid segments.

When a maximum point changes sign from negative to positive (this cannot happen with circle primitives, but can happen with ellipses) a new component is formed, and several connected particles are placed on it. When a minimum point changes sign from positive to negative, a hole is formed, and it is likewise seeded with a collection of connected particles (Figure 4). When a component or a hole ceases to exist, all particles that were on it are deleted.

In three dimensions, the situation is the same when a maximum point's value changes sign. When a minimum point's value goes negative, a hollow pocket is formed. This is different from a torus-type hole (discussed shortly), and it cannot be observed unless front clipping is used.

When a type I saddle point's value changes sign, two components are merged or separated. Figure 5 shows a configuration involving four blobby spheres. There are 4 maximum points, 4 type I saddle points and 1 type II saddle point. The value of the upper type I saddle point is positive, so the two maximum points it lies between are connected into 1 component. The other three type I saddle points have negative functional values, so the blobs they lie between are disconnected. If all of the type I saddle points in Figure 5 were positive, and the type II saddle point were negative, then the topology would be that of a torus. When a type II saddle point changes sign from negative to positive, the torus hole is filled in, forming a disk. The type II saddle point will not become positive until all of the maximum and type I saddle points surrounding it become positive first.

When a type I saddle point goes from negative to positive, two triangles are deleted and as few as 6 new triangles can reconnect their particles. A sign change from positive to negative is handled in the reverse manner, deleting the triangles connecting the components.

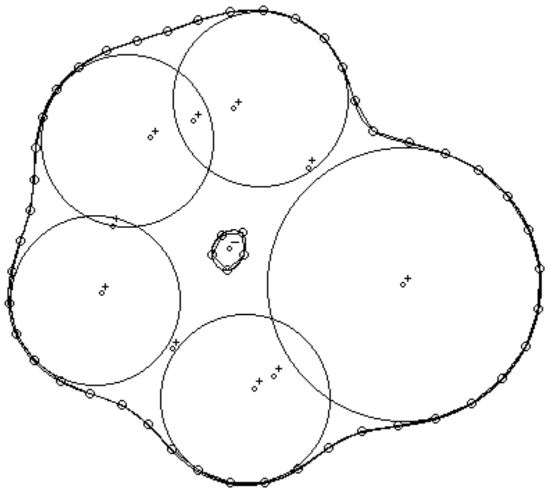


Figure 4: In two dimensions, when the functional value of a minimum point changes sign from positive to negative, a “lake” is created.

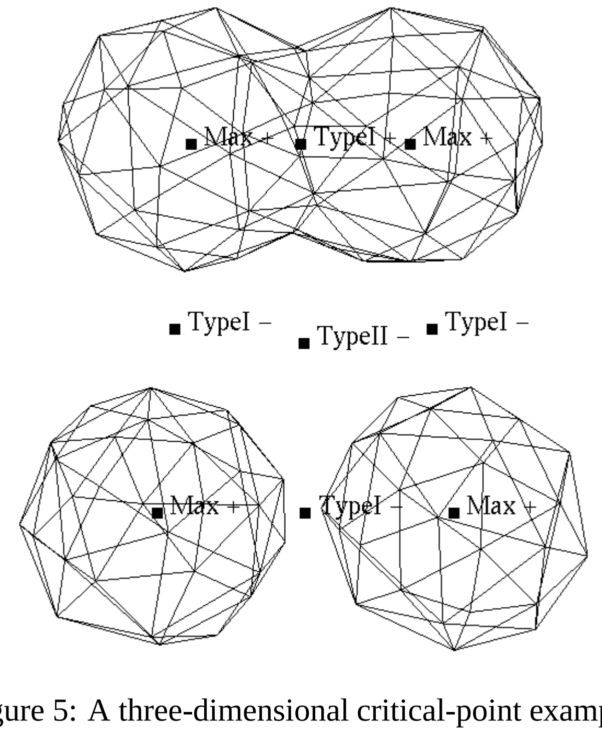


Figure 5: A three-dimensional critical-point example.

When a type II saddle point goes from positive to negative, again, two triangles are deleted and as few as 6 new triangles can reconnect their particles. The two deleted triangles remove the top and bottom of the hole and the new triangles form the tube of the hole. When a Type II saddle point goes from negative to positive, the process is reversed, deleting the triangles forming the hole and reconnecting the two ends of the hole.

The eigenvectors of the stability matrix at the saddle points form one line and one plane (depending on the type of saddle point) which intersect the polygonal regions that need to be rearranged. This simplifies the otherwise tedious task of finding the offending polygons.

4.3 Maintaining Geometry

If left unchecked, the geometry of the polygonal mesh degenerates as particles float about the surface. The resulting triangulation becomes inefficient and sometimes corrupt.

Several researchers have developed mesh optimization schemes [Hoppe *et al.*, 1993; Turk, 1992]. These techniques reduce the number of triangles in a polygonization while remaining true to the original topology and close to the original geometry. They employ edge addition, edge swapping, and edge deletion to optimize the polygonization. Where guaranteed bounds on geometry are required, van Overveld and Wyvill provide an in-depth qualitative analysis of a triangulation's accuracy, including error bounds [van Overveld & Wyvill, 1993].

Such an optimization scheme involving edge splitting, collapsing, and swapping keeps the geometry in good form [Hoppe *et al.*, 1993; Welch & Witkin, 1994]. If one particle passes across the opposite edge of one of its triangles, that triangle becomes invalid. This seldom happens when time steps are small because the optimization algorithm performs a swap when a particle gets close to an edge. Testing the sign of the dot product of the polygon normals with the normals of the surface quickly identifies this kind of error.

Another error which is more difficult to detect occurs when a particle on one component jumps to a nearby component, thus violating the integrity of mesh geometry and topology. This is mostly avoided by taking small steps and by updating particles with a high order differential equation solver such as Runge-Kutta.

In the current implementation, a particle's movement is constrained to be less than the distance to its closest neighbor. Experimentation indicates that this bound is sufficient to eliminate the above problems, though a tighter and provably robust bound is desired.

5 Conclusion

In summary, the work presented here provides a guarantee of topological correctness of an implicit surface polygonization. It also extends the interactive modeling system of Witkin and Heckbert to include connectivity information. The final rendering consists of polygons instead of points or disks, and with the guarantee (not necessarily interactively) that the topology of the polygonization is consistent with the topology of the actual implicit surface.

Techniques for detecting and tracking the critical points of an everywhere-smooth exponential function have been developed. Given enough time, the techniques can insure the topology of an implicit surface agrees with its polygonization, but can not currently maintain this guarantee in real time.

5.1 Implementation

Interactive environments of well over 10 frames per second, including topology maintenance, were obtained for simple scenes on a 200 MHz Silicon Graphics R4400 Indigo^2 XZ24 workstation. Scenes involving 6-8 blobby ellipsoids and two to three hundred particles run at about 2 frames per second when immediate topology verification is on, but approach 10 frames per second with topology verification suspended to when the user pauses.

5.2 Future Work

A robust and efficient method for avoiding the particle jumping problem of section section 4.3 is under development.

We have implemented a four-dimensional version of this methodology, which actually solves for the exact instant in time and location in space when the value of a critical point changes sign. This solution is much cleaner because critical points do not have to be chased around. It never misses a critical point's value changing sign even when they appear and disappear. Unfortunately, it is too slow to be interactive, largely due to the double root involved when demanding that a function and its derivatives are simultaneously zero.

Future work also includes extending the environment to include additional kinds of implicit surfaces. Given a degree d polynomial basis, tracking the critical points of its d derivatives would yield a topological guarantee without the use of interval analysis. However, the piecewise nature of polynomial bases interferes with the analytic properties required by the current system.

5.3 Acknowledgments

This research was supported in part by the NSF Research Initiation Award #CCR-9309210. This research was performed in the Imaging Research Laboratory. The authors would like to thank the SIGGRAPH '96 reviewers for rejecting this paper with constructive criticism and positive comments. The simplest of these changes have already made it into this version of the manuscript. Futher thanks are due to Dan Asimov, Jules Bloomethal and Jim Kajiya for there help in tracking down theorems in Morse theory, differential topology and combinatorial topology.

References

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

[Bloomenthal & Wyvill, 1990] Bloomenthal, J. and Wyvill, B. Interactive techniques for implicit modeling. *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, March 1990, pp. 109–116.

[Bloomenthal, 1988] Bloomenthal, J. Polygonization of implicit surfaces. *Computer Aided Geometric Design* 5(4), Nov. 1988, pp. 341–355.

[de Figueiredo *et al.*, 1992] de Figueiredo, L. H., de Miranda Gomes, J., Terzopoulos, D., and Velho, L. Physically-based methods for polygonization of implicit surfaces. In *Proceedings of Graphics Interface '92*, May 1992, pp. 250–257.

[Delmarcelle & Hesselink, 1994] Delmarcelle, T. and Hesselink, L. The topology of symmetric, second-order tensor fields. *Proceedings IEEE Visualization '94*, October 1994, pp. 140–147.

[Desbrun *et al.*, 1995] Desbrun, M., Tsingos, N., and Gascuel, M.-P. Adaptive sampling of implicit surfaces for interactive modeling and animation. *Implicit Surfaces '95 Proceedings*, April 1995, pp. 171–185.

[Gilmore, 1981] Gilmore, R. *Catastrophe Theory for Scientists and Engineers*. John Wiley and Sons, 1981.

[Hansen & Greenberg, 1983] Hansen, E. R. and Greenberg, R. I. An interval newton method. *Applied Mathematics and Computation* 12, 1983, pp. 89–98.

[Hansen, 1978] Hansen, E. A globally convergent interval method for computing and bounding real roots. *BIT* 18, 1978, pp. 415–424.

[Helman & Hesselink, 1991] Helman, J. L. and Hesselink, L. Visualizing vector field topology in fluid flows. *IEEE Computer Graphics and Applications*, May 1991, pp. 36–46.

[Hoppe *et al.*, 1993] Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., and Stuetzle, W. Mesh optimization. In Kajiya, J. T., ed., *Computer Graphics (SIGGRAPH '93 Proceedings)*, vol. 27, August 1993, pp. 19–26.

[Kalra & Barr, 1989] Kalra, D. and Barr, A. H. Guaranteed ray intersections with implicit surfaces. *Computer Graphics* 23(3), July 1989, pp. 297–306.

[Moore, 1966] Moore, R. E. *Interval Analysis.* Prentice Hall, 1966.

[Ning & Bloomenthal, 1993] Ning, P. and Bloomenthal, J. An evaluation of implicit surface tilers. *Computer Graphics and Applications* 13(6), Nov. 1993, pp. 33–41.

[Nishimura *et al.*, 1985] Nishimura, H., Hirai, M., Kawai, T., Kawata, T., Shirakawa, I., and Omura, K. Object modeling by distribution function and a method of image generation. In Proc. of *Electronics Communication Conference '85*, 1985, pp. 718–725. (Japanese).

[Norton, 1982] Norton, A. Generation and rendering of geometric fractals in 3-D. *Computer Graphics* 16(3), 1982, pp. 61–67.

[Ratschek & Rokne, 1984] Ratschek, H. and Rokne, J. *Computer Methods for the Range of Functions*. John Wiley and Sons, 1984.

[Reeves, 1983] Reeves, W. T. Particle systems – a technique for modeling a class of fuzzy objects. *ACM Trans. Graphics* 2, April 1983, pp. 91–108.

[Sederberg & Zundel, 1989] Sederberg, T. W. and Zundel, A. K. Scan line display of algebraic surfaces. In Lane, J., ed., *Computer Graphics (SIGGRAPH '89 Proceedings)*, vol. 23, July 1989, pp. 147–156.

[Snyder, 1991] Snyder, J. M. *Generative Modeling: An Approach to High Level Shape Design for Computer Graphics and CAD*. PhD thesis, California Institute of Technology, May 1991.

[Snyder, 1992] Snyder, J. M. Interval analysis for computer graphics. *Computer Graphics* 26(2), July 1992, pp. 121–130.

[Szeliski & Tonnesen, 1992] Szeliski, R. and Tonnesen, D. Surface modeling with oriented particle systems. In Catmull, E. E., ed., *Computer Graphics (SIGGRAPH '92 Proceedings)*, vol. 26, July 1992, pp. 185–194.

[Taylor, 1955] Taylor, A. E. *Advanced Calculus*. Ginn and Company, 1955.

[Turk, 1991] Turk, G. Generating textures for arbitrary surfaces using reaction-diffusion. In Sederberg, T. W., ed., *Computer Graphics (SIGGRAPH '91 Proceedings)*, vol. 25, July 1991, pp. 289–298.

[Turk, 1992] Turk, G. Re-tiling polygonal surfaces. In Catmull, E. E., ed., *Computer Graphics (SIGGRAPH '92 Proceedings)*, vol. 26, July 1992, pp. 55–64.

[van Overveld & Wyvill, 1993] van Overveld, C. and Wyvill, B. Potentials, polygons and penguins: An efficient adaptive algorithm for triangulating an equi-potential surface. *Proceedings of the Fifth Western Computer Graphics Symposium*, March 1993.

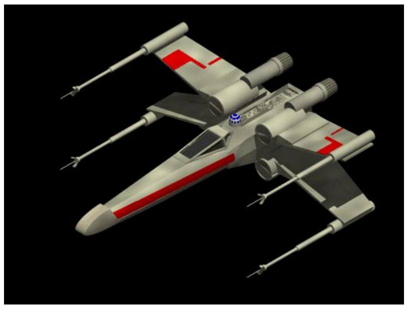
[Welch & Witkin, 1994] Welch, W. and Witkin, A. Free–Form shape design using triangulated surfaces. In Glassner, A., ed., *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series. ACM SIGGRAPH, ACM Press, July 1994, pp. 247–256. ISBN 0-89791-667-0.

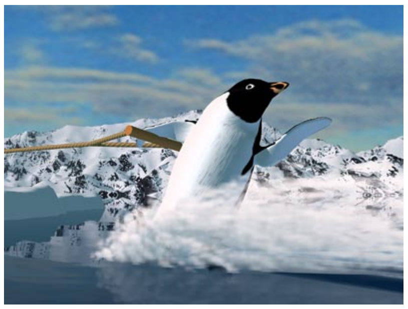
[Witkin & Heckbert, 1994] Witkin, A. P. and Heckbert, P. S. Using particles to sample and control implicit surfaces. In Glassner, A., ed., *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series. ACM SIGGRAPH, ACM Press, July 1994, pp. 269–278. ISBN 0-89791-667-0.

[Wyvill *et al.*, 1986] Wyvill, G., McPheeters, C., and Wyvill, B. Data structure for soft objects. *Visual Computer* 2(4), 1986, pp. 227–234.

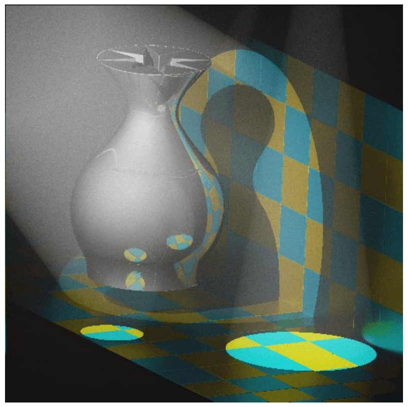
An Implicit Gallery

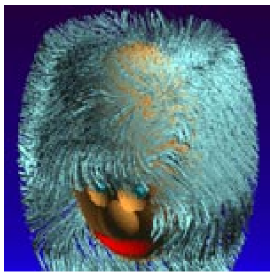
... a collage of implicit techniques


(a)


(b)

(c)


(d)


(e)

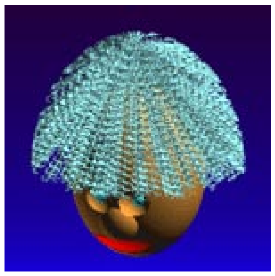

(f)

Figure 1: Examples generated using implicit techniques.

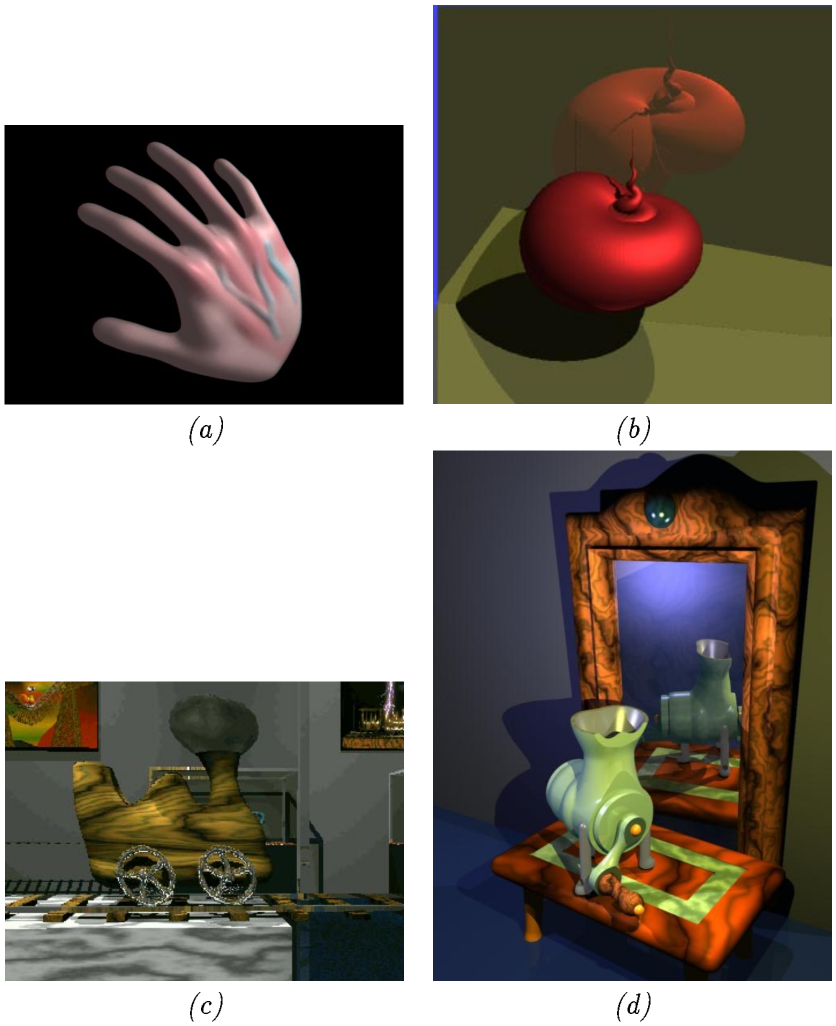


Figure 2: More examples generated using implicit techniques.