APPLICATIONS OF SPLINE MANIFOLDS TO PROBLEMS IN MODELING, RENDERING, AND ANALYSIS

by

William Michael Martin

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

School of Computing

The University of Utah

May 2013

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

STATEMENT OF DISSERTATION APPROVAL

The dissertation of William Michael Martin has been approved by the following supervisory committee members:

Elaine Cohen, Chair	September 24, 2012 Date Approved
Gershon Elber, Member	September 24, 2012 Date Approved
Richard F. Riesenfeld, Member	September 24, 2012 Date Approved
Peter Shirley, Member	September 24, 2012 Date Approved
Ross Whitaker, Member	September 24, 2012 Date Approved

and by Alan Davis, Chair of
The School of Computing

and by Donna M. White, Interim Dean of the Graduate School.

ABSTRACT

While boundary representations, such as nonuniform rational B-spline (NURBS) surfaces, have traditionally well served the needs of the modeling community, they have not seen widespread adoption among the wider engineering discipline. There is a common perception that NURBS are slow to evaluate and complex to implement. Whereas computer-aided design commonly deals with surfaces, the engineering community must deal with materials that have thickness. Traditional visualization techniques have avoided NURBS, and there has been little cross-talk between the rich spline approximation community and the larger engineering field.

Recently there has been a strong desire to marry the modeling and analysis phases of the iterative design cycle, be it in car design, turbulent flow simulation around an airfoil, or lighting design. Research has demonstrated that employing a single representation throughout the cycle has key advantages. Furthermore, novel manufacturing techniques employing heterogeneous materials require the introduction of volumetric modeling representations. There is little question that fields such as scientific visualization and mechanical engineering could benefit from the powerful approximation properties of splines. In this dissertation, we remove several hurdles to the application of NURBS to problems in engineering and demonstrate how their unique properties can be leveraged to solve problems of interest.

To my parents Michael and Julia and my beloved Graham.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I am indebted to many people who have aided and encouraged me during the course of my graduate career. First, I would like to thank my committee, whose patience was truly tested during my rather extended tenure. I would particularly like to express my gratitude to Elaine Cohen, who has generously given of her time, knowledge, and experience in helping to form the ideas in this dissertation. Pete Shirley has also provided a wealth of advice and timely humor, along with a finely honed research barometer. Finally, I would like to thank Rich Riesenfeld, Ross Whitaker, and Gershon Elber for their insightful questions and recommendations.

I have been blessed with many excellent role models from the graduate program. Gordon Kindlmann, Steven Parker, Simon Premože, Michael Stark, and David Weinstein have provided shining examples of how to think critically and pursue research of the highest standards. I am grateful to Amy and Bruce Gooch for their boundless creativity and enthusiasm, and their faithful friendship. Special thanks are due to Karen Feinauer in the School of Computing who constantly encouraged me to finish this degree. I would also like to thank my many friends at Visual Influence and Numira Biosciences who made working at a startup rewarding and a compelling distraction from my dissertation.

I could not have begun this work, much less finished it, without the love and support of my parents, Mike and Julia Martin. Finally, and most especially, I would like to thank Graham Gautschi for his support, encouragement, and love that made the journey worth traveling.

CHAPTER 1

INTRODUCTION AND AIMS

Nonuniform rational B-spline (NURBS) curves and surfaces form the foundation for many geometric modeling systems. There are a host of reasons for this. First, because splines contain the space of polynomials, they are flexible enough to describe complex functions. On the other hand, they are coordinate independent, meaning affine transformations do not alter qualitative shape. Further, a relatively small number of parameters are required to specify complex behaviour. Shape interpolating B-splines provide the designer a few descriptive handles (*e.g.,* control vertices) for easy modification of surface properties. Continuity between adjacent patches is ensured by the representation, and its degree is adjustable via the knots [3].

Relative to other smooth representations, NURBS also exhibit fast evaluation due to the local extent of their basis functions. Similarly, the impact on the model of modifying the control points is localized. The variation diminishing property of B-splines says, loosely, that a curve/surface will exhibit no more variation (oscillation) than its control polygon – that is, the B-spline function acts as a low-pass filter on its mesh. The convex hull property of B-splines allows us to intuitively isolate the location of a NURBS curve or surface in space and the refinement property facilitates the addition of control points without loss in precision. Finally, the B-spline basis possesses many useful integral and differential properties, some of which we will leverage in Chapter 7.

Despite these attractive features, splines have not been traditionally embraced by the engineering community at large. Downstream from the modeling phase in the design cycle, approximations in representation are frequently made, be it polygonization in rendering or the introduction of tetrahedra or hexahedra in a finite element simulation. Furthermore, traditional computer-aided design targets surface representations, whereas simulation and analysis frequently require volumetric models. Recent research in a new field called isogeometric analysis has demonstrated that there can be significant advantages to leveraging the same representation throughout the design cycle [4]; that is from modeling, to simulation, to manufacture.

The fundamental premise is that there is something in the original NURBS representation (or a smooth representation) that is worth preserving. With change in representation comes

the potential for approximation error. And because design is a feedback loop, be it lighting design or automobile design, it is necessary to relate results back to the original model so that adjustments can be made. It all comes back to the notion that defects in your computation ought to be due to fundamental aspects of your problem and not features introduced by your approximation. Our goal in this work has been to develop techniques that make the native NURBS representation amenable to problems in engineering. Among the fields we target are modeling, medical visualization, manufacture, and rendering. Our key aims in this work will be

1. Developing optimized methods for NURB evaluation.
2. Facilitating direct rendering of NURBS for visualization.
3. Extending surface-based modeling and rendering approaches to volumetric NURBS.
4. Providing methods for converting boundary representations into volumetric models for analysis.
5. Demonstrating how the approximational power of splines can be leveraged to solve a complex problem in engineering.

1.1 Organization

To this end, the dissertation is organized in the following way. In Chapter 2, we list some of the problems that have driven our work and discuss the previous research that has influenced our own. Next, because visualization is often key to problem exploration, we derive in Chapter 3 the necessary machinery for ray tracing NURBS directly. Part of this derivation yields a highly optimized evaluation algorithm. All of the details required for building a NURBS rendering system are provided, as well as indications of common pitfalls. In Chapter 4, we introduce the mathematics for volumetric NURBS. We make this representation amenable to common scientific computing and engineering tasks by providing methods for fitting them to data, modeling with them, and imbuing them with attribute data. We also develop methods for visualization and introduce a novel technique for optical path tracing. Chapters 5 and 6 provide operators for converting existing boundary representations into true surface and volumetric ones. They have the advantage of preserving the original boundary parameterization, a trait prescribed by the field of isogeometric analysis for many types of finite element analyses [4]. In Chapter 7, we demonstrate the approximation power of splines for capturing functions on manifolds, generalizing the attribute modeling techniques given in Chapter 4. Our example application characterizes

the radiance function for a global illumination simulation using a four-dimensional (4D) spline. Finally, we close with a brief recap of our contributions in Chapter 8.

CHAPTER 2

BACKGROUND

This dissertation relies on results from several areas of computer science, mathematics, and engineering. In this section, we briefly summarize some applications that have motivated us to pursue this work, as well as work related to ours in the field.

2.1 Motivating Applications

There are a number of fields that we believe could benefit from spline-based representations. We now survey some of these that have motivated the present work.

2.1.1 Scientific and Engineering Analysis

Scientific simulations often involve characterization and analysis of volumetric phenomena. Fluid dynamics simulations commonly take into account variables such as pressure, density, temperature, and velocity, which can vary continuously. Stress and fracture simulations may track force, density, stress, and deformation. For the field of molecular dynamics, isosurfaces of equal electrostatic potential provide an informative visualization.

In the area of meteorology, there are a number of quantities which are critical to weather and climate prediction. Among these are temperature, wind speed, barometric pressure, pollutant density, molecular concentration, and humidity. These typically vary globally and with altitude, indicating a volumetric model. Simulations are used for daily weather forecasting as well as predicting the incidence and behavior of such extreme weather phenomena as hurricanes and tornadoes.

Meteorological variables also have a direct impact on atmospheric optics. Localized changes in temperature and density are responsible for mirages. Due to a varying atmospheric index of refraction, light is guided along a curved path, making objects appear to be where they are not. The techniques we discuss later for lens modeling apply equally well here. Over a larger scale, similar phenomena are responsible for the twinkling of the stars, the colors in a sunset, color shifts due to atmospheric perspective, and have a direct impact on the quality of astronomical

observations. Whereas splines have a wide array of advantages when applied to functional approximation, they have not historically been leveraged to model and track these scientific variables. A key reason for this has been the historical perception that they are not suitable for simulation and analysis.

In the area of engineering design, models must frequently be synthesized and then analyzed. Computer-aided design (CAD) programs are usually employed to build the model, leveraging an underlying spline- or subdivision surface-based representation. The reasons for this representation are many, and they are well-documented in the literature [5]. This model must then be handed to a different team of engineers for analysis. Consider for example the aerodynamical analysis for a new air foil on a plane. In some situations, such as automotive design, there are aesthetic as well as fluid dynamics factors to be considered when building a part. The analysis phase frequently involves the transformation of the smooth model into a representation that is amenable for finite element analysis (FEA). The goal of FEA is to determine how a quantity such as heat or stress varies throughout the model, as these can indicate the potential for failure or unwanted side-effects such as turbulent flow. Hence, a smooth surface-based representation is traded for a tessellated volumetric one.

However, the process of design is a feedback loop. Results from the analysis phase must be applied to the design step so that modifications can be effected. The translation between representations can introduce errors into the feedback loop. Also, simulations can be particularly sensitive to minor deviations in domain geometry.

> Recent trends taking place in engineering analysis and high-performance computing are ... demanding greater precision and tighter integration of the overall modeling-analysis process. We note that a finite element mesh is only an approximation of the CAD geometry, which we view as 'exact.' This approximation can in many situations create errors in analytical results. The following examples may be mentioned: Shell buckling analysis is very sensitive to geometric imperfections, boundary layer phenomena are sensitive to the precise geometry of aerodynamic and hydrodynamic configurations, and sliding contact between bodies cannot be accurately represented without precise geometric descriptions. Automatic adaptive mesh refinement has not been as widely adopted in industry as one might assume from the extensive academic literature, because mesh refinement requires access to the exact geometry and thus seamless and automatic communication with CAD, which simply does not exist. Without accurate geometry and mesh adaptivity, convergence and high-precision results are impossible ([4, p. 2]).

The mesh translation process itself can be extremely costly – as much as 80% of the time spent in the analysis pipeline can be spent constructing geometry in a form that is amenable to processing [4]. This indicates that contrary to popular opinion, meshing is far from turnkey.

The recent area of isogeometric analysis has demonstrated advantages in leveraging a single NURBS-based representation in both the modeling and analysis phases of design [4]. While spline elements require more expensive evaluation than traditional basis functions, they make up for this in the quality of their results, their greater expressiveness in functional representation, and their lesser susceptibility to noise [4].

The problem remains that CAD-based modeling is a primarily surface-based endeavor. In Chapter 4, we introduce techniques for applying trivariate splines to problems in engineering. We provide methods for fitting volumetric NURBS to data and introduce modeling operators amenable to integration in CAD programs. In order to support modeling with heterogenous materials (discussed in Section 2.1.2.1), we introduce an attribute-based model which can support, among other things, material composition. We also introduce visualization and rendering operators to support the use of trivariate NURBS, and we conclude by modeling a GRIN lens and rendering its appearance.

In Chapters 5 and 6, we introduce methods for upgrading CAD boundary representations (B-Reps) into truly volumetric representations. One of the key aspects of our approach is that it preserves the parametrization of the original surface-based model. This means that values determined in the analysis process can be related directly back to the source model, decreasing the potential for error in translation.

2.1.2 Novel Materials

2.1.2.1 Functionally Gradient Material (FGM) Manufacturing

One of the most exciting developments in modern manufacturing has been the emergence of functionally graded materials (FGMs). FGMs are composites with the interesting property that the proportion of each constituent material can be varied continuously. Thus, for example, a turbine blade may have a steel (temperature resistant) edge and an aluminum (lightweight) interior with a smooth blend between the two. The gradient boundary between materials improves the wear life of the part, as failures tend to occur at discrete material boundaries (typically, on the side of the softer material). With the improved failure resistance of graded materials, turbine blades can be made lighter, thereby improving their efficiency. In fact, the concept of graded materials can be traced back to feudal Japan, where, analogously, the blades of swords possessed a soft but tough core, and a hardened edge [6]. Furthermore, many natural structures such as bamboo, plant stems, and bone are graded and provide strength in areas of high stress [6].

There are a number of competing technologies for manufacturing using functionally gradient materials. Among them are molecular beam epitaxy, vapor deposition, three-dimensional printing

(3DP), bulk and surface micromachining, and lithography. For the rapid-prototyping community, three-dimensional printing may be most familiar through analogy to stereolithography. To prepare a model for stereolithography, first the model is sliced into parallel layers of a prescribed thickness. The lithography machine then acts as a kind of ink jet printer, laying down a layer of material as specified by the bottom-most slice. Subsequent layers are laid down similarly until the part is completed. M.I.T.'s 3DP process [7] takes the ink jet analogy one step further by allowing mixtures of materials to be laid down in each layer (see Figure 2.1). Thus, in theory, gradient materials can be made from any material that can be reduced to a powder.

FGMs pose exciting possibilities for the future of manufacturing, since in theory, materials can be specifically tailored to function. This will in turn have an impact on approaches to modeling. For example, techniques such as pocketing to reduce weight in parts may become unnecessary as heavy materials can be graded to lighter ones. There are important implications to graphics research as well. If materials can be accurately tailored, then their corresponding reflectance properties can be collected in a database, allowing for precise renderings of parts from their modeling software descriptions. In Chapter 4, we introduce methods for augmenting traditional modeling systems with data structures and modeling operators required to fit and model with these materials.

2.1.2.2 Optics

As this dissertation is being written, novel lens technology is revolutionizing the optics community. So-called GRIN (GRadient INdex) lenses are unique in that their index of refraction is designed to vary continuously across the lens (see Figure 2.2). One implication is that lenses made from gradient material can be milled flat, and still focus light. A consequence is potentially

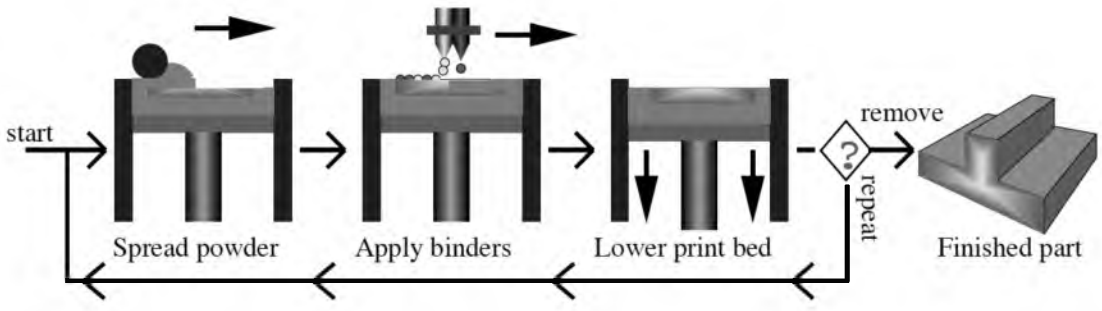

Figure 2.1: Diagram summarizing the three-dimensional printing technology (courtesy of MIT 3D Printing Lab – `http://web.mit.edu/tdp/www/whatis3dp.html`).

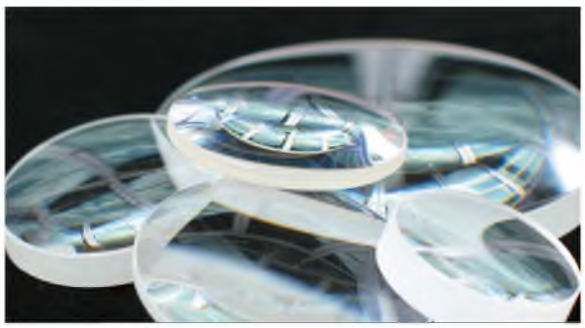

Figure 2.2: Examples of commercially available Gradium™ GRIN lenses (courtesy of LightPath Technologies – `http://www.lightpath.com`).

higher accuracy in lens fabrication, since gradient materials may be more precisely tuned than the traditional grinding process would allow. Typically, GRIN lenses have an index of refraction that varies radially from a central axis, although this need not be the case.

In reality, GRIN technology is not new. Every fax and copy machine contains an array of gradient lenses. The technology has also been used in fiber optic cable (see Figure 2.3). However, recent breakthroughs in fabrication technology have allowed GRIN lenses to be made to higher accuracy, at very large and very small sizes, and from a variety of materials. These advances have made GRIN lenses suitable for a range of applications, including solar power collection, digital communications (wave division multiplexing), medical instrumentation, and astronomical optics. In Chapter 4, we introduce methods for modeling and visualizing these lenses.

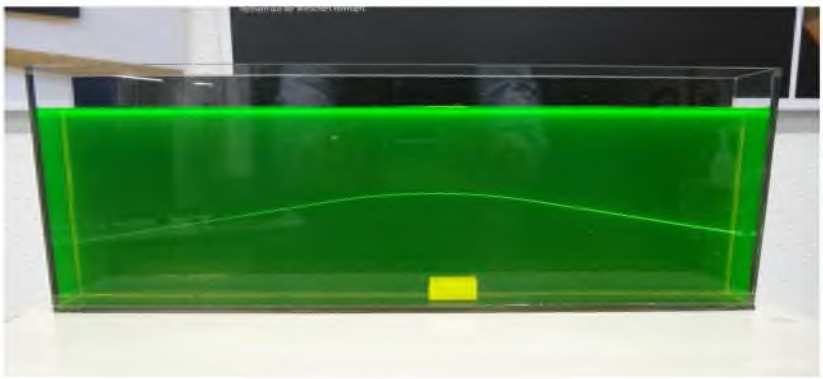

Figure 2.3: Light travels along a curved path in a medium with graded refractive index. This property is used in multi-mode optical fiber to reduce 'modal dispersion.' Image courtesy of Marc Achermann, Lucerne University of Applied Sciences and Arts, 6048 Horw, Switzerland.

2.1.3 Medical Visualization

In the area of medical diagnostics, volumetric data are commonly acquired through magnetic resonance imaging (MRI) or computed tomography (CT) scans. We briefly summarize how these machines work to impart some understanding of the quantities they record. An MRI machine (Figure 2.4) is essentially a machine capable of generating a strong magnetic field. When exposed to this field, the hydrogen atoms of the body tend to fall either into alignment or anti-alignment with the direction of the field. The machine generates an image by casting radio waves at a hydrogen-specific frequency toward the center of the measurement apparatus. This causes the atoms to precess (rotate 90 degrees) and generate a corresponding signal which can be recorded by the machine. Different tissues of the body contain different densities of hydrogen atoms, and the strength of the signal emitted is proportionate to this density. Thus, tissues can be differentiated in the resulting 3D photo.

A CT machine is a specialized form of X-ray machine. X-ray machines work by casting X-rays through the object of interest. Since different kinds of tissue absorb X-rays in different degrees, X-rays can be used to expose structures within the body. The CT machine has an X-ray device which rotates about the object, and produces slices of data, which can be used to reconstruct a 3D image. The output of both machines is three-dimensional volumetric data which are indicative of localized tissue type (see Figure 2.5).

In Chapter 4, we extend traditional methods for scientific data fitting and visualization to volumetric NURBS.

2.1.4 Realistic Rendering

The area of computer graphics has long been dominated by triangles. However, there are many reasons why it is a good fit for the application of spline-based techniques. First, because

Figure 2.4: An MRI machine.

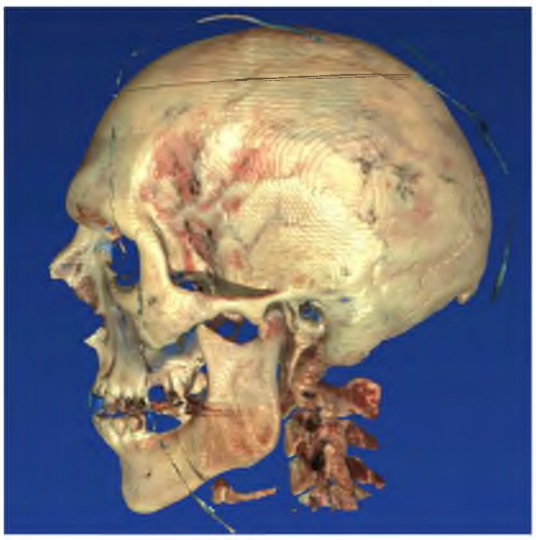

Figure 2.5: Visualization of CT data from the National Library of Medicine's Visible Human Project® (image courtesy of Steven Parker, University of Utah).

many of the models in production scenes are generated in CAD software, the models are typically spline (or subdivision surface)-based. As with finite element analysis, the initial step in the rendering process is often to convert smooth models into a triangular mesh. The reasoning has been that triangles are simpler to render, and because all smooth representations admit a tessellation [8], triangles become the common currency across rendering systems. Shortcomings of this approach are precisely parallel to those mentioned for isogeometric analysis. The models must frequently be tweaked in response to lighting conditions as well as the aesthetic opinions of the director (or engineer if we consider automotive design). This requires access to the underlying models. A solution employed by Pixar (in their Reyes architecture [9]) has been to delay tessellation until the final rendering step, tessellating to the granularity of a pixel and beyond in order to avoid visible seams, jaggies, and other artifacts of tessellated models. This approach is not generally applicable to interactive rendering as each rendered frame will entail many millions of polygons, composited in different layers, *etc*. Microtriangles also are not a programmatic simplification, as the rendering program must still deal with smooth surface representations and dice them into microfacets for the rendering step. If tessellation is applied earlier, one instead deals with the potential for visible seams – which can be magnified by shadows, specular highlights, and the presence of optical elements – as well as an explosion of data (and the resulting memory utilization) which often accompanies the approximation of curved surfaces using piecewise planar elements. Finally, we note that in recent years, ray tracing has become a legitimate alternative to raster-based rendering.

Particularly as the size of the models has increased, and it has done so exponentially [10], ray tracing has become attractive because its complexity scales in the number of pixels, and not in the number of elements. Because of the efficiency by which the ray tracing paradigm solves the visibility problem, the expense of evaluating primitives such as NURBS has become less of a concern.

In Chapter 3, we have developed a system for directly rendering NURBS surfaces which is optimized, general, and supports implementation within a parallel rendering system. We have successfully integrated our work into one such system, with promising results [1].

The global illumination problem (which we introduce in more detail in Chapter 7) aims to achieve photorealistic rendering of a scene by simulating how light interacts with, and is retransmitted by, interfaces in the scene. The radiance function, which captures this notion, is a function on manifolds – in its most general form, a function of (volumetric) position, wavelength, viewing angle, and time, a seven-dimensional function. However, in the absence of intervening media (fog, atmospheric dispersion, *etc.*), it is frequently framed as a surface-based function. The solution to the steady-state global illumination problem is often phrased as a finite element problem [11–13]. As we have discussed previously, this problem has historically been dominated by triangular mesh elements. However, smooth representations have all the advantages given in the preceding paragraphs, including potentially fewer elements. The basis functions used to represent radiance have traditionally been orthogonal functions which simplify quadrature of the global illumination problem – among these, spherical harmonics and Haar wavelets. A downside of these basis functions is that they can introduce oscillations in their reconstruction. One of the attractive aspects of spline approximation is that it can be constructed so as to avoid undue oscillation. Splines also possess integral properties which make them ideal candidates for quadrature. And intuitively, we observe that the radiance in most scenes appears to be smooth over large areas – and splines are nearly ideal for capturing such smooth functions. Obvious counterexamples include sharp shadows, specular reflections (caustics), and areas of high textural frequencies.

In Chapter 7, we generalize our attribute-based model from Chapter 4 to capture the view-dependent radiance function as a spline. We demonstrate how spline-based approximation techniques can be leveraged to simplify the integration of the global illumination equation, in the process demonstrating quadrature techniques which are readily amenable to other classes of engineering analysis problems. We generalize the gathering/scattering approach of radiant transfer to solve the global illumination problem, demonstrating how splines can be applied to FEA

problems outside the field of isogeometric analysis and we posit that our solutions to the rendering equation take the same form. We have implemented our spline-based radiance function in a real-time rendering system, allowing the radiance function to be interactively computed for novel views via a simple texture evaluation (see Chapter 7).

2.2 Related Work

Much work has preceded ours. The organization of this section mirrors the organization of the rest of our document, and details the work we consider most related or relevant to ours.

2.2.1 Ray Tracing NURBS Surfaces/Volumes

Several of the techniques we shall present in this dissertation entail ray tracing a trivariate NURBS volume. Such a technique requires ray intersection with the solid boundary and subsequent traversal of its interior. Since the parametric faces of a trivariate volume are themselves NURBS surfaces, we shall therefore require the capability to ray trace NURBS surfaces. Techniques for ray tracing NURBS can be divided into two broad classes — those that tessellate (*e.g.,* [14–18]) and those that work directly with the underlying representation (*e.g.,* [19–27]). The former are far more prevalent in commercial software systems. The techniques we propose are of the latter sort, for reasons we shall discuss in more detail later. We shall refer to these techniques as direct methods.

The seminal article on ray tracing parametric surfaces is that of Kajiya [22]. His method uses the theory of resultants to reduce the problem of ray tracing patches to that of finding the simultaneous roots of two parametric curves. Toth [26] introduces an intersection technique based on Newton-Raphson root finding. His algorithm subdivides the surface into intervals such that the Newton iteration is guaranteed to converge to a root, if one exists, from any start value in that interval. This guarantee is made, however, at the price of linear convergence, as his analysis only holds for a Newton interation with a fixed Jacobian (so-called linear Newton). These two articles by Kajiya and Toth contain the major ideas upon which most subsequent NURBS ray tracing papers have built.

The basic operation of a NURBS intersection routine can be summarized as follows:

- As a preprocess, refine the surface to facilitate ray culling using a hierarchy of bounding volumes. For Newton-type iterative schemes, this step will also determine good start values to ensure swift convergence.

- At run time, use the bounding volume hierarchy (BVH) to limit sections of the surface

under consideration, and apply the more costly patch intersection routines only if a ray successfully traverses to a leaf.

We summarize briefly the contributions of several other authors. Fournier and Buchanan [21] introduce Chebyshev polynomials and demonstrate that tight bounding volumes can be obtained directly from the coefficients of a Chebyshev bilinear patch. Their approach subdivides a surface patch until the subpatches can be well approximated by a bilinear representation. The bounding volumes are generated bottom-up using the coefficients of the Chebyshev representation, and intersections with the bilinear patches can be found exactly via a quadratic equation. Another novel approach, introduced by Nishita *et al.* [24], uses a technique called Bézier clipping to iteratively remove regions of the patch which do not intersect the ray. Their treatment also handles trimming curves using a technique of point classification, but requires that patches be represented in the Bézier-Bernstein basis.

There are a number of Newton-based techniques, which differ mainly in the way they form the bounding volume hierarchy and the manner in which initial values are chosen for the iteration. Barth and Stürzlinger [19] subdivide the NURBS surface until each subpatch can be approximated by a parallelogram. The hierarchy of patches is bounded by parallelipipeds. If a ray is not eliminated by intersections with the bounding volume hierarchy, then at the leaf, it is intersected with the approximating parallelogram. This intersection yields a start value for the Newton iteration on the leaf surface patch. Sweeney and Bartels [25] suggest refining the NURBS surface until the screen projection of each mesh facet is less than a few hundred pixels, and until the knot they associate with each control point constitutes a good starting value for the Newton iteration. The leaves of their bounding volume hierarchy are the axis-aligned rectangles which contain a particular refined control vertex and its 4-connected neighbors. If a ray succeeds in reaching a leaf node, a Newton iteration is started using the knot value associated with the vertex.

Yang [27] uses a modified spatial subdivision scheme to generate a BVH. First, points are generated on the surface using a user-specified parametric step size. An axis-aligned bounding box is produced for these points. Next, the bounding box is subdivided into 16 sub-boxes. Sub-boxes containing none of the surface points are discarded. If the limit depth has not been reached, the procedure is repeated for each remaining sub-box. If the limit depth has been reached, the parameter values for the points in the box are averaged to produce the start value for the Newton iteration. Lischinski and Gonczarowski [23] introduce a number of efficiency improvements to the NURBS ray tracing technique. Among these are development of a hybrid of BVH and spatial subdivision schemes, search tree caching for Toth's method, improved screen sampling order

based on the Peano curve, and a treatment of secondary rays which avoids trivial intersections at the ray origin.

2.2.2 Volumetric Representations and Techniques

We have noted that there are a variety of applications for which generating, analyzing, and visualizing volumetric quantities is key. This conclusion motivates the development of volumetric models which encapsulate the behavior of such systems. A number of representation techniques have been developed for volumetric data. The most commonly encountered of these is the voxel representation, which has become standard among the medical visualization community. One reason for this is that scanned data are typically captured in regularly spaced slices, each slice composed of a rectangular grid of values. The samples provided by the imaging machine must be reconstructed in order to generate a continuous volumetric function. One commonly used filter is a trilinear interpolant.

Among the scientific community, finite elements are a common tool for analysis. Finite elements (*e.g.*, tetrahedra) provide simplified connectivity information, allowing attributes, such as stress or temperature, to propagate across the volume. They are well suited to iterative schemes, and are commonly used in the solution of ordinary and partial differential equations. Both voxels and finite element meshes can be seen as spatial subdivision structures, which differ in regularity and shape.

Within the modeling community, volumes have been traditionally represented by their boundaries. Such representations include Bézier surfaces, NURBS, Steiner patches, Coons patches, and Hermite patches, among others. This has proven sufficient for models with homogeneous interior. However, the desire to apply proven modeling techniques to novel technologies, such as functionally gradient materials and GRIN lenses, has led to the development of higher dimensional parametric formulations. It is these representations which are most closely related to the present work, and we now discuss them in greater detail.

Farouki and Hinds [28] present a nice overview of parametric modeling techniques, including the generalization of bivariate NURBS surfaces to trivariate NURBS volumes. In the same year, Lasser [29] explored the Bernstein-Bézier volume representation, and extended techniques for evaluation and interpolation to them. Traditional modeling operations, such as ruling, extrusion, and revolution, were extended to trivariate NURBS by Paik in [30]. She also allowed volumes to represent surfaces which evolve through time by designating one coordinate as a temporal axis. A final contribution of that thesis was a physical simulation which incorporated a system of springs at the vertices. Madrigal and Joy [31] develop an algorithm for determining the boundary

of a trivariate solid that is swept through space. The method employs the results of Joy and Duchaineau [32] for determining the boundary of a trivariate solid, and then performing a sweep of that boundary to determine the resulting sweep envelope. As an alternative to parametric modeling, Wang and Kaufman [33] present a technique for volumetric modeling which is analogous to sculpting. A voxel representation is employed, and material can be removed or have its attributes (*e.g.,* color) modified using a variety of tools.

Historically, a number of papers have developed volumetric representations which incoporate attribute data and geometry. In a visionary 1977 paper, Stanton *et al.* [34] describe a batch system which uses a tricubic Hermite volume for stress analysis. Their results lead them to conclude that parametric volumes are "an important new analytical tool for solids of composite material." In a followup article, Casale and Stanton [35] develop a system of modeling volumes with carried attribute data. Their representation is tricubic, augments the geometry (vertices) with "data entries," and supports finite element analysis. Yen [36] generalizes this result with a scheme for performing finite element analyses on trivariate NURBS volumes. In his method, the volume is divided into regions, and attributes can be specified for each region. Subsequent meshing is performed to achieve a given simulation error tolerance. Yen further introduces a Boolean sum operator to generate volumes from boundary surfaces, in a spirit similar to the Coons patch. Dickinson *et al.* [37] use a B-spline volume to represent scalar and vector fields. Again providing a worthy foil, Kaufman [38] has combined modeled objects and measured data within a voxel-based volume visualization system, along with an algorithm for scan converting tricubic Béziers.

More recently, there has been a resurgence in interest in volumetric representations of heterogeneous materials, particularly as the applications of medical imaging and FGM manufacture become more common. Bonnell *et al.* [39] give a technique for determining discrete material interfaces for objects represented in a voxel formulation, provided that for each voxel, the proportion of volume filled by each material to total voxel volume is known. The result of their algorithm is a triangulated approximation to the interfaces between materials. Raviv and Elber [40, 41] employ trivariate B-splines for representing scalar fields and provide efficient algorithms for multiresolution sculpting using their coefficients. Park and Lee [42] use a rational NURBS volume to model fluid flow data. They present a data structure which couples attributes (such as flow density and flow velocity) to geometry and develop methods for nodal interpolation to the data.

Kumar and Dutta [43] develop an abstract mathematical representation by extending r-sets to

multimaterial objects. r-sets are an established mathematical representation of solid models [44]. However, they are limited to representing geometry. Their article extends the r-set to a new representation which contains both geometric and material information, called the r_m-set. The approach entails generalizing the traditional CSG operations to the new representation. The article also introduces a software framework for representing these models. The models presented in this work are limited however to objects with discrete boundaries and a single material per domain.

The authors correct this limitation in a followup article [45] which generalizes r_m-sets to handle funtionally graded materials (FGMs). Again, the CSG operations are extended to the new r_m-set representation. However, now care must be taken to assure that material fractions remain normalized. A software framework is introduced and a simple FGM object is generated using the aforementioned representation. Application to layered-manufacturing techniques is discussed. Kumar *et al.* [46] provide a good summary of the various mathematical representations for models found in the literature. They then proceed to generalize the notion of an r_m-set to that of a fiber bundle. Each material is a manifold in this representation. The CSG operations are extended to this "trivial" fiber bundle and an augmented software framework is introduced.

Marsan and Dutta [47] apply the preceding theory to the class of trivariate parametric functions. The traditional trivariate NURBS formulation is extended to represent material composition. This is done in one of two ways. The material can be represented as an explicit function over rectilinear coordinates (x, y, z), which also serve as the parametric domain. This is similar to the voxel formulation traditionally applied in the visualization literature. On the other hand, the coefficients of the trivariate volume can be extended to contain material information, resulting in a true parametric volume formulation, with attributes coupled to geometry. Methods of evaluation and data fitting, as well as applications to layered manufacturing, are discussed.

There have been several articles from M.I.T. supporting the use of heterogeneous materials in manufacture, with particular application to their 3DP technology. Liu *et al.* [48, 49] present a finite element-based system for designing FGM solids and develop an efficient distance transform so that composition can be automatically specified as a function of boundary proximity. They also produce a method to efficiently evaluate material composition at a point. Jackson [50] provides an extensive survey of volumetric techniques as applied to the problem of FGM modeling and local composition control. He provides analysis of the techniques in terms of memory utilization, and finds that cellular techniques afford the greatest efficiency. Cellular techniques decompose the geometry into cells, over which material properties can be defined. Among the region types

he introduces is a trivariate Bézier patch, with attribute data decoupled from geometric data, in the same spirit as the formulation we propose here.

2.2.3 Volume Visualization

In this dissertation, we will be describing a data structure which associates geometry and attributes in a single model. Such representations are by their nature high-dimensional, and it is typically difficult for designers and analysts to reason about this space. Therefore, techniques for encoding information in visualizations will play a key role in making these models usable and intuitive. A number of researchers have developed approaches to this problem, and we will leverage many of their ideas in our present work.

Perhaps the best known volume visualization algorithm is the so-called Marching Cubes technique for constructing isosurfaces, introduced by Lorensen and Cline [51]. For a given voxel and a particular isovalue, each vertex of the voxel is labelled with a 0 or 1 according to whether its attribute value is above or below the value sought. There is only a small set of ways a given isosurface can pass through a voxel, and the vertex labeling is used to lookup the tessellation for each corresponding configuration. The algorithm then marches forward to the next cell.

Isosurface rendering is one approach to extracting structure in volumetric datasets. Another technique is assigning colors and opacities to materials, and pulling out structural detail using traditional graphics shading models (see Figure 2.5). Such approaches may be classified under the moniker "direct volume rendering." Drebin *et al.* [52] introduce a technique for volume rendering which operates on data decomposed into material percentage volumes. Each material percentage volume tracks some material of interest, *e.g.,* skin or bone, its data values indicating the proportion of the material present in a given voxel. From this representation, the user associates a density with each material, and gradients can be calculated based on how quickly materials transition. Likewise, colors and opacity are associated with materials, and this together with gradient information is used to shade the volumetric data.

Shirley and Tuchman [53] provide a clever method for direct scalar volume rendering of datasets represented in a tetrahedral format. First, tetrahedra are classified according to their face orientation relative to the viewer. The tetrahedra are projected onto the viewing plane, and broken into a set of triangles, based on their classification. Ray integration is calculated at the thickest point of ray-tetrahedron traversal, and linear interpolation is used to generate the brightness and opacity across each triangle. The technique produced renderings of high quality with an order of magnitude increase in performance over ray traced techniques.

Nonetheless, given its simplicity, ray tracing is an attractive option for direct volume rendering. Levoy [54] describes methods for efficiently ray tracing volumetric data. The two techniques focused on are hierarchical gridding of data to speed ray-grid traversal, and adaptive ray termination based on accumulated opacity. Max [55] provides an overview of optical methods for direct volume rendering, with particular attention to multiple scattering effects, such as are seen in clouds.

There have been some techniques targeted specifically to rendering parametric volumes. Dickinson *et al.* [37] and Park and Lee [42] present methods for generating isosurfaces and streamlines for trivariate scalar and vector fields to effect feature segmentation. Raviv and Elber [41] achieve interactive visualization of complex scalar fields by fixing the viewpoint and preintegrating the B-spline functions along a line of sight. The user can freely modify the scalar coefficients and see the results in realtime. Chang *et al.* [56] develop a scanline technique for direct rendering of trivariate volumes using line-of-sight integration and compositing. Madi and Walton [57] allow visualization of multilayer objects using the concept of cuboids. Objects are discretized into layers of cube-like objects, allowing layers to be easily peeled away for display. Joy and Conkey [58] apply the results of [32] to visualize the envelope of a swept trivariate B-spline solid. The crux of the technique is to generate the trivariates formed from the parametric boundary of the surface and union them with the solid generated by the swept implicit boundary. Recently, Martin *et al.* [59] have employed a frustum-based approach to rendering the isosurfaces of trivariate volumes. Their technique leverages the subdivision and convex-hull properties of splines along with an optimized oriented bounding box hierarchy to expedite the isolation of isosurface-frustum intersections. A further advantage of their technique is that it trivially affords rendering the underlying geometry in the context of the isosurface.

2.2.4 Medial Axis Transforms

This dissertation will introduce a new modeling operator which is based on the concept of a medial axis and an associated generalized cylinder. The notion of a medial axis was formalized by Blum [60] and later reformulated using his famous "grass fire analogy" [61]. Given a closed curve in the plane, the medial axis of the curve can be found by setting a fire along the perimeter, and allowing the fire to propagate inward at a fixed rate. When two fire fronts collide at so-called "quench points," they are extinguished. The collection of all quench points is termed the medial axis. Other definitions are possible, as well. The medial axis can be seen as the locus of the centers of all maximal circles which are inscribed in a closed curve, and touch it at two or more points. It

can also be found contained in the Voronoi diagram of the curve boundary. The medial axis can be elevated to higher dimensions, by generalizing the grass fire analogy to a boundary surface, the maximal circle definition to spheres, and the 2D Voronoi to 3D. An object can be completely reconstructed from the maximal circle definition provided that the radius of the maximal sphere is recorded for each point on the axis. See Figure 2.6 for an example of the medial axis transform applied to a planar figure.

The medial axis in some sense forms the backbone of the object from which it is derived. This has made it popular for shape recognition in the vision community and as a handle for character manipulation in the animation community. The axis has also been applied in the field of robotics for collision avoidance. By following the medial axis, the robot is roughly speaking as far as possible from obstacles on the perimeter. There are a number of problems with the medial axis, however. It is notoriously slow to compute, numerically unstable, and very sensitive to noise. The slightest perturbation in a boundary will cause a spike whose magnitude is uncorrelated to the degree of perturbation. This has led Chuang [62–64] to develop an approximation to the medial axis which is less sensitive to noise, and easier to compute as well.

Ahuja and Chuang [62] develop a potential-based model for approximating the medial axis of a 2D polygonal region. Their idea is to treat the boundary of the region as possessing a charge. The force acting on an oppositely charged point in the interior can be calculated by integration over the boundary. This force reaches a local minimum when the particle is approximately equidistant from two boundaries. Thus, valleys in the potential function correspond roughly to the medial axis. Chuang [63] generalizes the potential approach to 3D for the purpose of obstacle avoidance and later [64] applies the technique to skeletonization of three-dimensional polyhedral objects. In three dimensions, the potential-based skeleton does not converge to the

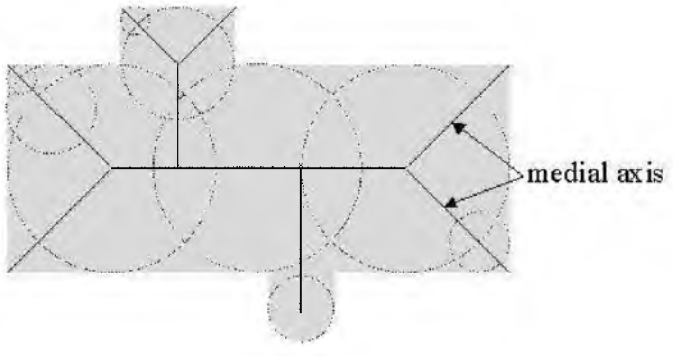

Figure 2.6: Medial axis of a planar figure.

medial axis. Chuang's potential-based skeleton remains a space curve, whereas the medial axis is in fact a series of joined surfaces, termed "medial surfaces." We shall discuss the exact form of the potential-based approach in later chapters.

There are many other excellent skeletonization algorithms. In particular, we have considered two. Lee *et al.* [65] modify the erosion operator to thin a voxelized model while preserving component connectivity to yield a thin skeleton. Cohen-Or *et al.* [66] employ a sequence of repulsive forces and edge collapses to produce a skeleton from a model.

A number of articles have further motivated our research. Yao *et al.* [67] present a simple method for computing the medial axis of a polygon and Chin *et al.* [68] manage it in linear time. Wolter [69] discusses the role of the medial axis in shape representation, and explores the relationship between the homotopy type of a solid and the homotopy type of its medial axis. Amenta *et al.* [70–72] develop methods for surface reconstruction from unorganized data by calculating the approximate medial axis of the point cloud.

2.2.5 Surface / Volume Completion

Surface completion is the process of "filling in" a surface from its boundary representation. As such, it bears some similarity to the problem of hole-filling, for which several methods have been introduced (*e.g.*, [73]). Often these techniques do not address the parameterization of the filled region. When the parameterization is addressed, it is often piecemeal, composed of a series of adjacent parametric patches.

There are a number of classic works on completing a surface from a series of bounding curves [74–77]. This work is most closely related to the algorithm we will develop. However, in contrast to our approach, these techniques generally assume the boundary can be naturally decomposed into n-faces, which can in turn be blended together. For example, schemes have been developed to complete a surface from 3, 4, 5, and 6-sided areas. There are many commonly occurring curve examples that do not easily admit such a decomposition

Similarly, volume completion builds a volume representation from a surface boundary representation. While this is not a new problem, the goal of completing a NURBS boundary while preserving its parameterization is relatively new. It is a point of view that has become more popular with the advent of the field called isogeometric analysis [4], as discussed above. The work of Martin *et al.* [78, 79] is most similar to ours, starting with a family of offsets projected inward towards a central midstructure. Their work diverges from ours on the question of how to deal with the difficult area near the axis. We have chosen to span this region with a separate NURBS volume, whereas they have opted for a hybrid representation, filling this region with

tetrahedra. Our work has the disadvantage of possible parametric distortion as the volume elements map to a one-dimensional figure. Theirs must deal with communicating values amongst two representations, with the possible introduction of errors or loss in precision.

Our volume completion operator amounts to a parametrization of the volume bounded by a boundary representation. A number of authors have dealt with the problem of parametrization, especially as it relates to surfaces. Two excellent overviews are provided by [80, 81]. The generalized cylinder (GC) representation was a starting point for our work. The definition of the medial axis as the locus of the centers of spheres that touch the boundary at two or more points suggests a method whereby the original boundary can be reconstructed by tracking a scaling function along the axis and centering a scaled sphere at that point. Our intuition of an offset-based approach to a simple midstructure was borne from this intuition. The GC-based representation was originally proposed by [82] and was recently extended by Chuang *et al.* [83] to their potential-based approach. An equally evocative idea was given by Blum's original work [60] that defined the medial axis as the quench points of fires placed on a boundary.

2.2.6 Global Illumination

In the area of photorealistic rendering, one often speaks of radiance, the function that characterizes the light leaving the surface of an object in a particular direction. Capturing this radiance function facilitates solution of the global illumination equation. Most approaches to radiance computation and storage have simplified the problem by either decreasing the dimensionality of the radiance function or assuming a simplified surface representation. The original radiosity approach to the problem captured view-independent irradiance at discrete points over piecewise-planar patches [11]. Troutman and Max [84] and Zatz [85] extended the basic radiosity approach to higher order functions of surface position. Ward's work on illuminance caching [86] allowed for nonuniform sampling of the diffuse radiance function over arbitrary surfaces, and the work of Greger *et al.* [87] generalized the representation to volumes, allowing for scenes with dynamic elements. Walter *et al.* [88] trace photons from light sources into the environment, and track the intersections of these packets to reconstruct the surface illumination function. Their method applies to polygonal models and does not render view-dependent shading effects (although it does capture view-dependent illumination effects). Photon maps [89] further generalize the approaches of Ward and Walter *et al.* by caching illumination as discrete oriented packets. At render time, the global illumination integral can be approximated at a surface point by considering the closest k packets and performing quadrature. In this way, arbitrary surfaces and a 4D radiance function

are supported. Difficulties include ensuring adequate sampling (thereby avoiding blurring) and determining the k significant neighbors.

A number of techniques for caching angularly varying radiance have been introduced. Walter *et al.* [90] employ Phong lobes as a basis for capturing nondiffuse lighting effects. The radiance function of each surface in the scene is approximated using virtual lights and Phong lobes. Hardware support for interpolated shading enables interactive walkthroughs of these scenes. However, the technique requires the results of a prior global illumination solution, has limited flexibility in the basis functions (*e.g.,* fixed, per-surface Phong exponents), and is restricted to polygons (GL shader). Another approach is to assume that the rendering equation can be factored into a sum-of-products of lower dimensional functions [91]. McCool *et al.* [92] apply factorization to the BRDF, allowing direct use of texture mapping hardware in conjunction with point source-based lighting. Latta and Kolb [93] apply the factorization to the entire GI integral. Some limitations of their approach are the assumption of isotropic materials and a distant environment (environment map). Object-object interactions are not treated, shadowing is due only to self-shadowing, and self-lighting is not accounted for.

Alternatively, spherical harmonics (SH) are a common representation for capturing directional radiance. Sillion *et al.* [12] generalize the basic radiosity approach to cache radiance as spherical harmonics coefficients at the vertices of planar surfaces. Surface colors are then determined by interpolation at render time. Stamminger *et al.* [94] extend this work to a multiresolution representation of the angular radiance. Cabral *et al.* [95] represent the surface reflectance (BRDF) in a 2D spherical harmonics representation by assuming isotropy and discretizing the remaining dimension. They assume an infinitely distant environment (environment map) which can be projected onto the SH basis, reducing the global illumination integral to a sum of coefficients. Sloan *et al.* [96] and Kautz *et al.* [97] take a similar approach, extending the model to include dynamic lighting, self-illumination, and arbitrary BRDFs. The environment is generally assumed to be distant, except for distinguished points in a local neighborhood of the object. Ramamoorthi and Hanrahan [98] have demonstrated that for diffuse materials and distant environments, 9 SH coefficients are sufficient to encode incident illumination.

Similar approaches for capturing surface radiance have been applied in the field of image-based rendering. Wood *et al.* [99] associate a piecewise linear 2D directional radiance map with each texel on the surface, allowing for real-time playback of photographic imagery. An idea related to the one we present was introduced by Malzbender *et al.* in their paper "Polynomial Texture Maps" [100]. Each texel stores 6 coefficients to a biquadratic function that approximates

surface luminance variation with respect to lighting direction. The model assumes directional lighting and a fixed viewing direction to reduce the dimensionality of the radiance function.

Several authors have explored hierarchical approaches to radiant transfer. Hanrahan *et al.* [101] develop a technique for patch-based energy exchange at multiple scales, thereby decreasing the number of interactions required in a radiosity solution. Their basic technique is limited to diffuse polygonal environments. Aupperle and Hanrahan [102] generalize the hierarchical technique to account for glossy reflections, assuming polygonal environments and that illumination is constant over a patch. The work of Gortler *et al.* [103] generalizes the Galerkin and hierarchical methods by introducing the multiscale wavelet basis to radiosity. Their method treats diffuse scenes, does not enforce continuity across patch boundaries (which can result in visible seams), and is implemented solely for polygonal environments (although they hint at an implementation for parametric patches). Yu and Peng [104] apply spline wavelets as a basis for representing irradiance at multiple scales over curved surfaces. Christensen *et al.* [105] demonstrate how wavelets and importance-driven refinement can be used to accelerate the radiosity solution for glossy environments. In contrast to our work, their approach is view dependent, requires a final gather, applies numerical integration in addition to approximation of the GI integrand, and is limited to convex patches. Bala *et al.* [106] introduce a bounded-error quadtree-based method for caching directional radiance on surfaces. The radiance function is generated using lazy evaluation, and whenever possible, colors are interpolated from previously cached values. It is an impressive system, but necessarily makes some concessions for the sake of efficiency and simplicity. In particular, it uses convex primitives, linear interpolation of radiance, and does not support general BRDFs, diffuse interreflections, or area light sources.

The approach we describe is most similar in spirit to that of Redner *et al.* [107]. Their paper develops the theory of B-spline density estimators and applies the resulting formulation to represent illumination functions across smooth surfaces. Our work differs from theirs in significant ways. Our radiance representation is directionally varying, incorporates arbitrary luminaires, addresses trimming curves, and provides for a hierarchical representation of radiance. On a more philosophical note, their work has the flavor of a radiosity formulation, with explicit projection onto basis functions. We take an approximation theoretic approach, and utilize the integration properties of B-splines to simplify the solution of the rendering equation.

CHAPTER 3

RAY TRACING TRIMMED NURBS SURFACES

3.1 Overview

A system is presented for ray tracing trimmed NURBS surfaces. While approaches to components are drawn largely from existing literature, their combination within a single framework is novel. This chapter also distinguishes itself from prior work in that the details of an efficient implementation are fleshed out. Throughout, emphasis is placed on practical methods suitable to implementation in general ray tracing programs.

3.2 Introduction

The modeling community has embraced trimmed NURBS as a primitive of choice. The result has been a rapid proliferation in the number of models utilizing this representation. At the same time, ray tracing has become a popular method for generating computer graphics images of geometric models. Surprisingly, most ray tracing programs do not support the direct use of untessellated trimmed NURBS surfaces. The direct use of untessellated NURBS is desirable because tessellated models increase memory use which can be detrimental to runtime efficiency on modern architectures. In addition, tessellating models can result in visual artifacts, particularly in models with transparent components.

Although several methods of generating ray-NURBS intersections have appeared in the literature [19–21, 23–27], widespread adoption into ray tracing programs has not occurred. We believe this lack of acceptance stems from both the intrinsic algebraic complexity of these methods, and from the lack of emphasis in the literature on clean and efficient implementation. We present a new algorithm for ray-NURBS intersection that addresses these issues. The algorithm modifies approaches already in the literature to attain efficiency and ease of implementation.

Our approach is outlined in Figure 3.1. We create a set of boxes that bound the underlying surface over a given parametric range. The ray is tested for intersection with these boxes, and for a particular box that is hit, a parametric value within the box is used to initiate root-finding. The key issues are determining which boxes to use, how to efficiently manage computing intersections

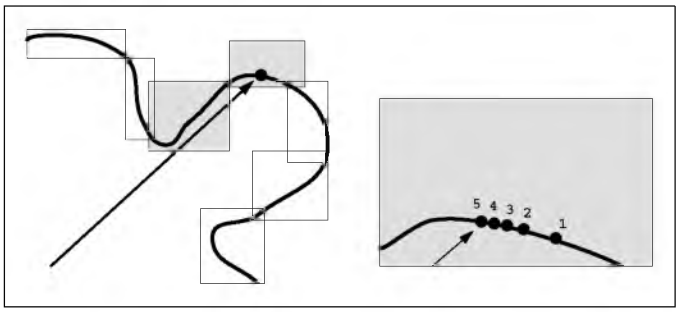

Figure 3.1: The basic method we use to find the intersection of a ray and a parametric object shown in a 2D example. Left: The ray is tested against a series of axis-aligned bounding boxes. Right: For each box hit, an initial guess is generated in the parametric interval the box bounds. Root-finding is then iteratively applied until a convergence or a divergence criterion is met.

with them, how to do the root-finding, and how to efficiently evaluate the geometry for a given parametric value.

We use refinement to generate the bounding volume hierarchy, which results in a shallower tree depth than other subdivision-based methods. We also use an efficient refinement-based point evaluation method to speed root-finding. These choices turn out to be both reasonable to implement and efficient.

In Section 3.3, we present the bulk of our method, in particular how to create a hierarchy of bounding boxes and how to perform root-finding within a single box to compute an intersection with an untrimmed NURBS surface. In Section 3.4, we describe how to extend the method to trimmed NURBS surfaces. Finally, in Section 3.5, we show some results from our implementation of the algorithm.

3.3 Ray Tracing NURBS

In ray tracing a surface, we pose the question "At what points does a ray intersect the surface?" We define a ray as having an origin and a unit direction

$$\mathbf{r}(t) = \mathbf{o} + \hat{\mathbf{d}} * t.$$

A nonuniform rational B-spline (NURBS) surface can be formulated as

$$\mathbf{S}^w(u,v) = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \mathbf{P}^w_{i,j} B_{j,k_u}(u) B_{i,k_v}(v)$$

where the superscript w denotes that our formulation produces a point in rational four space, which must be normalized by the homogeneous coordinate prior to display. The $\{\mathbf{P}^w_{i,j}\}_{i=0,j=0}^{M-1,N-1}$ are the control points $(w_{i,j}x_{i,j}, w_{i,j}y_{i,j}, w_{i,j}z_{i,j}, w_{i,j})$ of the $M \times N$ mesh, having basis functions B_{j,k_u}, B_{i,k_v} of orders k_u and k_v defined over knot vectors

$$\tau_u = \{u_j\}_{j=0}^{N-1+k_u}$$

$$\tau_v = \{v_i\}_{i=0}^{M-1+k_v}$$

formulated as

$$B_{j,k_u}(u) \equiv \begin{cases} 1 & \text{if } k_u = 1 \text{ and } u \in [u_j, u_{j+1}) \\ 0 & \text{if } k_u = 1 \text{ and } u \notin [u_j, u_{j+1}) \\ \frac{u-u_j}{u_{j+k_u-1}-u_j} B_{j,k_u-1}(u) + & \text{otherwise} \\ \frac{u_{j+k_u}-u}{u_{j+k_u}-u_{j+1}} B_{j+1,k_u-1}(u) & \end{cases}$$

$$B_{i,k_v}(v) \equiv \begin{cases} 1 & \text{if } k_v = 1 \text{ and } v \in [v_i, v_{i+1}) \\ 0 & \text{if } k_v = 1 \text{ and } v \notin [v_i, v_{i+1}) \\ \frac{v-v_i}{v_{i+k_v-1}-v_i} B_{i,k_v-1}(v) + & \text{otherwise.} \\ \frac{v_{i+k_v}-v}{v_{i+k_v}-v_{i+1}} B_{i+1,k_v-1}(v) & \end{cases}$$

(NB: For historical reasons, we refer to the "order" k_u or k_v of a surface in the u or v directions, respectively. Recently, it has been more commonplace to speak of the degree d_u or d_v. In this case, $d_u \equiv k_u - 1$ and $d_v \equiv k_v - 1$.) Such a surface $\mathbf{S}$ is defined over the domain $[u_{k_u-1}, u_N) \times [v_{k_v-1}, v_M)$. Each nonempty subinterval $[u_j, u_{j+1}) \times [v_i, v_{i+1})$ corresponds to a surface patch.

In this discussion, we assume that the reader has a basic familiarity with B-Splines. For further introduction, please refer to [5, 108–111].

Following the development by Kajiya [22], we rewrite the ray $\mathbf{r}$ as the intersection of two planes (Figure 3.2), $\{\mathbf{p} \mid \mathbf{P_1} \cdot (\mathbf{p}, 1) = 0\}$ and $\{\mathbf{p} \mid \mathbf{P_2} \cdot (\mathbf{p}, 1) = 0\}$, where $\mathbf{P_1} = (\mathbf{N_1}, d_1)$ and $\mathbf{P_2} = (\mathbf{N_2}, d_2)$. The normal to the first plane is defined as

$$\mathbf{N_1} = \begin{cases} (\hat{\mathbf{d}}_y, -\hat{\mathbf{d}}_x, 0) & \text{if } |\hat{\mathbf{d}}_x| > |\hat{\mathbf{d}}_y| \text{ and } |\hat{\mathbf{d}}_x| > |\hat{\mathbf{d}}_z| \\ (0, \hat{\mathbf{d}}_z, -\hat{\mathbf{d}}_y) & \text{otherwise.} \end{cases}$$

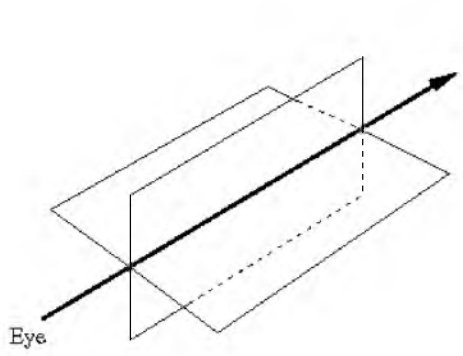

Figure 3.2: A ray formulated as the intersection of two planes.

Thus, $\mathbf{N_1}$ is always perpendicular to the ray direction $\hat{\mathbf{d}}$, as desired. $\mathbf{N_2}$ is simply

$$\mathbf{N_2} = \mathbf{N_1} \times \hat{\mathbf{d}}.$$

Since both planes contain the origin $\mathbf{o}$, it must be the case that $\mathbf{P_1} \cdot (\mathbf{o}, 1) = \mathbf{P_2} \cdot (\mathbf{o}, 1) = 0$. Thus,

$$\begin{aligned} d_1 &= -\mathbf{N_1} \cdot \mathbf{o} \\ d_2 &= -\mathbf{N_2} \cdot \mathbf{o}. \end{aligned}$$

An intersection point on the surface $\mathbf{S}$ must satisfy the conditions

$$\begin{aligned} \mathbf{P_1} \cdot (\mathbf{S}(u,v), 1) &= 0 \\ \mathbf{P_2} \cdot (\mathbf{S}(u,v), 1) &= 0. \end{aligned}$$

The resulting implicit equations can be solved for u and v using numerical methods.

Ray tracing a NURBS surface proceeds in a series of steps. As a preprocess, the control mesh is flattened using refinement. There are several reasons for this. For Newton to converge quadratically, our initial guess for the root (u_*, v_*) must be close. By refining the mesh, we can bound the various subpatches, and use the bounding volume hierarchy (BVH) both to cull the rays, and also to narrow the prospective parametric domain and so yield a good initial root estimate. It is important to note that the refined mesh does not persist in memory. It is used to

generate the BVH and then is discarded.

During the intersection process, if we reach a leaf of the BVH, we apply traditional numerical root finding to the implicit equations above. The result will determine either a single (u_*, v_*) value or that no root exists.

In the sections that follow, we discuss the details of flattening, generating the BVH, root finding, evaluation, and partial refinement. Together. these are all that is needed for computing ray-NURBS intersections.

3.3.1 Flattening

For Newton to converge both swiftly and reliably, the initial guess must be suitably close to the actual root. We employ a flattening procedure — *i.e.,* refining/subdividing the control mesh so that each span meets some flatness criteria — both to ensure that the initial guess is a good one, and for the purpose of generating a bounding volume hierarchy for ray culling.

A wealth of research exists on polygonization of spline surfaces, *e.g.,* [14–17], and for the most part, these approaches can be readily applied to the problem of spline flattening. Some differences merit discussion. First, in the case of finding the numerical ray-spline intersection, we are not so much interested in flatness as in guaranteeing that there are not multiple roots within a leaf node of the bounding volume hierarchy. We note that this guarantee cannot always be made, particularly for nodes which contain silhouettes according to the ray source. Fortunately, the convergence problems which these boundary cases entail can also be improved with the mesh flattening we prescribe. We would also like to avoid any local maxima and minima that would serve to delay or, worse yet, prevent the convergence of our scheme. The flatness testing utilized by tessellation routines can be used to prevent these situations.

As ray tracing splines is at the outset a complicated task, we recommend the application of as simple a flattening procedure as possible. We have examined two flattening techniques in detail. The first of these is an adaptive subdivision scheme given by Peterson in [16]. As the source for the *Graphics Gems* is publicly available, we will not discuss that method here, but instead refer the reader to the source.

The second approach we have considered is curvature-based refinement of the knot vectors. The number of knots to add to a knot interval is based on a simple heuristic which we now present.

Suppose we have a B-spline curve $\mathbf{c}(t)$. An oracle for determining the extent to which the span $[t_i, t_{i+1})$ should be refined is given by the product of its maximum curvature and its length over that span. Long curve segments should be divided in order to ensure that the initial guess for the numerical solver is reasonably close to the actual root. High curvature regions should be

split to avoid multiple roots. As we are utilizing maximum curvature as a measure of flatness, our heuristic will be overly conservative for curves other than circles. The heuristic value for the number of knots to add to the current span is given by

$$n_1 = C_1 * \max_{[t_i, t_{i+1})} \{\text{curvature}(\mathbf{c}(t))\} * \text{arclen}(\mathbf{c}(t))_{[t_i, t_{i+1})}.$$

We also choose to bound the deviation of the curve from its linear approximation. This notion will imbue our heuristic with scale dependence. Thus, for example, large circles will be broken into more pieces than small circles. Erring again on the conservative side, suppose our curve span is a circle with radius r, which we are approximating with linear segments. A measure of the accuracy of the approximation can be phrased in terms of a chord height h which gives the maximum deviation of the facets from the circle. Observing Figure 3.3, it can be seen that

$$\begin{aligned} h &= r - d \\ &= r - r\cos\frac{\theta}{2} \\ &\approx r(1 - (1 - \frac{\theta^2}{8})) \\ &= \frac{r\theta^2}{8}. \end{aligned}$$

The number of segments n_2 required to produce a curve within this tolerance is computed by

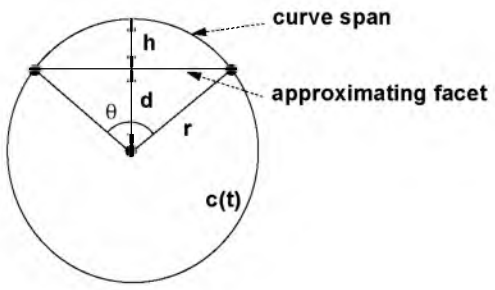

Figure 3.3: Illustration of the chord height tolerance heuristic.

$2\pi/\theta$. Thus, we have

$$\begin{aligned} n_2 &= \frac{2\pi}{\sqrt{\frac{8h}{r}}} \\ &= \frac{2\pi\sqrt{r}}{\sqrt{8h}} \\ &\approx C_2 * \sqrt{\text{arclen}(\mathbf{c}(t))_{[t_i,t_{i+1})}}. \end{aligned}$$

Combining the preceding oracles for curve behavior, our heuristic for the number of knots n to add to an interval will be $n_1 * n_2$:

$$n = C * \max_{[t_i,t_{i+1})} \{\text{curvature}(\mathbf{c}(t))\} * \text{arclen}(\mathbf{c}(t))^{3/2}_{[t_i,t_{i+1})}$$

where C allows the user to control the fineness. Since the maximum curvature and the arc length are in general hard to come by, we will estimate their values. The arc length of $\mathbf{c}$ over the interval is given by

$$\int_{t_i}^{t_{i+1}} |\mathbf{c}'(t)|dt = \text{avg}_{[t_i,t_{i+1})}\{|\mathbf{c}'(t)|\} * (t_{i+1} - t_i).$$

Curvature is defined as

$$\begin{aligned} \text{curvature}(\mathbf{c}(t)) &= \frac{|\mathbf{c}''(t) \times \mathbf{c}'(t)|}{|\mathbf{c}'(t)|^3} \\ &= \frac{|\mathbf{c}''(t)||\mathbf{c}'(t)||\sin\theta|}{|\mathbf{c}'(t)|^3} \\ &= \frac{|\mathbf{c}''(t)||\sin\theta|}{|\mathbf{c}'(t)|^2} \\ &\leq \frac{|\mathbf{c}''(t)|}{|\mathbf{c}'(t)|^2} \end{aligned}$$

We make the simplification $\text{curvature}(\mathbf{c}(t)) \approx \frac{|\mathbf{c}''(t)|}{|\mathbf{c}'(t)|^2}$. In general, this estimate of the curvature will be overstated. The error will be on the side of refining too finely rather than not finely enough, so it is an acceptable trade-off to get the speed of computing second derivatives instead of curvature.

We are interested in the maximum curvature over the interval $[t_i, t_{i+1})$

$$\max_{[t_i,t_{i+1})} \{\text{curvature}(\mathbf{c}(t)\} \approx \frac{\max_{[t_i,t_{i+1})}\{|\mathbf{c}''(t)|\}}{\text{avg}_{[t_i,t_{i+1})}\{|\mathbf{c}'(t)|\}^2}.$$

If we assume the curve is polynomial, then the first derivative restricted to the interval $[t_i, t_{i+1})$ is given by

$$\mathbf{c}'(t) = \sum_{j=i-k+2}^{i} \frac{(k-1)(\mathbf{P_j} - \mathbf{P_{j-1}})}{t_{j+k-1} - t_j} B_{j,k-1}(t)$$

where $\{\mathbf{P_j}\}$ are the control points of the curve. The derivative of a rational curve is considerably more complicated. While the polynomial formulation of the derivative is in general a poor approximation to the rational derivative, in the case of flattening, we have obtained reasonable results when applying the former to rational curves. For the remainder of this section, we shall assume that rational control points have been projected into Euclidean 3-space.

Since $\sum B_{j,k-1}(t) = 1$, we can approximate the average velocity by averaging the control points $\mathbf{V_j}$ of the derivative curve:

$$\text{avg}_{[t_i,t_{i+1})}\{\mathbf{c}'(t)\} \approx \frac{1}{k-1} \sum_{j=i-k+2}^{i} (k-1)\frac{(\mathbf{P_j} - \mathbf{P_{j-1}})}{t_{j+k-1} - t_j} = \frac{1}{k-1} \sum_{j=i-k+2}^{i} \mathbf{V_j}$$

where $\mathbf{V_j} = (k-1)\frac{(\mathbf{P_j} - \mathbf{P_{j-1}})}{t_{j+k-1} - t_j}$. The average speed is therefore

$$\text{avg}_{[t_i,t_{i+1})}\{|\mathbf{c}'(t)|\} \approx \frac{1}{k-1} \sum_{j=i-k+2}^{i} |\mathbf{V_j}|.$$

The second derivative over $[t_i, t_{i+1})$ is given by

$$\mathbf{c}''(t) = \sum_{j=i-k+3}^{i} (k-2)\frac{[\mathbf{V_j} - \mathbf{V_{j-1}}]}{t_{j+k-2} - t_j} B_{j,k-2}(t) = \sum_{j=i-k+3}^{i} \mathbf{A_j} B_{j,k-2}(t)$$

where $\mathbf{A_j} = (k-2)\frac{[\mathbf{V_j} - \mathbf{V_{j-1}}]}{t_{j+k-2} - t_j}$. Using the convex hull property again, the maximum magnitude

of the second derivative is approximated by the maximum magnitude of the second derivative curve control points $\mathbf{A_j}$:

$$\max_{[t_i,t_{i+1})}\{|\mathbf{c}''(t)|\} \approx \max_{i-k+3\le j\le i}\{|\mathbf{A_j}|\}.$$

Our heuristic is finally:

$$\begin{aligned} n &= C * \frac{\max_{[t_i,t_{i+1})}\{|\mathbf{c}''(t)|\} * [\text{avg}_{[t_i,t_{i+1})}\{|\mathbf{c}'(t)|\} * (t_{i+1}-t_i)]^{3/2}}{\text{avg}_{[t_i,t_{i+1})}\{|\mathbf{c}'(t)|\}^2} \\ &= C * \frac{\max_{[t_i,t_{i+1})}\{|\mathbf{c}''(t)|\} * (t_{i+1}-t_i)]^{3/2}}{\text{avg}_{[t_i,t_{i+1})}\{|\mathbf{c}'(t)|\}^{1/2}} \\ &= C * \frac{\max_{i-k+3\le j\le i}\{|\mathbf{A_j}|\}(t_{i+1}-t_i)^{3/2}}{(\frac{1}{k-1}\sum_{j=i-k+2}^{i}|\mathbf{V_j}|)^{1/2}}. \end{aligned}$$

For each row of the mesh, we apply the above heuristic to calculate how many knots need to be added to each u knot interval, the final number being the maximum across all rows. This process is repeated for each column in order to refine the v knot vector. The inserted knots are spaced uniformly within the existing knot intervals.

As a final step in the flattening routine, we transform all of the knot intervals in the refined knot vectors $\mathbf{t_u}$ and $\mathbf{t_v}$ into intervals with "open" end conditions. By this, we mean that we give multiplicity $k_u - 1$ to each internal knot of $\mathbf{t_u}$ and multiplicity k_u to each end knot. Similarly, we give multiplicities of $k_v - 1$ and k_v to the internal and external knots, respectively, of $\mathbf{t_v}$. The results are Bézier surface patches that correspond to each nonempty interval $[u_i, u_{i+1}) \times [v_j, v_{j+1})$ of $\mathbf{t_u} \times \mathbf{t_v}$, each of which can be bounded using the convex hull of the corresponding refined surface mesh points. This becomes critical in the next section.

The refined knot vectors determine the refinement matrix used to transform the existing mesh into the refined mesh. There are many techniques for generating this "alpha matrix." As it is not critical that this be fast, we refer the reader to several sources [109, 112–114].

Both adaptive subdivision and curvature-based refinement should yield acceptable results. Both allow the user to adjust the resulting flatness via a simple intuitive parameter. We have preferred the latter mainly because it produces its result in a single pass, without the creation of unnecessary intermediate points. Adaptive subdivision does have the advantage of inserting one knot value at a time, so one does not necessarily need to implement the full machinery of the Oslo algorithm [113]. Instead, one can opt for a simpler approach, such as that of Boehm [112]. It

is not clear which method produces more optimal meshes in general. On the one hand, adaptive subdivision computes intermediate results which it then inspects to determine where additional subdivision is required. On the other hand, our method utilizes refinement (we only subdivide in the last step), and this converges more swiftly to the underlying surface than does subdivision.

Neither technique is entirely satisfactory. Each considers the various parametric directions independently, while subdivision and refinement clearly impact both directions. The curvature-based refinement method refines a knot interval without considering the impact of that refinement on neighboring intervals. This can lead to unnecessary refinement. Neither makes any attempt to find optimal placement of inserted knots.

The adaptive subdivision and curvature-based refinement methods are the products of the inevitable compromise between refinement speed and quality. Both satisfy the efficiency and accuracy demands of the problem at hand.

A point which we do not wish to sweep under the carpet is that the selection of the flatness parameter is empirical and left to the user. As this parameter directly impacts the convergence of the root finding process, it should be carefully chosen. Too small a value may cause the numerical solver to fail to converge or to converge to one of several roots in the given parametric interval. This effect will probably be most noticeable along silhouette edges and patch boundaries. On the other hand, too large a value will result in over-refinement of the surface, leading to a deeper bounding volume hierarchy, and therefore, potentially more time per ray. We have found that after some experimentation, one develops an intuition for the sorts of parameters which work for a surface. For an example of a system which guarantees convergence without user intervention, see Toth [26]. This guarantee is made at the price of linear convergence in the root finding procedure.

3.3.2 Bounding Volume Hierarchy

We build a bounding volume hierarchy using the points of the refined control mesh we found in the previous section. The root and internal nodes of the tree will contain simple primitives which bound portions of the underlying surface. The leaves of the tree are special objects, which we call *interval objects*, and are used to provide an initial guess (in our case, the midpoint of the bracketing parametric interval) to the Newton iteration. We will now examine the specifics in more detail.

The convex hull property of B-spline surfaces guarantees that the surface is contained in the convex hull of its control mesh. As a result, any convex objects which bound the mesh will bound the underlying surface. We can actually make a stronger claim; because we converted our knot

vectors into "open" knot vectors in the last section (made the multiplicity of the internal knots $k-1$), each nonempty interval $[u_i, u_{i+1}) \times [v_j, v_{j+1})$ corresponds to a surface patch which is completely contained in the convex hull of its corresponding mesh points. Thus, if we produce bounding volumes for each of these intervals, we will have completely enclosed the surface (refer to Figure 3.4.) We form the tree by sorting the volumes according to the axis direction which has greatest extent across the bounding volumes, splitting the data in half, and repeating the process.

There remains the dilemma of which primitive to use as a bounding volume. Many different objects have been tried, including spheres [21], axis-aligned boxes [21, 25, 27], oriented boxes [21], and parallelepipeds [19]. There is generally a tradeoff between speed of intersection and tightness of fit. The analysis is further complicated by the fact that bounding volume performance depends on the type of scene being rendered.

We have preferred simplicity, narrowing our choice to spheres and axis-aligned boxes. Spheres have a very fast intersection test. However, spheres, by definition, can never be flat (see Figure 3.4). Since our intersection routines require surfaces which are locally "flat," spheres did not seem to be a natural choice.

Axis-aligned boxes have many advantages. First, they can become flat (at least along axis directions), so they can provide a tighter fit than spheres. The union of two axis-aligned boxes is easily computed. This computation is necessary when building the BVH from the leaves.

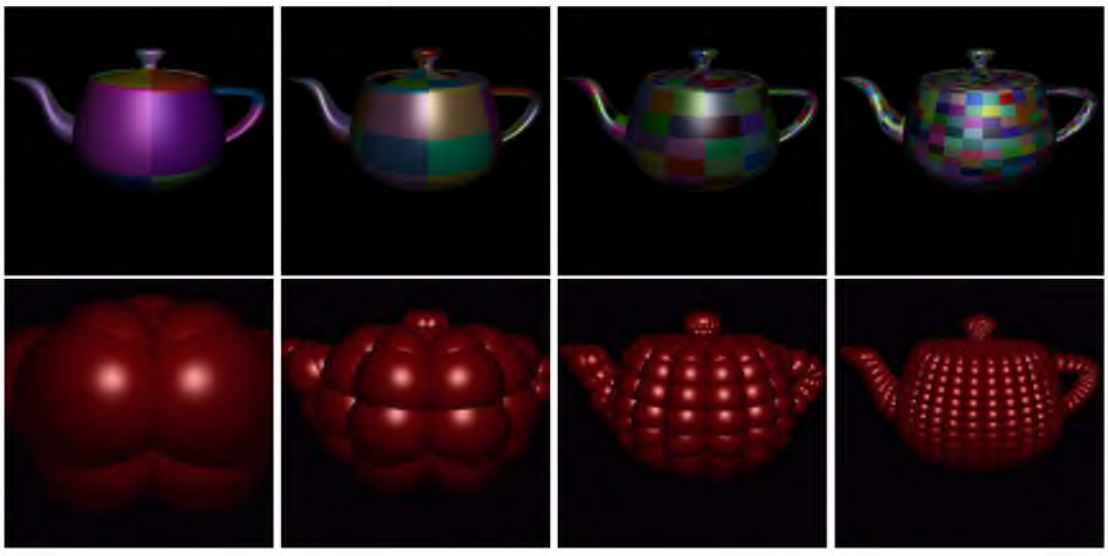

Figure 3.4: Convergence of the bounding volume hierarchy under 4-to-1 subdivision. The top figures show the parametric intervals. The bottom figures show the spheres bounding the corresponding control meshes.

With many other bounding volumes, the leaves of the subtree must be examined individually to produce reasonable bounding volumes. Finally, many scenes are axis-aligned, especially in the case of architectural walkthroughs. Axis-aligned boxes are nearly ideal in this circumstance.

A simple ray-box intersection routine is intuitive, and so we omit its discussion. An optimized version can be found in the paper by Smits [115].

3.3.3 Root Finding

Given a ray as the intersection of planes $\mathbf{P_1} = (\mathbf{N_1}, d_1)$ and $\mathbf{P_2} = (\mathbf{N_2}, d_2)$, our task is to solve for the roots (u_*, v_*) of

$$\mathbf{F}(u,v) = \begin{pmatrix} \mathbf{N_1} \cdot \mathbf{S}(u,v) + d_1 \\ \mathbf{N_2} \cdot \mathbf{S}(u,v) + d_2 \end{pmatrix}.$$

A variety of numerical methods can be applied to the problem. An excellent reference for these techniques is [116, pp 347-393]. We use Newton's method for several reasons. First, it converges quadratically if the initial guess is close, which we ensure by constructing a bounding volume hierarchy. Furthermore, the surface derivatives exist and are calculated at cost comparable to that of surface evaluation. This means that there is likely little computational advantage to utilizing approximate derivative methods such as Broyden.

Newton's method is built from a truncated Taylor's series. Our iteration takes the form

$$\begin{pmatrix} u_{n+1} \\ v_{n+1} \end{pmatrix} = \begin{pmatrix} u_n \\ v_n \end{pmatrix} - \mathbf{J}^{-1}(u_n, v_n) * \mathbf{F}(u_n, v_n)$$

where $\mathbf{J}$ is the Jacobian matrix of $\mathbf{F}$, defined as

$$\mathbf{J} = (\mathbf{F_u}, \mathbf{F_v}).$$

$\mathbf{F_u}$ and $\mathbf{F_v}$ are the vectors

$$\begin{aligned} \mathbf{F_u} &= \begin{pmatrix} \mathbf{N_1} \cdot \mathbf{S_u}(u,v) \\ \mathbf{N_2} \cdot \mathbf{S_u}(u,v) \end{pmatrix} \\ \mathbf{F_v} &= \begin{pmatrix} \mathbf{N_1} \cdot \mathbf{S_v}(u,v) \\ \mathbf{N_2} \cdot \mathbf{S_v}(u,v) \end{pmatrix}. \end{aligned}$$

The inverse of the Jacobian is calculated using a result from linear algebra:

$$\mathbf{J}^{-1} = \frac{\mathrm{adj}(\mathbf{J})}{\det(\mathbf{J})}.$$

The adjoint $\mathrm{adj}(\mathbf{J})$ is equal to the transpose of the cofactor matrix

$$\mathbf{C} = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

where $\mathbf{C}_{ij} = (-1)^{i+j}\det(\mathbf{J}_{ij})$ and $\mathbf{J}_{ij}$ is the submatrix of $\mathbf{J}$ which remains when the ith row and jth column are removed. We find that

$$\mathrm{adj}(\mathbf{J}) = \begin{pmatrix} \mathbf{J}_{22} & -\mathbf{J}_{12} \\ -\mathbf{J}_{21} & \mathbf{J}_{11} \end{pmatrix}.$$

We use four criteria, drawn from Yang [27], to decide when to terminate the Newton iteration. The first condition is our success criterion: if we are closer to the root than some predetermined ϵ

$$||\mathbf{F}(u_n, v_n)|| < \epsilon$$

then we report a hit. Otherwise, we continue the iteration. The other three criteria are failure criteria, meaning that if they are met, we terminate the iteration and report a miss. We do not allow the new (u_*, v_*) estimate to take us farther from the root than the previous one:

$$||\mathbf{F}(u_{n+1}, v_{n+1})|| > ||\mathbf{F}(u_n, v_n)||.$$

We also do not allow the iteration to take us outside the parametric domain of the surface:

$$u \notin [u_{k_u-1}, u_N), v \notin [v_{k_v-1}, v_M).$$

We limit the number of iterations allowed for convergence:

$$iter > MAXITER.$$

We set $MAXITER$ around 7, but the average number of iterations needed to produce convergence is 2 or 3 in practice.

A final check is made to assure that the Jacobian $\mathbf{J}$ is not singular. While this would seem to be a rare occurrence in theory, we have encountered this problem in practice. In the situation where $\mathbf{J}(u_k, v_k)$ is singular, either the surface is not regular ($\mathbf{S_u} \times \mathbf{S_v} = 0$) or the ray is parallel to a silhouette ray at the point $\mathbf{S}(u_k, v_k)$. (A proof of this assertion is given in the Appendix.) In either situation, to determine singularity, we test

$$|\det(\mathbf{J})| < \epsilon.$$

If the Jacobian is singular, we perform a jittered perturbation of the parametric evaluation point,

$$\begin{pmatrix} u_{k+1} \\ v_{k+1} \end{pmatrix} = \begin{pmatrix} u_k \\ v_k \end{pmatrix} + .1 * \begin{pmatrix} \text{drand48}() * (u_0 - u_k) \\ \text{drand48}() * (v_0 - v_k) \end{pmatrix}$$

and initiate the next iteration. This operation tends to push the iteration away from problem regions without leaving the basin of convergence.

Because any root (u_*, v_*) produced by the Newton iteration is approximate, it will almost definitely not lie along the ray $\mathbf{r} = \mathbf{o} + t * \hat{\mathbf{d}}$. In order to obtain the closest point along $\mathbf{r}$ to the approximate root (u_*, v_*), we perform a projection

$$t = (\mathbf{P} - \mathbf{o}) \cdot \hat{\mathbf{d}}.$$

The approximate nature of the convergence also impacts other parts of the ray tracing system. Often, a tolerance ϵ is defined to determine the minimum distance a ray can travel before reporting an intersection. This prevents self-intersections due to errors in numerical calculation. The potential for error is larger in the case of numerical spline intersection than, say, ray-polygon intersection. Thus, the tolerances will need to be adjusted accordingly. Failure to make this adjustment will result in "surface acne" [117] (see Figure 3.5).

An enhanced method for abating acne would test the normal at points less than ϵ along the ray to determine whether these points were on the originating surface. Unfortunately, we have found that we cannot rely on modeling programs to produce consistently oriented surfaces. Therefore, our system utilizes the coarser ϵ condition above.

Figure 3.5: Left: Failure to adjust tolerances may result in surface acne. Right: Properly adjusted tolerances solve the problem. Black regions result from truncated ray depth.

3.3.4 Evaluation

The Newton iteration requires us to perform surface and derivative evaluation at a point (u, v) on the surface. In this section, we examine how this can be accomplished efficiently. Prior work on efficient evaluation can be found in [114, 118].

We begin by examining the problem in the context of B-spline curves. We then generalize the result to surfaces. The development parallels that found in [119].

We evaluate a curve $\mathbf{c}(t)$ by using refinement to stack $k-1$ knots (where k is the order of the curve) at the desired parameter value t_*. The refined curve is defined over a new knot vector $\mathbf{t}$ with basis functions $N_{i,k}(t)$ and new control points $\omega_i \mathbf{D}_i$.

Recall the recurrence for the B-Spline basis functions:

$$N_{i,k}(t) = \begin{cases} 1 & \text{if } k = 1 \text{ and } t \in [t_i, t_{i+1}) \\ 0 & \text{if } k = 1 \text{ and } t \notin [t_i, t_{i+1}) \\ \frac{t-t_i}{t_{i+k-1}-t_i} N_{i,k-1}(t) + \frac{t_{i+k}-t}{t_{i+k}-t_{i+1}} N_{i+1,k-1}(t) & \text{otherwise.} \end{cases}$$

Let $t_* \in [t_\mu, t_{\mu+1})$. As a result of refinement, $t_* = t_\mu = \ldots = t_{\mu-k+2}$. According to the definition of the basis functions, $N_{\mu,1}(t_*) = 1$. There are only two potentially nonzero basis functions of order $k = 2$, namely, those dependent on $N_{\mu,1}$: $N_{\mu,2}$ and $N_{\mu-1,2}$. From the recurrence,

$$\begin{aligned} N_{\mu,2}(t_*) &= \frac{t_* - t_\mu}{t_{\mu+k-1} - t_\mu} N_{\mu,1}(t_*) + \frac{t_{i+k} - t_*}{t_{\mu+k} - t_{\mu+1}} N_{\mu+1,1}(t_*) \\ &= 0 * 1 + \frac{t_{i+k} - t_*}{t_{\mu+k} - t_{\mu+1}} * 0 \\ &= 0 \end{aligned}$$

and,

$$\begin{aligned} N_{\mu-1,2}(t_*) &= \frac{t_* - t_{\mu-1}}{t_{\mu+k-2} - t_{\mu-1}} N_{\mu-1,1}(t_*) + \frac{t_{\mu+k-1} - t_*}{t_{\mu+k-1} - t_\mu} N_{\mu,1}(t_*) \\ &= 0 * 0 + 1 * 1 \\ &= 1. \end{aligned}$$

Likewise, the only nonzero order $k = 3$ terms will be those dependent on $N_{\mu-1,2}$: $N_{\mu-1,3}$ and $N_{\mu-2,3}$.

$$\begin{aligned} N_{\mu-1,3}(t_*) &= \frac{t_* - t_{\mu-1}}{(\cdots)} * 1 + (\cdots) * 0 \\ &= 0 \end{aligned}$$

$$\begin{aligned} N_{\mu-2,3}(t_*) &= (\cdots) * 0 + \frac{t_{\mu-2+k} - t_*}{t_{\mu-2+k} - t_{\mu-1}} * 1 \\ &= 1 \end{aligned}$$

The pattern that emerges is that $N_{\mu-k+1,k}(t_*) = 1$. A straightforward consequence of this result is

$$\mathbf{c}(t_*) = \frac{\sum_i N_{i,k}(t_*)\omega_i \mathbf{D_i}}{\sum_i N_{i,k}(t_*)\omega_i} = \frac{\omega_{\mu-k+1}\mathbf{D_{\mu-k+1}}}{\omega_{\mu-k+1}} = \mathbf{D_{\mu-k+1}}.$$

The point with index $\mu - k + 1$ in the refined control polygon yields the point on the curve.

A further analysis can be used to yield the derivative. Given a rational curve

$$\mathbf{c}(t) = \frac{\sum_i N_{i,k}(t)\omega_i \mathbf{D_i}}{\sum_i N_{i,k}(t)\omega_i} = \frac{\sum_i N_{i,k}(t)\mathbf{D_i^\omega}}{\sum_i N_{i,k}(t)\omega_i} = \frac{\mathbf{D}(t)}{\omega(t)},$$

where $\mathbf{D_i^\omega} = \omega_i \mathbf{D_i}$, the derivative is given by the quotient rule

$$\mathbf{c}'(t) = \frac{\omega(t)(\mathbf{D}^\omega)'(t) - \mathbf{D}^\omega(t)\omega'(t)}{\omega(t)^2}.$$

By the preceding analysis, $\mathbf{D}^{\omega}(t_*) = \mathbf{D}^{\omega}_{\mu-\mathbf{k}+\mathbf{1}}$. Likewise, $\omega(t_*) = \omega_{\mu-k+1}$. The derivative of the B-Spline basis function is given by

$$N'_{i,k}(t) = (k-1)\left[\frac{N_{i,k-1}(t)}{t_{i+k-1}-t_i} - \frac{N_{i+1,k-1}(t)}{t_{i+k}-t_{i+1}}\right].$$

Evaluating the derivative at t_*, we have

$$\begin{aligned}
(\mathbf{D}^{\omega})'(t_*) &= \sum_i N'_{i,k}(t_*)\mathbf{D}^{\omega}_{\mathbf{i}} \\
&= (k-1)\sum_i \mathbf{D}^{\omega}_{\mathbf{i}}\left(\frac{N_{i,k-1}(t_*)}{t_{i+k-1}-t_i} - \frac{N_{i+1,k-1}(t_*)}{t_{i+k}-t_{i+1}}\right) \\
&= (k-1)\left[\sum_i \mathbf{D}^{\omega}_{\mathbf{i}}\frac{N_{i,k-1}(t_*)}{t_{i+k-1}-t_i} - \sum_i \mathbf{D}^{\omega}_{\mathbf{i}}\frac{N_{i+1,k-1}(t_*)}{t_{i+k}-t_{i+1}}\right].
\end{aligned}$$

We know the only nonzero basis function of order $k-1$ is $N_{\mu-k+2,k-1}(t_*) = 1$. Therefore,

$$(\mathbf{D}^{\omega})'(t_*) = (k-1)\left[\frac{\mathbf{D}^{\omega}_{\mu-\mathbf{k}+\mathbf{2}}}{t_{\mu+1}-t_{\mu-k+2}} - \frac{\mathbf{D}^{\omega}_{\mu-\mathbf{k}+\mathbf{1}}}{t_{\mu+1}-t_{\mu-k+2}}\right].$$

Analogously,

$$\omega'(t_*) = (k-1)\left[\frac{\omega_{\mu-k+2}}{t_{\mu+1}-t_{\mu-k+2}} - \frac{\omega_{\mu-k+1}}{t_{\mu+1}-t_{\mu-k+2}}\right].$$

Plugging in for $\mathbf{c}'(t)$

$$\begin{aligned}
\mathbf{c}'(t_*) &= (k-1)\left[\frac{\frac{\mathbf{D}^{\omega}_{\mu-\mathbf{k}+\mathbf{2}}-\mathbf{D}^{\omega}_{\mu-\mathbf{k}+\mathbf{1}}}{t_{\mu+1}-t_{\mu-k+2}}\omega_{\mu-k+1} - \mathbf{D}^{\omega}_{\mu-\mathbf{k}+\mathbf{1}}\frac{\omega_{\mu-k+2}-\omega_{\mu-k+1}}{t_{\mu+1}-t_{\mu-k+2}}}{\omega^2_{\mu-k+1}}\right] \\
&= \frac{k-1}{(t_{\mu+1}-t_{\mu-k+2})\omega_{\mu-k+1}}\left[\mathbf{D}^{\omega}_{\mu-\mathbf{k}+\mathbf{2}} - \mathbf{D}^{\omega}_{\mu-\mathbf{k}+\mathbf{1}}\frac{\omega_{\mu-k+2}}{\omega_{\mu-k+1}}\right] \\
&= \frac{k-1}{(t_{\mu+1}-t_{\mu-k+2})\omega_{\mu-k+1}}\left[\omega_{\mu-k+2}\mathbf{D}_{\mu-\mathbf{k}+\mathbf{2}} - \omega_{\mu-k+1}\mathbf{D}_{\mu-\mathbf{k}+\mathbf{1}}\frac{\omega_{\mu-k+2}}{\omega_{\mu-k+1}}\right] \\
&= \frac{(k-1)\omega_{\mu-k+2}}{(t_{\mu+1}-t_*)\omega_{\mu-k+1}}[\mathbf{D}_{\mu-\mathbf{k}+\mathbf{2}} - \mathbf{D}_{\mu-\mathbf{k}+\mathbf{1}}].
\end{aligned}$$

The result for surface evaluation follows directly from the curve derivation, due to the inde-

pendence of the parameters in the tensor product, so we shall simply state the results:

$$\begin{aligned}
\mathbf{S}(u_*, v_*) &= \mathbf{D}_{\mu_\mathbf{v}-\mathbf{k_v}+\mathbf{1},\mu_\mathbf{u}-\mathbf{k_u}+\mathbf{1}} \\
\mathbf{S_u}(u_*, v_*) &= \frac{(k_u-1)\omega_{\mu_v-k_v+1,\mu_u-k_u+2}}{(u_{\mu_u+1}-u_*)\omega_{\mu_v-k_v+1,\mu_u-k_u+1}}[\mathbf{D}_{\mu_\mathbf{v}-\mathbf{k_v}+\mathbf{1},\mu_\mathbf{u}-\mathbf{k_u}+\mathbf{2}} - \mathbf{D}_{\mu_\mathbf{v}-\mathbf{k_v}+\mathbf{1},\mu_\mathbf{u}-\mathbf{k_u}+\mathbf{1}}] \\
\mathbf{S_v}(u_*, v_*) &= \frac{(k_v-1)\omega_{\mu_v-k_v+2,\mu_u-k_u+1}}{(v_{\mu_v+1}-v_*)\omega_{\mu_v-k_v+1,\mu_u-k_u+1}}[\mathbf{D}_{\mu_\mathbf{v}-\mathbf{k_v}+\mathbf{2},\mu_\mathbf{u}-\mathbf{k_u}+\mathbf{1}} - \mathbf{D}_{\mu_\mathbf{v}-\mathbf{k_v}+\mathbf{1},\mu_\mathbf{u}-\mathbf{k_u}+\mathbf{1}}].
\end{aligned}$$

The normal $\mathbf{n}(u, v)$ is given by the cross product of the first order partials:

$$\mathbf{n}(u_*, v_*) = \mathbf{S_u}(u_*, v_*) \times \mathbf{S_v}(u_*, v_*).$$

If the surface is not regular (*i.e.*, $\mathbf{S_u} \times \mathbf{S_v} = 0$), then our computation may erroneously generate a zero surface normal. We avoid these problem areas in our numerical solver by perturbing the parametric points (see Section 3.3.3).

3.3.5 Partial Refinement

We still need to explain how to calculate the points in the refined mesh so that we can evaluate surface points and derivatives. What follows is drawn directly from Lyche *et al.* [114], tailored to our specialized needs. We again formulate our solution in the context of curves, and then generalize the result to surfaces.

Earlier, we proposed to evaluate the curve $\mathbf{c}$ at t_* by stacking $k-1$ t_*-valued knots in its knot vector τ to generate the refined knot vector $\mathbf{t}$. The B-spline basis transformation defined by this refinement yields a matrix $\mathbf{A}$ which can be used to calculate the refined control polygon $\mathbf{D}^\omega$ from the original polygon $\mathbf{P^w}$:

$$\mathbf{D}^\omega = \mathbf{A}\mathbf{P^w}.$$

We are not interested in calculating the full alpha matrix $\mathbf{A}$, but merely rows $\mu-k+2$ and $\mu-k+1$, as these are used to generate the points $\mathbf{D}^\omega_{\mu-\mathbf{k}+\mathbf{2}}$ and $\mathbf{D}^\omega_{\mu-\mathbf{k}+\mathbf{1}}$ which are required for point and derivative evaluation.

Suppose $t_* \in [\tau_{\mu'}, \tau_{\mu'+1})$. We can generate the refinement for row $\mu+k-1$ using a triangular

scheme

$$\begin{array}{ccc} & & \alpha'_{\mu',0} \\ & \alpha'_{\mu'-1,1} & \alpha'_{\mu',1} \\ & \vdots & \vdots \\ \alpha'_{\mu'-\nu,\nu} & \cdots & \alpha'_{\mu',\nu} \end{array}$$

where ν is the number of knots we are inserting and

$$\begin{aligned} \alpha'_{j,1} &= \delta_{j,\mu'} \\ \alpha'_{j,p+1} &= \gamma_{j,p}\alpha'_{j,p} + (1-\gamma_{j+1,p})\alpha'_{j+1,p} \\ \gamma_{j,p} &= \begin{cases} (t_* - \tau_{\mu'-p+j-(k-1-\nu)})/d, & \text{if } d = \tau_{\mu'+1+j} - \tau_{\mu'-p+j-(k-1-\nu)} > 0 \\ \text{arbitrary} & \text{otherwise.} \end{cases} \end{aligned}$$

$\mathbf{A}_{\mu-k+1,j} = \alpha'_{j,\nu}$ for $j = \mu'-\nu, \cdots, \mu'$ and $\mathbf{A}_{i,j} = 0$ otherwise. If n knots exist in the original knot vector τ with value t_*, then $\nu = \max\{k-1-n, 1\}$ – that is to say, we always insert at least 1 knot. The quantity ν is used in the triangular scheme above to allow one to skip those basis functions which are trivially 0 or 1 due to repeated knots. As a result of this triangular scheme, we generate basis functions in place and avoid redundant computation of α' values for subsequent levels.

The procedure of knot insertion we propose is analogous to Bézier subdivision. In Figure 3.6, a Bézier curve has been subdivided at $t = .5$, generating a refined polygon $\{\mathbf{p_i}\}$ from the original polygon $\{\mathbf{P_i}\}$. Recall that a Bézier curve is simply a B-spline curve with open end conditions, in this case, with knot vector $\tau = \{0,0,0,0,1,1,1,1\}$. The refined knot vector is then $\mathbf{t} = \{0,0,0,0,.5,.5,.5,1,1,1,1\}$. According to our definitions, $\mu = 6$, $\mu' = 3$. Thus, the point on the surface should be indexed $\mu - k + 1 = 6 - 4 + 1 = 3$, which agrees with the figure. We observe that $\mathbf{p_3}$ is a convex blend of $\mathbf{p_2}$ and $\mathbf{p_4}$.

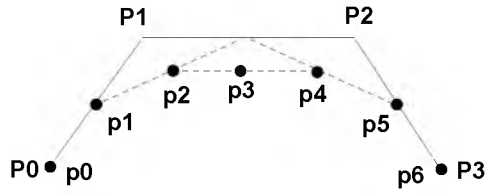

Figure 3.6: Original mesh and refined mesh which results from Bézier subdivision.

Likewise, in the refinement scheme we propose, the point on the curve $\mathbf{D}^{\omega}_{\mu-\mathbf{k}+\mathbf{1}}$ will be a convex blend of the points $\mathbf{D}^{\omega}_{\mu-\mathbf{k}}$ and $\mathbf{D}^{\omega}_{\mu-\mathbf{k}+\mathbf{2}}$. The blend factor will be $\gamma_{\mu',0}$. The dependency graph shown in Figure 3.7 will help to clarify. The factor $\gamma_{\mu',0}$ is introduced at the first level of the recurrence. The leaf terms can be written as

$$\alpha'_{j,\nu} = (1 - \gamma_{\mu',0})l_{j,\nu} + \gamma_{\mu',0}r_{j,\nu}$$

with $j = \mu' - \nu, \cdots, \mu'$. $\{l_{j,\nu}\}$ and $\{r_{j,\nu}\}$ are those terms dependent on $\alpha'_{\mu-1,1}$ and $\alpha'_{\mu,1}$, respectively. They are the elements of the alpha matrix rows $\mu-k$ and $\mu-k+2$ with $\mathbf{A}_{\mu-k,j} = l_{j,\nu}$ and $\mathbf{A}_{\mu-k+2,j} = r_{j,\nu}$ for $j = \mu' - \nu, \cdots, \mu'$. We can generate the $\{l_{j,\nu}\}$ by setting $\alpha'_{\mu'-1,1} = 1$ and $\alpha'_{\mu',1} = 0$ and likewise, generate $\{r_{j,\nu}\}$ by setting $\alpha'_{\mu'-1,1} = 0$ and $\alpha'_{\mu',1} = 1$. Thus, $\mathbf{A}_{\mu-k,j}$ and $\mathbf{A}_{\mu\ k+2,j}$ can be generated in the course of generating $\mathbf{A}_{\mu\ k+1,j}$ at little additional expense.

The procedure above generalizes easily to surfaces, allowing us to generate the desired rows of the refinement matrices $\mathbf{A_u}$ and $\mathbf{A_v}$. The refined mesh $\mathbf{D}^{\omega}$ is derived from the existing mesh $\mathbf{P^w}$ by:

$$\mathbf{D}^{\omega} = \mathbf{A_v}\mathbf{P^w}\mathbf{A_u^T}.$$

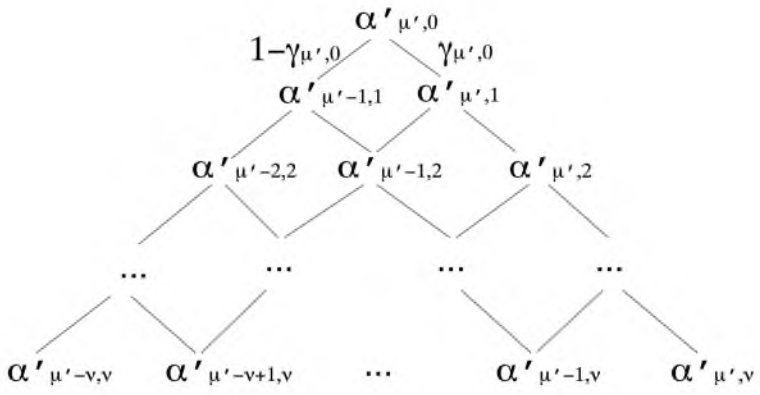

Figure 3.7: Graph showing how the factor $\gamma_{\mu',0}$ propagates through the recurrence.

To produce the desired points, we only need to evaluate

$$\begin{pmatrix} \mathbf{D}^{\omega}_{\mu_\mathbf{v}-\mathbf{k_v}+1,\mu_\mathbf{u}-\mathbf{k_u}+1} & \mathbf{D}^{\omega}_{\mu_\mathbf{v}-\mathbf{k_v}+1,\mu_\mathbf{u}-\mathbf{k_u}+2} \\ \mathbf{D}^{\omega}_{\mu_\mathbf{v}-\mathbf{k_v}+2,\mu_\mathbf{u}-\mathbf{k_u}+1} & \mathbf{D}^{\omega}_{\mu_\mathbf{v}-\mathbf{k_v}+2,\mu_\mathbf{u}-\mathbf{k_u}+2} \end{pmatrix} =$$
$$\begin{pmatrix} (\mathbf{A_v})_{\mu_v+k_v+1,[\mu'_v-\nu_v\ldots\mu'_v]} \\ (\mathbf{A_v})_{\mu_v+k_v+2,[\mu'_v-\nu_v\ldots\mu'_v]} \end{pmatrix} \mathbf{P}^{\mathbf{w}}_{[\mu'_\mathbf{v}-\nu_\mathbf{v}\ldots\mu'_\mathbf{v}][\mu'_\mathbf{u}-\nu_\mathbf{u}\ldots\mu'_\mathbf{u}]} \begin{pmatrix} (\mathbf{A_u})_{\mu_u+k_u+1,[\mu'_u-\nu_u\ldots\mu'_u]} \\ (\mathbf{A_u})_{\mu_u+k_u+2,[\mu'_u-\nu_u\ldots\mu'_u]} \end{pmatrix}^T$$

This can be made quite efficient. We have been able to calculate approximately 150K surface evaluations (with derivative) per second on a 300MHz MIPS R12K using this approach.

3.4 Trimming Curves

Trimming curves are a common method for overcoming the topologically rectangular limitations of NURBS surfaces. They result typically when designers wish to remove sections from models which are not aligned with the underlying parameterization. In this section, we will define what we mean by trimming curves.

A trimming curve is a closed, oriented curve which lies on a NURBS surface. For our purposes, the curve will consist of piecewise linear segments in parametric space $\{\mathbf{p_i} = (u_i, v_i)\}$. (In principle, there is no reason one could not extend our hierarchical technique to higher order trimming curves. [24] deals with Bézier trims.) Often other data are available, such as the real-world coordinates over the various curve vertices, but we will not make use of this information. It is important to note that these curves are not necessarily convex.

We calculate the orientation of the curve using the method of Rokne [120] for computing the area of a polygon. Given parametric points $\{\mathbf{p_i} = (u_i, v_i)\},\ i = 0 \ldots n$, the signed area can be computed by

$$A = \frac{1}{2}\sum_{i=0}^{n} u_i v_{(i+1) \bmod n} - u_{(i+1) \bmod n} v_i.$$

If A is negative, the curve has a clockwise orientation. Otherwise, the orientation is counter-clockwise.

The orientation of a trimming curve determines which region of the surface is to be kept. We use the convention that the part of the surface to be kept is on the right side of the curve [as you walk in the direction of its orientation]. Inconsistencies in orientation that would result in an ambiguous determination of whether to trim are not allowed (see Figure 3.8).

An important characteristic of the trimming curves we use is that they are not allowed to cross.

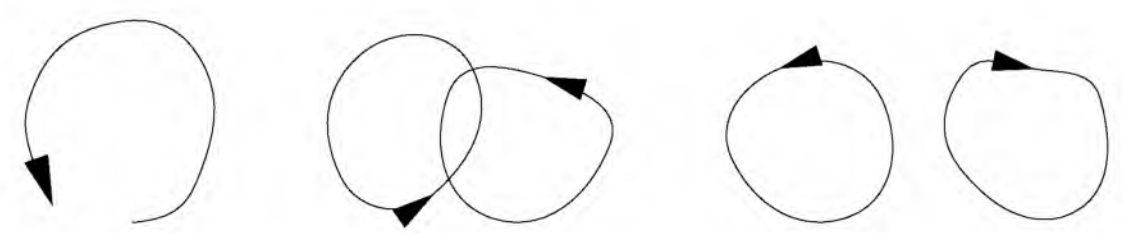

Figure 3.8: Invalid trimming curves: a curve which is not closed, curves which cross, and curves with conflicting orientation.

Trimming curves can contain trimming curves, and can share vertices and edges. Areas inscribed by counter-clockwise curves are often termed "holes," while those inscribed by clockwise curves are termed "regions."

3.4.1 Building a Hierarchy

Given a set of trimming curves on a particular B-spline surface, we can build a hierarchy based on containment (see Figure 3.9). Since the curves are not allowed to cross, there are only three possible relationships between two curves $\mathbf{c_1}$ and $\mathbf{c_2}$. $\mathbf{c_1}$ can contain $\mathbf{c_2}$, be contained in $\mathbf{c_2}$, or share no regions in common with $\mathbf{c_2}$. Each node in our hierarchy is a list of trims, and each trim can refer to yet another list of trims which fall inside of it. The procedure for building the hierarchy is given in Figure 3.10.

The *contains* function for trims needs a bit of clarification. Since trims can share edges and vertices, proper containment tests – those that test only the vertices – will not always work. Instead, we perform inside/outside tests on the midpoints of each trim segment. In comparing $\mathbf{c_1}$ and $\mathbf{c_2}$, $\mathbf{c_1}$ is judged to be contained in $\mathbf{c_2}$ if and only if the midpoint of some segment of $\mathbf{c_1}$

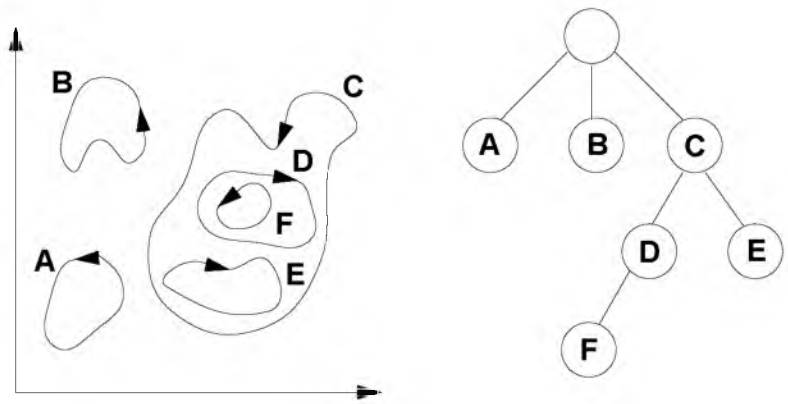

Figure 3.9: A set of trimming curves and the resulting hierarchy.

```
Insert (Trim newtrim, TrimList tl)
  for each Trim t in TrimList tl do
    if t contains newtrim then
      Insert (newtrim, t.trimlist)
      return
    else
      if newtrim contains t then
        Insert(t,newtrim.trimlist)
        Remove(t,tl)
      end if
    end if
  end for
  tl.Add(newtrim)
```

Figure 3.10: Algorithm for adding a trimming curve to the trimming hierarchy.

falls inside $\mathbf{c_2}$. Since curves cannot cross, any such midpoint will do. The inside/outside test is performed with regard to some ϵ so as to counteract round-off error.

For each trim curve, and for each trim list, we store a bounding box which we will use to speed culling in the following ray tracing step. Once the trim hierarchy is created, we perform a quick pass through the surface patches, removing those patches which are completely trimmed away. This is an optimization step which reduces the size of the BVH and the number of patches which must be examined by the intersection routines. The procedure given in Figure 3.11 can be used by encoding the parametric boundary of the patch to be tested as the trimming curve crv.

```
IsTrimmed(TrimList tl, Trim crv)
  for each Trim t in TrimList tl do
    if t contains crv then
      return IsTrimmed(t.tl, crv)
    else
      if t crosses crv then
        return false
      end if
    end if
  end for
  return !tl.is_clockwise
```

Figure 3.11: Algorithm to determine whether a closed curve in paramteric space is trimmed away by the trimming hierarchy.

3.4.2 Ray Tracing Trimmed NURBS

We ray trace trimmed NURBS by first performing ray intersection with the untrimmed surface. If an intersection point (u_*, v_*) is found, we then look to the trim hierarchy to determine whether it is to be culled or returned as a hit. Please see Figure 3.12 for details. Because ambiguous orientations are not allowed, trims at the same level of the hierarchy will have the same orientation. This orientation is referenced above as *tl.is_clockwise*. The variable *keep* determines whether the point should be culled.

3.5 Results

We have generated some images (see Figures 3.13, 3.14, 3.15, 3.16, 3.17, and 3.18) and timings (refer to Table 3.1) for datasets rendered using our technique.

The results in this section were published in 2000 [121] on hardware that was state-of-the-art at that time. All timings are for a single 300MHz R12K MIPS processor with an image resolution of 512x512. All models were Phong shaded and timings include shadow rays.

We have implemented our method in a parallel ray tracing system, and have obtained interactive rates with scenes of moderate geometric complexity. For a discussion of that system, we refer the reader to Parker *et al.* [1]. See also Figure 3.14 for the images included in that paper.

Source code and other material related to the system which we have described can be found online at http://www.acm.org/jgt/papers/MartinEtAl00.

```
Inside(Point p, TrimList tl, boolean& keep)
  keep = !tl.is_clockwise
  if tl.boundingbox contains p then
    for each Trim t in TrimList tl do
      if t.boundingbox contains p then
        if t contains p then
          Inside(p,t.tl,keep)
          return
        end if
      end if
    end for
  end if
```

Figure 3.12: Algorithm for determining whether a ray-surface intersection should be trimmed (reported as a miss).

Figure 3.13: A scene containing NURBS primitives. All of the objects on the table are spline models which have been ray traced using the method presented in this chapter.

Figure 3.14: Ray traced Bézier surfaces from the interactive ray tracing paper by Parker *et al.* [1]. Reprinted with permission. © Copyright 1999 by ACM, Inc. Definitive version: `http://doi.acm.org/10.1145/300523.300537`.

Figure 3.15: Teapot scene from different viewpoints.

Figure 3.16: Rendering of a NURBS scene featuring left) a metallic goblet, center) a bump-mapped glass, and right) a scratched metallic tabletop.

Figure 3.17: Mechanical parts produced by the Alpha_1 [2] modeling system (crank, Crank1A, and allblade).

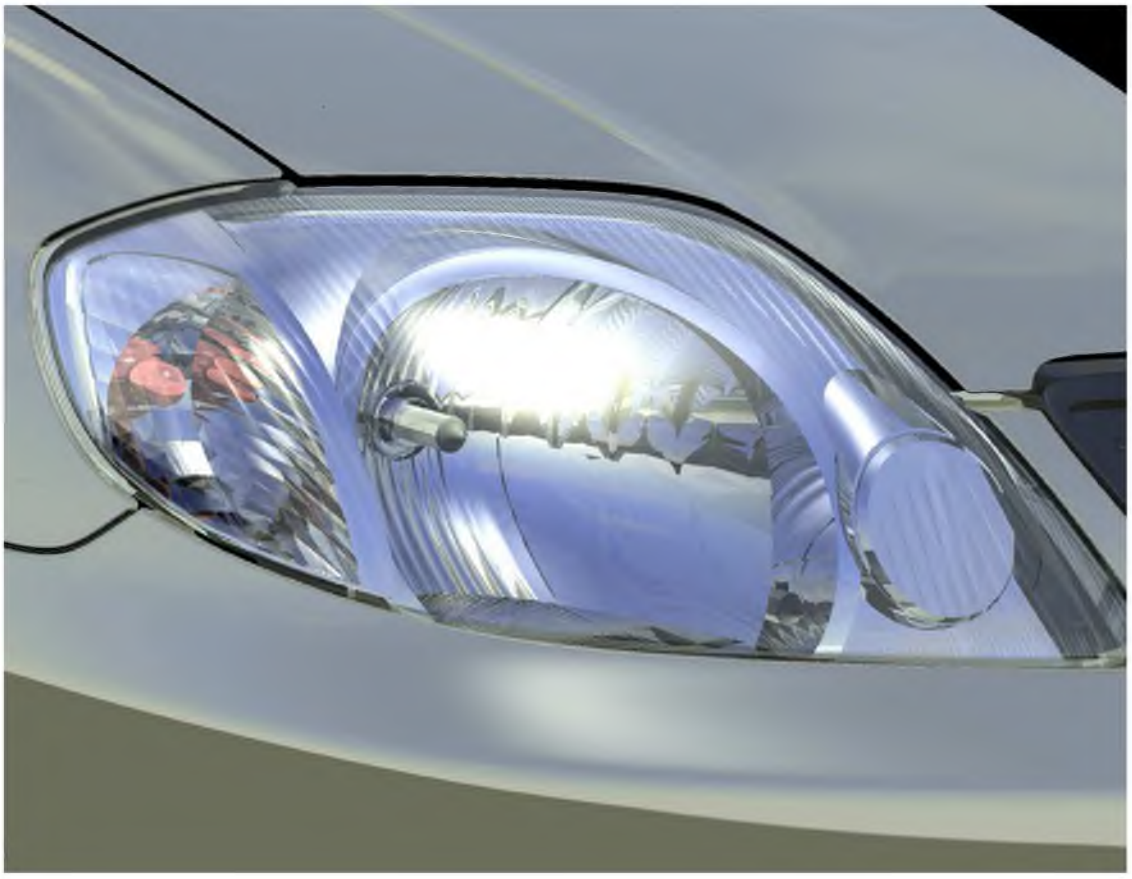

Figure 3.18: A commercial headlight rendered using our system.

Table 3.1: Statistics for our technique. "Light BV intersections" are generated by casting shadow rays and are treated (and measured) separately from ordinary BV intersections. "NURBS tests" give the number of numerical NURBS surface intersections performed. "Total NURBS time"and "Avg time per NURBS" give the total and mean time spent on numerical surface intersections, respectively. "NURBS hits" denote the number of numerical intersections which yielded a hit. "Reported hits" give the number of successful numerical hits which were not eliminated by trimming curves or by comparison with the previous closest hit along the ray.

Statistics	teapot	teapot-solid	spoon	pencil
Number of Surfaces	32	10	3	17
Number of Trims	0	5	0	9
Number of Rays	262144	262144	262144	262144
Total Time (sec.)	14	17	4.8	10
BV Intersections	12642506	10861260	3265298	3439462
Light BV Intersections	6542330	5098600	902120	1714142
NURBS tests	431620	642511	155118	475934
Total NURBS time (sec.)	9.35	12.17	3.42	8.28
Avg Time per NURBS (sec.)	2.17E-5	1.89E-5	2.20E-5	1.74E-5
NURBS hits	367307	458061	79387	226927
(% of tot tests)	(85.1%)	(71.3%)	(51.2%)	(47.7%)
Reported hits	119597	196051	21753	36577
(% of tot tests)	(27.7%)	(30.5%)	(14.0%)	(7.7%)

Statistics	goblet	Crank1A	crank	allblade
Number of Surfaces	1	20	73	351
Number of Trims	0	18	64	0
Number of Rays	262144	262144	262144	262144
Total Time (sec.)	7.8	78	40	61
BV Intersections	3756604	23818532	14716416	47701788
Light BV Intersections	1638038	6669190	3334258	17060564
NURBS tests	320622	2287689	1306480	2340071
Total NURBS time (sec.)	5.97	39.39	20.82	43.46
Avg Time per NURBS (sec.)	1.86E-5	1.72E-5	1.59E-5	1.86E-5
NURBS hits	226753	1488391	542912	1319143
(% of tot tests)	(70.7%)	(65.1%)	(41.6%)	(56.4%)
Reported hits	100001	209344	103525	445496
(% of tot tests)	(31.2%)	(9.2%)	(7.9%)	(19.0%)

CHAPTER 4

REPRESENTATION OF VOLUMETRIC DATA USING TRIVARIATE SPLINES

4.1 Overview

Our goal in this chapter is to leverage traditional strengths from the geometric design and scientific visualization communities to produce a tool valuable to both. We present a method for representing and specifying attribute data across a trivariate NURBS volume. Some relevant attribute quantities include material composition and density, optical indices of refraction and dispersion, and data from medical imaging. The method is independent of the granularity of the physical geometry, allowing for a decoupling of the resolution of the carried data from that of the volume. Volume attributes can be modeled or fit to data.

A method is presented for efficient evaluation of trivariate NURBS. We incorporate methods for data analysis and visualization, including isosurface extraction, planar slicing, volume ray tracing, and optical path tracing, all of which are grounded in refinement theory for splines. The applications for these techniques are diverse, including such fields as optics, fluid dynamics, and medical visualization.

4.2 Introduction

Volumes have long been important in the fields of scientific and medical visualization. MRI and CAT scanning devices produce a 3-dimensional photograph of the internal state of a subject. In the field of fluid dynamics, pressure and velocity are spatially varying quantities whose values are critical to analysis. Likewise, turbidity and pollutant density play a large role in the simulation of atmospheric optics. The ability to deal with volumetric data is clearly a requirement.

True volumetric primitives are encountered less frequently in the field of computer-aided geometric design. Traditionally, boundary representations have been utilized extensively. This certainly makes sense for modeling solids having uniform interior. However, recent advances have led to manufacturing technologies supporting heterogeneous materials. For example, there are now machines with the capability to combine different source materials by percentage. In the

area of optical design, lenses with continuously varying index of refraction are coming available — so-called GRIN (Gradient Index) lenses. With these advances has come the need to model the interiors as well as the boundaries of objects. Finally, among the engineering community, there is often the desire to perform analyses on the products of design. These tests, such as temperature and stress simulation, generally involve propagation of attributes across an object's interior.

Traditionally, volumetric primitives have been grid-based. This has served well among the scientific community, where the data are often regularly spaced along grid lines. This preference may change as adaptive 3D scanning technology becomes more common. Among the geometric design community, NURBS have been the *de facto* primitive of choice. In this article, we advocate that a trivariate NURBS model may well serve the needs of both communities.

There are many advantages to the model we propose. First, it decouples geometric representation from attribute representation. This means that complicated geometries with simple attributes, and vice versa, may be represented at the resolution that best suits them. The result may be a large savings in storage and execution time. Furthermore, noise is an important variable in any visualization involving measured data. NURBS generally provide a robust representation for a signal containing moderate noise. Splines are a terse representation. By this, we mean that compared with polygons, or higher dimensional analogues such as voxels, splines generally represent a smooth function with fewer points.

For scientific and medical applications, either shape approximating or interpolating splines may be used with the attribute data, depending on whether a qualitative or more quantitative approach is required. From the computer-aided geometric design (CAGD) perspective, an extended NURBS representation means that all of the existing algorithms can be applied in the new problem domain. We can consider modeling both the geometry and the attributes carried by the volume. Methods for visualization for design analysis can be borrowed from the visualization community.

We begin with a review of the existing literature and introduce our representation in Section 4.3. Section 4.4 deals with techniques for fitting data and modeling shape. Section 4.5 introduces an efficient method for evaluating trivariates, which is critical for large datasets. In Section 4.6, we adapt visualization techniques to our aggregate spline representation. We conclude and give future work in Section 4.7.

4.3 Background

Trivariate NURBS representations have been considered by a number of researchers. Early, Farouki and Hinds [28] gave a unified approach to curves, surfaces, and volumes. Lasser [29] explored the Bernstein-Bézier volume representation, and extended techniques for evaluation and interpolation to them. In her thesis, Paik [30] explored trivariates for modeling, and in particular, modeling operators, deformations, and animations. Kaufman [38] has combined modeled objects and measured data within a volume visualization system, along with an algorithm for scan converting tricubic Béziers. Chang *et al.* [56] provided a method for rendering volumes using line of sight integration and compositing, and an attribute volume representation similar to the one described here. A sculpting method introduced by Raviv and Elber [40] allows the user to sculpt a three-dimensional object by modifying the scalar coefficients of a trivariate NURBS equation. Lee and Park [42] introduce an attribute model whose coefficients are generated from fluid flow data. Finally, Joy and Duchaineau [32] generate a complete representation for the boundary of a trivariate by unioning the faces with an implicit equation based on the Jacobian.

4.3.1 Trivariate Volumes

A nonuniform rational B-spline (NURBS) volume is a mapping $\mathbf{P}^w : \mathbb{R}^3 \rightarrow \mathbb{P}^3$ that can be formulated as

$$\mathbf{P}^w(u_1, u_2, u_3) = \sum_{i_1=0}^{N_1} \sum_{i_2=0}^{N_2} \sum_{i_3=0}^{N_3} \mathbf{P}^w_{i_1,i_2,i_3} B_{i_1,k_1}(u_1) B_{i_2,k_2}(u_2) B_{i_3,k_3}(u_3)$$

where the superscript w denotes that our formulation produces a point in rational four space. The omission of the superscript, as in $\mathbf{P}$, will denote the function which results from dividing $\mathbf{P}^w$ by its rational coordinate, a mapping $\mathbb{R}^3 \rightarrow \mathbb{R}^3$. The $\{\mathbf{P}^w_{a,b,c}\}$ are the control points $w_{a,b,c}(x_{a,b,c}, y_{a,b,c}, z_{a,b,c}, 1)$ of the $(N_1+1) \times (N_2+1) \times (N_3+1)$ mesh, having basis functions B_{i_j,k_j} of orders k_j with knot vectors $\tau^{\mathbf{j}} = \{\mathbf{u}^j_{i_j}\}_{i_j=0}^{N_j+k_j}$ for $j = 1, 2, 3$. For notational convenience, we may occasionally assign names to the variables such as u, v, w, where $(u, v, w) = (u_1, u_2, u_3)$.

Such a volume $\mathbf{P}^w$ is defined over the domain $\prod_{j=1}^{3} [\mathbf{u}^{\mathbf{j}}_{k_j-1}, \mathbf{u}^{\mathbf{j}}_{N_j+1})$, where we use $\prod$ to denote the Cartesian product. Each nonempty subinterval $\prod_{j=1}^{3} [\mathbf{u}^{\mathbf{j}}_{i_j}, \mathbf{u}^{\mathbf{j}}_{i_j+1})$ corresponds to a volume fragment.

The partial derivatives of $\mathbf{P}^w$ are likewise NURBS. Their control points are simply scaled differences of adjacent points in the original control mesh. For example, the control points for

$\frac{\partial \mathbf{P}^w}{\partial u_1}$ are given by

$$(D_{u_1}\mathbf{P}^w)_{i_1,i_2,i_3} = \frac{k_1 - 1}{u^1_{i_1+k_1} - u^1_{i_1+1}}(\mathbf{P}^w_{i_1+1,i_2,i_3} - \mathbf{P}^w_{i_1,i_2,i_3}).$$

The corresponding derivative volume is then given by

$$\frac{\partial \mathbf{P}^w}{\partial u_1}(u_1,u_2,u_3) = \sum_{i_1=0}^{N_1-1}\sum_{i_2=0}^{N_2}\sum_{i_3=0}^{N_3}[(D_{u_1}\mathbf{P}^w)_{i_1,i_2,i_3}B_{i_1+1,k_1-1}(u_1)B_{i_2,k_2}(u_2)B_{i_3,k_3}(u_3)] \tag{4.1}$$

4.3.2 Aggregate Data Types

We now provide a representation which incorporates an arbitrary number of volume attributes. Each volume consists of a geometric description $\mathbf{P}^w$ and a set of attribute descriptions $\{\mathbf{A}^w_p\}$. Each attribute is represented as an independent trivariate volume

$$\mathbf{A}^w_p(u_1,u_2,u_3) = \sum_{i_1=0}^{N_{p1}}\sum_{i_2=0}^{N_{p2}}\sum_{i_3=0}^{N_{p3}}\mathbf{A}^w_{p\,i_1,i_2,i_3}B_{i_1,k_{p1}}(u_1)B_{i_2,k_{p2}}(u_2)B_{i_3,k_{p3}}(u_3)$$

All aspects of the representation, *e.g.*, order, dimensions of the control mesh, and knot vectors, are independent of the geometric trivariate representation. The only requirement is that all the volumes share the same parametric domain. In our implementation, we normalize all the domains to the unit cube I^3.

Two advantages of this representation merit mention. First, by decoupling the representation of the geometry from the attributes, and likewise, the attributes from one another, each function may be specified only to its required resolution. As volume data can be large, the so-called "curse of dimensionality," this may result in substantial savings. Furthermore, evaluation times will benefit for functions which are represented at differing orders (degrees). The second advantage of this scheme is that a trivariate NURBS package can be easily extended to include this representation. It does not increase the dimensionality of the points nor does it slow down existing routines.

4.4 Modeling and Data Fitting

As is a common paradigm in modeling systems, trivariate splines can easily be built up from lower dimensional primitives. Consider the example in Figure 4.1. Here, a line segment

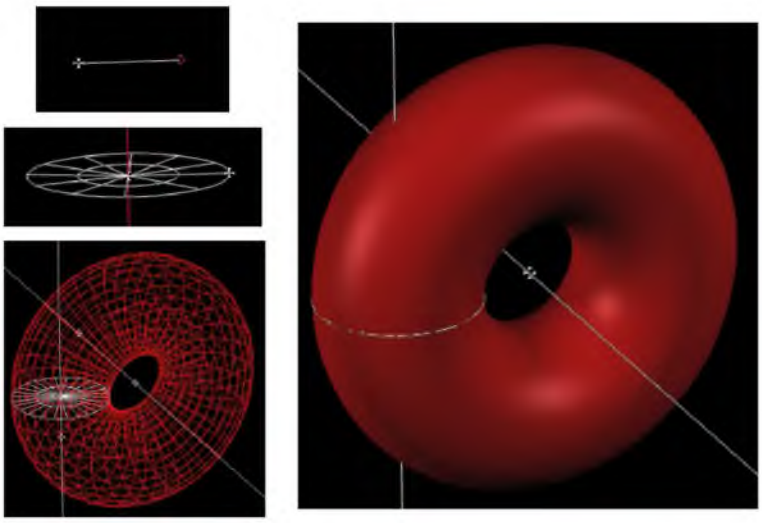

Figure 4.1: Building a trivariate using simple modeling operators. We implemented these modeling operators in Alpha_1 [2] as a straightforward extension of the surface-based operators. This is possible because of the tensor-product nature of the primitives.

is constructed from two points. Next, the segment is rotated about an axis to form a disk. Finally, this disk is then rotated about another axis to sweep out a torus. Each step entails an increase in dimension, from curve, to surface, to volume, and each has a well-defined curve, surface, and volume NURBS representation. The modeling system constructs a dependency graph which encapsulates how complex models are built up from these primitives. Subsequent changes to the underlying pieces – for example, altering the shape of our segment in Figure 4.1 – will immediately propagate to downstream primitives that depend upon them. Consider also Figure 4.2 that constructs a trivariate solid through extrusion.

Because the attributes of our system are represented as NURBS volumes, they can be modeled using any of the traditional operators. For example, we can model an attribute curve, extrude it into a surface, and rotate the surface about an axis to obtain a volume. While this might be useful, it can lead to some nonintuitive results, as the values of the attributes are contained in the coordinate values of the resulting objects. In the absence of underlying geometry, it is not immediately clear what meaning those operations have on the attribute functions.

For the purpose of modeling, we have developed a hierarchy of aggregate data types. Aggregate curves contain a geometry curve and an arbitrary number of attribute curves. Similarly, there are aggregate surfaces. At any stage of design, an attribute object may be combined with a geometric object to form an aggregate object. Additional attributes may be added to the

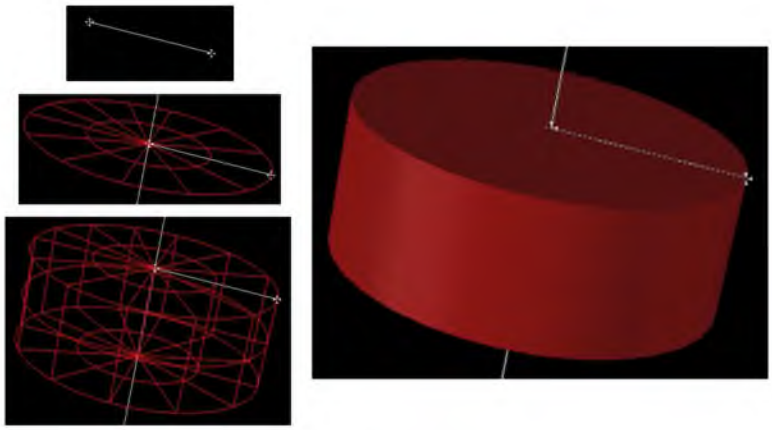

Figure 4.2: A swept cylindrical solid build up from primitives.

aggregate data type at any later stage of design, as well. Once an attribute has been added to an aggregate data type, it becomes carried data. It can still be edited explicitly, but the default target of modification becomes the geometry. If the aggregate data type is mapped one level up the hierarchy, for example, by extruding the geometry curve into a surface, the carried attribute data are also generalized to a higher parametric dimension. By default, a ruled object is created from the attribute objects contained in the original aggregate data types. If the geometric modeling operator only took one aggregate parameter, such as ruling or rotation about an axis, each attribute object is ruled with itself. That is, in the absence of additional data, we assume that the attribute varies only along the original parametric dimensions. If the mapping is applied to several aggregate objects, the ruling is applied across the corresponding attribute objects.

Let us consider a brief modeling example. Consider the two curves in Figure 4.3. The curve on the top left corresponds to geometry, and the curve on the bottom left represents a particular attribute. Here, the y-coordinate contains the relevant attribute data, and it can be seen that the attribute varies with t in a nonlinear way. We form an aggregate curve from these two curves, and then extrude the geometry to form a surface. The attribute will be constant along the extrusion direction. Finally, surface is rotated about an axis to form a volume of revolution. If the axis of revolution is the 'z-axis' and perpendicular to the extrusion direction, then it is clear that the attribute data varies with 'z', and is constant in planes perpendicular to 'z'.

If we wish to edit the attribute curve, we can do so, and because of the dependency graph used by our modeling environment, all changes will propagate through the structure. Furthermore, we can explicitly request to modify the attribute objects at any level of the hierarchy. Thus,

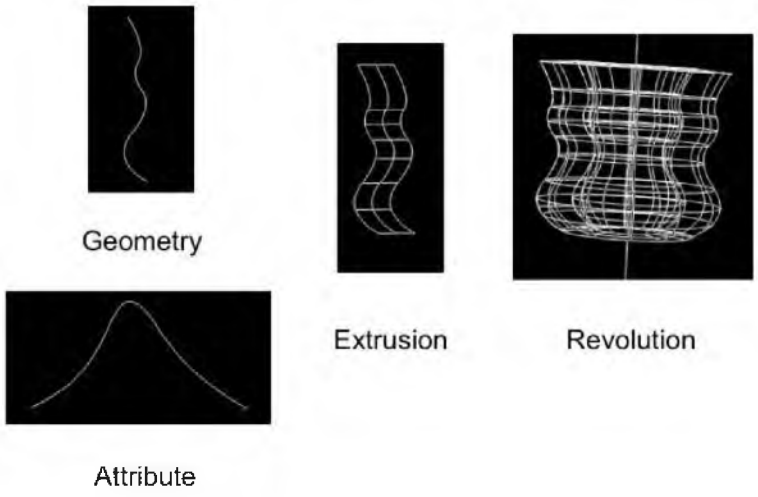

Figure 4.3: Design example using aggregate objects.

if we want to allow variation along other parametric axes, we can do that now, in light of the completed geometric shape. The visualization techniques discussed in Section 4.6 can provide further information to guide these edits.

The visualization community often produces data by measurement or simulation. In such an instance, data fitting becomes a primary concern. If the qualitative shape is the most important thing, then the data points may simply be used as control points in the trivariate representation. In many cases, it is desired that the functional approximation interpolate the data. It is this problem that we now address.

If the data points are $\mathbf{c}_{j_1,j_2,j_3}$, the problem is to find the control points $\mathbf{P}_{i_1,i_2,i_3}$ such that

$$\sum_{i_1,i_2,i_3} \mathbf{P}_{i_1,i_2,i_3} B_{i_1}(\tilde{u}^1_{j_1}) B_{i_2}(\tilde{u}^2_{j_2}) B_{i_3}(\tilde{u}^3_{j_3}) = \mathbf{c}_{j_1,j_2,j_3}$$

where $\tilde{u}^1_{j_1}, \tilde{u}^2_{j_2}, \tilde{u}^3_{j_3}$ are particular parameter values that correspond to the region of maximum influence for $\mathbf{P}_{j_1,j_2,j_3}$, called *nodal values* or Greville abscissas. The nodal values in u_1 for knot vector $\tau^1 = \{\mathbf{u}^1_j\}$ are given by

$$\tilde{u}^1_{i_1} = \frac{1}{k_1 - 1} \sum_{j=1}^{k_1 - 1} \mathbf{u}^1_{i_1+j}$$

There are analogous formulations for $\tilde{u}^2_{i_2}$ and $\tilde{u}^3_{i_3}$.

In similar fashion to [42], we make the following definitions. Let

$$\mathbf{P}_{i_2,i_3}(\tilde{u}^1_{j_1}) \equiv \sum_{i_1=0}^{N_1} \mathbf{P}_{i_1,i_2,i_3} B_{i_1,k_1}(\tilde{u}^1_{j_1}) \tag{4.2}$$

$$\mathbf{P}_{i_3}(\tilde{u}^1_{j_1}, \tilde{u}^2_{j_2}) \equiv \sum_{i_2=0}^{N_2} \mathbf{P}_{i_2,i_3}(\tilde{u}^1_{j_1}) B_{i_2,k_2}(\tilde{u}^2_{j_2}) \tag{4.3}$$

Thus, we have

$$\sum_{i_3=0}^{N_3} \mathbf{P}_{i_3}(\tilde{u}^1_{j_1}, \tilde{u}^2_{j_2}) B_{i_3,k_3}(\tilde{u}^3_{j_3}) = \mathbf{c}_{j_1,j_2,j_3}$$

For each fixed j_1, j_2 pair, we can solve N_3+1 equations for N_3+1 unknowns to get $\{\mathbf{P}_{i_3}(\tilde{u}^1_{j_1}, \tilde{u}^2_{j_2})\}$. Likewise, for each fixed i_3, j_1 pair, we can solve Equation (4.3) for $\{\mathbf{P}_{i_2,i_3}(\tilde{u}^1_{j_1})\}$. Finally, for each i_2, i_3, we can solve Equation (4.2) for $\{\mathbf{P}_{i_1,i_2,i_3}\}$, thereby solving the interpolation problem.

Modeling and data fitting can certainly be combined. For example, suppose the object in Figure 4.2 represents a lens, with radially varying index of refraction. The attribute curve could have been generated by the data fitting technique discussed above. By importing the curve into our modeling program, it can now be used as a primitive for lens design.

4.5 Evaluation

Efficient evaluation schemes are critical to the success of any data representation — ever more so as the size of the dataset increases. In this section, we present a rapid evaluation scheme based in refinement.

For simplicity, we briefly restate some of the results of Section 3.3.4 and Section 3.3.5. We evaluate a curve $\mathbf{c}(t)$ with order k and knot vector τ by using refinement to stack $k-1$ knots (recall that k is one more than the degree of the curve) at the desired parameter value t_*. The refined curve is defined over a new knot vector $\mathbf{t} = \{\mathbf{t_i}\}$ with basis functions $N_{i,k}(t)$ and new control points $\omega_i \mathbf{D}_i$.

Let $t_* \in [t_\mu, t_{\mu+1})$. Then,

$$\begin{aligned} \mathbf{c}(t_*) &= \mathbf{D}_{\mu-\mathbf{k}+\mathbf{1}} \\ \mathbf{c}'(t_*) &= \frac{(k-1)\omega_{\mu-k+2}}{(t_{\mu+1}-t_*)\omega_{\mu-k+1}}[\mathbf{D}_{\mu-\mathbf{k}+\mathbf{2}} - \mathbf{D}_{\mu-\mathbf{k}+\mathbf{1}}] \end{aligned} \tag{4.4}$$

(see [121]).

Analogously, in order to evaluate a trivariate volume $\mathbf{P}$ having knot vectors τ^u, τ^v, τ^w and orders k_u, k_v, k_w, we stack $k_u - 1$, $k_v - 1$, and $k_w - 1$ knots valued u_*, v_*, and w_*. If, with regard to the new knot vectors, $u_* \in [u_{\mu_u}, u_{\mu_u+1})$, $v_* \in [v_{\mu_v}, v_{\mu_v+1})$, and $w_* \in [w_{\mu_w}, w_{\mu_w+1})$, then

$$\mathbf{P}(u_*, v_*, w_*) = \mathbf{D}_{\mu_\mathbf{u}-\mathbf{k_u}+\mathbf{1},\mu_\mathbf{v}-\mathbf{k_v}+\mathbf{1},\mu_\mathbf{w}-\mathbf{k_w}+\mathbf{1}} \tag{4.5}$$

and

$$\begin{aligned} &\mathbf{P_u}(u_*, v_*, w_*) = \\ &\frac{(k_u-1)\omega_{\mu_u-k_u+2,\mu_v-k_v+1,\mu_w-k_w+1}}{(u_{\mu_u+1}-u_*)\omega_{\mu_u-k_u+1,\mu_v-k_v+1,\mu_w-k_w+1}} \\ &[\mathbf{D}_{\mu_\mathbf{u}-\mathbf{k_u}+\mathbf{2},\mu_\mathbf{v}-\mathbf{k_v}+\mathbf{1},\mu_\mathbf{w}-\mathbf{k_w}+\mathbf{1}} - \\ &\mathbf{D}_{\mu_\mathbf{u}-\mathbf{k_u}+\mathbf{1},\mu_\mathbf{v}-\mathbf{k_v}+\mathbf{1},\mu_\mathbf{w}-\mathbf{k_w}+\mathbf{1}}] \end{aligned} \tag{4.6}$$

$$\begin{aligned} &\mathbf{P_v}(u_*, v_*, w_*) = \\ &\frac{(k_v-1)\omega_{\mu_u-k_u+1,\mu_v-k_v+2,\mu_w-k_w+1}}{(v_{\mu_v+1}-v_*)\omega_{\mu_u-k_u+1,\mu_v-k_v+1,\mu_w-k_w+1}} \\ &[\mathbf{D}_{\mu_\mathbf{u}-\mathbf{k_u}+\mathbf{1},\mu_\mathbf{v}-\mathbf{k_v}+\mathbf{2},\mu_\mathbf{w}-\mathbf{k_w}+\mathbf{1}} - \\ &\mathbf{D}_{\mu_\mathbf{u}-\mathbf{k_u}+\mathbf{1},\mu_\mathbf{v}-\mathbf{k_v}+\mathbf{1},\mu_\mathbf{w}-\mathbf{k_w}+\mathbf{1}}] \end{aligned}$$

$$\mathbf{P_w}(u_*, v_*, w_*) = \frac{(k_w - 1)\omega_{\mu_u - k_u + 1, \mu_v - k_v + 1, \mu_w - k_w + 2}}{(w_{\mu_w + 1} - w_*)\omega_{\mu_u - k_u + 1, \mu_v - k_v + 1, \mu_w - k_w + 1}} [\mathbf{D}_{\mu_u - \mathbf{k_u} + \mathbf{1}, \mu_v - \mathbf{k_v} + \mathbf{1}, \mu_w - \mathbf{k_w} + \mathbf{2}} - \mathbf{D}_{\mu_u - \mathbf{k_u} + \mathbf{1}, \mu_v - \mathbf{k_v} + \mathbf{1}, \mu_w - \mathbf{k_w} + \mathbf{1}}]$$

In order to perform these evaluations, we must first generate the refinement matrices which map the old control points into the new ones. For curves, the refinement problem can be written as $\mathbf{D} = A_u \mathbf{P}$, and for surfaces $\mathbf{D} = A_u \mathbf{P} A_v^T$. However, as the parametric dimensionality increases, this type of notation no longer suffices. We introduce a new notation which generalizes to any number of parametric dimensions.

We define the operator $\otimes^k$ such that

$$\mathbf{D} = A \otimes^k \mathbf{C} \equiv (\mathbf{D})_{c_0, c_1, \ldots, c_{k-1}, i, c_{k+1}, \ldots c_M} = \sum_j (A)_{i,j} (\mathbf{C})_{c_0, c_1, \ldots, c_{k-1}, j, c_{k+1}, \ldots c_M}$$

For curves, then

$$\mathbf{D^0} = A_u \otimes^0 \mathbf{C} \equiv (\mathbf{D^0})_i = \sum_j (A_u)_{i,j} (\mathbf{C})_j = A_u \mathbf{C}$$

Likewise, for surfaces,

$$\begin{aligned} \mathbf{D}^{1,0} &= A_v \otimes^1 (A_u \otimes^0 \mathbf{C}) \\ &= A_v \otimes^1 \mathbf{D^0} \\ &= (\mathbf{D}^{1,0})_{i,j} \equiv \sum_l (A_v)_{j,l} (\mathbf{D^0})_{i,l} \\ &= \mathbf{D^0} A_v^T \\ &= A_u \mathbf{C} A_v^T \end{aligned}$$

which is what would be expected. The operator generalizes to n dimensions. For trivariates, the

evaluation refinement given at the beginning of this section can then be denoted

$$A_w \otimes^2 (A_v \otimes^1 (A_u \otimes^0 \mathbf{P}^w))$$

Returning to the curve case of Equation (4.4), we are not interested in calculating the full alpha matrix $\mathbf{A}$, but merely rows $\mu - k + 2$ and $\mu - k + 1$, as these are used to generate the points $\mathbf{D}^{\omega}_{\mu-\mathbf{k}+\mathbf{2}}$ and $\mathbf{D}^{\omega}_{\mu-\mathbf{k}+\mathbf{1}}$ which are required for point and derivative evaluation.

Suppose $t_* \in [\tau_{\mu'}, \tau_{\mu'+1})$. We can generate the refinement for row $\mu + k - 1$ using a triangular scheme

$$\begin{array}{ccc} & & \alpha'_{\mu',0} \\ & \alpha'_{\mu'-1,1} & \alpha'_{\mu',1} \\ & \vdots & \vdots \\ \alpha'_{\mu'-\nu,\nu} & \cdots & \alpha'_{\mu',\nu} \end{array}$$

where ν is the number of knots we are inserting and

$$\begin{aligned} \alpha'_{j,1} &= \delta_{j,\mu'} \\ \alpha'_{j,p+1} &= \gamma_{j,p}\alpha'_{j,p} + (1 - \gamma_{j+1,p})\alpha'_{j+1,p} \\ \gamma_{j,p} &= (t_* - \tau_{\mu'-p+j-(k-1-\nu)})/d \end{aligned}$$

$\mathbf{A}_{\mu-k+1,j} = \alpha'_{j,\nu}$ for $j = \mu' - \nu, \cdots, \mu'$ and $\mathbf{A}_{i,j} = 0$ otherwise. If n knots exist in the original knot vector τ with value t_*, then $\nu = \max\{k - 1 - n, 1\}$ — that is to say, we always insert at least 1 knot. The quantity ν is used in the triangular scheme above to allow one to skip those basis functions which are trivially 0 or 1 due to repeated knots. As a result of this triangular scheme, we generate basis functions in place and avoid redundant computation of α' values for subsequent levels.

In the refinement scheme we propose, the point on the curve $\mathbf{D}^{\omega}_{\mu-\mathbf{k}+\mathbf{1}}$ will be a convex blend of the points $\mathbf{D}^{\omega}_{\mu-\mathbf{k}}$ and $\mathbf{D}^{\omega}_{\mu-\mathbf{k}+\mathbf{2}}$. The blend factor will be $\gamma_{\mu',0}$. The dependency graph shown in Figure 4.4 will help to clarify. The factor $\gamma_{\mu',0}$ is introduced at the first level of the recurrence. The leaf terms can be written as

$$\alpha'_{j,\nu} = (1 - \gamma_{\mu',0})l_{j,\nu} + \gamma_{\mu',0}r_{j,\nu}$$

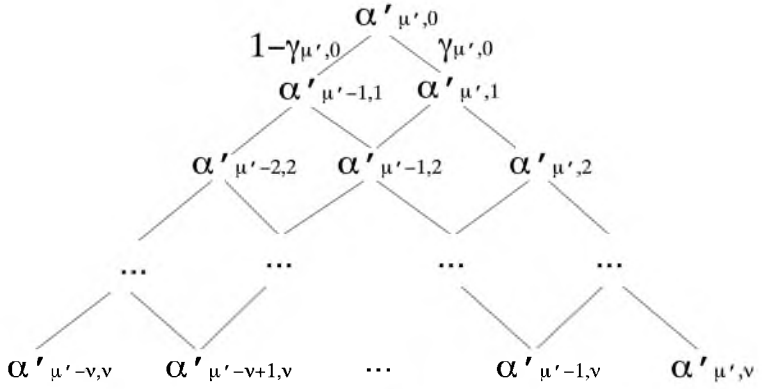

Figure 4.4: Graph showing how the factor $\gamma_{\mu',0}$ propagates through the recurrence.

with $j = \mu' - \nu, \cdots, \mu'$. $\{l_{j,\nu}\}$ and $\{r_{j,\nu}\}$ are those terms dependent on $\alpha'_{\mu-1,1}$ and $\alpha'_{\mu,1}$, respectively. They are the elements of the alpha matrix rows $\mu-k$ and $\mu-k+2$ with $\mathbf{A}_{\mu-k,j} = l_{j,\nu}$ and $\mathbf{A}_{\mu-k+2,j} = r_{j,\nu}$ for $j = \mu' - \nu, \cdots, \mu'$. We can generate the $\{l_{j,\nu}\}$ by setting $\alpha'_{\mu'-1,1} = 1$ and $\alpha'_{\mu',1} = 0$ and likewise, generate $\{r_{j,\nu}\}$ by setting $\alpha'_{\mu'-1,1} = 0$ and $\alpha'_{\mu',1} = 1$. Thus, $\mathbf{A}_{\mu-k,j}$ and $\mathbf{A}_{\mu-k+2,j}$ can be generated in the course of generating $\mathbf{A}_{\mu-k+1,j}$ at little additional expense.

To produce the desired points for Equation (4.5) we only need to evaluate

$$\begin{aligned}
&\mathbf{D}^{\omega}_{\mu_\mathbf{u}-\mathbf{k_u}+\mathbf{1},\mu_\mathbf{v}-\mathbf{k_v}+\mathbf{1},\mu_\mathbf{w}-\mathbf{k_w}+\mathbf{1}} = \\
&\quad (\mathbf{A_u})_{\mu_u+k_u+1,[\mu'_u-\nu_u\ldots\mu'_u]} \otimes^0 \\
&\quad (\mathbf{A_v})_{\mu_v+k_v+1,[\mu'_v-\nu_v\ldots\mu'_v]} \otimes^1 \\
&\quad (\mathbf{A_w})_{\mu_w+k_w+1,[\mu'_w-\nu_w\ldots\mu'_w]} \otimes^2 \\
&\quad \mathbf{P}^{\mathbf{w}}_{[\mu'_\mathbf{u} \quad \nu_\mathbf{u}\ldots\mu'_\mathbf{u}][\mu'_\mathbf{v} \quad \nu_\mathbf{v}\ldots\mu'_\mathbf{v}][\mu'_\mathbf{w} \quad \nu_\mathbf{w}\ldots\mu'_\mathbf{w}]}
\end{aligned}$$

To calculate Equation (4.7), we substitute $(\mathbf{A_u})_{\mu_u+k_u+2,[\mu'_u-\nu_u\ldots\mu'_u]}$ for $(\mathbf{A_u})_{\mu_u+k_u+1,[\mu'_u-\nu_u\ldots\mu'_u]}$ in the above expression to obtain $\mathbf{D}^{\omega}_{\mu_\mathbf{u}-\mathbf{k_u}+\mathbf{2},\mu_\mathbf{v}-\mathbf{k_v}+\mathbf{1},\mu_\mathbf{v}-\mathbf{k_v}+\mathbf{1}}$, and perform similar substitutions to obtain $\mathbf{D}^{\omega}_{\mu_\mathbf{u}-\mathbf{k_u}+\mathbf{1},\mu_\mathbf{v}-\mathbf{k_v}+\mathbf{2},\mu_\mathbf{v}-\mathbf{k_v}+\mathbf{1}}$ and $\mathbf{D}^{\omega}_{\mu_\mathbf{u}-\mathbf{k_u}+\mathbf{1},\mu_\mathbf{v}-\mathbf{k_v}+\mathbf{1},\mu_\mathbf{v}-\mathbf{k_v}+\mathbf{2}}$. This can be made quite efficient.

4.6 Visualization Techniques

4.6.1 Isosurfacing and Slicing

In this section, we provide a unified approach to two common methods of data visualization. An isosurface with respect to a data attribute $\mathbf{A}_p$ is the set of points within the volume having

a particular value for $\mathbf{A}_p$. The display of isosurfaces within a volume gives useful information about the variation of $\mathbf{A}_p$. Another visualization technique is to create a planar slice of the volume, and color code it according to a given attribute (see Figure 4.5).

The isosurfacing problem can be formalized as finding the set of points which obey the equation

$$\begin{aligned}
\mathbf{A}_p(\mathbf{u}) &= \frac{\sum_{i_1,i_2,i_3} w_{p_{i_1,i_2,i_3}} \mathbf{A}_{p_{i_1,i_2,i_3}} \mathbf{B}_{i_1,i_2,i_3}(u_1,u_2,u_3)}{\sum_{i_1,i_2,i_3} w_{p_{i_1,i_2,i_3}} \mathbf{B}_{i_1,i_2,i_3}(u_1,u_2,u_3)} \\
&= \mathbf{A}_* \\
&= \frac{\sum_{i_1,i_2,i_3} w_{p_{i_1,i_2,i_3}} \mathbf{A}_* \mathbf{B}_{i_1,i_2,i_3}(u_1,u_2,u_3)}{\sum_{i_1,i_2,i_3} w_{p_{i_1,i_2,i_3}} \mathbf{B}_{i_1,i_2,i_3}(u_1,u_2,u_3)}
\end{aligned}$$

Thus,

$$\begin{aligned}
0 &= \frac{\sum_{i_1,i_2,i_3} w_{p_{i_1,i_2,i_3}} (\mathbf{A}_{p_{i_1,i_2,i_3}} - \mathbf{A}_*) \mathbf{B}_{i_1,i_2,i_3}(\mathbf{u})}{\sum_{i_1,i_2,i_3} w_{p_{i_1,i_2,i_3}} \mathbf{B}_{i_1,i_2,i_3}(\mathbf{u})} \\
&= \sum_{i_1,i_2,i_3} w_{p_{i_1,i_2,i_3}} (\mathbf{A}_{p_{i_1,i_2,i_3}} - \mathbf{A}_*) \mathbf{B}_{i_1,i_2,i_3}(\mathbf{u})
\end{aligned}$$

If all the weights $w_{p_{i_1,i_2,i_3}}$ are positive, then for a root to be possible on an interval I, the bounding box of the associated control points must contain the origin. In the case of scalar $\mathbf{A}_*$, this means that the difference $(\mathbf{A}_{p_{i_1,i_2,i_3}} - \mathbf{A}_*)$ must change signs.

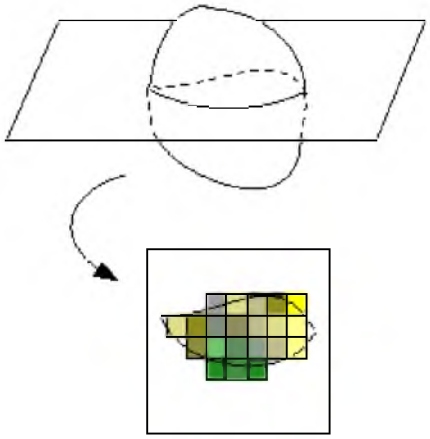

Figure 4.5: Planar cut.

If we are given a plane $\mathbf{p} = \{\mathbf{x} : (a,b,c,d) \cdot (\mathbf{x},1) = 0\}$, then the points in $\mathbf{P}^w$ which are sliced by $\mathbf{p}$ are given by

$$\begin{aligned} 0 &= \frac{\sum_{i_1,i_2,i_3} w_{i_1,i_2,i_3}(\mathbf{P}_{i_1,i_2,i_3} \cdot (a,b,c,d))\mathbf{B}_{i_1,i_2,i_3}(\mathbf{u})}{\sum_{i_1,i_2,i_3} w_{i_1,i_2,i_3}\mathbf{B}_{i_1,i_2,i_3}(\mathbf{u})} \\ &= \sum_{i_1,i_2,i_3} (w_{i_1,i_2,i_3}\mathbf{P}_{i_1,i_2,i_3}) \cdot (a,b,c,d)\mathbf{B}_{i_1,i_2,i_3}(\mathbf{u}) \end{aligned}$$

In both the case of isosurfacing and planar slicing, we are left with the problem of finding the zeros of a trivariate spline having the form

$$\mathbf{D}(u_1,u_2,u_3) = \sum_{i_1,i_2,i_3} \mathbf{D}_{i_1,i_2,i_3}\mathbf{B}_{i_1,i_2,i_3}(u_1,u_2,u_3).$$

To solve the problem, we first break the $\mathbf{D}$ into Bézier volumes and place these into a search list. For each element in the list, we test whether the control points are within an epsilon box of zero. If so, this volume is added to the root list. If not, we test whether the bounding box of the points contains the origin. If the answer is yes, the volume is subdivided and the resulting volumes are appended to the list. On the other hand, if the answer is no, the volume is discarded. This procedure continues until no further volumes are in the search list.

The result of the outlined procedure is a list of volumes which contain the roots of $\mathbf{D}$ (refer to Figures 4.6 and 4.7). We now have to refine $\mathbf{P}^w$ at the interval values which describe the domains for the volumes in the root list. The resulting geometric volumes may need to be further refined according to a flatness criterion. A polygonal approximation can then be displayed. In the case of planar slicing, the resulting polygons may be colored according to the average isovalue in each volume.

4.6.2 Direct Volume Rendering

It is a common technique to map scalar values to colors and visualize volumetric data by passing rays through it. This is frequently termed "direct volume rendering" (see Figure 4.8). Mathematically, this operation can be formulated as the calculation

$$\int_{t_a}^{t_b} \alpha(\mathbf{o}+t\mathbf{d})\mathbf{C}(\mathbf{o}+t\mathbf{d})dt,$$

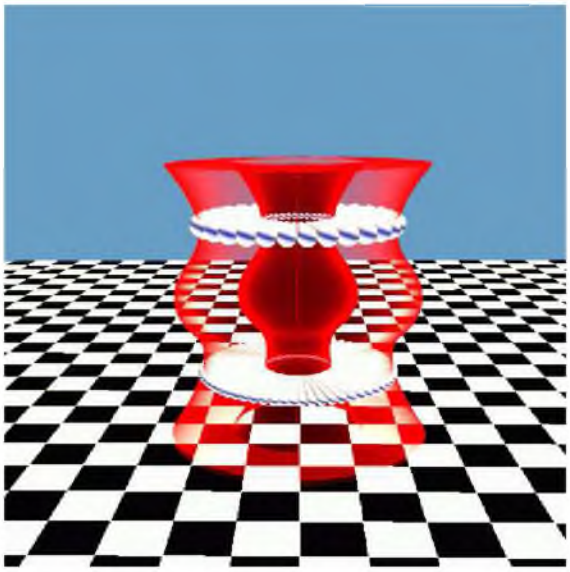

Figure 4.6: Visualization of subdivision-based isosurface extraction.

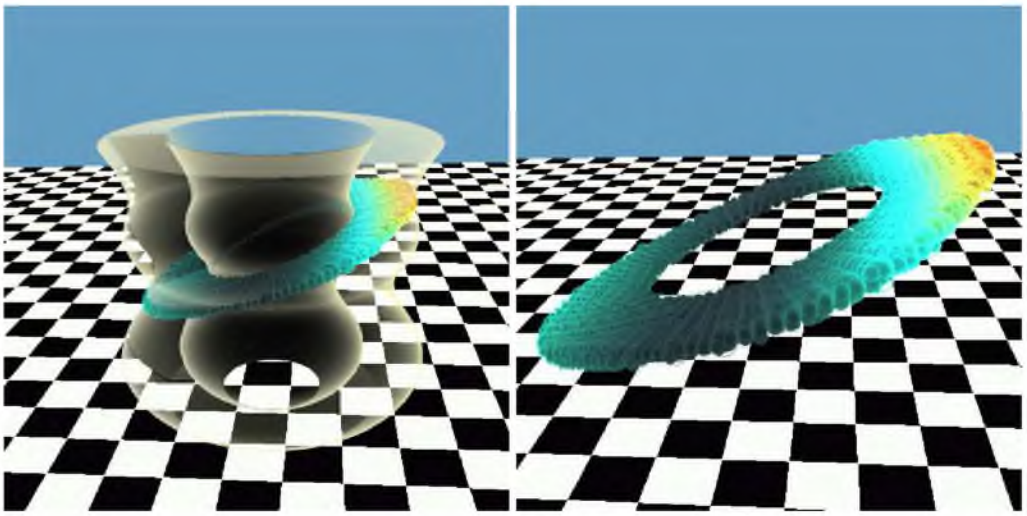

Figure 4.7: Visualization of subdivision-based planar cut extraction.

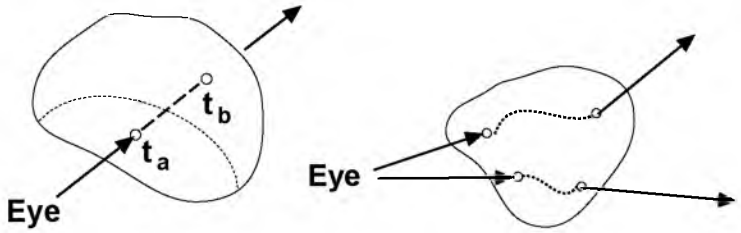

Figure 4.8: Direct volume rendering (left) compared with optical path tracing (right).

where $\alpha(\mathbf{x})$ and $\mathbf{C}(\mathbf{x})$ are the accumulated opacity and color values corresponding the attributes at the point $\mathbf{x}$, and t_a, t_b are the entry and exit points of the ray $\mathbf{o}+t\mathbf{d}$ with the volume $\mathbf{P}(u, v, w)$. It is clear that under energy conservation, $\alpha(\mathbf{x}) \leq 1$. In this section, we provide the machinery for ray tracing volumetric splines.

A trivariate spline $\mathbf{P}(u, v, w)$ is a mapping from the rectangular cell $I_u \times I_v \times I_w$ to $\mathbb{R}^3$. The faces of the domain cell, $\partial I_u \times I_v \times I_w$, $I_u \times \partial I_v \times I_w$, and $I_u \times I_v \times \partial I_w$ map to surfaces in $\mathbb{R}^3$. These surfaces necessarily bound a closed volume, as they share boundaries and are collectively equivalent topologically to a cube. However, as shown in [32], the faces need not enclose the same volume as does $\partial\mathbf{P}(u, v, w)$.

Theorem 1 *Given a rectangular cell $B = [u_a, u_b] \times [v_a, v_b] \times [w_a, w_b]$ and a trivariate B-spline function $\mathbf{P}(u, v, w)$ defined over B, the surface boundary of the solid $\mathbf{P}$ is contained within the union of the faces of the solid over B and the points where the determinant of the Jacobian of $\mathbf{P}$ over B vanishes.*

This theorem was first brought to our attention in [32] and a proof can be found in [122].

To trace a ray through the volume $\mathbf{P}$, we first wish to find the closest point at which the ray $\mathbf{o} + t\mathbf{d}$ contacts the boundary surface $\partial\mathbf{P}$. If a ray is defined as the intersection of two planes $\mathbf{p_1}, \mathbf{p_2}$, where $\mathbf{p_l} = \{\mathbf{x} : (a_l, b_l, c_l, d_l) \cdot (\mathbf{x}, 1) = 0\}, l = 1, 2$, then a ray intersecting $\partial\mathbf{P}$ will satisfy at least one of the following relations:

$$\mathbf{F^k}(s, t) \cdot (a_l, b_l, c_l, d_l) = 0, \text{ for } l = 1, 2 \tag{4.7}$$

or

$$\begin{aligned} \mathbf{P}(u,v,w) \cdot (a_l, b_l, c_l, d_l) &= 0, \text{ for } l = 1,2 \\ \mathcal{J}\mathbf{P}(u,v,w) &= 0 \end{aligned} \tag{4.8}$$

where $\mathbf{F^k}, k = 1..6$ are the faces of $\mathbf{P}$ and $\mathcal{J}\mathbf{P}$ is the determinant of the Jacobian matrix for $\mathbf{P}$. This is simply a restatement of Theorem 1.

As the formulas in Equation (4.7) and Equation (4.8) are implicit equations, we can apply Newton's method to find the roots, and therefore, the intersection of the ray with $\partial\mathbf{P}$, provided we have a good initial guess. With each face, we generate and store a bounding volume hierarchy using subdivision according to a flatness criteria. This is a preprocessing step. See [121] for a detailed discussion. Boxes which do not intersect the ray are culled, and we apply a Newton's method to Equation (4.7) using the starting value associated with each of the remaining boxes.

To handle the implicit boundaries described by Equation (4.8), we store with each volume $\mathbf{P}$ a bounding hierarchy obtained by subdividing $\mathbf{P}$ until each piece is within a maximum volume tolerance. We cull those boxes whose volumes cannot contain a zero Jacobian determinant according to the following method due to [32].

A cone is determined by a normalized axis vector $\hat{\mathbf{C}}$ and a spread angle θ (Figure 4.9). In what follows, a "ˆ", as in $\hat{\mathbf{v}}$, will denote the normalized form of a vector. Given a set of vectors $\{\mathbf{v_i}\}$, we can fit a bounding cone to them using the following algorithm due to [123]. Set $\hat{\mathbf{C}}_0 = \mathbf{v}_0/||\mathbf{v}_0||$, and $\theta_0 = 0$. For each subsequent vector $\mathbf{v_i}$, the angle α between $\mathbf{v_i}$ and $\hat{\mathbf{C}}_{\mathbf{i-1}}$ is given by $\alpha = \cos^{-1}(\hat{\mathbf{v}}_\mathbf{i} \cdot \hat{\mathbf{C}}_{\mathbf{i-1}})$. For $\alpha \leq \theta_{i-1}$, $\hat{\mathbf{C}}_\mathbf{i} = \hat{\mathbf{C}}_{\mathbf{i-1}}, \theta_i = \theta_{i-1}$. Otherwise, we compute an

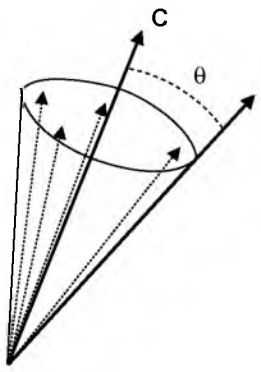

Figure 4.9: A bounding cone.

intermediate vector $\mathbf{v_t}$

$$\mathbf{v_t} = \begin{cases} \mathbf{v_{i-1}}, & \text{if } \theta_{i-1} = 0 \\ \cot\theta_{i-1} \sin\alpha \hat{\mathbf{C}}_{\mathbf{i-1}} - \hat{\mathbf{v}}_{\mathbf{i}}, & \text{otherwise.} \end{cases}$$

We have that

$$\hat{\mathbf{C}}_{\mathbf{i}} = \frac{\hat{\mathbf{v}}_{\mathbf{i}} + \hat{\mathbf{v}}_{\mathbf{t}}}{||\hat{\mathbf{v}}_{\mathbf{i}} + \hat{\mathbf{v}}_{\mathbf{t}}||}, \quad \cos\theta_i = \hat{\mathbf{C}}_{\mathbf{i}} \cdot \hat{\mathbf{v}}_{\mathbf{t}}$$

(Since the publication of our work in 2001, an algorithm for finding optimal bounding cones using linear programming has been published. We refer the reader to [124].)

We extend the dot product and cross product operators to cones in the following way. Given two cones C_1, C_2, $C_1 \cdot C_2$ is the range of scalar product values for vectors bounded by C_1 and C_2. Analogously, $C_1 \times C_2$ is the cone C_3 which bounds the cross-products of vectors bounded by C_1 and C_2. A conservative estimate of the cross product is accomplished by crossing the cone axes, and calculating the spread angle θ_3 via:

$$\theta_3 = \sin^{-1}\left(\frac{\sqrt{\sin^2\theta_1 + 2\sin\theta_1\sin\theta_2\cos\beta + \sin^2\theta_2}}{\sin\beta}\right)$$

where β is the smaller of the angles between the two cone axes [123].

We know that the partial derivatives of a NURBS volume $\mathbf{P}^w$ are again NURBS volumes (Equation (4.1)). Consider the cones C_u, C_v, C_w which bound the homogenized ($\mathbb{R}^3$) control points denoted $(\mathcal{D}_u\mathbf{P})_{i_1,i_2,i_3}$, $(\mathcal{D}_v\mathbf{P})_{i_1,i_2,i_3}$, $(\mathcal{D}_w\mathbf{P})_{i_1,i_2,i_3}$, respectively, of the derivative volumes. By virtue of the convex hull property, we have that

$$\mathcal{J}(\mathbf{P}) \equiv \mathcal{D}_u\mathbf{P} \cdot (\mathcal{D}_v\mathbf{P} \times \mathcal{D}_w\mathbf{P}) \subseteq L(C_u \cdot (C_v \times C_w)),$$

where L is an interval of positive values [32]. This implies that $\mathcal{J}\mathbf{P} \neq 0$ in the given volume if $0 \notin C_u \cdot (C_v \times C_w)$. We can remove bounding boxes containing such volumes from consideration. As before, we can also cull those boxes which the ray does not intersect, and apply the Newton iteration to Equation (4.8) using the start (*e.g.*, average parameter) values stored in the remaining boxes.

The result will be a list of points where the ray $\mathbf{p_0} + \mathbf{v_0}t$ intersects $\partial\mathbf{P}$. For each volume,

there should be a pair of points: an entry and an exit point. Let the point closest to the ray origin have coordinates $\mathbf{u_1}$, $\mathbf{p_1}$, where $\mathbf{u_1} = (u_1, u_2, u_3)_1^T$. We evaluate the attribute data at the point $\mathbf{u_1}$, and obtain an opacity α_1 and a color $\mathbf{C_1}$. The accumulated color $\mathbf{C_{acc}} = \alpha_1 * \mathbf{C_1}$ and the accumulated opacity $\alpha_{acc} = 1 - \alpha_1$.

Now, we begin to traverse the volume. Starting from $\mathbf{p_1}$ and traveling a small distance Δt along the vector $\mathbf{v_1}$, we arrive at the point $\mathbf{p}_*$. Since $\mathbf{p}_*$ is close to $\mathbf{p_1}$, we are justified in the approximation $\mathbf{p}_* - \mathbf{p_1} \approx \mathbf{JP}(\mathbf{u_1})(\mathbf{u}_* - \mathbf{u_1})$, where $\mathbf{JP}(\mathbf{u})$ is the Jacobian matrix, $\mathbf{u}_k = (u_1, u_2, u_3)_k^T$ and we know $(u_1, u_2, u_3)_1^T$ from the previous Newton iteration. This leads to the Newton iteration

$$[\mathbf{JP}(\mathbf{u_k})]^{-1}(\mathbf{p}_* - \mathbf{P}(\mathbf{u_k})) + \mathbf{u}_k = \mathbf{u}_{k+1} \tag{4.9}$$

Note that the functions $\mathbf{P}$ and $\mathbf{JP}$ lack the superscript ω. This denotes a projection into $\mathbb{R}^3$. $\mathbf{P}(\mathbf{u_k})$ is found by dividing $\mathbf{P}^\omega(\mathbf{u_k})$ by its rational coordinate. In similar fashion, $\mathbf{JP}(\mathbf{u_k}) = [\mathbf{P_u}(\mathbf{u_k})\ \mathbf{P_v}(\mathbf{u_k})\ \mathbf{P_w}(\mathbf{u_k})]$ is found by computing $\mathbf{P_u^\omega}(\mathbf{u_k})$, $\mathbf{P_v^\omega}(\mathbf{u_k})$, and $\mathbf{P_w^\omega}(\mathbf{u_k})$ and homogenizing each.

From Equation (4.9), we obtain $\mathbf{u}_*$, which we can again use to calculate the attribute data and corresponding color and opacity functions, $\mathbf{C_2}$ and α_2. We increment the accumulated colors and opacities $\mathbf{C_{acc}} = \mathbf{C_{acc}} + \alpha_{acc} * \alpha_2 * \mathbf{C_2}$ and $\alpha_{acc} = \alpha_{acc} * (1 - \alpha_2)$.

This process is repeated until the ray exits the volume. The color of the ray can be written as $\mathbf{C_{acc}} = \sum_{i=1}^{\infty} \alpha_i \mathbf{C_i} \prod_{j=1}^{i-1}(1 - \alpha_j)$. An image rendered in this fashion is shown in Figure 4.10.

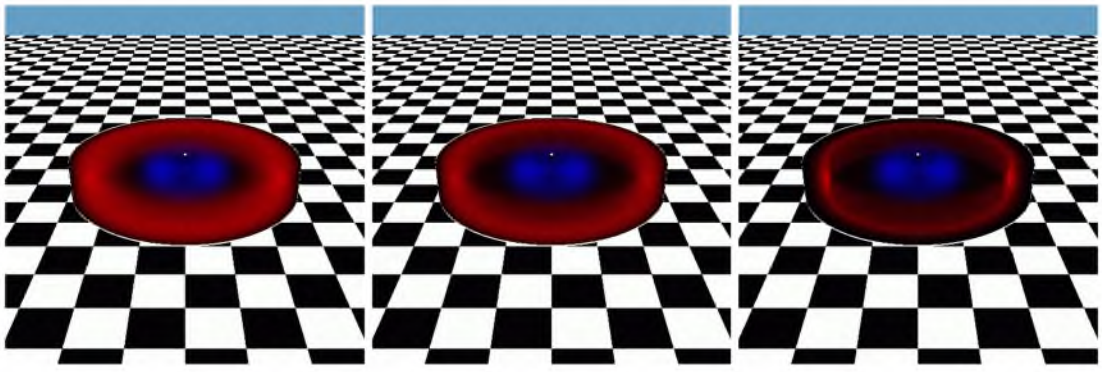

Figure 4.10: A sequence of progressively sharper isosurfaces extracted using direct volume rendering.

4.6.3 Optical Path Tracing

An application that sometimes occurs in optics is the desire to trace the path of a ray which is perturbed by a spatially varying refractive index. Some example applications are visualizing atmospheric effects such as thermal clines near the ground, metropolitan pollution, atmospheric perspective, and cutting-edge optical lenses such as GRIN (gradient index of refraction) lenses. The attribute data for such a volume would include a model of the refractive index, $\eta(u, v, w)$, at each point of its interior.

The well-known Snell's law formula at the interface between discrete media is $\eta_{\text{in}} \sin\theta_{\text{in}} = \eta_{\text{out}} \sin\theta_{\text{out}}$, where $\theta_{\text{in}}, \theta_{\text{out}}$ are measured between the normal and the ray. See Figure 4.11 for an illustration. The formula also holds true in a volume with a varying refractive index. The interface in the discrete formula corresponds to the isosurface with constant η in the volume. A ray with direction $\mathbf{v}$ is perturbed with respect to the normal $\mathbf{n}$ of the isosurface which it contacts. Since this normal will by necessity point in the direction of maximum change in η, $\mathbf{n} = \nabla\eta$. It follows that the path of a ray will in general trace a curved path through a medium with continuously varying refractive index. See Figure 4.8.

The gradient $\nabla\eta$ is given by $(\partial\eta/\partial x, \partial\eta/\partial y, \partial\eta/\partial z)$. By the chain rule, we have that

$$\begin{pmatrix} \partial\eta/\partial u \\ \partial\eta/\partial v \\ \partial\eta/\partial w \end{pmatrix} = \begin{pmatrix} \partial x/\partial u & \partial y/\partial u & \partial z/\partial u \\ \partial x/\partial v & \partial y/\partial v & \partial z/\partial v \\ \partial x/\partial w & \partial y/\partial w & \partial z/\partial w \end{pmatrix} \begin{pmatrix} \partial\eta/\partial x \\ \partial\eta/\partial y \\ \partial\eta/\partial z \end{pmatrix}$$

yielding that

$$\nabla\eta = (\mathbf{J}\mathbf{P}^T)^{-1}(\partial\eta/\partial u, \partial\eta/\partial v, \partial\eta/\partial w)^T. \tag{4.10}$$

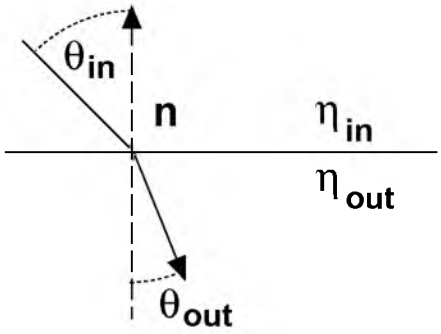

Figure 4.11: Snell's law.

The method described in Section 4.6.2 may be modified as follows. We calculate the intersection of the ray $\mathbf{p_0} + t\mathbf{v_0}$ with the boundary of the volume $\partial\mathbf{P}$, yielding points $\mathbf{u_1}$ and $\mathbf{p_1}$. We evaluate η and $\mathbf{n_1} = \nabla\eta$ at $\mathbf{u_1}$ to generate a new ray direction $\mathbf{v_1}$. $\mathbf{n_1}$ is reflected, if necessary, so that $\mathbf{n_1} \cdot \mathbf{v_0} < 0$. It must be the case that $\mathbf{v_1}$ is in the plane containing $\mathbf{v_0}$ and $\mathbf{n_1}$. We can generate a vector tangent to the isosurface $\mathbf{t_1} = \mathbf{n_1} \times (\mathbf{v_0} \times \mathbf{n_1})$. As a byproduct of this computation, we obtain $\sin(\theta_0) = ||\mathbf{v_0} \times \mathbf{n_1}||$. The angle between $-\mathbf{n_1}$ and $\mathbf{v_1}$ is given by $\theta_1 = \arcsin(\eta_0 \sin(\theta_0)/\eta_1)$, where η_0 is 1 for air. The outgoing ray direction is then given by $\mathbf{v_1} = \sin\theta_1\mathbf{t_1} + \cos\theta_1\mathbf{n_1}$. We walk a distance Δt as before, determining points $\mathbf{u_2}$, $\mathbf{p_2}$, and the perturbation is calculated using the η_1, $\eta_2 = \eta(\mathbf{u_2})$, and $\mathbf{n_2} = \nabla\eta(\mathbf{u_2})$. Note that in the process of calculating Equation (4.9), we have calculated the Jacobian needed for Equation (4.10). Also note that the stepsize Δt can be made to depend on the gradient $\nabla\eta$. The larger the gradient magnitude, the shorter the distance we can cover without missing something.

Figure 4.12 demonstrates our path tracing technique. Both lenses are geometrically flat. The one on the left has a constant index of refraction, whereas the lens on the right has a radially varying index of refraction.

4.6.4 Summary of Ray Tracing

We summarize the ray tracing algorithm in Figure 4.13. Here, we have integrated the developments from the previous sections to produce a general approach.

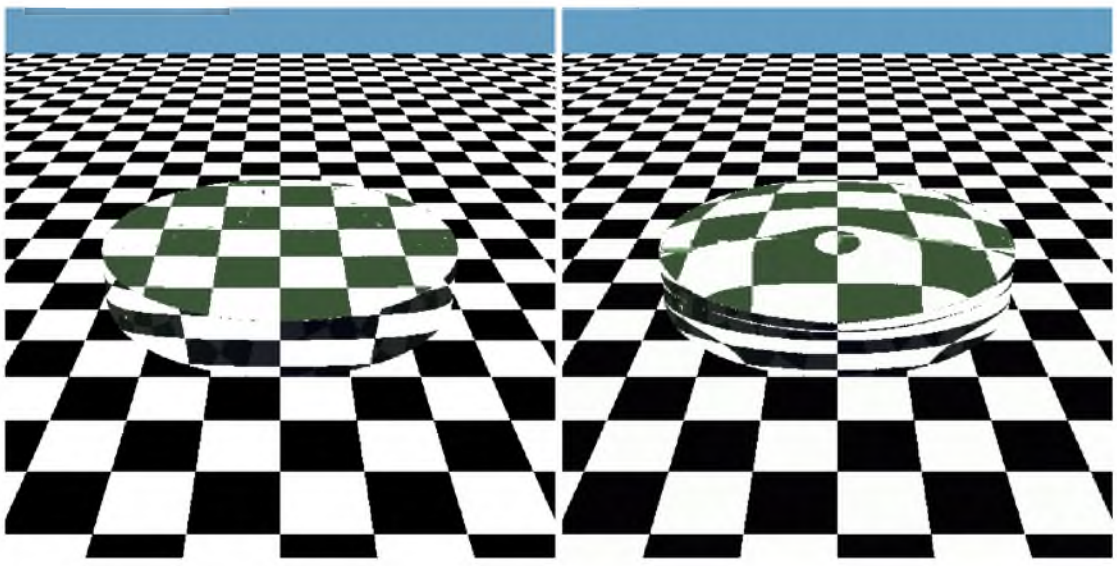

Figure 4.12: Lens with constant index of refraction (left) and varying index of refraction (right).

```
TraceRay ( Environment env, Ray r, Color C)
  loop
    for each Volume vol in env do
      IntersectBoundary(r,vol,hitlist)
    end for
    if hitlist = NULL then
      return
    end if
    vol = hitlist.closest
    uv = r.hit.uv
    p = r.hit.p
    n = r.hit.normal
    while p ∈ vol do
      r.o = p
      PerturbRayDirection(r,vol,uv)
      ComputeColor(C,vol,uv)
      p = r.o + Δt * r.v
      CalcParametricPoint(r.o,p,uv)
    end while
    r.o = p
    PerturbRayDirection(r,vol,uv)
  end loop
```

Figure 4.13: Algorithm for tracing rays through a trivariate volume.

4.7 Conclusion and Future Work

This chapter has introduced a framework for representing attribute data orthogonally to geometric data within a trivariate NURBS volume. It extends existing modeling and data fitting techniques to this new representation and presents an efficient algorithm for volume evaluation. In addition, we have incorporated techniques for data visualization such as planar slicing, iso-surfacing, ray tracing, and optical path tracing which may serve as invaluable aids to composite design or data analysis.

From the modeling standpoint, there is much yet to be done. Tensor-product surfaces are a deformation of a rectangle, and therefore limited topologically. In the past, trimming curves have provided added flexibility to surface design. In the case of volumes, the problem is decidedly harder, and a practical solution is not yet known. The scheme we prescribe generalizes in a straight-forward manner to subdivision surfaces. On the other hand, greater flexibility leads to a reduction in performance.

The visualization techniques depend critically on how well-behaved the target models are. For example, if the mapping from parameter space to geometry is not one-to-one, then the search for preimage as prescribed in the optical path tracing section is not well-defined. Likewise, topological information such as adjacency may need to be available for our routines. As an

example, consider a cylindrical volume of revolution (see Figure 4.14). A cylinder has three exterior faces, whereas a trivariate NURBS must have six. Three of the parametric boundary faces are therefore interior to the geometric model. Two of the faces meet to form an internal boundary. This boundary must be ignored for the ray tracing algorithm to perform properly. Likewise, one of the faces is singular — it maps to a line segment. A ray tracing algorithm must be aware of this singularity or unexpected results may occur. Our method for computing the closure of a B-rep will induce such a singularity at the medial axis. Knowledge of these singularities should be encoded in the data structure to ensure proper algorithm performance. In our programs, we have dealt with these singularities in an ad hoc manner, leaving their proper identification and resolution to future work. Splines lend themselves to multiresolution methods. For example, modeling can be expedited if changes are made from coarse to fine. As another example, evaluations often only need to be made within a given error tolerance, opening the door to potential savings. We suspect further exploration of these traits will prove fruitful.

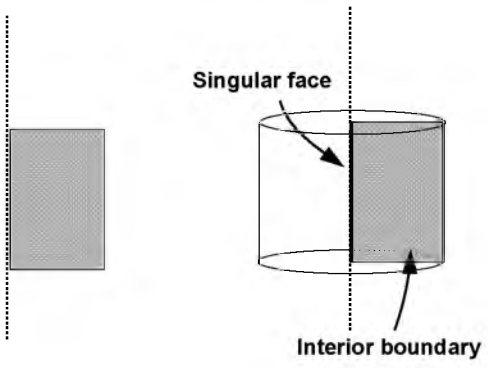

Figure 4.14: Demonstration of boundary singularities. A rectangle is revolved about a vertical axis to create a solid (left). The resulting volume (right) has two coincident internal faces (shown in gray) and one singular (one-dimensional) parametric boundary along the axis.

CHAPTER 5

SURFACE COMPLETION FROM AN IRREGULAR BOUNDARY CURVE

5.1 Overview

It is frequently necessary to complete the design of a surface from a specification of its boundary. This chapter introduces a technique for completing the surface when the boundary is described by a non-self-intersecting, closed, planar, B-spline curve. The mapping produces a tensor product B-spline surface whose outer boundary is the input curve, and whose parameterization generalizes the polar parameterization of the disc.

5.2 Introduction

In this chapter, we propose a new operator for generating a planar surface from a closed, non-self-intersecting piecewise polynomial boundary in the plane. We consider this approach to be a novel step towards the larger goal of surface completion from a freeform curve boundary. This is a common problem arising in geometric modeling. Examples include "capping" extrusions and filling holes where adjacent patches come together. Holes also commonly occur in scanned datasets. There are many applications where such models must be made "watertight."

Given the importance of the problem, a number of methods have been proposed for surface completion. Rather than attempt to warp a rectangular uv domain to an irregularly shaped region, a common approach is to employ tensor product surfaces whose parameter domains are further restricted by trimming curves. Generally, the boundary must be densely sampled to accurately represent this subset and the parameterization originally associated with the boundary curve is lost. Moreover, the representation does not lend itself easily to further modeling operations.

Several methods have been introduced for hole-filling (*e.g.,* [73]). Often, these techniques do not address the parameterization of the filled region. When the parameterization is addressed, it is often piecemeal, composed of a series of adjacent parametric patches.

Finally, there are a number of classic works on completing a surface from a series of bounding curves [74–77]. This work is most closely related to the algorithm we will develop. However,

in contrast to our approach, these techniques generally assume the boundary can be naturally decomposed into n-faces, which can in turn be blended together. For example, schemes have been developed to complete a surface from 3, 4, 5, and 6-sided areas. There are many commonly occurring curve examples that do not easily admit such a decomposition (see Figure 5.1). Another drawback is that many techniques are tailored for a certain "sided-ness." Finally, the parameterization, when it is developed, is not always a straightforward mapping from a rectangular domain.

The starting point for surface completion algorithms is a description of the boundary. Therefore, an intuitive approach to completion is a parameterization that starts on the boundary and works its way inward (see Figure 5.2). This idea is evocative of offset curves. Take the sequence of curves generated by successively moving each point on the curve a fixed distance in the direction of its normal. The union of such a sequence can be used to parameterize the interior of the boundary. However, this type of completion has a problem in that all points do not generally come together simultaneously. Thus, as seen in Figure 5.2, portions of the offsets will begin to cross and must be clipped to avoid singularities in the parameterization. This results in a complex parameterization. Another possible technique is to use variable offsets. The problem now becomes how to choose the offset distances. This chapter offers a solution — concentric parameterization.

A related technique that does not have the crossing problem is based on level set methods [125]. However, these methods are grid-based and do not produce parameterizations. Our technique uses parametric functions and therefore has straightforward application in most commonly used geometric modeling systems.

Our parameterization is inspired by the standard (r, θ) parameterization of the disc. Such a surface parameterization respects the parameterization of the outer circle which is its boundary (see Figure 5.3). One parameter can be seen as traversing the boundary, whereas the other selects

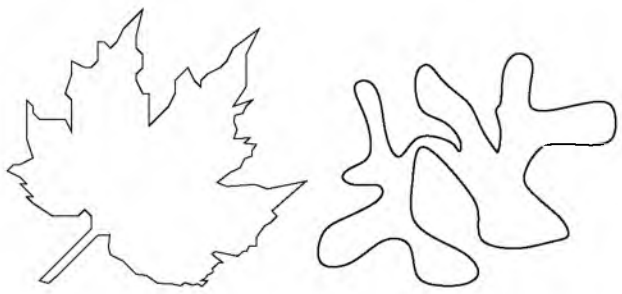

Figure 5.1: Motivating examples for our work.

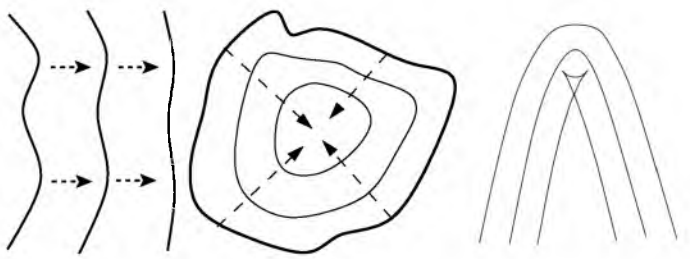

Figure 5.2: Moving curves and a problem with offsets

the successive scalings of the boundary which work their way to the center. The main goal of this chapter is to find a method of surface completion which simulates this (r, θ)-type relationship while assuring that the successive offsets meet simultaneously (see Figure 5.3, rightmost).

The medial axis is the natural generalization of the circle's centerpoint to objects with more complex boundaries. The medial axis of a figure is defined to be the locus of the centers of all maximal inscribed circles. Such a circle will touch the boundary at at least 2 points. Since there is a corresponding point on the medial axis for each boundary point, it is natural to consider the trivial surface completion operator

$$(1-t)\gamma(s) + t\mathcal{MA}(\gamma(s))$$

where $\gamma(s)$ is a point on the boundary curve, and $\mathcal{MA}(\gamma(s))$ maps this point to the medial axis. The left frame of Figure 5.4 shows this mapping applied to a rectangle, and the middle panel shows the resulting surface parameterization.

This parameterization is subject to a considerable distortion — in particular, every isoline (s, t_0) travels through the corners (Figure 5.4, middle). This certainly does not capture the intuitive notion of a disc-like parameterization. Another problem is that $\mathcal{MA}$ is not always a

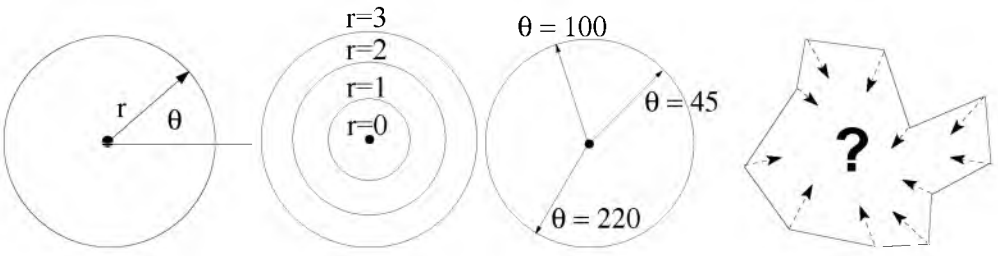

Figure 5.3: The inspiration for our parameterization

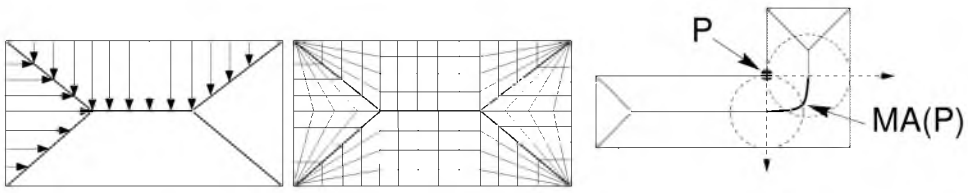

Figure 5.4: Weaknesses in a direct medial mapping.

function. A concave vertex, for example, maps to an entire curve segment along the medial axis (Figure 5.4, right).

5.3 The Concentric Parameterization

Figure 5.5 contains a brief review of nomenclature. The medial axis can be divided into roughly two types of curves. *Sheets* are portions of the medial axis that do not touch the boundary. They are joined to the boundary (and in particular to the convex points) by the *seams*. It was the inclusion of these seams in the surface completion that led to the severe distortions of Figure 5.5. Our new mapping projects solely to the sheet, using the seams to guide the contraction.

5.3.1 Polygonal Boundary

Let us first consider the case of a polygonal boundary. The right panel of Figure 5.5 illustrates that the medial axis divides the polygon into regions. Each region is bounded by two seams, a portion of the boundary curve, and a portion of the sheet (which can degenerate to a point). Each point contained in a region is closer to its boundary and sheet segment than it is to any of the

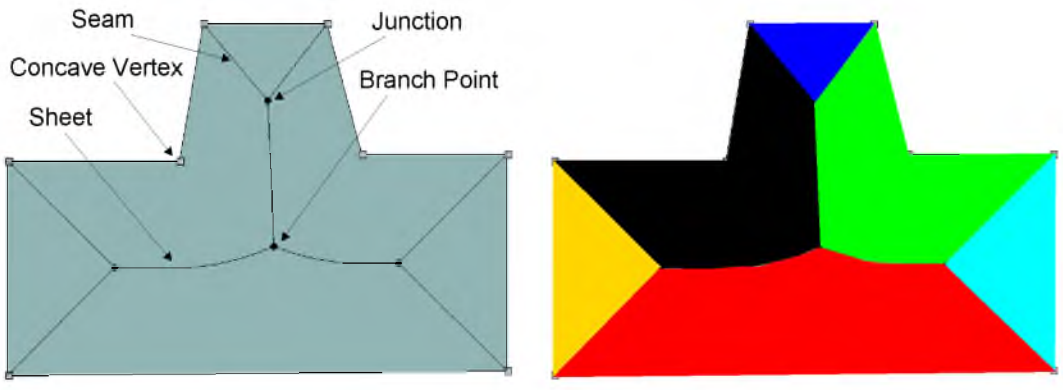

Figure 5.5: Nomenclature review; regions formed by the medial axis.

other boundary or sheet segments. In particular, the boundary segment is closer to the part of the sheet bounding its region than it is to any other part of the sheet. (Similar observations have been made by [126].) Thus, it is natural to consider a mapping of each region boundary onto its corresponding sheet segment. Our contraction is based on a particular approximation to the sheet:

> **Definition 5.1** *The* concentric axis *of a polygon is a piecewise linear approximation to the medial axis sheet whose vertices, termed* concentric vertices *satisfy the following two properties:*
>
> - *a) every vertex of the polygon corresponds to a concentric vertex and*
> - *b) every concentric vertex has a corresponding vertex on the boundary polygon of each region it borders.*

If condition b) is not satisfied, the surface completion may contain holes [126]. The set of concentric vertices will include the junction and branch points (Figure 5.5). The convex vertices naturally map to the junction points of the medial axis. To fully satisfy condition a), we must find a mapping of the concave boundary vertices onto the medial axis. Considering the right pane of Figure 5.4, it is reasonable to select any point in the range of $\mathcal{MA}$ corresponding the concave vertex as an addition to the concentric axis. If there are preexisting concentric vertices in this range, we may select the closest one to simplify the next step.

We now form a curve from the concentric axis; this curve will be blended with the boundary curve to complete the surface.

> **Definition 5.2** *The* concentric control polygon *or more simply* concentric polygon *is a sequence of concentric vertices determined by traversing the boundary polygon in the direction of increasing parametric value, and inserting the concentric vertex corresponding to each boundary point encountered.*

> **Definition 5.3** *A* concentric curve *is a piecewise linear B-spline defined by a concentric control polygon and a corresponding knot vector (termed the* concentric knot vector*).*

We want to preserve the original parameterization of the boundary curve as an isoparametric direction in the surface completion. Since even the case of a polygonal boundary admits a

nonuniform parameterization, we assume that there is a knot vector associated with the boundary. Each vertex of the boundary therefore has an associated parameter, which is used to assign parameters to the corresponding concentric vertices, and form a knot vector for the concentric curve.

In order to satisfy condition b) of Definition 1, it is necessary to ensure that each concentric vertex has a boundary vertex correspondence for each region it borders. If a boundary correspondence is lacking, one will be inserted as follows. Because we have imbued the concentric curve with a parameterization, we can approximate the parameter value of the concentric vertex for this region, and add this value to the concentric knot vector. We add the concentric vertex to the corresponding location in the concentric polygon. When we bring the boundary curve and concentric curve into the same spline space for the final blend, the refinement will find the boundary correspondence automatically. The mapping based on this blend is demonstrated for a rectangle in Figure 5.6.

5.3.2 Generalization to Higher Order Curves

Direct application of this technique to arbitrary curves is somewhat difficult. One wishes to perform the sort of contraction introduced above on a finite number of points, but one also generally wants to avoid discretizing the curve. The B-spline representation provides a tractable solution to this problem. Because the B-splines are a) defined by a control polygon, b) possess the convex hull property with respect to the control polygon, and c) are variation diminishing with respect to the control polygon, the contraction of the boundary control polygon onto its concentric polygon implies a mapping of the boundary curve to the concentric curve. Since the boundary control polygon converges to the boundary curve under refinement, the medial axis of the control polygon will converge to that of the boundary curve in the limit. Hence, the technique of the previous section provides a good approximation to the continuous case with sufficient refinement. The quality of the parameterization is largely dependent on how well the boundary control polygon approximates the boundary curve.

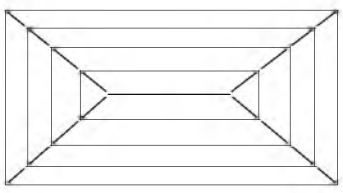

Figure 5.6: Parameterization of the rectangle which results from our mapping.

The concentric completion algorithm for curves is an extension of the algorithm for polygons. The method of determining the parameterization of the concentric curve is a straightforward generalization. We associate with each boundary vertex the nodal value of its associated B-spline basis function. This is the parameter where, to first approximation, its influence is greatest. We summarize the concentric algorithm in Figure 5.7. Figure 5.8 demonstrates the algorithm on a simple, uniform, cubic curve (Figure 5.8(a)).

Figure 5.9 shows a simple case where using the concentric axis of the boundary control polygon fails, because this approximate skeleton crosses the boundary curve. This example violates our assumption that the boundary control polygon is a good approximation to the shape of the underlying curve. However, we can detect cases where the concentric curve crosses the boundary curve rather easily. We simply test each concentric polygon edge for crossings. One way to do this is to cast a ray along each edge. If there is an intersection with the boundary curve between the endpoints of the segment, then the concentric axis is not valid. We note that for low order curves, this intersection can be accomplished analytically. A similar method can be used to determine the quality of the approximation by calculating distance to the boundary curve [127].

If the concentric axis approximation is found inadequate, one option is to refine the boundary control polygon, and restart the concentric algorithm. However, we want to avoid an unnecessary

---

1. Assign a parameterization to the boundary vertices using the nodal values (Greville abscissas) of the spline space (Fig. 5.8(b)).
2. Calculate the medial axis of the control polygon (Fig. 5.8(c)).
3. For each concave vertex, insert a concentric vertex (Fig. 5.8(d-e)).
4. Form the concentric axis control polygon: for each boundary vertex, find the closest concentric vertex along the seam, and add it to the concentric axis. Add the associated parameter (nodal value from step 1) to the concentric knot vector (Fig. 5.8(f-l)).
5. For each region, if there are internal concentric vertices, insert them at the appropriate place in the concentric control polygon. Calculate the interpolated parameter value, and add it to the concentric knot vector (Fig. 5.8(m-n)).
6. Degree raise the concentric linear B-spline to match the degree of the boundary.
7. Refine the boundary and concentric curves using the union of their knot values.
8. Form the sweep surface (Fig. 5.8(o)).

Figure 5.7: Basic concentric completion algorithm.

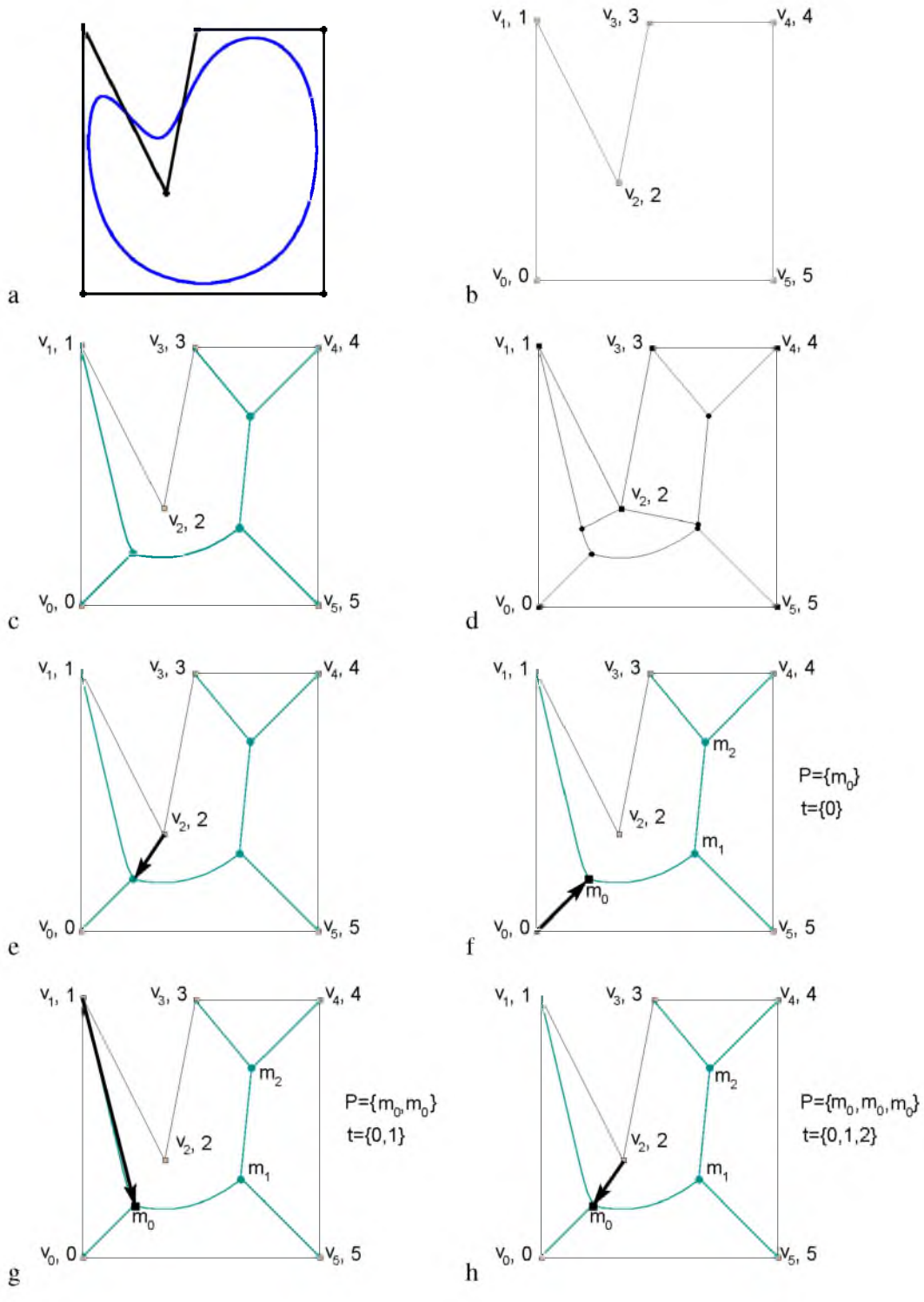

Figure 5.8: The concentric completion algorithm, illustrated.

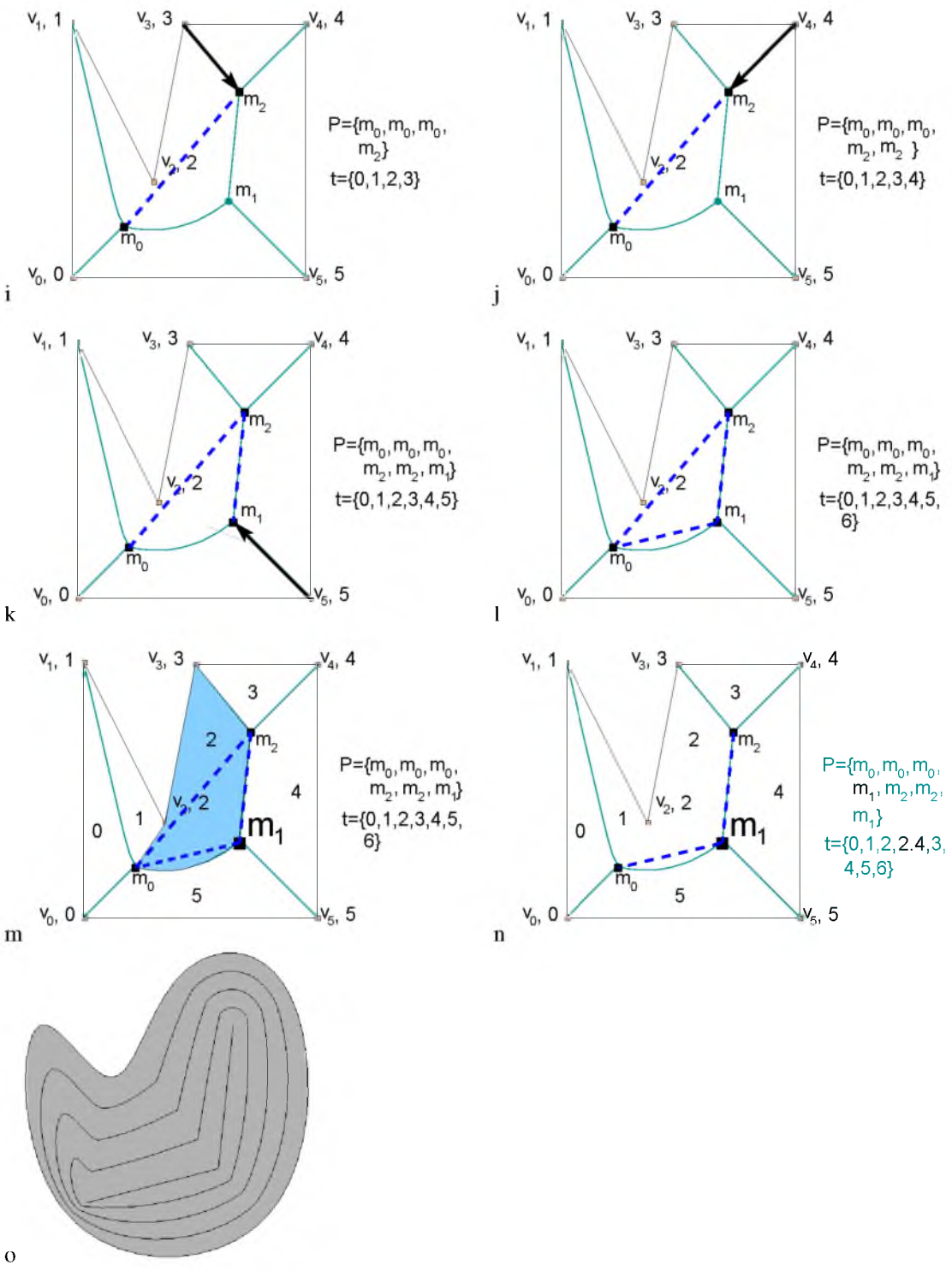

Figure 5.8: (continued)

explosion in the number of surface control points. An alternative is to move the concentric curve until it is seated within the boundary curve. It is sometimes possible to do this by calculating the medial axis of the refined control polygon and moving the concentric axis into alignment. When successful, this can eliminate the need for additional control points. This technique was used in Figure 5.10. The tradeoff between approximation quality and the number of control points is readily apparent.

5.4 Conclusions

We have presented a technique for completing a surface from a non-self-intersecting, closed, planar B-spline curve. Figure 5.11 shows that the algorithm works as expected for convex curves. Figures 5.12 and 5.13 demonstrate the algorithm on some more complicated curves. On the right of Figure 5.13, we again demonstrate the results of moving the axis to avoid refining the boundary. Our method has produced reasonable parameterizations of a variety of complex figures where existing completion techniques would fail or experience difficulty. Presently, we pursue a generalization of the technique to nonplanar boundary curves and methods for accommodating further modeling operations involving the boundary and the completed surface. The present technique is not ideal for boundaries with detail at many scales. Such boundaries tend to have secondary and tertiary branches that have little to do with the basic shape of the surface. We are developing methods to deal with these issues.

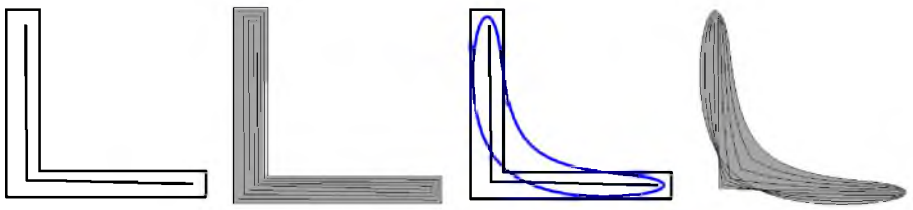

Figure 5.9: An example where the basic algorithm fails (degrees 1 and 3).

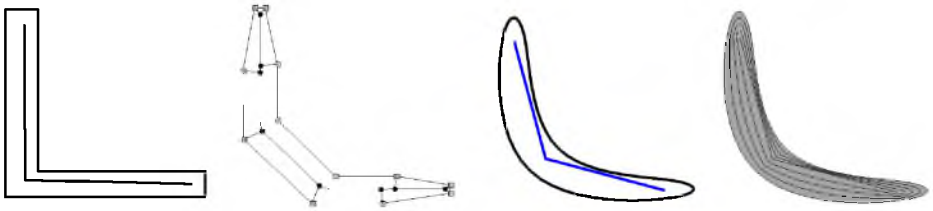

Figure 5.10: Moving the coarse concentric axis to its refined position.

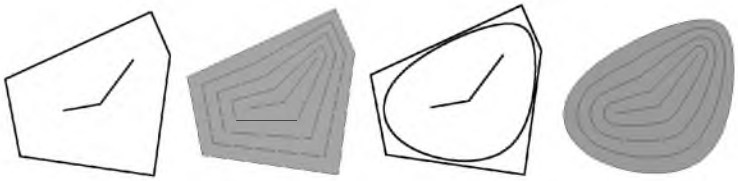

Figure 5.11: Our algorithm applied to a simple convex curve (degrees 1 and 3).

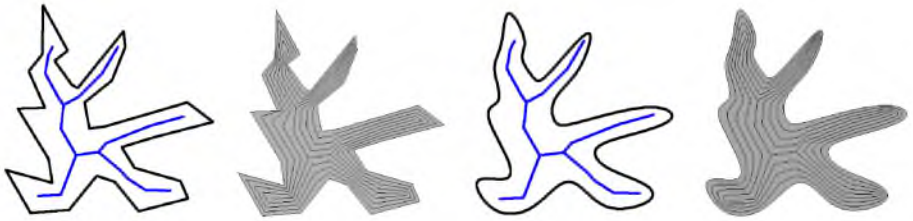

Figure 5.12: A more complicated example for degrees 1 and 3.

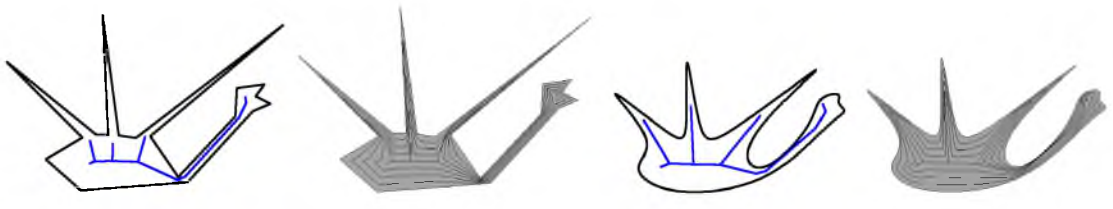

Figure 5.13: A sharp polygon for degrees 1 and 3.

CHAPTER 6

VOLUME COMPLETION FROM A BOUNDARY REPRESENTATION

6.1 Overview

There are many occasions where it is useful to relate a boundary representation of a solid to a true volumetric representation. Examples include heterogeneous manufacture, scientific and engineering simulation, and medical visualization. Traditionally, obtaining this volumetric representation entails discretizing the solid into volume elements. However, this can be nonoptimal for a variety of reasons, including a potential explosion of data, loss of precision, artifacts related to sampling, and loss of any associated parameterization — with the corresponding loss of encoded information. Other volumetric representations, such as those based on implicits, result in a fundamentally different mode of interaction with the resulting volume, producing a loss of familiarity and intuition. We desire a representation which is more generally compatible with an iterative design cycle, and preserves the functional relationships present in the source representation.

We introduce a method for synthesizing a parameterized volume from a boundary representation. The ideas apply generally to boundary representations possessing the convex hull property with respect to a finite set of control points. Due to their widespread utilization in modeling and manufacture, our target domain is the tensor-product NURBS surfaces. In this case, the resulting volume is likewise a tensor product. The volume is synthesized through the application of an approximate medial axis transform to produce a swept volume terminating at a central axis. The resulting parameterization is concentric in nature, preserving the original parameterization along the boundary. The radial parameter is an index into a family of offset surfaces, resulting in a natural correspondence between parametric and geometric locations within the volume.

6.2 Introduction

Traditional models from animation, rendering, and geometric design are boundary representations. However, when simulations are desired, volumetric representations are frequently required.

It is useful to preserve the carefully crafted surface representation in the volumetric model. Some example applications are heat transfer, stress simulation, heterogeneous manufacture, and hole filling. A burgeoning field called isogeometric analysis has married NURBS-based modeling to finite element analysis, demonstrating favorable results when a single representation is used throughout the design cycle [4].

The process of "upgrading" a surface-based boundary representation to a true volumetric representation is called volume completion. Our approach to volume completion involves the generation of some well-chosen offsets that converge at a central axis, thereby parameterizing the model. The (r, θ) parameterization of a circle in the plane, where r indexes a family of offsets, is evocative of our approach (see Figure 6.1). The question with more complex figures is how to generate these offsets without introducing discontinuities in the mapping or necessitating a different parameterization of the boundary. Recall the problem of uniform offsets (Figure 6.2).

Often, research in the field refers to this radially indexed family of offsets as a "generalized cylinder" representation. This representation is frequently associated with the medial axis transform. While the medial axis transform has a number of advantages, we eschew it for a few significant disadvantages. Foremost among these is that given an $n-1$ manifold embedded in n

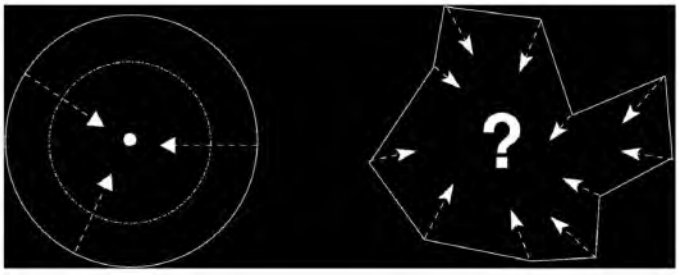

Figure 6.1: The basic idea behind the concentric parameterization.

Figure 6.2: Uniform offsets may introduce crossings.

dimensional space, the medial axis is itself an $n-1$ degree manifold. So whereas in the plane, the medial axis is 1D, it becomes a "medial surface" in 2D. This is shown for a simple configuration in the left of Figure 6.3.

The key difficulty with mapping the boundary onto a 2D axis is in matching the parameterizations of surfaces as they meet across the axis. Consider the extremely simple situation pictured in Figure 6.4. In order to traverse the medial surface, we would need to track, for each point on the "medial surface," which parameters map to that point – in essence encoding a 2D trimming surface from the field of computer-aided geometric design. Furthermore, it is nearly impossible to ensure that the boundary surfaces abut perfectly at the medial surface. This is because the medial surface is piecewise biquadratic. While the splines include the space of biquadratic functions, the tensor product nature of B-spline surfaces means they can only perfectly capture these functions along certain orientations – consider a diagonal line on a raster display. The "jaggies" are analogous to the imperfect characterization of the off-axis function.

The medial axis transform is also highly susceptible to noise. A single small notch on the boundary can result in a spur in the medial surface. That is, small boundary perturbations can result in disproportionately large changes in the medial surface.

In order to resolve these shortcomings, we turn to skeletonization algorithms. There are a large variety of these in the literature, and some of the most promising techniques were summarized in [66]. The mapping of a boundary onto a 1D skeleton by no means escapes the problems indicated in Figure 6.4. However, this problem area is well isolated and has a low probability of being encountered during volume traversal. It also is in some sense as far from the boundary as possible, which in many simulations is where the interesting features lie.

To summarize, we shall prefer a 1D axis for our generalized cylinder representation because:

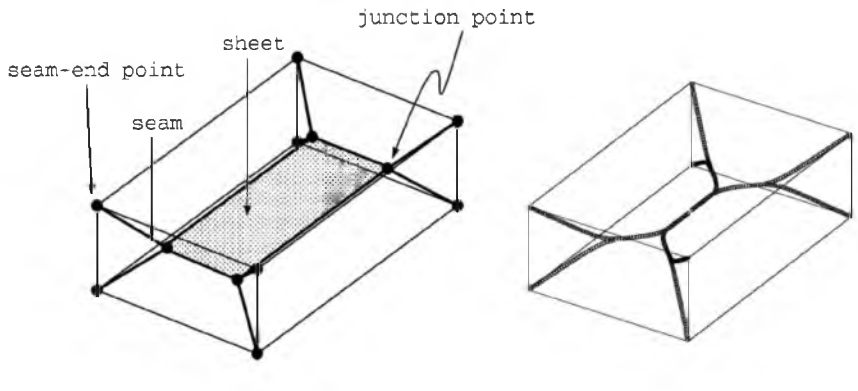

Figure 6.3: Comparing the medial axis to the potential-based skeleton.

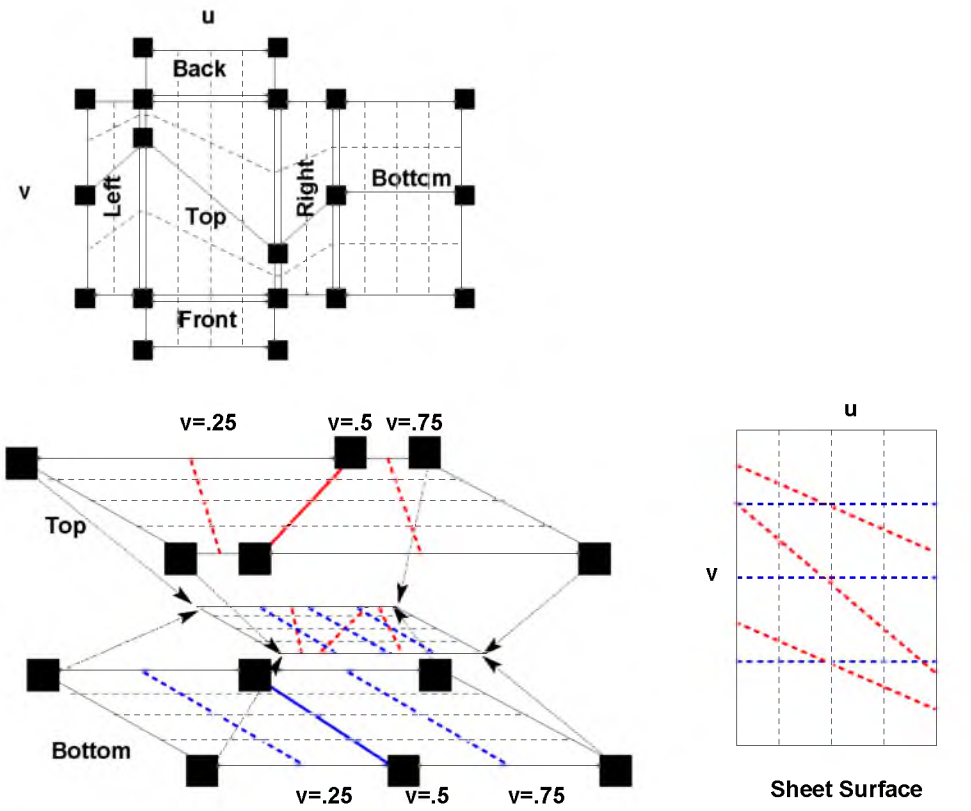

Figure 6.4: A parameterization mismatch resulting from contracting the boundary onto a medial surface. Shown are the six boundary surfaces of a box. The top and bottom surfaces represent identical geometries. However, the control points on the top are not uniformly spaced, resulting in isoparametric lines that are not parallel. Hence, the parameterizations do not match when projected on the medial axis sheet.

1. It is easier to control and less noise-sensitive than the medial axis.

2. Matching parameterizations across the center is greatly simplified.

3. Discontinuities are "hidden" in a narrow seam.

4. Techniques we develop will generalize to higher dimensions.

5. It possesses overall simplicity of implementation.

Our technique comprises two parts: 1) the extraction of a skeleton and 2) the derivation of a driver function that maps the surface boundary onto the skeleton in a continuous manner. As in the previous chapter, we will compute the skeleton and the driver function with respect to the control

polygon/mesh that determine the B-spline boundary. This allows us to focus on the simpler polygonal/polyhedral problem. Because of the convex hull and variation diminishing properties of splines, the skeletal projection of the boundary we prescribe will indicate a corresponding projection of the smooth boundary. Some care has to be taken to get this projection right, and the details follow below. The control polygon/mesh converges very quickly to the underlying smooth representation, allowing us to compute skeletons of arbitrarily high quality (albeit at a tradeoff in performance).

Early in our work, we discovered an elegant idea by Chuang *et al.* [62–64], that defines the skeleton as the 1D minimum of the potential due to a charged boundary. Appealing characteristics of this skeleton are that it can be made to converge to the medial axis in 2D (see Figure 6.5), and

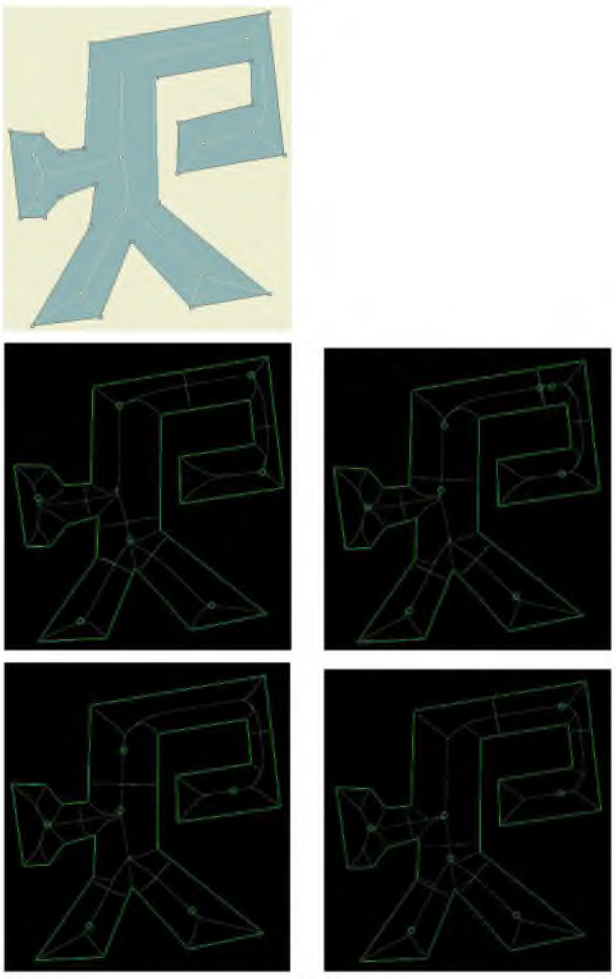

Figure 6.5: A planar figure with its medial axis, compared with the potential-based skeleton of degree 1, 2, 3 , and 9. Notice the progressive convergence to the medial axis shape.

in higher dimensions, the skeleton remains 1D (a curve) [64]. Another seemingly positive feature is that the potential function can also be used as the driver function for producing the family of offsets toward the skeleton. We will examine why that may not be advantageous in a later section.

A major shortcoming of the papers by Chuang *et al.* is an incomplete formulation of the algorithm for computing the potential-based skeleton. In this chapter, we shall develop methods for computing this skeleton.

The key contributions of this chapter are:

1. An improved method for calculating the skeleton of a potential field inscribed by a polygonal or polyhedral figure.

2. A method for computing the potential-based skeleton of a closed planar curve or closed freeform surface.

3. A method for parameterizing the interior of a closed polygonal figure in the plane.

4. A method for parameterizing the interior of a closed planar curve in the plane.

5. A method for parameterizing the interior of a closed polyhedral figure.

6. A method for parameterizing the interior of a closed freeform surface.

6.3 Potential-Based Skeletonization

The key idea of the potential-based skeleton is to imbue the boundary surface with a charge. Intuitively, each boundary point has an impact everywhere in space. Hence, the potential function is C^{∞}. Now, consider a particle with opposite charge dropped inside the boundary. The force acting on this particle is given by the gradient. The gradient to the potential function determines a vector field which is everywhere continuous – and particles advected by the field never cross. In fact, lines of force only meet where the force magnitude is 0. These points are the sinks and saddles of the system, and the (locally minimal) ridgelines that connect them form a skeleton for the figure. This skeleton is generally 1D (however, it can be higher dimensional in instances of exceptional symmetry – consider a sphere within a sphere). We formalize this idea below.

6.3.1 Potential Field Formulation

The potential at a point P is given by

$$\phi(P) = \int_S \frac{dS}{R^m}, m \geq 2 \tag{6.1}$$

where R is the distance from P to the surface S. The potential is divergent on S, and falls off with increasing R. The force at P is then given by the gradient

$$\nabla\phi(P) = \nabla \int_S \frac{dS}{R^m}, m \geq 2$$

In the plane, there exists a closed form for a polygonal boundary, and we have provided the derivation in the Appendix. In R^3, the situation is more complex. However, there exists a closed form for this computation when the boundary is polyhedral, and the degree of potential falloff, m, is 3. This result is derived in Chuang *et al.* [63], so we merely restate it here, with substantially more detail to ease implementation. The gradient (force) due to the charged boundary is given by

$$\begin{aligned}
\nabla\phi(P)_3 &= \nabla \int_S \frac{dS}{R^3} \\
&= \sum_{S_i} \left(\sum_{E_{ij}} \left(\nabla\phi_x^{E_{i,j}}(v_{i,j}^l) * l_{i,j} + \nabla\phi_y^{E_{i,j}}(v_{i,j}^l) * u_{i,j} + \nabla\phi_z^{E_{i,j}}(v_{i,j}^l) * n_i \Big|_{l=l_0}^{l=l_1} \right) \right. \\
&\quad \left. -\alpha/v_z^2 * n_i \right)
\end{aligned} \tag{6.2}$$

where S_i is one of the polygonal facets forming the boundary, $E_{i,j}$ is the jth line segment forming the ith facet, $l_{i,j}$ is the vector in the direction of $E_{i,j}$ in the plane of S_i, $u_{i,j}$ is the vector perpendicular to $l_{i,j}$ in S_i, n_i is the normal to S_i, $v_{i,j}$ is the vector from the $E_{i,j}$ to P reparameterized into the coordinate system $\{l_{i,j}, u_{i,j}, n_i\}$, l_0 and l_1 are similarly the endpoints of $E_{ij,}$ in this local system (see Figure 6.6), and

$$\nabla\phi_x^{E_{i,j}}(v) = \begin{cases} 0 & \text{if } v_y = 0 \\ \frac{v_y}{(v_x{}^2+v_y{}^2)*\sqrt{v_x{}^2+v_y{}^2+v_z{}^2}} & \text{otherwise.} \end{cases}$$

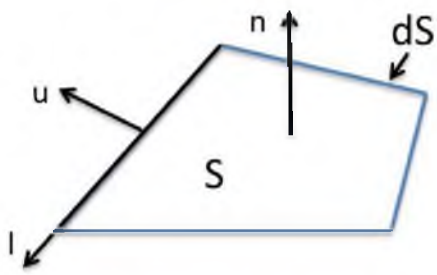

Figure 6.6: Configuration for computing the potential due to a polygonal facet.

$$\nabla\phi_y^{E_{i,j}}(v) = \begin{cases} 0 & \text{if } v_x = 0 \\ \frac{-v_x*(v_x^2+2*v_y^2+v_z^2)}{\sqrt{v_x^2+v_y^2+v_z^2}*(v_x^2+v_y^2)*(v_y^2+v_z^2)} & \text{otherwise.} \end{cases}$$

$$\nabla\phi_z^{E_{i,j}}(v) = \begin{cases} 0 & \text{if } v_x = 0 \\ 0 & \text{if } v_y = 0 \\ 0 & \text{if } v_z = 0 \\ \frac{1}{v_z^2} * \frac{v_x*v_y*v_z}{(v_y^2+v_z^2)*\sqrt{v_x^2+v_y^2+v_z^2}} - \arctan\left(\frac{v_x*v_y}{v_y*\sqrt{v_x^2+v_y^2+v_z^2}}\right) & \text{otherwise.} \end{cases}$$

$$\alpha = \begin{cases} 0 & \text{Projection of } P \text{ onto } S_i \text{ is outside} \\ \theta & \text{Projection of } P \text{ onto } S_i \text{ falls on a vertex whose edges form angle } \theta \\ \pi & \text{Projection of } P \text{ onto } S_i \text{ falls on an edge} \\ 2\pi & \text{Projection of } P \text{ onto } S_i \text{ falls on the interior} \end{cases}$$

6.3.2 Properties of the Potential Skeleton

1. Except in situations of high symmetry, it is 1D (Figure 6.3).

2. It converges to the medial axis in 2D and something like a curve approximation to the medial surface in 3D. See Figure 6.5.

3. It requires no topology – as indicated in the Equation (6.2), the potential field computation and its gradient consider the facets independently.

4. It does not require closed figures – again, this is a by-product of considering the facets

independently.

5. It is C^∞ except at the sources, sinks, and saddles. This is clear from the formulation of Equation (6.1).

6. As a corollary, the potential field provides nonintersecting trajectories for our swept offsets. For example, see Figures 6.7, 6.8, 6.9.

7. It is relatively noise-insensitive. The degree m of the generalized Newtonian potential gives a knob for trading noise-insensitivity against closeness to the medial axis in 2D.

8. It has an inner and outer formulation (see Figure 6.10). This is a very interesting feature of the potential-based skeleton. Particles can be advected inward toward the skeleton, but they can equally well be advected outward. These paths are also guaranteed not to cross, so they can be used to generate outer offsets as well.

9. It works with contours/holes – again without tracking topology. See for example Figure 6.11.

10. It lends itself to importance (Figure 6.12).

11. It is appropriate for path planning. Because paths cannot cross, and because a robot navigating the scene will be repelled from the boundary by the potential field, it makes an excellent candidate for path planning (see [63] for more details).

12. It generalizes to higher dimensions. For an $n-1$ manifold embedded in R^n, the axis

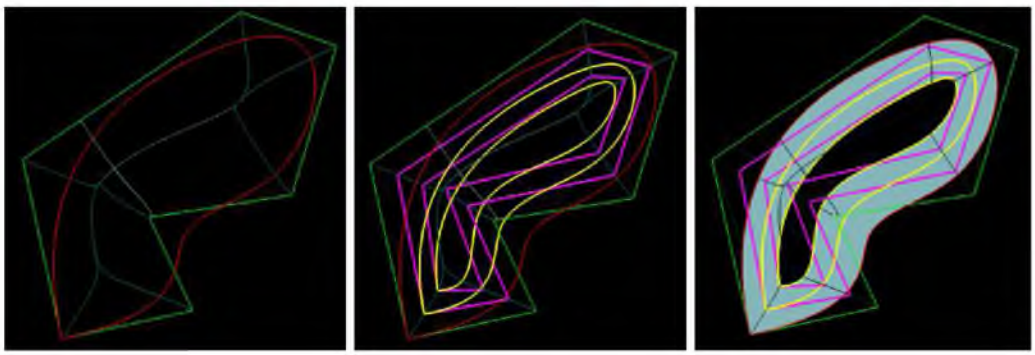

Figure 6.7: Uniform speed offsets generated inward from the boundary, and the swept surface that results from their blend.

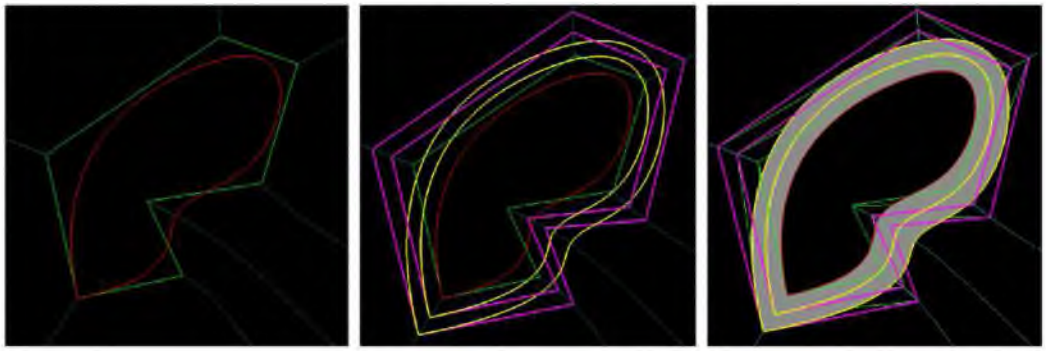

Figure 6.8: Uniform speed offsets generated outward from the boundary, and their swept surface.

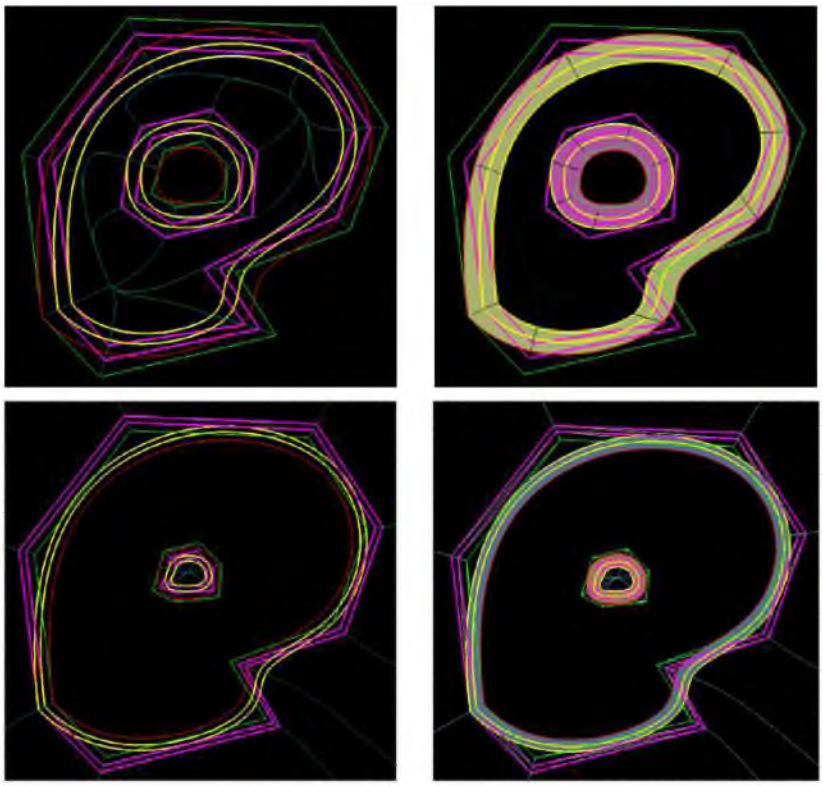

Figure 6.9: Sweeps are trivially generated from contours as well.

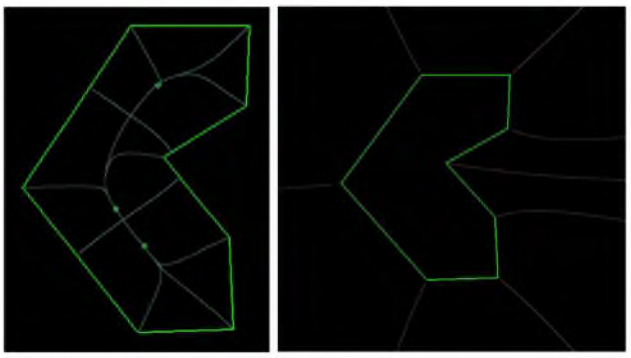

Figure 6.10: Inner and outer force lines generated by a potential field.

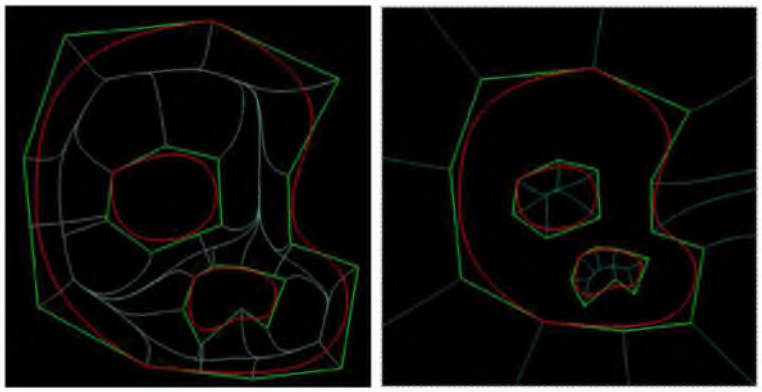

Figure 6.11: A potential-based approach requires no tweaking to handle contours. Shown left and right are the same curves with inside and outside reversed.

remains 1D.

6.3.3 Improving Evaluation Speed

One of the disadvantages of the potential-based approach is its speed of evaluation. The time to evaluate the potential or force at a point in space is linear with respect to the number of facets. Hence, evaluation becomes extremely tedious for larger models. We have applied Algorithm 1 to improve the performance of our technique. At run time, when an evaluation is requested, a search that is average-case logarithmic in the number of triangles (tetrahedra) is applied to find the triangle (tetrahedron) to evaluate, and then a constant time operation is performed to compute the barycentric interpolant of the vertex forces.

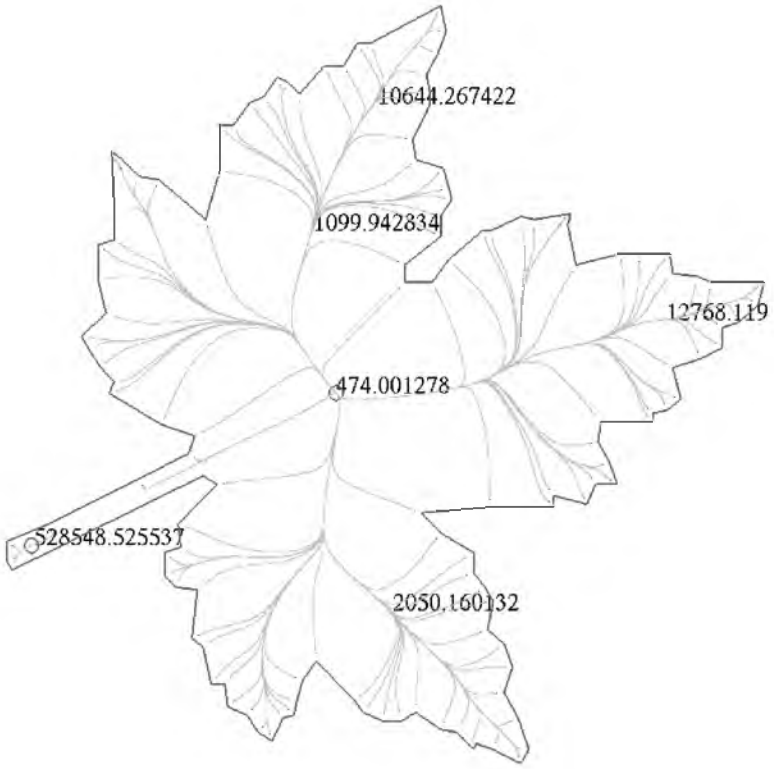

Figure 6.12: Because the gradient to the potential field becomes roughly equivalent to the medial axis at higher degrees, we can use the potential (or the nth root of the potential) to determine feature size in much the same way as the medial axis. Note how the magnitude of the potential function evaluated on the skeleton can be used to gauge the size of the corresponding feature (higher values correspond to smaller features).

Algorithm 1 A technique for improving the evaluation of the potential function.

1: Triangulate (2D) or tetrahedralize (3D) the interior of the figure.
2: For each triangle (tetrahedron), compute the force at the vertices.
3: Evaluate the quality of the approximation given by the barycentric interpolation of the force versus direct evaluation. If the quality is too low, subdivide using a 4-1 (12-1) subdivision. And repeat the quality evaluation with the children.
4: If a triangle's area (tetrahedron's volume) is less than a user-specified tolerance, mark the triangle (tetrahedron) as needing direct evaluation and return.

6.3.4 Determination of the Skeleton

The sinks and saddles of the system, and the (locally minimal) ridgelines that connect them form a skeleton for the figure. The basic algorithm for determining the skeleton is given by Algorithm 2.

Algorithm 2 High-level algorithm for computing the potential-based skeleton.

1: Find all the sinks by tracing the boundary points along the lines of force.
2: Locate the saddles by using the Poincaré index theorem to isolate the regions where they reside.
3: Trace the saddle to sink paths to compute the skeleton.
4: Does this skeleton (based on the control mesh) fall outside the actual surface? If so, refine uniformly and repeat.

6.3.4.1 Determination of the sinks

As we mentioned previously, the sinks are discovered by walking from the boundary in the direction opposite to the gradient to the potential field until we encounter a zero force (gradient). Trajectories that have a common endpoint improve our estimate of the sinks.

6.3.4.2 Locating the saddles

Saddles are traditionally much more difficult to isolate because vector fields diverge from these points except along separatrices. However, we are fortunate in that we have already isolated the sources at the boundary and the sinks in the previous step. Hence, we can reliably use the Poincaré Index to isolate these remaining critical points. In the plane, the index theorem says that given any continuous vector field, and any closed curve in that field, the vectors measured on the boundary of that curve sweep out angles that are some integer multiple of 2π. The number of sweeps k is given by

$$k = n_+ + n_- - n_x$$

where n_+, n_-, and n_x are the number of sources, sinks, and saddles enclosed by the curve. The danger in applying this formula is that critical points can cancel each other, thereby hiding their presence. However, since we know the location of the sinks (and there are no sources on the interior), we can account for their impact when necessary, and reliably locate saddles.

The same basic intuition applies to 3D vector fields. However, here rather than covering the Gaussian circle an integral number of times, the vector directions over a closed surface cover the Gaussian sphere. Mann and Rockwood showed how to use geometric algebra to compute signed areas on the Gaussian sphere [128]. The GAIGEN package can be used to generate an algebra for tracking the summation of these signed areas [129]. An equally effective, albeit slower (but more accessible) approach is to tessellate the Gaussian sphere and accumulate the vector coverage in bins. We have employed both approaches with equal success.

The final piece of machinery needed is a method for decomposing the interior of our model into closed curves/surfaces. In the plane, we utilize the software called *triangulate* from UNC-Chapel Hill [130]. For freeform surfaces, we make use of TetGen [131]. Once our figure has been decomposed into closed triangles/tetrahedra, we sample the boundary of these regions and accumulate the angles to determine the index of that region. Regions with negative index (after subtracting out the number of sinks) are subdivided using a 4-1 (triangle) or 12-1 (tetrahedron) subdivision. We stop when the area or volume of the enclosed region is less than a given tolerance, and designate the centroid as the approximate saddle location.

6.3.4.3 Tracing the saddle-sink paths

The output of the last step was a set of triangles or tetrahedra containing saddles. The index of each tells us what kind of saddle this region contains. Most commonly we will have 1-saddles, which connect two sinks. In situations of extraordinary symmetry (as may occur for products of computer-aided design), we may encounter n-saddles, which connect $n + 1$ sinks. As almost all vectors flow away from a saddle, it is a simple thing to perform a search for outbound directions on the enclosing triangle or tetrahedron to discover the unique saddle-sink paths. The diameter of the enclosing triangle/tetrahedron gives the radius at which it is safe to perform this search. Figure 6.13 shows this algorithm applied to a planar curve, and Figure 6.14 does the same for surfaces.

6.3.5 Saddle-sink Connectivity Graph

Sinks are connected to one another through saddles. Hence, it is possible to gain a sense of the connectivity of a figure by computing the graph of its saddle-sink connections. If the nodes are sinks and the arrows are saddles, then this graph decomposes a model into star-shaped regions. We can further make use of this graph to improve the shape of the potential-based skeleton. Let us define leaf-sinks as sinks that are connected to only one saddle point. Leaf sinks can exhibit the poor behavior shown in the left of Figure 6.15, where force lines are pushed very close together but do not actually meet in a sink until much farther along. This behavior does not produce a desirable mapping. We can detect this sort of behavior and improve the mapping by expanding our skeleton to the point where the trajectories approach within epsilon. We have implemented this improvement, and demonstrate it in the center and right panels of Figure 6.15.

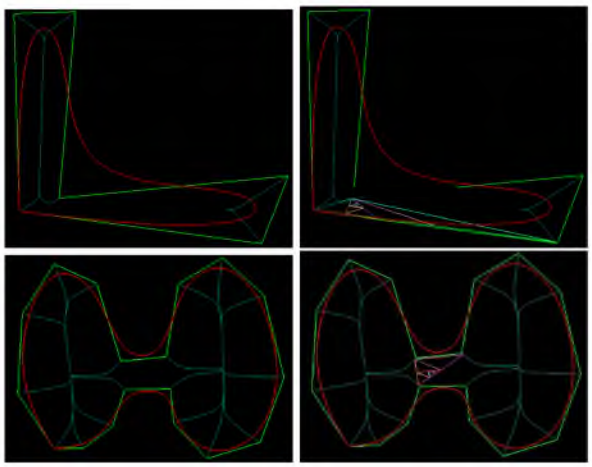

Figure 6.13: 2D figures illustrating how the index theorem is used to isolate saddle points of the skeleton (see Section 6.3.4.2). The saddle in the middle of the 'L' is quickly isolated using subdivision. The triangles evaluated to achieve isolation are shown.

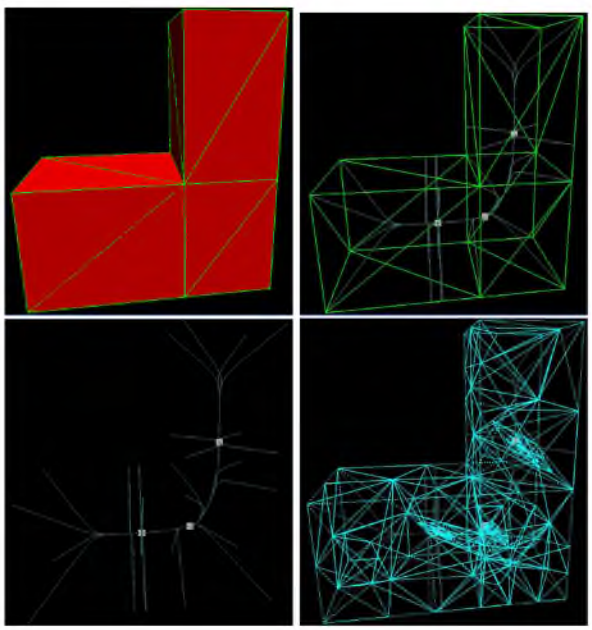

Figure 6.14: A visualization of the tetrahedra generated to isolate the saddles of a 3D potential field. This 3D 'L' has three saddle points. The three dense clusters in the figure on the right visualize the tetrahedra that are evaluated as shown in Section 6.3.4.2.

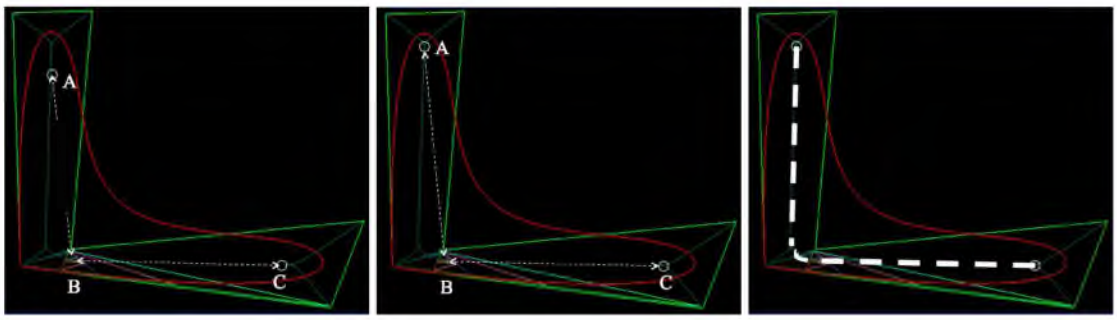

Figure 6.15: Improvement of the concentric axis at graph endpoints to avoid 'pinching.'

6.3.6 Application to Curves/Surfaces

The curves and surfaces we consider in this dissertation all have two important properties: 1) convex hull and 2) convergence under refinement. In the case of B-spline curves and surfaces, uniform refinement of the control polygon (mesh) results in a sequence of refined control polygons (meshes) that converge quadratically to the underlying curve (surface) [132]. This property means that relatively little refinement of the initial control polygon (mesh) is needed to produce an excellent approximation to the potential-based skeleton (see Figure 6.16). The technique for computing the skeleton of areas/volumes bounded by B-spline curves/surfaces is given by Algorithm 3.

6.4 Surface Completion of Planar Curves

Before proceeding, we briefly note that the technique of Chapter 5 applies directly if the medial axis is traded for the potential-based axis. Hence, we will not revisit the planar case here. Please see Figure 6.17 for a summary.

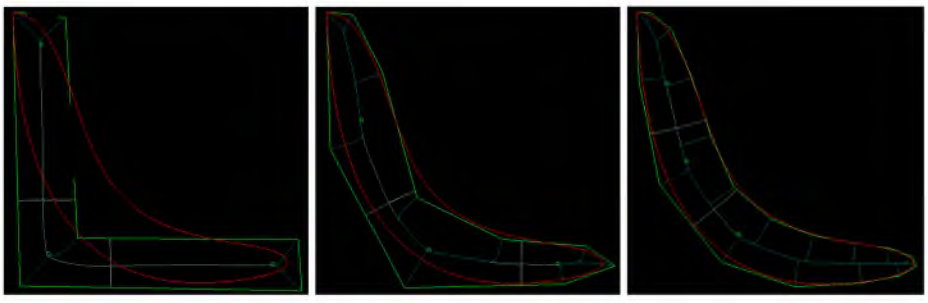

Figure 6.16: Because control polygons converge rapidly to the shape of the curve/surface they describe, so too does the potential skeleton. In the example shown, only one level of subdivision is required to obtain a reasonable skeleton for the continuous curve.

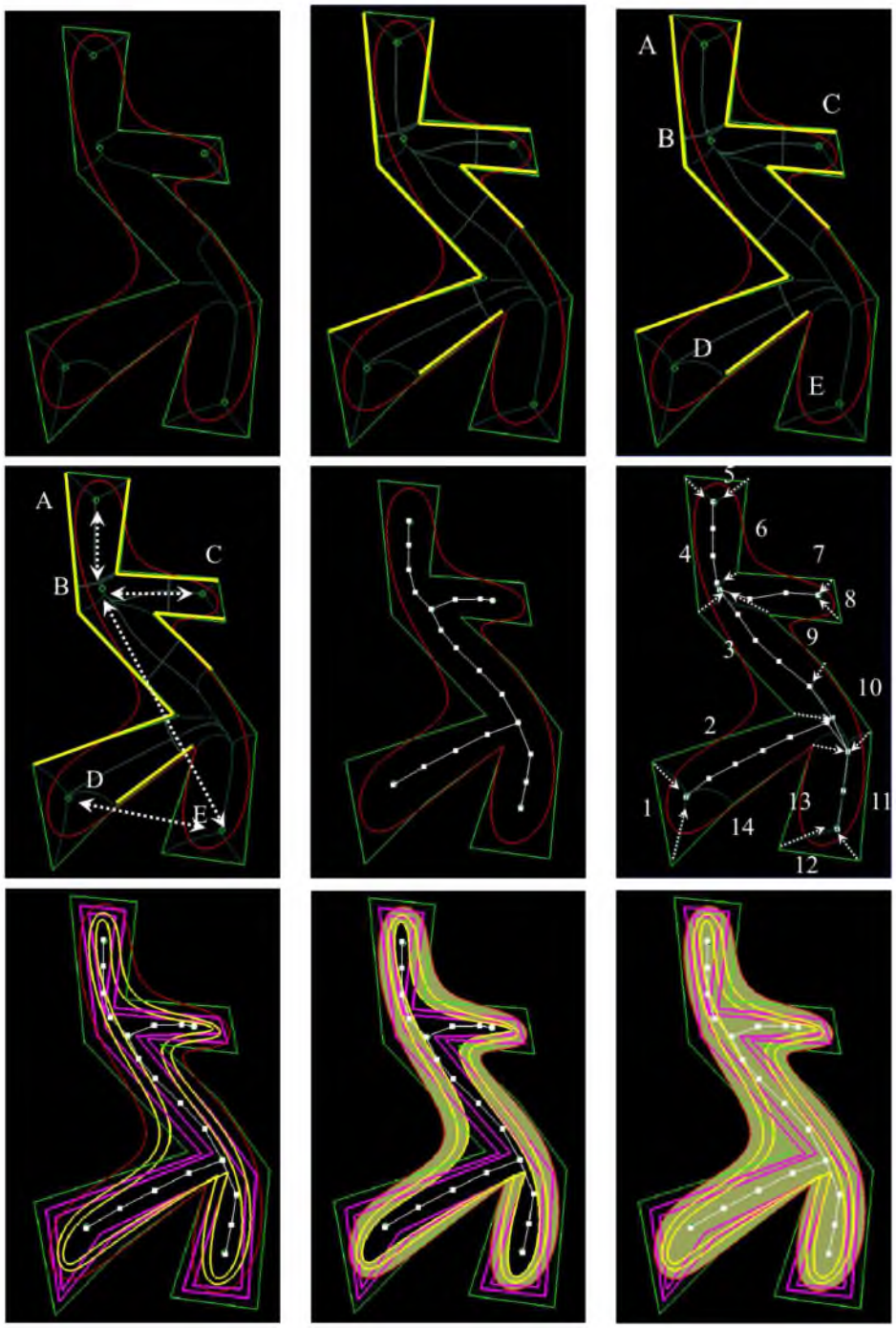

Figure 6.17: Every point on the boundary has a corresponding mapping on the axis. The highlighted boundary segments span saddles. The sinks can be thought of as nodes of the graph, and they are connected through saddles, which can be thought of as graph edges. This graph can be used to reconstruct the skeleton as well as to determine the mapping regions. Once the mapping has been determined, offsets can be generated or the surface can be completed. See Chapter 5 for more details.

Algorithm 3 Method for calculating the potential-based skeleton of a B-spline curve/surface.

1: Apply the method of Section 6.3.4 to the control mesh to produce a coarse approximation to the skeleton.
2: Use uniform subdivision to refine the boundary.
3: Compute the refined skeleton.
4: Compute the difference between the skeletons.
5: If the difference is too large, repeat from 2.

6.5 Surface Completion of Space Curves

Consider a space curve embedded in R^3. If there exists a continuous homeomorphic deformation of the curve onto the plane, then we can project the curve onto the plane, complete its interior, and then reverse the projection to obtain the surface. Because this represents a trivial extension of the our planar technique, we will not dwell on it here. Obviously, the difficulty lies in deriving an appropriate projection operator, and that is beyond the scope of this work.

6.6 Decoupling the Skeletonization Operator from the Driver

Our technique for completing a manifold from its boundary requires two things: a skeleton and a function that propels the boundary onto the skeleton. Again, our intuition is summarized in Figure 6.1. Because the boundary may require refinement in order to improve the shape of the offsets and the resulting parameterization, it is appropriate that the skeleton be decoupled from the driving function. Furthermore, in the case of the potential function, the computation scales in the number of control points – so recomputing the skeleton each time may result in very long recomputation. Our optimization of Section 6.3.3 alleviates this somewhat. However, it is still inconvenient to recompute the skeleton with each boundary refinement.

Instead of a forward technique, where the goal destination is implicit, a better approach rephrases the problem as a boundary value problem as in [78]. This allows the source (boundary) and destination (skeleton) functions to be adjusted independently. Hereafter, we assume that the skeleton of the boundary has been computed using one of various techniques [63, 64, 66, 133], where we have given one candidate above. In the next section, we will introduce our technique for determining a driver.

6.7 Harmonic Analysis

Rephrasing the application as a boundary value problem allows us to specify the source and target functions for the driver independently. Hence, boundary refinements will not result in

motion of the skeleton. We take the approach of Martin *et al.* [78, 79], employing the Finite Element Method with discrete harmonic functions satisfying Laplace's equation:

$$\nabla^2 u = 0$$

The idea is to tetrahedralize the interior of the boundary representation, assign functional values of $u = -1$ and $u = 1$ to vertices on the boundary and skeleton, respectively, and solve for the values of u at the internal vertices. u is then represented as a trilinear interpolant of its vertices:

$$u(\mathbf{x}) = \sum_{i=0}^{n} u_i B_i(\mathbf{x})$$

where

$$B_i(\mathbf{x}) = \begin{cases} 0 & \mathbf{x} \text{ outside tet defined by } \mathbf{P_i} \\ 1 & \mathbf{x} \equiv \mathbf{P_i} \\ c \in (0,1) & \text{otherwise.} \end{cases} \tag{6.3}$$

To be more specific, if $\mathbf{x}$ falls within a tetrahedron defined by $\mathbf{P_0}, \mathbf{P_1}, \mathbf{P_2}, \mathbf{P_3}$, then the basis functions $\mathbf{B}$ have values determined by

$$\mathbf{B} * \begin{pmatrix} \mathbf{P_0} & \mathbf{P_1} & \mathbf{P_2} & \mathbf{P_3} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

and

$$\mathbf{B} = \begin{pmatrix} \mathbf{P_0} & \mathbf{P_1} & \mathbf{P_2} & \mathbf{P_3} \end{pmatrix}^{-1} \tag{6.4}$$

and then $\mathbf{c}$ from Equation (6.3) is given by:

$$\mathbf{c} = \mathbf{B}\mathbf{x}$$

The gradient of $\mathbf{c}$ is then

$$\nabla \mathbf{c} = \mathbf{B} \nabla \mathbf{x} = \mathbf{B} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Returning to the Finite Element Method, recall that our function u has the property

$$\nabla^2 u = 0$$

The weak formulation states that for any smooth functions v where $v \equiv 0$ on the boundary of the domain $d\Omega$ where

$$\nabla^2 u = f$$

that the following relationship holds

$$\int_\Omega \nabla^2 uv ds = \int_\Omega f v ds$$

Using Green's Theorem, the left-hand side can be rewritten as

$$\begin{aligned} \int_\Omega \nabla^2 uv ds &= \nabla uv|_{d\Omega} - \int_\Omega \nabla u \cdot \nabla v ds \\ &= -\int_\Omega \nabla u \cdot \nabla v ds \end{aligned}$$

This last simplification is made possible because $v \equiv 0$ on the boundary. Hence,

$$-\int_\Omega \nabla u \cdot \nabla v ds = \int_\Omega f v ds$$

In our case, we have $f \equiv 0$, hence,

$$\begin{aligned} 0 &= \int_\Omega \nabla u \cdot \nabla v ds \\ &= \int_\Omega \nabla \sum_{i=0}^{n} u_i B_i(\mathbf{x}) \cdot \nabla v ds \\ &= \sum_{i=0}^{n} u_i \int_\Omega \nabla B_i(\mathbf{x}) \cdot \nabla v ds \end{aligned}$$

The functions v are termed "test functions," and they are what provide sufficient constraints to make the Finite Element Method soluble. We note that our basis functions B_j at the tetrahedral vertices satisfy the requirements of a test function, so long as they do not touch the boundary. Hence, for each B_j corresponding to the $\mathbf{P_j} \in \mathbf{V_I}$, the internal set of m vertices for our tetrahedralization, we have the equations

$$\begin{aligned} 0 &= \sum_{i=0}^{n} u_i \int_\Omega \nabla B_i(\mathbf{x}) \cdot \nabla v ds \\ &= \sum_{i=0}^{n} u_i \int_\Omega \nabla B_i(\mathbf{x}) \cdot \nabla B_j(\mathbf{x}) ds, \mathbf{P_j} \in \mathbf{V_I} \end{aligned}$$

Let $\mathcal{T}(i,j)$ be the set of tetrahedra that include vertices $\mathbf{P_i}$ and $\mathbf{P_j}$. Then, for a particular test function B_j, we have

$$0 = \sum_{i=0}^{n} u_i \sum_{T_k \in \mathcal{T}(i,j)} \nabla B_i^k \cdot \nabla B_j^k V(T_k), \mathbf{P_j} \in \mathbf{V_I} \tag{6.5}$$

where $V(T_k)$ is the volume of the tetrahedron T_k. Equation (6.5) expresses a subtle point – whereas the gradients ∇B_i are piecewise constant, they will be different for each tetrahedron the vertex P_i is involved in.

Let us define the sets $\mathbf{P_i} \in \mathbf{V_B}$ and $\mathbf{P_i} \in \mathbf{V_S}$ as the set of vertices on the boundary and skeleton, respectively. We know the values of the u_i at these points. Hence, we can split

Equation (6.5) into knowns and unknowns

$$
\begin{aligned}
0 &= \sum_{P_i \in V_B} (-1) \sum_{T_k \in \mathcal{T}(i,j)} \nabla B_i^k \cdot \nabla B_j^k V(T_k) \\
&\quad + \sum_{P_i \in V_S} (1) \sum_{T_k \in \mathcal{T}(i,j)} \nabla B_i^k \cdot \nabla B_j^k V(T_k) \\
&\quad + \sum_{P_i \in V_I} u_i \sum_{T_k \in \mathcal{T}(i,j)} \nabla B_i^k \cdot \nabla B_j^k V(T_k) \\
\sum_{P_i \in V_B} \sum_{T_k \in \mathcal{T}(i,j)} \nabla B_i^k \cdot \nabla B_j^k V(T_k) & \\
- \sum_{P_i \in V_S} \sum_{T_k \in \mathcal{T}(i,j)} \nabla B_i^k \cdot \nabla B_j^k V(T_k) &= \sum_{P_i \in V_I} u_i \sum_{T_k \in \mathcal{T}(i,j)} \nabla B_i^k \cdot \nabla B_j^k V(T_k) \qquad (6.6)
\end{aligned}
$$

Because we have a formulation like Equation (6.6) for each of the m internal vertices corresponding to the m test functions B_j, we have m equations and m unknowns, and can solve for the internal u_i:

$$b = Au$$

The procedure is summarized in Algorithm 4.

Algorithm 4 Solve Laplace's Equation.

```
for all tets tet_i do
    for all vertices v_j in tet_i do
        tet_i.v_j.coeff = coefficient from Equation (6.4)
    end for
end for
Initialize a sparse matrix A of dimension n to 0
Initialize b to 0
for all tets tet_i do
    for all edges (j,k) in tet_i do
        s = Dot( grad(tet_i.v_j), grad(tet_i.v_k) ) * Volume(tet_i)
        A(j,k) += s
        A(k,j) += s
    end for
end for
Apply boundary constraints, Dot( A_i,x ) = +/- 1 for boundary vertices
Solve A * x = b
```

Once we have have computed the driver function u, we can compute the mapping of the boundary vertices onto the skeleton using any number of streamline advection algorithms. In our work, we have used the streamline functionality of VTK [134]. In Figure 6.18, we show a simple 3D figure with these streamlines, the computed skeleton, and a sequence of offsets toward that skeleton.

6.8 Volumetric Completion of a B-Spline Boundary Representation

Our volume completion algorithm has two prerequisites: the computation of a skeleton and the definition of a mapping that moves the boundary onto the skeleton. In previous sections, we have provided examples of how to satisfy these prerequisites, although many other solutions are possible. In this section, we assume a piecewise-linear skeleton has been provided and that nonintersecting paths from the boundary vertices onto this skeleton can be readily computed.

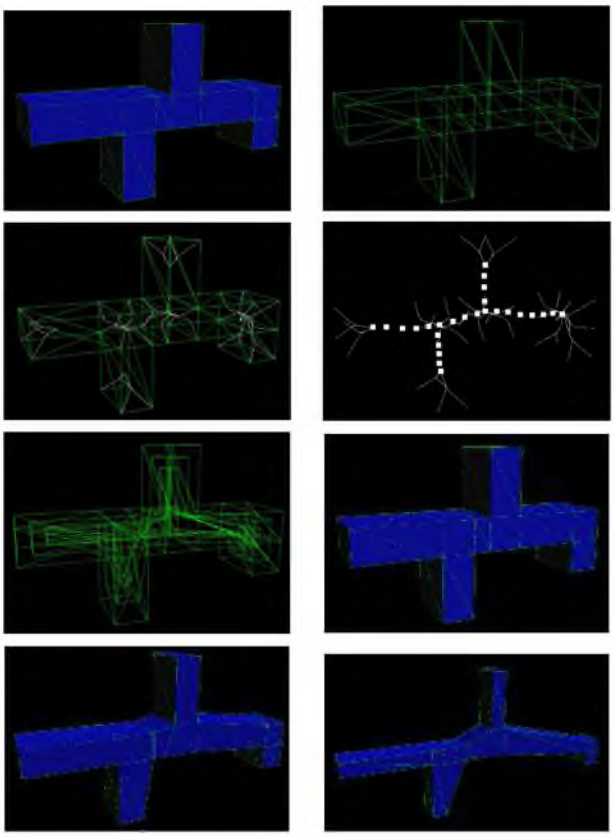

Figure 6.18: A simple 3D shape, shown with its potential field, its extracted 1D potential skeleton, and the sequence of blended polygons that parameterized its interior.

In the case of a B-spline boundary, simply pushing the control points onto the skeleton does not guarantee a closed figure – the geometry of the mapped surfaces is not guaranteed to match the shape of the skeleton. We will introduce some requirements to guarantee that the surfaces meet in exactly the 1D figure that is the skeleton:

1. We will require that the surfaces be linear on the axis. This will have the result that patches will map to line segments.

2. We will require that patches cannot have any internal skeleton points. This means that surfaces will interpolate the shape of the skeleton.

The left side of Figure 6.19 illustrates for a simple planar configuration why smooth midstructures can cause difficulties. Patches (or in this case curves) whose control vertices abut at the axis may still trace out different shapes. For this reason, we require that the midstructure be piecewise linear. This does not restrict the offset surfaces to be linear however – we will degree-raise the midstructure so that we can blend it with the higher order offsets. The right side of Figure 6.19 illustrates a difficulty with allowing patches to span skeleton segments. Even though both boundary patches (in this case, curves) are piecewise linear, their mappings do not abut because Curve A spans a skeleton vertex. We will ensure that patches do not map across a vertex in the midstructure by applying subdivision. The implication is that patches will map onto the midstructure as either "pyramids" or "wedges," as shown in Figure 6.20 for the 2D and 3D case.

However, the refinements required to achieve this configuration could potentially introduce many control points into the offset sequence, depending on the number of skeleton points, and also on how closely the contraction paths approach the skeleton points. To avoid the possible explosion of control points, we pursue a two-part technique:

1. Generate a sequence of offsets from the original boundary that come within an ϵ tube of the central skeleton – we know we can do this because of the continuous nature of the driving function – however, we may need to refine the boundary to prevent intersection of the offset layers and intersection with the central skeleton. We can detect these intersections by looking for singularities in the Jacobian as prescribed in [58].

2. Once we have this family, we generate a separate volume that spans the distance from the innermost offset and the skeleton. The skeleton will be represented as a linear spline surface that is made compatible with the innermost offset in terms of its spline space.

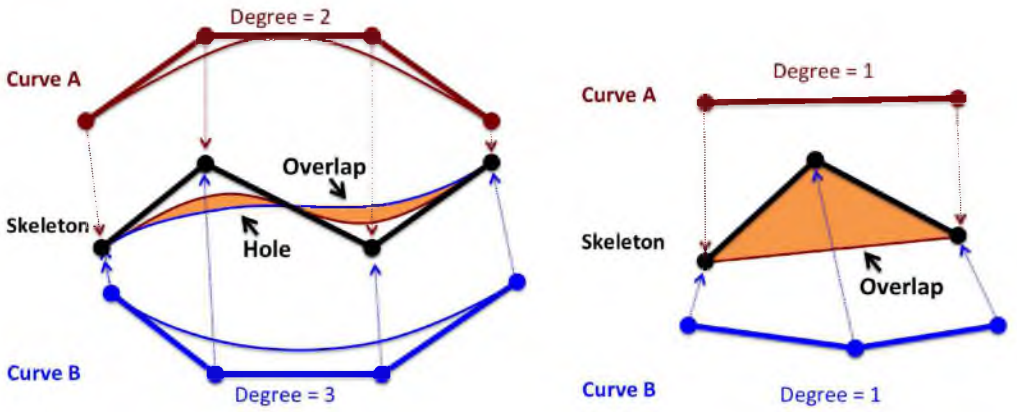

Figure 6.19: Simple configurations illustrating potential problems with mapping boundary representations onto a central axis. On the left, we demonstrate how mapping onto a curved midstructure can fail – simply moving the control vertices of the boundary to the axis does not ensure that the resulting curves trace out the same shape. This can result in gaps or overlaps in the completed surface, and the same is true for the volumetric case. On the right, we demonstrate that even linear boundaries can fail to match if inflection points in the skeleton are missed.

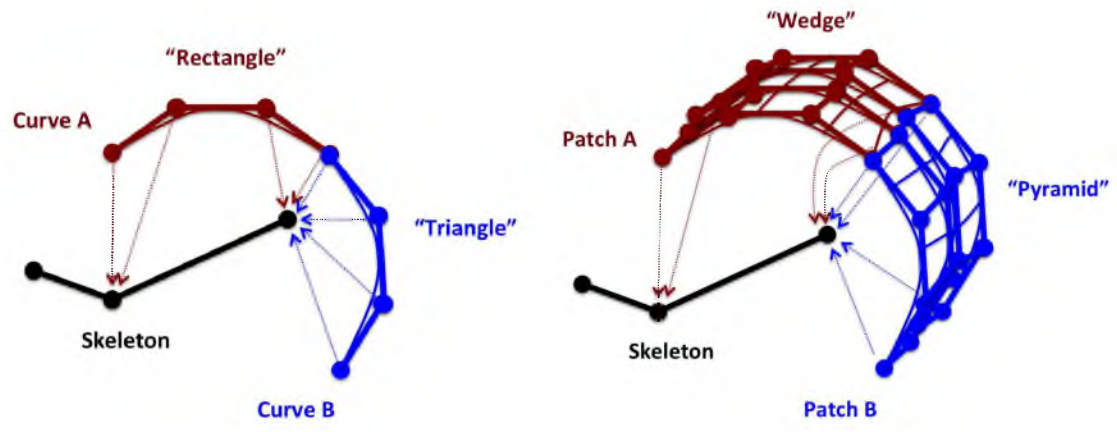

Figure 6.20: We require that patches map to either a point or a segment on the skeletal midstructure. In 2D, this results in either a "triangular" or "rectangular" completion surface, respectively. In 3D, surface patches will generate "pyramid" or "wedge" volume completions, respectively.

This high-level approach is summarized in Algorithm 5.

Algorithm 5 High-level summary of the surface/volume completion algorithm.

1: Compute the skeleton.
2: Compute the sequence of offsets.
3: Compute the sweep of the offsets to produce a nearly filled in figure.
4: Create a linear curve/surface that represents the shape of the axis.
5: Make the axial control mesh and the innermost offset curve compatible (in both the geometric and parametric sense – that is, embed them in the same spline space).
6: Perform a linear blend to fill the gap.

We begin by computing the parameterized family of offsets. The ideas are summarized in Algorithm 6.

Algorithm 6 General approach for producing the generalized cylinder offsets.

1: For each boundary patch, move the patch towards the skeleton.
2: Using the set of bounding boxes computed from a refined version of the patch, test for intersection with the axis. The techniques from Section 3.3.4 are readily amenable to this type of analysis.
3: If the patch intersects the axis, and the maximum distance of the patch to the axis is greater than the user-specified ϵ, then we need to refine the boundary, recompute the axis, and repeat from above.
4: Otherwise, we proceed to test the next patch.
5: We are guaranteed, under refinement, that we can achieve a sequence of offsets that approach the skeleton within ϵ. However, the surface becomes progressively more densely sampled, which is what we are hoping to avoid.

The detailed recipe for computing the family of offsets to a B-spline boundary is given in Algorithm 7.

Algorithm 7 Compute the family of offsets that terminate within epsilon of the axis.

```
repeat
  repeat
    for each patch do
      move the points along the lines of force until they are within epsilon of the axis
      {test for intersection with the axis}
      refine the patch to high resolution
      test each bbox against the axis
      optionally refine further and test again.
      if the patch is pierced by the axis then
        uniformly insert knots into a refined knot vector.
      end if
    end for
    if patches were found that penetrate the axis then
      refine using the refined knot vector
    end if
  until no patches penetrate the axis
  Compute the family of patches that fall between the boundary surface and the terminal
  offset
  if The sweep is singular – passes through itself, Apply Joy et al. to test this then
    refine problem areas
  end if
until swept volume is nonsingular
Compute clamped (i.e., nonfloating) sweep that starts at the boundary and terminates at the
innermost offset, with knot values {0....1 − t_ε}
```

Our final task is to produce a surface mesh that is geometrically identical to the skeleton and provides a correspondence between the control points of the innermost offset and the axis. We can then perform a simple sweep to produce the spanning volume. First, some definitions:

- Let $m(P_i, t)$ be the mapping that takes a control point P_i to the central axis c at $t = 1$.
- Let $s_u(i)$, $s_v(j)$ be the ith and jth Greville abscissa for the knot vectors τ_u and τ_v for a particular patch.
- Let $len_c(p, q)$ be the distance from p to q as measured along the central axis c.
- Let C_i, t_i^u and D_j, T_j be the control polygons and knot vectors for the axial curves obtained by mapping the rows/columns (respectively) of the innermost offset onto c. These curves will be blended to form the central (1D) surface to be swept with the innermost offset.

The approach for computing the spanning patch is given in Algorithm 8. Figure 6.21 demonstrates the approach.

Algorithm 8 Compute the linear span from the innermost offset to the skeleton.

1: **for** each inner offset surface **do** {build a surface from the potential axis}
2: **for** each row i in the offset surface's control mesh **do**
3: **for** each span $P_{i,j}$, $P_{i,j+1}$ **do**
4: add $m(P_{ij}, 1)$ to C_i
5: add $s_u(j)$ to t_i^u
6: apply Dijkstra's algorithm to compute the path from $m(P_{i,j}, 1)$ and $m(P_{i,j+1}, 1)$ on c
7: **for** each axis point a_k falling between $m(P_{i,j}, 1)$ and $m(P_{i,j+1}, 1)$ on c **do**
8: add a_k to C_i
9: add $(1 - len_c(m(P_{ij}, 1)), a_k)/len_c(m(P_{i,j}, 1), m(P_{i,j+1}, 1))) * s_u(j) + len_c(m(P_{i,j}, 1), a_k)/len_c(m(P_{i,j}, 1), m(P_{i,j+1}, 1)) * s_u(j + 1)$ to t_i^u
10: **end for**
11: **end for**
12: add $m(P_{in}, 1)$ to C_i
13: add $s_u(n)$ to t_i^u
14: **end for**
15: Make the spline spaces of the rows compatible via refinement, to form a unified knot vector t^u and control mesh C.
16: **for** each column j of C **do**
17: **for** each span C_{ij}, C_{i+1j} **do**
18: add C_{ij} to D_j
19: add $s_v(i)$ to T_j
20: apply Dijkstra's algorithm to compute the path from C_{ij} and C_{i+1j} on c
21: **for** each axis point a_k falling between C_{ij} and C_{i+1j} **do**
22: add a_k to D_j
23: add $(1-len_c(C_{ij}, a_k)/len_c(C_{ij}, C_{i+1j}))*s_v(i)+len_c(C_{ij}, a_k)/len_c(C_{ij}, C_{i+1j})* s_v(i + 1)$ to T_j
24: **end for**
25: **end for**
26: add P_{mj} to D_j
27: add $s_v(m)$ to T_j
28: **end for**
29: Make the spline spaces of the columns compatible via refinement, to form a unified knot vector T and control mesh D
30: Degree raise the medial surface to match the boundary.
31: Make the spline spaces of the terminal offset and the medial surface compatible via refinement
32: Compute the swept surface – the two w knots will be $1 - t_\epsilon$ and 1.
33: **end for**

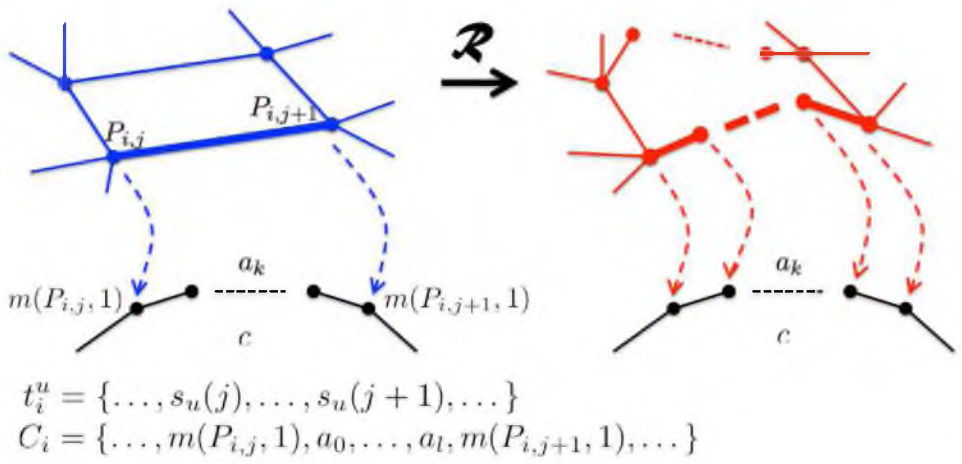

Figure 6.21: Demonstration of our algorithm for mapping the innermost offset onto the skeletal axis c. For each row / column of the innermost offset, we will compute a skeletal curve to which it maps. On the left, we consider the jth span of the ith row of the control polygon for a patch in the innermost offset. The function m is our driver function that maps the boundary onto the skeleton. Given the mapping of the jth span onto c, we apply Dijkstra's algorithm to determine the skeletal vertices a_k that fall between $m(P_{i,j})$ and $m(P_{i,j+1})$. These vertices are added to the control polygon C_i of the skeletal curve for row i. We assign knots corresponding to $m(P_{i,j})$ and $m(P_{i,j+1})$ using the j and $j+1$ nodal values of the innermost offset surface's knot vector, τ_u. We assign knots to the a_k using linear interpolation of the aforementioned nodal values based on the geodesic distance traveled along c, as detailed in Algorithm 8. On the right, we show the result of refining the innermost offset surface to make it compatible with the skeletal curve we built on the left. Postrefinement, we can guarantee that the mapping of the refined jth span onto the skeleton will contain none of the original skeletal vertices, except at the end points (recall, we wish to avoid the situation in Figure 6.19). By building a skeletal curve for every row of the innermost offset, making these compatible, and blending them into a surface, and then repeating for the columns, we ensure that for the resulting blend, no patches violate the mapping configurations indicated in Figure 6.20.

6.9 Conditions for Success

Theorem 1 *Given a closed surface B with parametric domain X comprised of NURBS patches B_i with adjacency information and compatible spline spaces across patch boundaries (that is the parameter spaces can be brought into accord along shared boundaries, and data structures are maintained to allow boundary traversal across patches, as is common in CAD systems); and provided that the control meshes form a closed surface, as will often be the case as a by-product of design or can often be ensured using refinement without losing exactness of the representation; and given a straight line skeleton S that is homotopy equivalent to B, our algorithm produces a homotopic mapping $H(x,0) = B$ and $H(x,1) = S$, for $x \in X$.*

Proof: We require that the control polygons form a closed surface boundary as this is needed for application of the Finite Element technique. The requirement can be relaxed somewhat if the potential field is used as the driving function, as mentioned in Section 6.3.2. We require that the patches have compatible parameterizations and possess cross-boundary traversal data structures because then, we can operate on shared rows and columns, thereby ensuring that no black holes or cross-patch discontinuities are introduced in mapping the patches to the skeleton. Many skeletonization algorithms produce boundary-homotopic skeletons – the potential-based skeleton that was discussed in Section 6.3 as well as that of [66] satisfy this requirement. Obviously, there are an infinite number of homotopy equivalent skeletons for a given boundary, and the quality of the parameterization will depend in part on how well the skeleton approaches the shape of the surface. Below, we will implicitly assume "skeleton" refers to a homotopy equivalent skeleton. Because the control mesh of a NURBS surface converges quadratically to the limit surface under refinement [132], we can derive a skeleton using the suitably refined control mesh as a surrogate.

We model the mapping of the boundary onto the skeleton as a boundary value problem using harmonic analysis. Because the resulting harmonic function is continuous and satisfies the "maximum principle," the mapping of the boundary onto the axis is also a homotopy [78]. Furthermore, it is shown in [78] that the discrete approximation to this function reconstructed using finite elements is also a homotopy provided care is taken in handling degeneracies. We refer the reader to that work for details.

Our algorithm has two complexities not addressed by the preceding statements. First, our boundary is not a continuous set of points being mapped onto the axis, but rather a discrete set of control vertices. Second, the boundary being moved to the axis is a surrogate for a piecewise polynomial surface, and merely mapping the boundary to the axis is not sufficient to ensure a closed, nonsingular figure. These considerations are demonstrated in Figure 6.19.

The algorithm comprises two parts: 1) Generating a family of offsets to within a user-specified ϵ of the axis and 2) computing a blend between this innermost offset and the axis. In 1), we are guaranteed that we can offset the boundary to within ϵ of the skeletal midstructure (without passing through it), because the boundary and skeleton can be made arbitrarily close to the continuous case by employing refinement. The intermediate offsets can be made to conform arbitrarily closely to the continuous function, again via refinement. Since the continuous case does not have degeneracies, we can avoid intersections in the offset surfaces. The algorithm for achieving this is straightforward and indicated in Algorithm 7. Hence, the volume swept from the boundary to the innermost offset will be continuous.

Now, we must ensure that the mapping from the innermost offset onto the skeleton will be continuous. Our approach is to subdivide the innermost offset so that the refined patches map to line segments on the axis, as shown in Figure 6.20. Given a patch, for every row interval and column interval in the control mesh, we consider its mapping onto the skeleton, as shown in Figure 6.21. We employ Dijkstra's algorithm to find the path between the two corresponding skeleton vertices. Because the control vertices were path connected in the mesh, and because the skeleton is homotopic to the mesh boundary, we know that there exists a path between these points on the skeleton. That is, homotopic mappings preserve path connectivity. As described in Algorithm 8, we add knots to the refinement vector of the innermost offset so that sufficient points will be inserted in the control interval to achieve a correspondence with the desired path on the skeleton. By doing this for every row and column interval in the control mesh for the innermost offset, and treating the skeleton as a linear spline, we can ensure that every refined patch on the innermost offset maps to a single linear span on the skeleton. Hence, the mapping path sweeps out a wedge or pyramid. Because of the convex hull and variation diminishing properties of B-splines, the resulting family of surfaces cannot pierce the patch of B that caps the wedge, nor the envelope of the mesh sweep up to the axis. Hence, the only place where we need to ensure that the family of swept surfaces behaves is at the internal parametric boundary, which is the skeleton axis.

Finally, because refinement produces a geometrically identical surface, the innermost blend of part 2) and the sweep formed from the offsets from part 1) will meet with C^0.

□

6.10 Results

We have applied the volumetric completion algorithm of Section 6.8 to both polyhedral and spline models. In Figure 6.22, we show one of these models, its skeleton, and a contraction onto the skeleton.

6.11 Parametric Stretch

While we are guaranteed a continuous mapping of the boundary onto the skeleton, given sufficient refinement, the quality of the parametrization can be a concern. Inherent to this particular generalized cylinder model is some distortion of the parametric domain in the vicinity of the axis. Square domains are mapped to either points or lines on the axis, resulting in pyramidal or wedge-shaped volumes. Our approach has been to isolate these regions away from the boundary where areas of interest often lie. However, parametric distortion can occur throughout the volume depending on characteristics of the driver function such as the "speed" at which it modifies the sweep. For simplicity, we have used approximately arc-length uniform offsets to generate intermediate offset surfaces between the boundary and the interior. One aspect worth noting is that the contraction speed can be modified to improve the parameterization. So long as control points start at the boundary and end at the innermost offset, and move exclusively forward, the contraction remains valid. Hence, the degree of stretch can be optimized by tuning the local speed of the contraction function, the refinement of the boundary, and the number of intermediate offsets. One measure of parametric stretch can be found by tracking the Jacobian of the volume $\mathbf{V}$:

$$\det \mathcal{J}\mathbf{V}(u,v,w) = \frac{d\mathbf{V}}{du} \cdot \left(\frac{d\mathbf{V}}{dv} \times \frac{d\mathbf{V}}{dw} \right)$$

In other words, how well preserved is the volume of the (u,v,w) parallelepiped when passed through $\mathbf{V}$? Other measures track the lengths of the sides of the parallelepiped, $||\frac{d\mathbf{V}}{du}||$, $||\frac{d\mathbf{V}}{dv}||$, and $||\frac{d\mathbf{V}}{dw}||$, or consider the angles between the partials, *e.g.*, $\arctan(||\frac{d\mathbf{V}}{du} \times \frac{d\mathbf{V}}{du}||, \frac{d\mathbf{V}}{du} \cdot \frac{d\mathbf{V}}{dv})$. Optimizing these metrics remains future work.

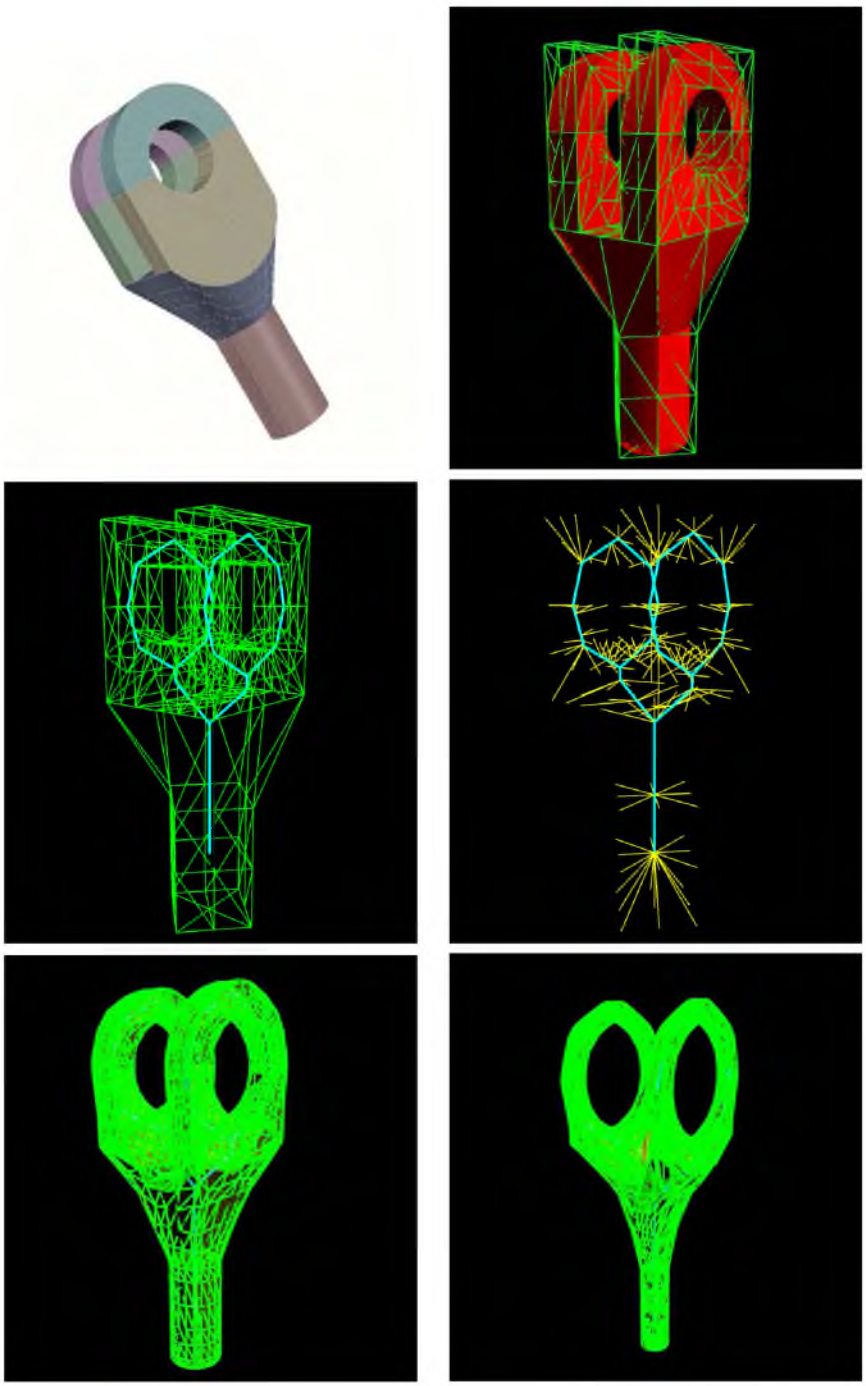

Figure 6.22: Our contraction-based technique, applied to a mechanical part. First, the skeleton of the part is computed, based on its control polygon. Depending on the quality desired, refinement may be used to develop a closer approximation to the surface before skeletonization. This first refinement is merely for the purpose of computing a skeleton, and may be discarded prior to the next step. Next, the contraction paths are computed from the driver function. Based on the user-selected epsilon, refinement may be required so that the innermost offset can more closely approximate the shape of the midstructure. This is an essential tradeoff of our method. (Model courtesy of David Johnson, University of Utah.)

6.12 Conclusions and Future Work

In this chapter, we have presented methods for

1. computing the skeleton of a boundary representation,
2. computing a driver function that maps the boundary onto the skeleton, and
3. computing the closure of a boundary representation.

We have assumed a B-rep comprised of NURBS in our development, although the methods apply more generally. Future work includes extending our development to T-splines [135], LR-splines [136], and hierarchical splines [137]. These representations would allow us greater control over our mapping onto the midstructure with reduced concern for the explosion in geometric complexity that accompanies tensor product formulations. In Chapter 7, we demonstrate how a hierarchical spline formulation can be applied to greatly reduce computational complexity and memory requirements when utilizing 4D splines in the solution of a finite element problem. Another area of future work is minimizing parametric distortion due to the mapping onto the midstructure. This can be accomplished by better refining the boundary in areas of high distortion, and adjusting the speed (*e.g.,* based on local curvature) by which the control vertices are mapped to the skeleton.

CHAPTER 7

SPLINES FOR GLOBAL ILLUMINATION

7.1 Overview

We introduce a spline-based approach for representing radiance as a texture over surfaces. Our technique enables speedy capture and playback of view-dependent lighting effects for a wide variety of surface types, illumination conditions, and material elements. The B-spline formulation facilitates the solution of the global illumination equation. Once computed, the surface radiance representation is view independent, can be evaluated quickly, and is equally suited for incorporation into ray tracing or hardware rasterization algorithms.

7.2 Our Approach

Our goal in this chapter is to enable interactive exploration of realistically rendered scenes containing complex illumination, material, and surface types. A central consideration in any rendering program is determining the colors of the visible surface points. Due to complex surface-surface lighting interactions, on-the-fly solutions to the general global illumination problem are not presently possible. Instead, it is common to maintain cached representations of (incident or exitant) radiance throughout the scene that can be queried both as part of the global illumination solution preprocess and at render time. For a given wavelength of light, surface radiance is a 4D function that depends on both surface position (u,v) and viewing direction (θ,ϕ). Key considerations in selecting a radiance representation are compactness, expressiveness, accuracy, and speed of evaluation.

Existing approaches to representing radiance can generally be divided into two classes: those that vary continuously in the spatial domain but neglect angular variation, and those that represent directional radiance at discrete points. In the present work, we introduce a unified representation for surface radiance in the form of 4D tensor product B-splines. The B-spline representation can be viewed as a general type of directionally dependent smooth texture map that is compactly represented, yet is easy and fast to evaluate. In particular, our formulation properly generalizes the polynomial textures of Malzbender *et al.* [100] and the B-spline illumination maps of Redner

et al. [107]. The representation leads directly to a global illumination algorithm analogous to gathering in radiosity. By suitably representing the integrand with a B-spline function, we can solve the resulting integral exactly. Because the B-spline radiance representation is itself sufficient for final rendering quality and is a full view-independent solution, the need for a view-dependent final gather is eliminated.

We have chosen the B-spline basis to represent angular variation instead of spherical harmonics for a number of reasons. While spherical harmonics have the advantage of being parameterized over the sphere, they also carry several well-known drawbacks. They have nonlocal support, meaning that capturing local variations in texture has global consequences. Increasing the number of basis functions to achieve finer control results in polynomials of higher degree. The number of coefficients that must be evaluated increases with the desired accuracy and the associated basis functions become progressively more expensive to evaluate. These higher degree polynomials that can assume negative values can result in visible ringing, an intrinsic characteristic of the representation.

The B-spline basis, on the other hand, is nonnegative, local, and can be refined without increasing the polynomial degree. However, the canonical mapping to the sphere generates unacceptable distortions near the poles. Hence, we introduce a new, low-distortion mapping of the plane to the sphere that can be quickly evaluated. The tensor-product representation has two traditional drawbacks: square domains and nonlocal refinement. We address the former concern by extending our radiance representation to trimmed surfaces. In answer to the latter concern, we generalize the basic approach to a hierarchical B-spline representation of radiance.

7.3 Summary of Contributions

Our approach to the GI problem is to approximate the integrand with a tensor-product spline. This can be interpreted as low-pass filtering the incident radiance function. Mitchell and Netravali [138] found that B-spline filters were those filters that best preserved edge fidelity while eliminating noise. Our extension of the technique to hierarchical splines (H-splines) allows us to selectively band limit the signal, affording greater precision where necessary. It also allows the incorporation of arbitrary sampling schemes (such as density estimation and importance), without an explosion in memory and computation. Once the integrand is approximated, the integral itself is computed exactly, due to special properties of the B-spline basis. The resulting 4D radiance function is view-independent, fast to evaluate (because of the tensor product approach), and smoothly interpolates as the view changes (due to the variation diminishing property of splines).

In summary, we:

- Introduce a 4D B-spline representation of radiance that
  - unifies the representation of both angular and spatial variation of radiance over general parametric surfaces,
  - is fast to evaluate because it is over low-degree polynomials,
  - is capable of representing complex local variations in texture and lighting because the basis functions are local,
  - implicitly maintains continuity across surface elements,
  - reduces unwanted undulations due to the variation diminishing property,
  - requires very few basis function evaluations,
  - does not exhibit ringing because the basis functions are nonnegative and low-degree,
  - encodes both local and distant lighting effects,
  - is suitable for both the incremental gather and final rendering,
  - is a function over rectangular textures and therefore easily encoded in hardware;
- Develop a fast, low-distortion mapping from the plane to the sphere;
- Introduce a technique for numerically integrating the global illumination equation by approximating the integrand with a spline;
- Extend the representation of radiance to trimmed surfaces;
- Extend the technique to a multiresolution spline scheme.

7.4 Mathematical Problem Formulation

In this section, we present a mathematical formulation of the problem of representing and computing surface radiance. Additionally, we recall some relevant properties of B-splines.

We assume the scene environment $\mathcal{E}$ consists of a collection of surfaces $\mathcal{E} = \{S_1, \ldots, S_p\}$ in 3-space, each S_i having a regular parametric representation $s_i(u, v)$, where the parameter domain

is a connected subset of $[0,1] \times [0,1]$. At each surface point $s(u,v)$ of some $S \in \mathcal{E}$, there is a local coordinate system, or local frame, the coordinate axes of which are given by

$$\hat{\mathbf{x}}_S = \frac{\mathbf{s}_u}{\|\mathbf{s}_u\|}, \qquad \hat{\mathbf{z}}_S = \frac{\mathbf{s}_u \times \mathbf{s}_v}{\|\mathbf{s}_u \times \mathbf{s}_v\|}, \qquad \hat{\mathbf{y}}_S = \hat{\mathbf{z}}_S \times \hat{\mathbf{x}}_S,$$

where $\|\cdot\|$ denotes Euclidean distance. Note that the local coordinate axes are functions of the parameters u and v and that the vector $\hat{\mathbf{z}}_S$ is the unit surface normal. The local frame coordinates of a vector $\mathbf{d}$ at the surface point $s(u,v)$ are given by $(\mathbf{d}\cdot\hat{\mathbf{x}}_S, \mathbf{d}\cdot\hat{\mathbf{y}}_S, \mathbf{d}\cdot\hat{\mathbf{z}}_S)$. The local frame also induces a coordinate system in (θ,ϕ) for the local unit hemisphere above the surface, used to specify a direction on the surface. If r is a point in space different from the surface point $s(u,v)$, then the spherical coordinates of the direction $\mathbf{d} = r - s(u,v)$ are given by

$$\theta = \arccos \frac{\mathbf{d}\cdot\hat{\mathbf{z}}_S}{\|\mathbf{d}\|}, \qquad \phi = \arctan\left(\mathbf{d}\cdot\hat{\mathbf{y}}_S, \mathbf{d}\cdot\hat{\mathbf{x}}_S\right)$$

where the function $\arctan(a,b)$ denotes the two-dimensional inverse tangent function. Note the spherical coordinates are dependent on the local frame, which in turn varies with surface position.

7.5 Radiance Integrals

At a fixed wavelength, each surface S in the environment has an intrinsic emissive radiance function $L_S^{\text{emit}}(u,v,\theta,\phi)$, which is zero except for the scene light sources, and a Bi-directional Reflectance Distribution Function (BRDF) $\rho_S(u,v,\theta,\phi,\theta^{\text{in}},\phi^{\text{in}})$. The outgoing radiance L_S on a surface is determined by integrating the incoming radiance L_S^{in} against the BRDF ρ over the hemisphere above the surface and adding the result to the emitted radiance,

$$L_S(u,v,\theta,\phi) = L_S^{\text{emit}}(u,v,\theta,\phi) + \int_\Omega L_S^{\text{in}}(u,v,\theta,\phi)\rho(u,v,\theta,\phi,\theta^{\text{in}},\phi^{\text{in}})\cos\theta^{\text{in}}\, d\omega \qquad (7.1)$$

where $d\omega = \sin\theta^{\text{in}} d\theta^{\text{in}} d\phi^{\text{in}}$. Note that the outgoing radiance $L_S(u,v,\theta,\phi)$ is a function of surface position (u,v) and direction (θ,ϕ) in the local frame. In general, S can be illuminated by radiance from all the other surfaces in the scene, including itself. Our goal is to approximate L_S with a tensor product B-spline.

When Equation (7.1) is written for each $S \in \mathcal{E}$, a system of coupled integral equations results. The global illumination problem is the problem of solving this system of integral equations. We

need to compute the final, or steady-state, surface radiance after reflected light is sufficiently attenuated.

First, consider the case where the scene consists of only two objects R and E, and reformulate the contribution to the receiving surface R from the emitting surface E as illustrated in Figure 7.1. Suppose E is parameterized by $e(s,t)$, with derivatives and local frame coordinates as above, and that E has an outgoing radiance function $L_E(s,t,\theta,\phi)$. If there is no attenuation along an unobstructed ray path, a change of variables can be applied to the integral of Equation (7.1), to obtain the surface integral

$$\begin{aligned} L_R(u,v,\theta,\phi) &= L_R^{\text{emit}}(u,v,\theta,\phi) \\ &+ \int_E \text{vis}(r,e) L_E(s,t,\theta_E,\phi_E) \rho(u,v,\theta,\phi,\theta^{\text{in}},\phi^{\text{in}}) \frac{\cos\theta^{\text{in}} \cos\theta_E}{d^2} dE \end{aligned}$$

where $\text{vis}(r,e) = 1$ if the point $e(s,t)$ is visible from $r(u,v)$ and 0 otherwise. The surface integral can be expressed directly in terms of the surface function e over

$$L_R = L_R^{\text{emit}} + \int_0^1 \int_0^1 \chi_e \text{vis}\, L_E \rho \frac{(e-r)\cdot \mathbf{z}_R\, (r-e)\cdot \mathbf{z}_E}{\|r-e\|^4} \|\mathbf{e}_s \times \mathbf{e}_t\|\, dsdt \tag{7.2}$$

where χ_e denotes the characteristic function of the parameter domain of $e(s,t)$: $\chi_e(s,t)$ is 1

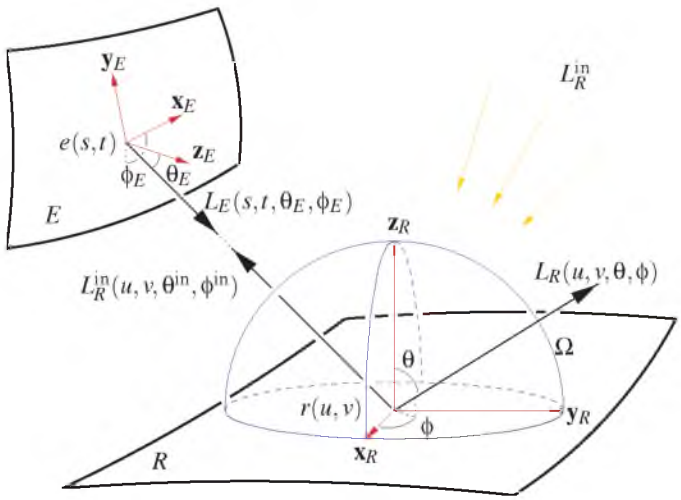

Figure 7.1: Geometry for surface radiance and light transport. The outgoing surface radiance along a ray L_R is a function of both surface position as well as the angle the ray makes with the local coordinate axes. L_R can be computed by integrating incoming radiance over the hemisphere Ω above the surface, or from a surface integral over all emitting surfaces E.

if (s,t) is in the parameter domain of $e(s,t)$, and 0 otherwise. For the sake of brevity, we have omitted the function parameters in Equation (7.2). In general, a surface integral over every surface in the environment must be added to Equation (7.2), including one for R itself.

7.6 A Review of B-Spline Approximation

The use of B-spline curves and surfaces has been well established in the graphics community since they were introduced by Riesenfeld as modeling primitives [3]. However, B-splines were studied by Schoenberg many years earlier as a technique for approximating functions [139]. Outside of graphics, they have found a solid place in numerical analysis as a powerful approximation technique, and have numerous applications to industrial problems.

Because every spline can be written as a linear combination of B-splines, a spline-space $S_{d,\boldsymbol{\tau}}$ of splines of degree d with knots $\boldsymbol{\tau}$ is defined as

$$S_{d,\boldsymbol{\tau}} = \text{span}\{B_0, \ldots, B_{n-1}\},$$

where $B_i(x) = B_{i;d,\boldsymbol{\tau}}(x)$ is the i^{th} B-spline of degree d on the *knot vector* $\boldsymbol{\tau} = (\tau_0, \ldots, \tau_{n+d})$ of nondecreasing real values. The B-spline B_i is a nonnegative piecewise polynomial, is zero outside the interval $[\tau_i, \tau_{i+d+1}]$, and $\sum_i B_i(x) = 1$ for all $x \in [\tau_d, \tau_n)$. Moreover B_i has $d-m$ continuous derivatives at a knot $x = \tau_j$ which occurs m-times among $\tau_i, \ldots, \tau_{i+d+1}$. If $f(x) = \sum_{i=0}^{n-1} a_i B_i(x)$ is a spline of degree d and $x \in [\tau_I, \tau_{I+1})$, then $f(x)$ is a sum of only $d+1$ terms

$$f(x) = \sum_{i=I-d}^{I} a_i B_i(x) \tag{7.3}$$

Because splines are piecewise polynomials, they are easily differentiated and integrated and there are stable and efficient algorithms for computing with them [5]. We will make use of the formula for integrating a spline of degree d on $\boldsymbol{\tau}$

$$\int_{\tau_1}^{\tau_{n+d}} \sum_{i=0}^{n-1} a_i B_i(x)\, dx = \sum_{i=0}^{n-1} a_i \frac{\tau_{i+d+1} - \tau_i}{d+1}. \tag{7.4}$$

7.6.1 B-Spline Approximation

Approximating a function f entails two tasks: a choice of a suitable spline space $S_{d,\boldsymbol{\tau}}$ (*i.e.*, knot vector) and a way of computing the B-spline coefficients a_i of a spline $g \in S_{d,\boldsymbol{\tau}}$ so that $g = \sum_{i=0}^{n-1} a_i B_i$ is a good approximation to f. Existing methods for computing spline approximations can be divided into two classes, local and global methods. A method is local if the value of the approximation $g(x)$ at a point x depends only on values of f in a neighborhood of x and global otherwise. A global method generally requires solving an n-by-n linear system of equations for the unknown B-spline coefficients. Examples of global methods are spline interpolation and least squares methods. In a local method, the coefficients are given explicitly. For example, in Schoenberg's variation diminishing spline approximation method, the i^{th} B-spline coefficient is given by $a_i = f(t_i^*)$, with the evaluation node t_i^* an average of contiguous knot values

$$t_i^* = \frac{\tau_{i+1} + \cdots + \tau_{i+d}}{d}.$$

The Schoenberg method belongs to a class of methods known as *quasi-interpolants* [140, 141]. We obtain an example of a quadratic quasi-interpolant by choosing the B-spline coefficient of the i^{th}quadratic B-spline $B_{i,2}$ as

$$a_i = -\frac{1}{2} f(\tau_{i+1}) + 2f(t_i^*) - \frac{1}{2} f(\tau_{i+2}),$$

where $t_i^* = (\tau_{i+1} + \tau_{i+2})/2$. This method has approximation order $O(h^3)$, while the Schoenberg method is $O(h^2)$ for all degrees d. $O(h^4)$ cubic quasi-interpolants exist as well [141]. Convergence of B-spline approximation is well established (*e.g.*, [142]).

7.6.2 Tensor Product B-spline Functions

There are several natural ways of generalizing univariate B-spline functions to multivariate functions. Since speed is an issue in this chapter, the tensor product form of B-splines is a natural choice.

Tensor product spline functions are defined on a rectangular grid using a knot vector in each spatial dimension. In the two-dimensional case, there are two knot vectors $\boldsymbol{\tau}_x$ and $\boldsymbol{\tau}_y$, two degrees d_x and d_y, and two dimensions m and n. The coefficients a_{ij} can be stored in a $m \times n$ matrix,

and the tensor product spline function is written as the double summation

$$f(x,y) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} a_{ij} B_{j;d_y,\boldsymbol{\tau}_y}(y) B_{i;d_x,\boldsymbol{\tau}_x}(x).$$

Higher-dimensional tensor product B-spline functions follow naturally. The coefficient matrix becomes a general tensor. A k-dimensional tensor product spline function is the k-fold sum

$$f(x_1,\ldots,x_k) = \sum_{i_1=0}^{n_1-1} \cdots \sum_{i_k=0}^{n_k-1} a_{i_1\ldots i_k} B_{i_k}(x_k) \cdots B_{i_1}(x_1),$$

where the coefficients form a k^{th}order tensor. Notice the knot vector and degree subscripts on the basis functions have been omitted for brevity. In general, we will use an even simpler notation, omitting variable dependence and summation limits: for a 4D tensor product B-spline, we write simply

$$\sum_{ijkl} a_{ijkl} B_l B_k B_j B_i.$$

7.6.3 Evaluation of Tensor Product B-Spline Functions

Evaluating a tensor product spline is fast because for a spline of degree d, at most $d+1$ B-splines are nonzero at any parameter value. For example, a two-dimensional cubic spline has only four nonzero basis functions in each variable, and the summation in this case can be written as the matrix product

$$[B_{I-3} \cdots B_I] \begin{bmatrix} a_{I-3,J-3} & \cdots & a_{I-3,J} \\ \vdots & & \vdots \\ a_{I,J-3} & \cdots & a_{I,J} \end{bmatrix} \begin{bmatrix} B_{J-3} \\ \vdots \\ B_J \end{bmatrix}$$

where I and J are determined by the knot intervals as in Equation (7.3). This 4×4 matrix product is amenable to hardware vectorization.

Higher-order evaluations require a more involved formulation, but the idea is the same. It is worth noting that if fast computation of 1D and 2D B-splines is available (*e.g.,* in hardware) a simple re-association of the tensor product summations shows how 3D and 4D splines can be

quickly evaluated:

$$\sum_{ijk} a_{ijk} B_k B_j B_i = \sum_i \left(\sum_{jk} a_{ijk} B_k B_j \right) B_i$$
$$\sum_{ijkl} a_{ijkl} B_l B_k B_j B_i = \sum_{ij} \left(\sum_{kl} a_{ijkl} B_l B_k \right) B_j B_i$$

and of course higher-dimensional extensions follow.

7.7 Surface Radiance Textures

This section is concerned with the approximation of the radiance on the surface, assuming that the actual value of L_S is available at each point. As $L_S(u, v, \theta, \phi)$ is a four-variate function, a four-dimensional tensor product B-spline is indicated. We need to construct an appropriate spline space and approximation method.

7.8 Lambertian Surfaces

To develop the B-spline approximation technique, we first assume that the surface is Lambertian, that is, the reflected or scattered radiance is independent of the direction. The surface radiance reduces to a function $L(u, v)$ of only two variables. To construct an approximation $\tilde{L}(u, v)$ to L involves choosing the degrees and the knot vectors in u and v, and applying an approximation method (Schoenberg or quasi-interpolation).

For knot vectors, we can for example use uniform knots with multiple knots at each end:

$$[\underbrace{a, \ldots, a}_{d+1}, a + h, a + 2h, \ldots, b - h, \underbrace{b, \ldots, b}_{d+1}] \tag{7.5}$$

where d is the degree, $m \geq 0$ is the number of internal knots, $h = (b - a)/(m + 1)$ and the domain of the parameter (u or v) is the closed interval $[a, b]$. Normally, the domain $[a, b]$ is the unit interval $[0, 1]$, but for practical reasons, the radiance function might not be defined on the edge of a surface patch, so we use instead $[\varepsilon, 1 - \varepsilon]$ for some small ε, this having the effect of pulling evaluation points away from the edges. This makes parameter values less than ε or greater than $1 - \varepsilon$ technically outside the domain of the B-spline function, but in practice, this causes little difficulty due to the continuous dependence of a spline as a function of its knots.

7.8.1 Nonuniform Domains; Trims

The knot vectors need not be uniform. If there is extra detail on one part of the surface, for example, the knots can be clustered there. However, because we are using tensor products, knot lines extend through the entire domain, as illustrated in Figure 7.2. An obvious problem arises when the parameter domain for the surface is not rectangular, such as is the case for a trimmed B-spline surface [5, 121], because the evaluation nodes (Section 7.6.1) may lie outside the domain. A simple and remarkably effective solution is to move such nodes just inside the boundary. If done in a consistent manner, this effectively extends the function being approximated to the entire rectangle. Again, Figure 7.2 provides an illustration.

7.9 View-Dependent Radiance

We now turn to the problem of approximating directionally dependent surface radiance, which is intrinsically a four-dimensional function and thus requires a 4D tensor product B-spline. The spherical coordinate parameterization for the hemisphere given by (θ, ϕ) above is a candidate for the directional spline parameters, as the domain is rectangular. The periodicity required in the ϕ-variable can be handled by a simple periodic extension of the corresponding knot vector. A more serious difficulty, however, is that the standard spherical parameterization is highly nonuniform: knot values in ϕ get pinched together near the pole (Figure 7.3).

7.9.1 Mapping to the Sphere

What is needed is a smooth, low distortion mapping of the unit square to the unit hemisphere that is fast to evaluate because the inverse mapping must be computed at every radiance evaluation. Shirley and Chiu [143] present a fast square to hemisphere mapping, but it contains

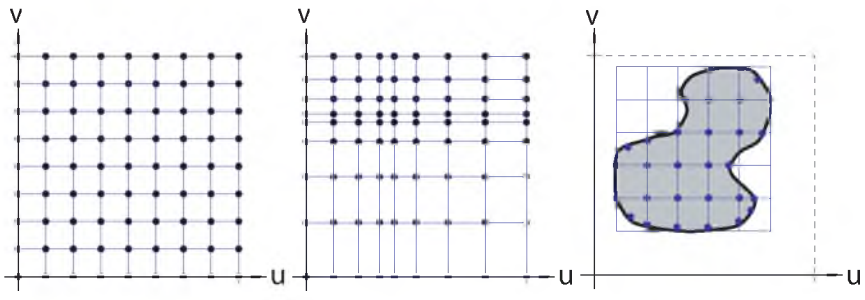

Figure 7.2: Tensor product knots and knot lines. (Left) uniform knot spacing. For nonuniform spacing (middle), the knots can be chosen independently in each dimension, but not arbitrarily over the domain. In the case of a nonrectangular domain (right), evaluation nodes can be moved inside the domain.

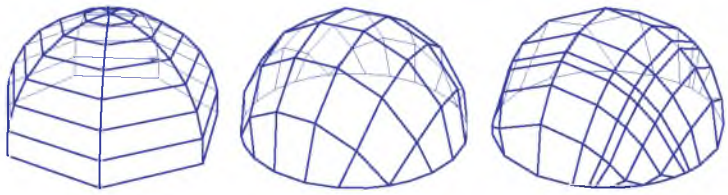

Figure 7.3: Angular parameters on the hemisphere. The standard θ, ϕ spherical parameterization results in poor knot spacing (left). Our trimmed square-to-sphere mapping is much more uniform (middle). Nonuniform knot spacing can be used with our mapping, to better approximate a highlight for example (right).

derivative discontinuities that result in visible artifacts when we apply it to the B-spline representation.

Instead, we apply the equal area projection of the disc to the hemisphere, trimming away points in parameter space outside the disc. Accordingly, we use a different set of local parameters α, β, with $-1 \leq \alpha \leq 1$, $-1 \leq \beta \leq 1$, for the angular parameters in the spline approximation. If $r = \sqrt{\alpha^2 + \beta^2}$, then the rectangular coordinates (in the local frame) of the corresponding point on the hemisphere are given by

$$z = 1 - r^2 \tag{7.6}$$

$$x = \frac{\sqrt{1 - z^2}}{r}\alpha \tag{7.7}$$

$$y = \frac{\sqrt{1 - z^2}}{r}\beta. \tag{7.8}$$

Notice the mapping is undefined outside the unit disc, so we "pull in" node values outside the disc: if $r > R$ then α and β are scaled by R/r. Thus, every point in the plane outside the unit disc is moved radially to the disc of radius R. Normally, R would be set to 1, but radiances are properly zero (otherwise undefined) at normal angle $\pi/2$, and this could cause unwanted darkening at grazing angles—the opposite of the Fresnel effect exhibited by many reflective materials. If the maximum normal angle θ is to be $\pi/2 - \delta$, then $R \approx \sqrt{1 - \delta}$. We denote by $\theta = \theta(\alpha, \beta)$ and $\phi = \phi(\alpha, \beta)$ the local spherical coordinates of the point (x, y, z) given by Equations (7.6)–(7.8). In terms of the spline parameters α and β,

$$\theta(\alpha, \beta) = \arccos(1 - \min(\alpha^2 + \beta^2, 1 - \delta))$$

$$\phi(\alpha, \beta) = \arctan(\beta, \alpha)$$

In practice, often only the rectangular coordinates are needed, because BRDF models are often expressed in terms of the incoming and outgoing vectors.

7.10 Approximation

Given the mappings $\theta(\alpha, \beta)$ and $\phi(\alpha, \beta)$ of the previous section, we can approximate the radiance $L(u, v, \theta, \phi)$ for a given surface by a 4-dimensional tensor product spline in the variables (u, v, α, β). Thus,

$$L(u, v, \theta(\alpha, \beta), \phi(\alpha, \beta)) \approx \tilde{L}(u, v, \alpha, \beta),$$

where

$$\tilde{L}(u, v, \alpha, \beta) = \sum_{i,j,k,l} a_{ijkl} B_l B_k B_j B_i. \tag{7.9}$$

For the variables u and v, we use the knot vector given by Equation (7.5), while for α and β, we use

$$[\underbrace{-R, \ldots, -R}_{d+1}, -R + h, -R + 2h, \ldots, R - h, \underbrace{R, \ldots, R}_{d+1}],$$

where $h = 2R/(m + 1)$, and m is the number of internal knots.

The coefficients are computed using the Schoenberg method, the surface radiance evaluated at the Schoenberg node values

$$a_{ijkl} = L(u_i^*, v_j^*, \theta(\alpha_k^*, \beta_l^*), \phi(\alpha_k^*, \beta_l^*)).$$

For notational convenience, we write θ_{kl}^* and ϕ_{kl}^* for $\theta(\alpha_k^*, \beta_l^*)$ and $\phi(\alpha_k^*, \beta_l^*)$, respectively. Figure 7.4 gives pseudocode for rendering surface points using the B-spline approximation. We note that the coefficients could also be constructed using a quasi-interpolation method.

B-Spline Shading Functions

```
radiance BSPLINE_SHADE (surface S, real u, real v, point eye )
   // Returns the B-spline radiance at u, v, as viewed from eye.
{
   r ← S(u, v)
   d̂ ← (eye − r)/‖eye − r‖
   compute the surface frame x_S, y_S, z_S at u, v.
   compute ŵ, the direction of d̂ in the local frame at u,v
   map ŵ to spline parameters α, β
   return S.L(u, v, α, β)
}


radiance BSPLINE_SHADE (surface S, point x, unit_vector d̂ )
   // Returns the B-spline radiance function at x, situated on the
   // surface S as viewed from the direction d̂
{
   compute the u, v parameter values of x on the surface S
   compute the surface frame x_S, y_S, z_S at u, v.
   compute ŵ, the direction of d̂ in the local frame at u,v
   map ŵ to spline parameters α, β
   return S.L(u, v, α, β)
}
```

Figure 7.4: Functions to render a surface point using a B-spline radiance function. The first is more suited to hardware rasterization; the second, to ray tracing.

7.11 Global Illumination

In this section, we consider solving the integral equations for global illumination. The global (hemispherical) formulation of Equation (7.1) as well as the surface integral formulation of Equation (7.2) form a coupled system of integral equations. Our solution method is based on an iterative gathering approach, where an approximate radiance is stored on each surface, and an updated approximate radiance is computed by direct evaluation of the radiance integrals from the earlier approximation.

The goal is to construct a B-spline radiance approximation in the form of Equation (7.9) on each surface by evaluating the radiance integrals. Consider first the hemispherical integral formulation Equation (7.1). For the radiance approximation, the B-spline coefficients are computed by evaluating the equation at the Schoenberg evaluation nodes

$$a_{ijkl} = L_R^{\text{emit}}(u_i^*, v_j^*, \theta_{kl}^*, \phi_{kl}^*) + \int_\Omega L_R^{\text{in}}(u_i^*, v_j^*, \theta^{\text{in}}, \phi^{\text{in}}) \rho(u_i^*, v_j^*, \theta_{kl}^*, \phi_{kl}^*, \theta^{\text{in}}, \phi^{\text{in}}) \cos\theta^{\text{in}} d\omega. \tag{7.10}$$

Assuming that L^{in} can be computed (in our implementation, it is done with ray tracing from the previous radiance approximation), the integral of Equation (7.10) over θ^{in}, ϕ^{in} could be evaluated using any numerical method that uses point sampling. For example, the integral can be computed using an importance sampling method based on the shape of the BRDF.

Our approach to evaluating the integral is to perform a B-spline approximation to the integrand, then evaluate this exactly using existing techniques. The B-spline approximation to the integrand of Equation (7.10) has the form

$$\begin{aligned}\int_\Omega L_R^{\text{in}}\rho\cos\theta^{\text{in}}\,d\omega &= \int_0^{2\pi}\int_0^{\pi/2} L_R^{\text{in}}\rho\cos\theta^{\text{in}}\sin\theta^{\text{in}}\,d\theta^{\text{in}}\,d\phi^{\text{in}} \\ &\approx \sum_{ijkl}\left(\int_0^{2\pi}\int_0^{\pi/2}\sum_{pq} c_{ijklpq}B_qB_p\;d\theta^{\text{in}}\,d\phi^{\text{in}}\right)B_lB_kB_jB_i. \qquad (7.11)\end{aligned}$$

Applying the B-spline integration formula from Equation (7.4) yields a formula for the surface radiance coefficients

$$a_{ijkl} = L_R^{\text{emit}}(u_i^*, v_j^*, \theta_{kl}^*, \phi_{kl}^*) + \sum_{pq} c_{(ijkl)pq}\frac{\tau_{q+d_{\phi^{\text{in}}}+1}-\tau_q}{d_{\phi^{\text{in}}}+1}\frac{\sigma_{p+d_{\theta^{\text{in}}}+1}-\sigma_p}{d_{\theta^{\text{in}}}+1} \qquad (7.12)$$

where the c_{ijklpq} are the integrand of Equation (7.11) evaluated at node values u_i^*, v_j^*, θ_{kl}^*, ϕ_{kl}^*, $\theta_p^{\text{in}*}$, and $\phi_q^{\text{in}*}$. Note that the choice of the first four evaluation nodes is fixed by the knot vectors for the B-spline radiance representation on R, but the knot vectors in θ^{in} and ϕ^{in} for the integration can be chosen arbitrarily, and could even be different for each integral evaluation.

The surface integral formulation can be approximated in the same way:

$$\begin{aligned}&\int_0^1\int_0^1 \chi_e\text{vis}\,L_E\rho\,\frac{(e-r)\cdot\mathbf{z}_R\,(r-e)\cdot\mathbf{z}_E}{\|r-e\|^4}\|\mathbf{e}_s\times\mathbf{e}_t\|\,ds dt \\ &= \int_0^1\int_0^1 f(u,v,\theta,\phi,s,t)\,ds dt \\ &\approx \sum_{ijkl}\left(\int_0^1\int_0^1\sum_{pq} f(u_i^*, v_j^*, \theta_{kl}^*, \phi_{kl}^*, s_p^*, t_q^*)B_qB_p\,ds dt\right)B_lB_kB_jB_i \\ &= \sum_{ijkl}\left(\sum_{pq} f(u_i^*, v_j^*, \theta_{kl}^*, \phi_{kl}^*, s_p^*, t_q^*)\frac{\tau_{q+d_t+1}-\tau_q}{d_t+1}\frac{\sigma_{p+d_s+1}-\sigma_p}{d_s+1}\right)B_lB_kB_jB_i \qquad (7.13)\end{aligned}$$

and thus gives the explicit formula for the B-spline coefficients for the outgoing radiance $\tilde{L}_R$ reflected off of R from E. Summing the corresponding coefficients for all other surfaces in the scene, including R itself if it is nonconvex, results in the coefficients of the radiance B-spline on R.

7.12 A Gathering Algorithm

In this section, we describe a gathering approach for global illumination similar to that used in classical radiosity. The algorithm uses the data structures of Figure 7.5. A `Bspline4D` object contains the knot vectors as well as the tensor of coefficients. A `surface_radiance` object stores three `Bspline4D` objects: one for the cumulative gathered radiance, one for an incremental radiance from the most recent gather, and one temporary value for the gathered radiance. There is one `surface_radiance` object for each scene surface, and initially, the Le field contains the emissive radiance for the surface if that surface is a light source. Figure 7.5

```
struct Bspline4D {
 radiance_tensor4D C          // B-spline coefficients
 real              knots[4][] // knot vectors
 int               degree[4]  // B-spline degrees
}
struct surface_radiance {
 Bspline4D         Le         // cumulative radiance
 Bspline4D         L          // incremental radiance
 Bspline4D         Ltmp       // temporary radiance
}
```

Gathering Algorithm

```
// Direct Lighting Phase
for each surface S do
   S.L ← 0
   for each emitting surface E do
      gather from E to S, store the radiance in S.L
   end for
   // add the gathered radiance to the cumulative radiance
   S.Le ← S.Le + S.L
end for

// Indirect Lighting Phase
while each .L is too large do
   for each surface S do
      gather globally from the scene to S.Ltmp
      // the emission is taken from the .L fields on other surfaces
   end for
   for each surface S do
      add the gathered radiance S.Ltmp to S.Le
      replace the incremental radiance S.L with S.Ltmp
   end for
end while
```

Figure 7.5: The global illumination gathering algorithm, and the associated data structures.

outlines the method.

The first iteration is the "direct lighting" stage, where surfaces are only illuminated by light sources. Generally, there are only a few light sources, and each source subtends only a small solid angle as from most receiver points. It is therefore more efficient to use the surface integral formulation for this computation.

After the direct lighting computation, the L field for each surface contains the radiance due to direct illumination. From these, we gather to each surface from the entire environment, using the hemispherical integral, and the resulting radiance (the indirect lighting after a single reflection) is placed into the Ltmp field. Then we add Ltmp to the cumulative radiance Le, replace L with Ltmp, and repeat. Notice that after iteration n, the L fields contain the indirect lighting after exactly n reflections, while the cumulative radiance Le stores the total illumination. After a sufficient number of indirect iterations, determined either visually or by a convergence criterion on the incremental radiances, the algorithm terminates and Le contains the GI solution. We emphasize that at the end of the algorithm, Le is valid for all views and is used directly for our renderings. There is no need for a final gather.

It might seem more natural to gather from the cumulative radiance Le, and indeed this would require only two B-spline representations for each surface during the global illumination computation. However, working with the incremental radiance has some important advantages. First, it allows the direct lighting computation to be handled separately, and this is advantageous because frequently, this must be done more accurately than the indirect lighting computations. In fact, we can usually get away with a much coarser representation for the subsequent incremental radiances. Figure 7.6 shows how our algorithm performs with respect to glossy interreflections.

7.13 Transmission

The discussion thus far has assumed the surfaces are only reflective. If a surface also has a Bi-directional Transmission Distribution Function (BTDF), the radiance function must be extended to directions below the local surface tangent plane, *i.e.,* having normal angle $\theta > \pi/2$. We do this by adding a second radiance B-spline for the transmitted radiance. The gathering algorithm must be modified accordingly. Figure 7.7 demonstrates our approach applied to transmitted illumination.

7.14 Hierarchical Radiance Textures

Figure 7.8 illustrates the behavior of our radiance model with respect to sharp features as the number of samples and the spline degree are increased. These are global changes to the spline

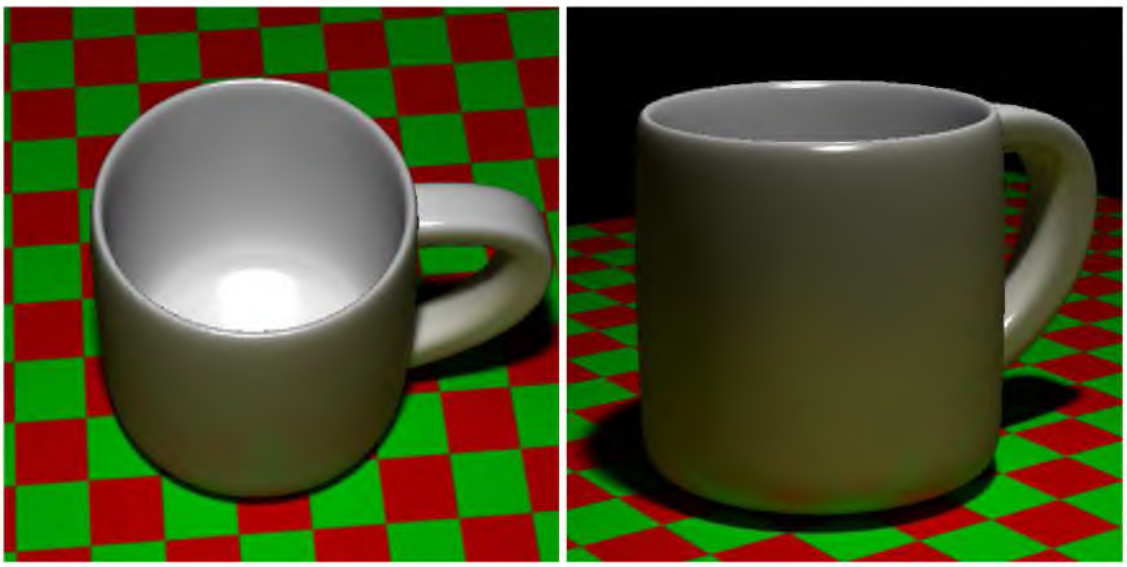

Figure 7.6: A demonstration of glossy interreflections (ray traced using the B-spline shader).

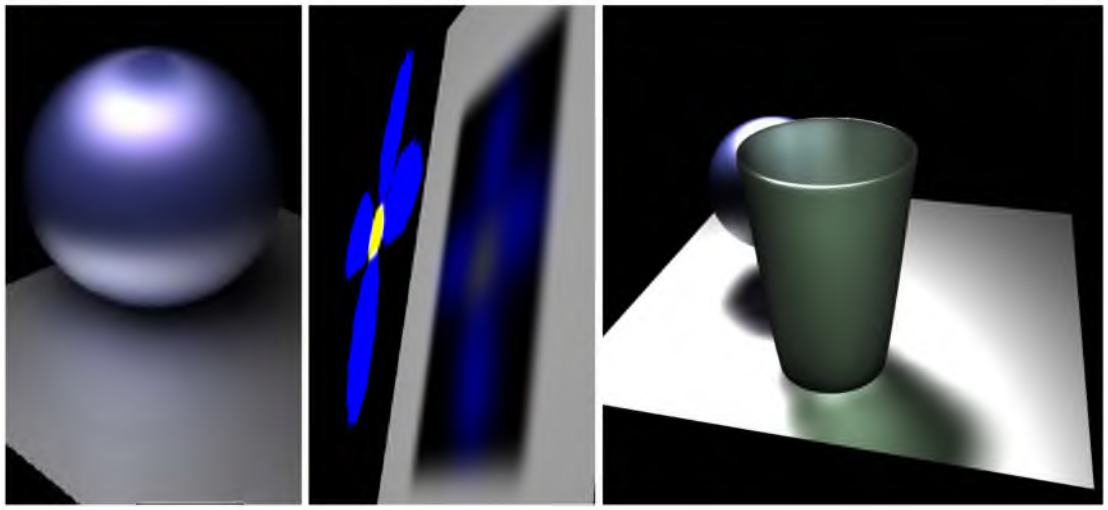

Figure 7.7: An anisotropic sphere, a transmissive rectangle, and a glass with a Phong-like transmission (the images are ray traced, using the B-spline shader).

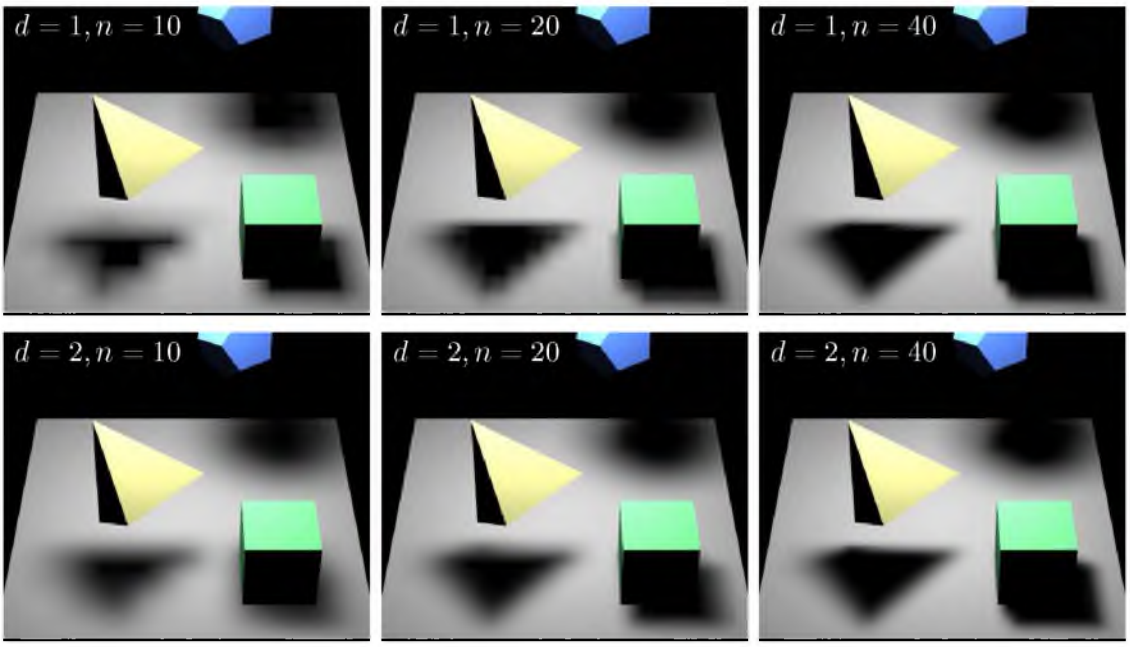

Figure 7.8: 2D B-splines for a diffuse surface with varying degree and knot density. Note the degree 1 case reduces to linear interpolation. The tensor product nature is most noticeable on shadow edges diagonal to the knot lines.

representation. For the sake of memory efficiency and speed of computation, we would like to add the facility for making localized adjustments to the radiance function. The standard approach to adding detail to tensor product splines is via knot refinement. However, because knot lines extend to the patch boundary, refinement is a nonlocal operation. Obtaining local control with tensor product splines can therefore lead to an explosion in coefficients. In our case, it can also lead to unnecessary gathers in locations where the radiance is already well represented by the coarser mesh.

The desire to preserve the advantages of the tensor product approach while allowing for detail at multiple scales has led to a number of multiresolution approaches. To extend our notion of radiance textures, we have chosen hierarchical splines (H-splines) [137], but we note that spline wavelets [144] might be equally well suited. The chief observation of the H-spline approach is that while refinement is a global operation, the impact of editing a control point is localized to a small number of patches. The remainder of the refined surface is identical to the coarse level surface. Forsey and Bartels introduce the concept of an overlay, which represents just that part of the surface impacted by editing the refined control mesh. The coefficients of the overlay patch are the offsets representing the modifications to the refined mesh. Naturally, overlays can possess overlays, which are offsets from offsets, and so on.

The H-spline is represented in a tree data structure. In order to evaluate the H-spline at a particular parameter $(u_0, ..., u_n)$, we traverse the associated overlays L_i until we reach a leaf, and the radiance L is given by

$$L(u_0, \ldots, u_n) = L_0(u_0, \ldots, u_n) + \sum_{i=1}^{l-1} L_i(u_0, \ldots, u_n) \tag{7.14}$$

To allow for isolated refinements of the GI integrals (Equations (7.12) and (7.13)) over both the spatial and directional domains, we assume that the integrands can be represented as an H-spline. The resulting gather algorithm is compatible with our basic approach. Suppose we have a patch $\mathcal{P}$ that we determine has not been sufficiently refined. We insert some knots into the patch to add local detail, and this results in a new overlay at level l. The radiance function is represented by Equation (7.14), and the incremental gather for the overlay is calculated as an integral

$$\begin{aligned} L(u,v,\theta,\phi) &= \int F(u,v,\theta,\phi,\theta^{\text{in}},\phi^{\text{in}}) d\theta^{\text{in}} d\phi^{\text{in}} \\ L_0(u,v,\theta,\phi) + \sum_{i=1}^{l} L_i(u,v,\theta,\phi) &= \int F_0(u,v,\theta,\phi,\theta^{\text{in}},\phi^{\text{in}}) + \sum_{i=1}^{l} F_i(u,v,\theta,\phi,\theta^{\text{in}},\phi^{\text{in}}) d\theta^{\text{in}} d\phi^{\text{in}} \\ L_l(u,v,\theta,\phi) &= \int F_l(u,v,\theta,\phi,\theta^{\text{in}},\phi^{\text{in}}) d\theta^{\text{in}} d\phi^{\text{in}} \end{aligned}$$

This last equation is not useful for evaluation, because we never explicitly generate F_l. However, we note that so long as we adjust vertices within overlay l, the coarser level terms will cancel, leaving us with the overlay solely, and we can calculate the coefficients for new samples of the global illumination integral using the formula

$$\begin{aligned} L_l(u,v,\theta,\phi) = &\left(\int F_0(u,v,\theta,\phi,\theta^{\text{in}},\phi^{\text{in}}) + \sum_{i=1}^{l} F_i(u,v,\theta,\phi,\theta^{\text{in}},\phi^{\text{in}}) d\theta^{\text{in}} d\phi^{\text{in}} \right) \\ &- \left(L_0(u,v,\theta,\phi) + \sum_{i=1}^{l-1} L_i(u,v,\theta,\phi) \right) \end{aligned}$$

In other words, the leaf nodes represent the residual with respect to the coarse and refined approximations. This final equation enables direct use of the gathering algorithms from Section 7.11.

Figure 7.9 demonstrates the application of the H-spline approach to the floor of the Cornell Box. For this example, the H-spline approach results in an order of magnitude decrease both in

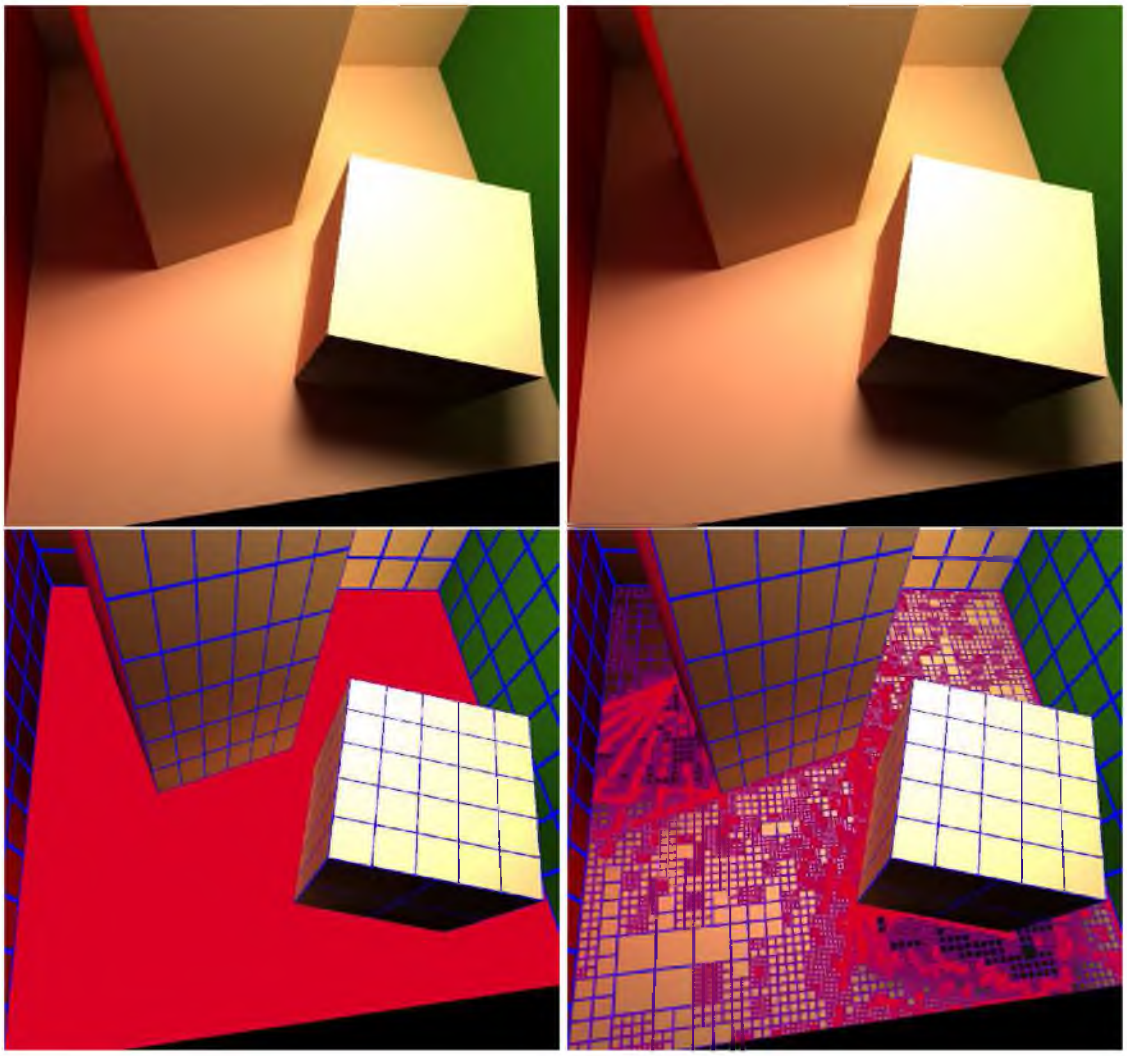

Figure 7.9: A comparison of tensor product and hierarchical splines. To meet the same error metric (< 1 intensity level change under refined gather), the uniform subdivision tensor product patch (left) requires $264,196$ coefficients. The uniformly refined hierarchical representation requires only $17,781$ coefficients. Furthermore, the gather step is roughly an order of magnitude faster for the H-spline. The bottom figure shows knot lines for each.

the number of coefficients and in compute time. Note that the image accurately reproduces the shadow discontinuity on the floor of the box.

7.15 Implementation

We have implemented the 4D B-spline techniques presented here in the context of a standard ray tracer for both rendering and global illumination computation. Each coefficient is stored as four floats (RGB with a pad to 16 bytes), although for better spectral representation, we could use a 5D tensor product B-spline, with the extra variable for the wavelength. Our scenes consist of triangles, parallelograms, Bézier patches, and trimmed NURBS surfaces. For the global illumination computation, ray tracing is required for evaluating the visibility function in the surface integral formulation, and for evaluating the L^{in} value in the hemispherical integral. Therefore, our ray tracer must be able to compute ray intersections with these surfaces and the corresponding parameter values as well as surface derivatives to compute the local frame [121].

Regardless of how a surface radiance B-spline is computed (or captured), it can be treated as a general surface shader (Figure 7.4) that is dependent on surface position and view direction. Virtually any rendering system that can render the scene surfaces can render the scene using the B-spline radiances. We incorporated the B-spline shader into a GL-based renderer. We divide each surface into microfaceted polygons and render each using smooth shading. The vertex colors are computed per-frame in software using our B-spline surface function, with the eye point corresponding to the center of projection. Figure 7.10 shows a simple scene rendered in this way.

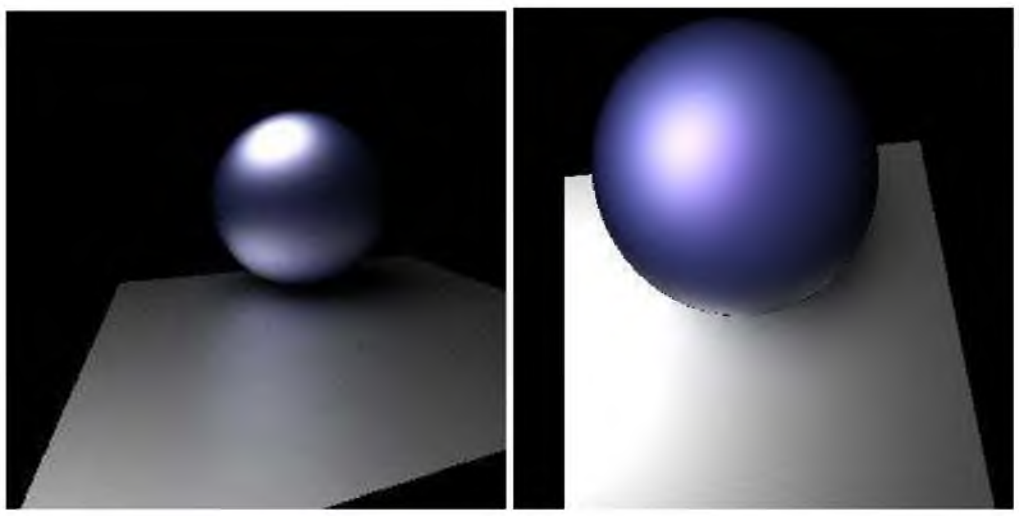

Figure 7.10: OpenGL renderings of a simple scene (global illumination computed offline). Each object is rendered with 128^2 Gouraud shaded micropolygons, with vertex colors from the B-spline shader. The framerate is about 10 Hz.

7.16 Results

Figures 7.6 – 7.12, Table 7.1, and Figure 7.13 show some results of our method. Figures 7.8 and 7.10 are rendered using OpenGL as discussed above; Figures 7.6–7.7, 7.9, and 7.11–7.13 are ray traced using the B-spline shader. In Table 7.1, we outline the configuration used in the table scene of Figure 7.13. The surfaces are polygons, Bézier patches, and trimmed NURBS. For non-Lambertian surfaces, we generally used the anisotropic Ashikhmin-Shirley BRDF [145], because it is energy conserving and exhibits the Fresnel behavior of increased specularity at glancing angles. The specular exponents were limited to be under 100.

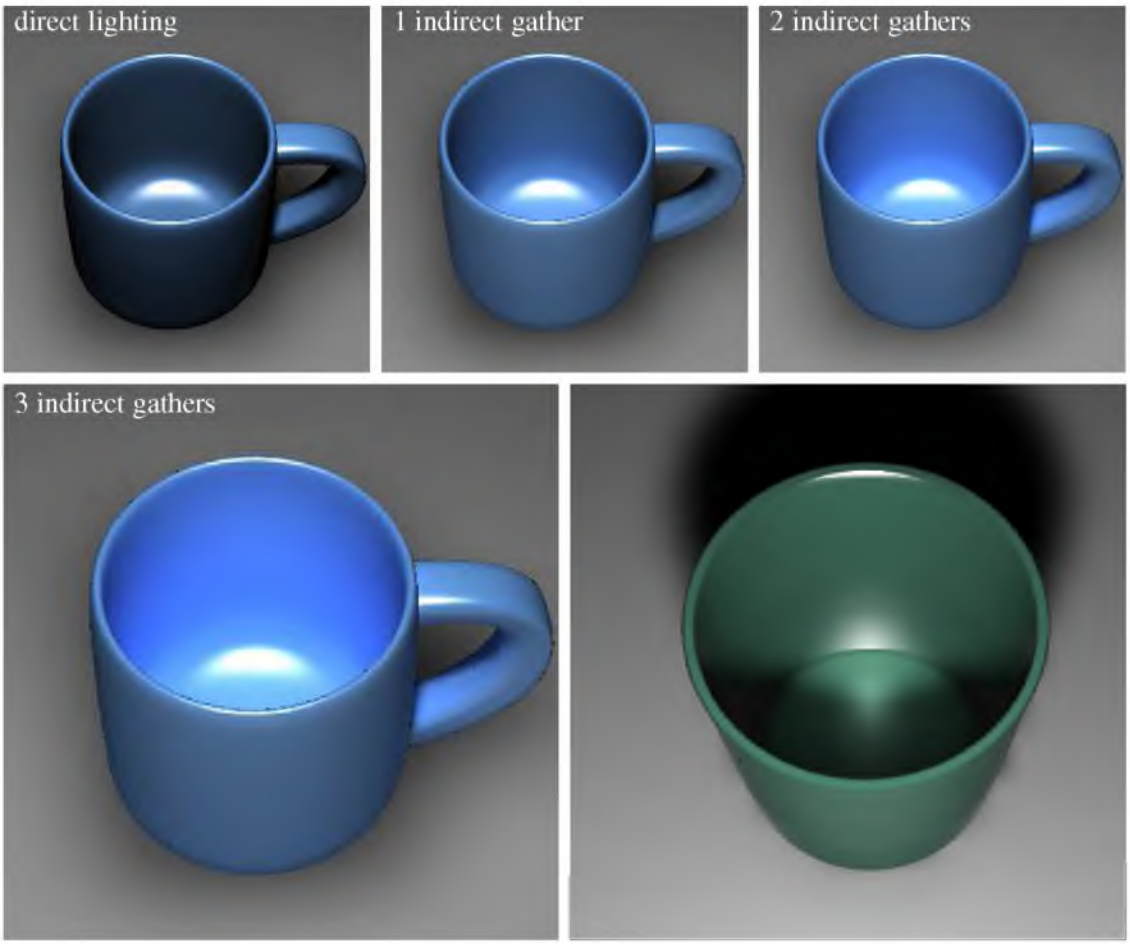

Figure 7.11: Indirect lighting in a mug and a cup. The blue mug radiance B-spline has 10 knots in each angular parameter and 41 in the spatial parameters. The green cup exhibiting a caustic has 17 knots in each parameter. The mug is modeled with only two NURBS surfaces; the cup, only one.

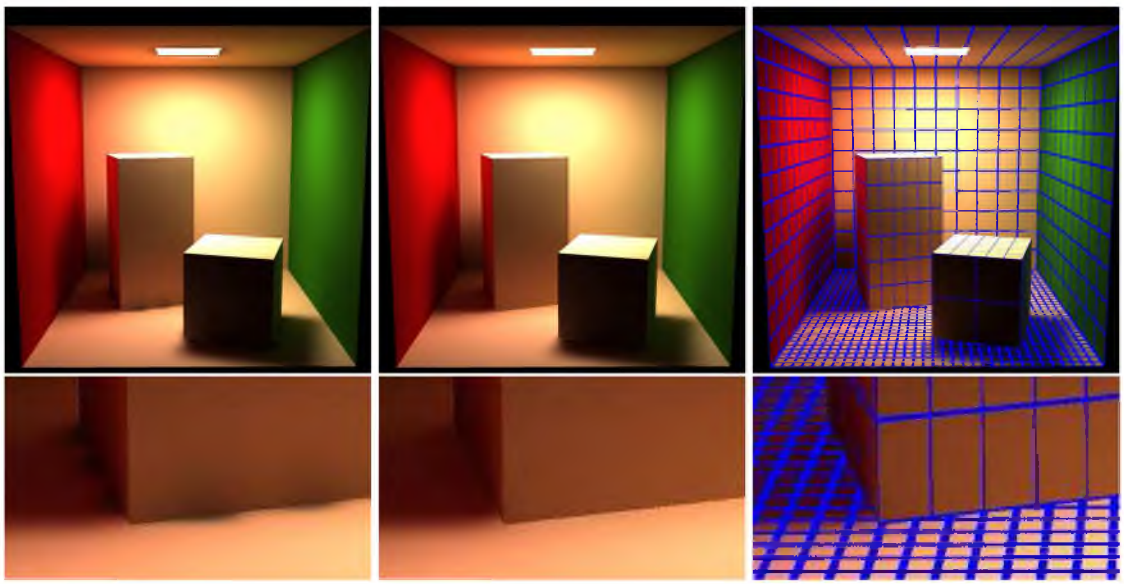

Figure 7.12: Our (nonhierarchical) method applied to the Cornell Box. The left images, which have untrimmed surfaces, exhibit artifacts where surfaces meet. Trimming has been applied to correct this in the middle images, with essentially no performance penalty. The right images have the knot lines shown in blue. The GI solution, with 3 indirect iterations, was coarsely computed with 15 knots in each of θ and ϕ for the hemispherical integration. The GI computation time was about 15 seconds on a single processor. Note: our hierarchical approach addresses the problem of representing sharp shadows. Please see Figure 7.9.

Table 7.1: Details of the B-spline radiance representations used for the table scene (Figure 7.13), giving the degrees in the spatial and angular dimensions, the number of coefficients, and the total memory required for each object. These numbers are for the basic (nonhierarchical) approach with uniform knot lines.

Object	Type	Degrees (spat/ang)	Coefficients (u,v,α,β)	Memory (kB)
lamp body	NURBS (1)	2,2	14,14,10,10	314
lamp socket	NURBS (1)	2,2	14,14,3,3	56
lamp shade	NURBS (1)	2,2	3,3,3,3	1
mug	NURBS (2)	2,2	14,14,10,10	627
teapot	Bézier (32)	2,2	8,8,14,14	6423
metal sphere	sphere	2,2	10,10,10,10	320
glass sphere	sphere	2,2	20,20,20,20	5120
table top	rectangle	2,0	183,183,1,1	535
table bottom	rectangle	2,0	4,4,1,1	1
table sides	rectangle	2,0	4,10,1,1	1
table legs	rectangle	0,0	1,1,1,1	1
walls	rectangle	2,0	24,24,1,1	9
floor	rectangle	2,0	24,24,1,1	9
TOTAL				13 416

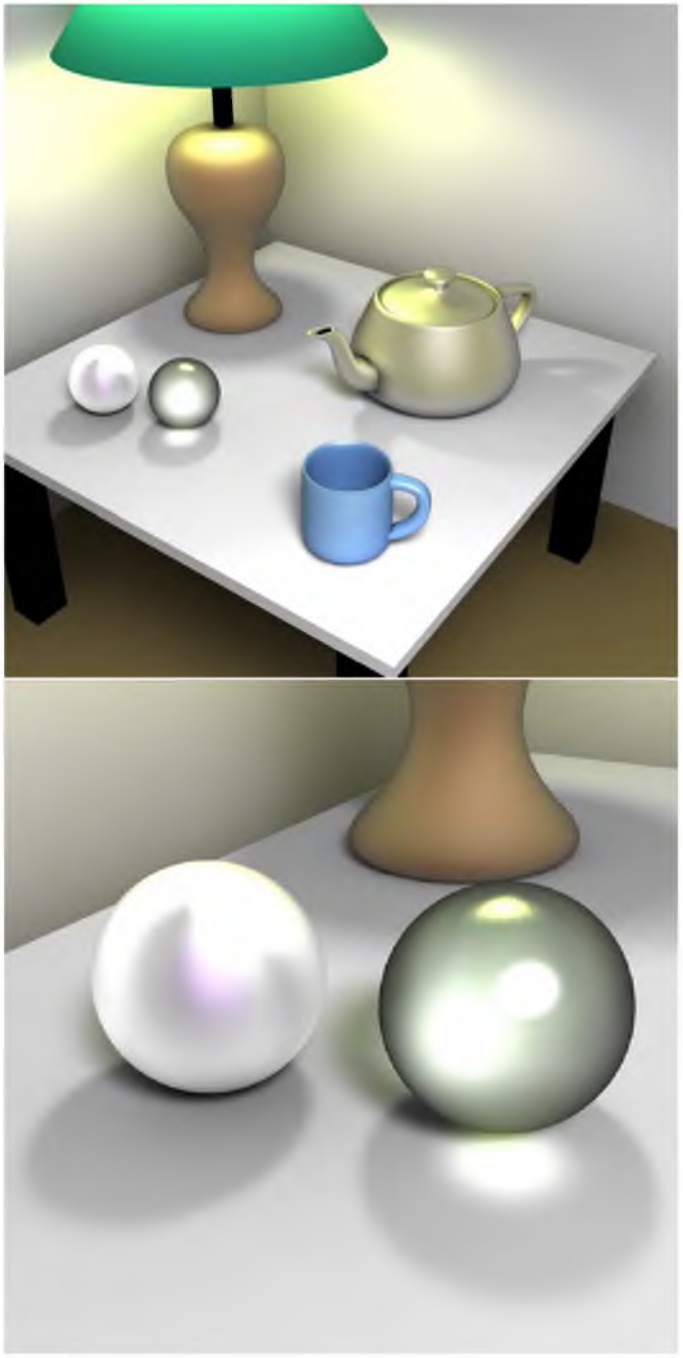

Figure 7.13: A more complex scene, with Bézier patches, NURBS, anisotropic reflection, transmission, and caustics. The source inside the lamp is in the shape of a light bulb, and there is an area source on the ceiling. The Lambertian table top has 180×180 coefficients to faithfully represent the shadow edges. The full GI solution is very finely sampled, with 35×35 knots for the integration, and required about 20 hours on a single processor. Table 7.1 gives knot vector sizes and memory statistics.

7.17 Representation and Global Illumination

B-spline approximations have established convergence properties, so the representation should be effective provided enough knots are chosen. However, a memory explosion can result: for example, a modest 100 coefficients in each dimension requires 1.6 GB of memory. We have extended our basic approach to hierarchies so that the sample budget can be utilized where it is most needed.

In general, we use quadratic (degree 2) splines in all four variables, because quadratic splines tend to represent the shape of the approximated function better than higher-order splines, and they are faster to evaluate. Quadratic splines are a great improvement over linear interpolation because they have C^1 continuity. Theoretically, the C^2 discontinuities might be noticeable, but we have not found this to be a problem. In some cases, however, it appears that cubic splines work better in areas of fine detail.

We have found that using 16 coefficients in each of the four dimensions is sufficient to represent radiance for a typical surface patch — note this amounts to as many coefficients as there are pixels in a 256×256 (nondirectional) texture map. Glossy surfaces with a small diffuse component tend to require fewer spatial knots and more angular knots, while diffuse surfaces require more spatial knots. Surfaces that exhibit near mirror reflection, however, can require many knots in both directions.

For the gathering algorithm, the primary question is what knot vectors to use for the integrations. We have found that 10 knots in each dimension for the direct lighting phase is usually sufficient. For the hemispherical gather, 15 knots in θ and 30 knots in ϕ usually avoids undersampling, although as few as 7 knots in each is sometimes enough. Again, our hierarchical extension allows for more sophisticated sampling schemes. Undersampling artifacts from the hemispherical integration can occur when there is a bright surface of small solid angle. A simple workaround is to treat such a surface as having no emission in the hemispherical integration, then separately compute the contribution from the bright surface using the surface integral formulation.

7.18 Performance

(*N. B.*: The results in this section were generated in 2002 [146] on hardware that was state-of-the-art at that time.) The most important performance consideration is the evaluation of the 4D tensor product B-spline functions, which we do in software. Figure 7.14 shows a graph of some benchmarks. The general upward trend of evaluation time with larger dimension is primarily due to degrading cache coherence. Quadratic 4D splines of modest dimensions can generally be

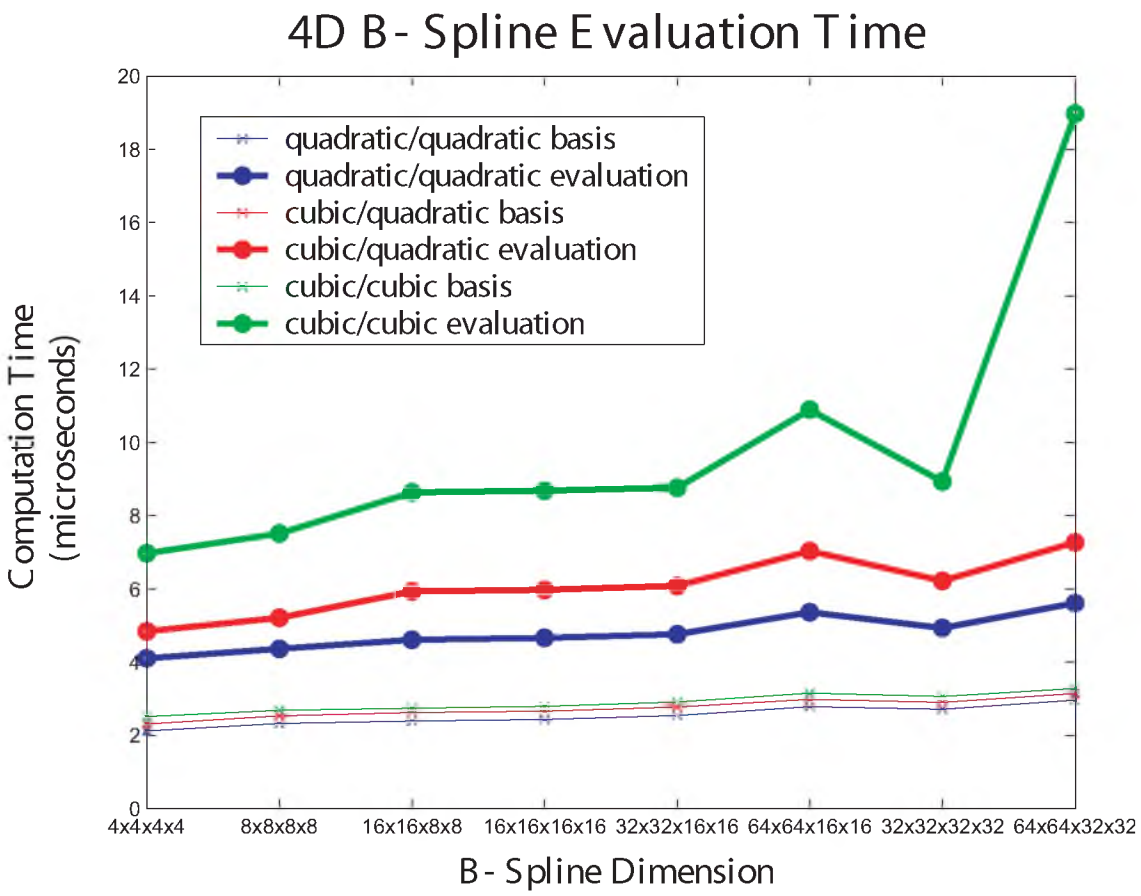

Figure 7.14: Benchmarks for 4D tensor product B-spline evaluation (single MIPS R12K, 400 MHz). The thick lines show the total computation time per evaluation, including the hemispherical mapping. The thin lines show the time of computing the B-spline basis functions. Three different degree combinations are shown, with degree in u and v first. The horizontal labels indicate the number of knots in each parametric direction.

evaluated at rates of better than 200 kHz.

In our ray tracer, the cost of finding the surface intersections generally outweighs the cost of evaluating the surface B-splines, particularly when there are many NURBS surfaces in the scene. For the GL implementation, this is not the case. In both implementations though, the B-spline evaluation time dominates the sphere mapping and conversion to the local frame. The simple scene illustrated in Figure 7.10 runs at roughly 10 frames per second on a 1.8 GHz Pentium 3 PC.

A scene with only a few surfaces and modest sampling (a coarse solution uses 7×7 knots points for the surface integration and 10×10 points on the hemispherical integration) and only one or two indirect gathers takes from a few seconds to a few minutes. We did not attempt to accelerate the GI computation for this work because we view it as a preprocess.

7.19 Conclusion and Future Work

In this chapter, we have demonstrated that surface radiance can be effectively approximated using a suitably crafted tensor product spline combined with an appropriate spherical mapping. The representation is compact, fast to evaluate, and as the examples illustrate, the representation is useful for a wide variety of phenomena. Furthermore, we have used the representation for a global illumination algorithm, and the representation computed by the algorithm can be directly rendered at interactive rates.

For this work, we have taken a rather direct approach to the approximation and integral evaluation. Knot clustering, for example, could improve the representation on a surface with nonuniformly distributed detail. Our approach to evaluating the illumination integrals is also very straightforward and could be improved. For example, the hemispherical integral could be importance sampled according to the BRDF. Also, hierarchical or clustering methods [13, 147, 148] would almost certainly improve performance. The former follow directly from our H-spline extension.

Finally, we note some natural extensions. Time varying radiance is a relatively simple extension, as is the representation of volumetric phenomena, including scattering.

CHAPTER 8

CONCLUSION

The main thrust of this dissertation was to demonstrate the utility of splines and remove barriers to their use in engineering applications. To this end, we have developed a suite of tools and techniques which includes

1. Efficient methods for evaluating NURBS functions and their derivatives.

2. Ray tracing techniques for NURBS surfaces and volumes, suitable for inclusion in an interactive parallel rendering system.

3. A novel representation based on NURBS curves, surfaces, and volumes which encodes attributes decoupled from geometry.

4. Extensions of traditional modeling operators to facilitate intuitive design of heterogeneous solids.

5. Nodal interpolation techniques to support data fitting for engineering applications.

6. An extension of traditional visualization techniques to trivariate models in order to increase intuition about these high-dimensional representations. These methods include isosurfacing, planar slicing, direct volume rendering, and optical path tracing.

7. A novel modeling operator which upgrades a boundary representation to a volumetric representation to facilitate simulation and analysis.

8. An original algorithm for characterizing radiance and solving the global illumination problem by leveraging the innate properties of splines.

APPENDIX A

PAPERS

The author has published the following papers during his graduate career. The papers in **bold** directly relate to this dissertation.

1. **Steven Parker, William Martin, Peter-Pike Sloan, Peter Shirley, Brian Smits, and Charles Hansen, "Interactive Ray Tracing," 1999 ACM Symposium on Interactive 3D Graphics. pp. 119-126, 1999.**

2. **William Martin, Elaine Cohen, Russell Fish, and Peter Shirley, "Practical Ray Tracing of Trimmed NURBS Surfaces," Journal of Graphics Tools, Volume 5, No. 1, 2000 pp. 27-52.**

3. Peter-Pike Sloan, William Martin, Amy Gooch, and Bruce Gooch, "The Lit Sphere: A Model for Capturing NPR Shading from Art," Graphics Interface 2001. pp. 143-150, 2001.

4. **William Martin and Elaine Cohen, "Representation and Extraction of Volumetric Attributes Using Trivariate Splines: A Mathematical Framework," Solid Modeling 2001.**

5. **Michael M. Stark, William Martin, Elaine Cohen, Tom Lyche, and Richard F. Riesenfeld, "B-Splines for Physically-Based Rendering," University of Utah Technical Report UUCS-02-008.**

6. William Martin, Erik Reinhard, Peter Shirley, Steven Parker, and William Thompson "Temporally Coherent Interactive Ray Tracing," Journal of Graphics Tools. 7(2), pp. 41-48, 2002.

7. **William Martin and Elaine Cohen, "Surface Completion of an Irregular Boundary Curve Using a Concentric Mapping," Curve and Surface Design: Saint-Malo 2002,**

Tom Lyche, Larry L. Schumaker, and Marie-Laurence Mazure, editors.

8. Huy T. Vo, Steven P. Callahan, Nathan Smith, Claudio T. Silva, William Martin, David Owen, and David Weinstein, "iRun: Interactive Rendering of Large Unstructured Grids, Eurographics Symposium on Parallel Graphics and Visualization 2007.

9. SX Vasquez, F Gao F, F Su, V Grijalva, J Pope, W Martin, J Stinstra, M Masner, N Shah, DM Weinstein, R Farias-Eisner, ST Reddy, 2011. "Optimization of MicroCT Imaging and Blood Vessel Diameter Quantitation of Preclinical Specimen Vasculature with Radiopaque Polymer Injection Medium. PLoS ONE 6(4): e19099. doi:10.1371/ journal.pone.0019099

APPENDIX B

SINGULAR JACOBIANS AND THE RAY INTERSECTION FUNCTION

In this section, we show that for regular surfaces, singular Jacobian matrices of the intersection function $\mathbf{F}$ (Section 3.3.3) evaluated at (u_0, v_0) are encountered only for rays which are parallel to silhouette rays at (u_0, v_0).

Definition Suppose we are given a parametric surface $\mathbf{S}(u, v)$ and a ray formulated as $\mathbf{r}(t) = \mathbf{o} + \hat{\mathbf{d}} * t$. $\mathbf{r}(t)$ is called a *silhouette ray* of $\mathbf{S}(u_0, v_0)$ if $\mathbf{F}(u_0, v_0) = 0$ and $(\mathbf{S_u}(u_0, v_0) \times \mathbf{S_v}(u_0, v_0)) \cdot \hat{\mathbf{d}} = 0$ — that is, if the ray both intersects $\mathbf{S}$ at (u_0, v_0) and lies in the tangent plane of $\mathbf{S}$ at (u_0, v_0). The point $\mathbf{S}(u_0, v_0)$ where the silhouette ray contacts the surface $\mathbf{S}$ is called a *silhouette point.*

We wish to relate a zero of $\det(\mathbf{J})$ to rays parallel to silhouettes.

Theorem *Let* $\mathbf{S}(u, v)$ *be a parametric surface such that* $\mathbf{S_u} \times \mathbf{S_v} \neq 0$ *(i.e.,* $\mathbf{S}$ *is a regular parameterization). Let* $\mathbf{F}$ *be the function*

$$\mathbf{F}(u, v) = \begin{pmatrix} \mathbf{N_1} \cdot \mathbf{S}(u, v) + d_1 \\ \mathbf{N_2} \cdot \mathbf{S}(u, v) + d_2 \end{pmatrix}$$

whose roots at (u_0, v_0) *determine the intersection of a ray described by the intersection of two planes with the surface* $\mathbf{S}$ *at* (u_0, v_0)*. The Jacobian of* $\mathbf{F}(u_0, v_0)$ *is singular if. and only if. the ray is parallel to a silhouette ray at that point.*

Proof. Consider a ray $\mathbf{o} + \hat{\mathbf{d}} * t$, which lies along the intersection of two planes $\mathbf{P_1}$ and $\mathbf{P_2}$, with normals $\mathbf{N_1}$ and $\mathbf{N_2}$, respectively, so that $\mathbf{N_1} \times \mathbf{N_2}$ is parallel to $\hat{\mathbf{d}}$. The derivative of $\mathbf{F}$ is the Jacobian matrix

$$\mathbf{J}(u, v) = \begin{pmatrix} \mathbf{N_1} \cdot \mathbf{S_u} & \mathbf{N_1} \cdot \mathbf{S_v} \\ \mathbf{N_2} \cdot \mathbf{S_u} & \mathbf{N_2} \cdot \mathbf{S_v} \end{pmatrix}$$

Suppose first that $\mathbf{J}(u_0, v_0)$ is singular. Thus, the columns of $\mathbf{J}(u_0, v_0)$ are linearly dependent:

$$a \begin{pmatrix} \mathbf{N_1} \cdot \mathbf{S_u}(u_0, v_0) \\ \mathbf{N_2} \cdot \mathbf{S_u}(u_0, v_0) \end{pmatrix} + b \begin{pmatrix} \mathbf{N_1} \cdot \mathbf{S_v}(u_0, v_0) \\ \mathbf{N_2} \cdot \mathbf{S_v}(u_0, v_0) \end{pmatrix} = 0$$

with a, b not both zero. This gives us

$$\begin{pmatrix} a\mathbf{N_1} \cdot \mathbf{S_u}(u_0, v_0) + b\mathbf{N_1} \cdot \mathbf{S_v}(u_0, v_0) \\ a\mathbf{N_2} \cdot \mathbf{S_u}(u_0, v_0) + b\mathbf{N_2} \cdot \mathbf{S_v}(u_0, v_0) \end{pmatrix} =$$
$$\begin{pmatrix} \mathbf{N_1} \cdot (a\mathbf{S_u}(u_0, v_0) + b\mathbf{S_v}(u_0, v_0)) \\ \mathbf{N_2} \cdot (a\mathbf{S_u}(u_0, v_0) + b\mathbf{S_v}(u_0, v_0)) \end{pmatrix} = \begin{pmatrix} \mathbf{N_1} \cdot \mathbf{A} \\ \mathbf{N_2} \cdot \mathbf{A} \end{pmatrix} = 0$$

where $\mathbf{A} = a\mathbf{S_u}(u_0, v_0) + b\mathbf{S_v}(u_0, v_0)$ is in the tangent plane. Since $\mathbf{N_1}$ and $\mathbf{N_2}$ are perpendicular to both $\hat{\mathbf{d}}$ and $\mathbf{A}$, $\hat{\mathbf{d}}$ is parallel to $\mathbf{A}$ and is therefore parallel to the tangent plane. Then, $\hat{\mathbf{d}}$ is parallel to a silhouette ray at $\mathbf{S}(u_0, v_0)$.

Now, suppose that $\hat{\mathbf{d}}$ is parallel to a silhouette ray at $\mathbf{S}(u_0, v_0)$. Thus, $\hat{\mathbf{d}} \cdot (\mathbf{S_u}(u_0, v_0) \times \mathbf{S_v}(u_0, v_0)) = 0$, and $\hat{\mathbf{d}}$ can be written as the linear combination $\hat{\mathbf{d}} = a\mathbf{S_u}(u_0, v_0) + b\mathbf{S_v}(u_0, v_0)$. Recalling that $\hat{\mathbf{d}}$ is perpendicular to both $\mathbf{N_1}$ and $\mathbf{N_2}$, we have

$$\begin{aligned} \mathbf{N_1} \cdot \hat{\mathbf{d}} &= \mathbf{N_1} \cdot (a\mathbf{S_u}(u_0, v_0) + b\mathbf{S_v}(u_0, v_0)) = a\mathbf{N_1} \cdot \mathbf{S_u}(u_0, v_0) + b\mathbf{N_1} \cdot \mathbf{S_v}(u_0, v_0) = 0 \\ \mathbf{N_2} \cdot \hat{\mathbf{d}} &= \mathbf{N_2} \cdot (a\mathbf{S_u}(u_0, v_0) + b\mathbf{S_v}(u_0, v_0)) = a\mathbf{N_2} \cdot \mathbf{S_u}(u_0, v_0) + b\mathbf{N_2} \cdot \mathbf{S_v}(u_0, v_0) = 0 \end{aligned}$$

which implies linear dependence of the columns of $\mathbf{J}(u_0, v_0)$. Thus, $\mathbf{J}(u_0, v_0)$ is singular. □

APPENDIX C

DERIVATION OF POTENTIAL EQUATIONS IN THE PLANE

The potential for a charged boundary can be summarized using

$$\int_\gamma \frac{d\gamma}{||P-\gamma(t)||^m}$$

If the boundary is piecewise linear, then the generalized Newtonian potential due to each segment $\gamma_i(t) = P_i + t(P_{i+1} - P_i)$ is given by

$$\begin{aligned} \Phi(P) &= \int_0^1 \frac{||\Delta P_i||}{||P-\gamma(t)||^m} dt \\ &= \int_0^1 \frac{||\Delta P_i||}{||P-P_i-t(P_{i+1}-P_i)||^m} dt \end{aligned} \tag{C.1}$$

where $\Delta P_i \equiv P_{i+1} - P_i$. Since

$$\begin{aligned} ||P-\gamma(t)|| &= \langle (P-P_i) - t\Delta P_i, (P-P_i) - t\Delta P_i\rangle^{1/2} \\ &= (\langle P-P_i, P-P_i\rangle - 2t\langle P-P_i, \Delta P_i\rangle + t^2\langle \Delta P_i, \Delta P_i\rangle)^{1/2} \\ &= (||P-P_i||^2 - 2t\langle P-P_i, \Delta P_i\rangle + t^2||\Delta P_i||^2)^{1/2}, \end{aligned}$$

$$
\begin{aligned}
||P-\gamma(t)||^m &= (||P-P_i||^2 - 2t\langle P-P_i, \Delta P_i\rangle + t^2||\Delta P_i||^2)^{m/2} \\
&= ||\Delta P_i||^m \left(\frac{||P-P_i||^2}{||\Delta P_i||^2} - 2t\frac{\langle P-P_i, \Delta P_i\rangle}{||\Delta P_i||^2} + t^2\right)^{m/2} \\
&= ||\Delta P_i||^m \left[\left(t - \frac{\langle P-P_i, \Delta P_i\rangle}{||\Delta P_i||^2}\right)^2 - \frac{\langle P-P_i, \Delta P_i\rangle^2}{||\Delta P_i||^4} + \frac{||P-P_i||^2}{||\Delta P_i||^2}\right]^{m/2} \\
&= ||\Delta P_i||^m \left[\left(t - \frac{\langle P-P_i, \Delta P_i\rangle}{||\Delta P_i||^2}\right)^2 + \frac{A}{||\Delta P_i||^4}\right]^{m/2} \\
&= \frac{||\Delta P_i||^m A^{m/2}}{||\Delta P_i||^{2m}} \left[\frac{||\Delta P_i||^4}{A}\left(t - \frac{\langle P-P_i, \Delta P_i\rangle}{||\Delta P_i||^2}\right)^2 + 1\right]^{m/2} \\
&= \frac{A^{m/2}}{||\Delta P_i||^m} \left[\left(\frac{||\Delta P_i||^2}{A^{1/2}}t - \frac{\langle P-P_i, \Delta P_i\rangle}{A^{1/2}}\right)^2 + 1\right]^{m/2} \qquad \text{(C.2)}
\end{aligned}
$$

where $A \equiv ||P-P_i||^2||\Delta P_i||^2 - \langle P-P_i, \Delta P_i\rangle^2$.

Returning to the original integral in Equation (C.1), if we apply the substitution

$$u = \frac{||\Delta P_i||^2}{A^{1/2}}t - \frac{\langle P-P_i, \Delta P_i\rangle}{A^{1/2}},$$

and the result Equation (C.2), we obtain

$$
\begin{aligned}
\Phi(P) &= \int_0^1 \frac{||\Delta P_i||}{||P-P_i-t(P_{i+1}-P_i)||^m} dt \\
&= \int_0^1 \frac{||\Delta P_i||}{\frac{A^{m/2}}{||\Delta P_i||^m}(1+u^2)^{m/2}} dt \\
&= \int_{\underline{u}_i(P)}^{\overline{u}_i(P)} \frac{||\Delta P_i||}{\frac{A^{m/2}}{||\Delta P_i||^m}(1+u^2)^{m/2}} \frac{A^{1/2}}{||\Delta P_i||^2} du \\
&= \frac{||\Delta P_i||^{m-1}}{A^{(m-1)/2}} \int_{\underline{u}_i(P)}^{\overline{u}_i(P)} \frac{du}{(1+u^2)^{m/2}} \\
&= \frac{||\Delta P_i||^{m-1}}{A^{(m-1)/2}} F_i(P)\Big|_{\underline{u}_i(P)}^{\overline{u}_i(P)}
\end{aligned}
$$

where

$$\underline{u}_i(P) = -\frac{\langle P - P_i, \Delta P_i \rangle}{A^{1/2}} \tag{C.3}$$

$$\overline{u}_i(P) = \frac{||\Delta P_i||^2}{A^{1/2}} + \underline{u}_i(P) \tag{C.4}$$

and

$$F_i(P) = \int \frac{du}{(1+u^2)^{m/2}}$$

$$= \begin{cases} u \sum_{k=0}^{n-1} \frac{\prod_{j=1}^{k} 2(n-j)}{(1+u^2)^{n-k-1/2} \prod_{j=0}^{k} 2(n-j)-1} & m = 2n+1 \text{ (odd)} \\ u \sum_{k=0}^{n-2} \frac{\prod_{j=1}^{k} 2(n-j)-1}{(1+u^2)^{n-k-1} \prod_{j=0}^{k} 2(n-j-1)} + \frac{\prod_{j=1}^{n-2} 2(n-j)-1}{\prod_{j=0}^{n-2} 2(n-j-1)} \arctan(u) & m = 2n \text{ (even)} \end{cases}$$

So there is a closed, albeit somewhat complicated closed form for the generalized potential due to a polygonal boundary. The force exerted on a point in space due to the charged boundary is given by $\nabla\Phi(P) = (\frac{d}{dx}\Phi(P), \frac{d}{dy}\Phi(P))$. Applying the chain rule, we have

$$\frac{d}{dx}\Phi(P) = ||\Delta P_i||^{m-1} \left[A^{(1-m)/2} \left(\frac{d}{dx} \int_{\underline{u}_i(P)}^{\overline{u}_i(P)} \frac{du}{(u^2+1)^{m/2}} \right) + \left(\frac{d}{dx} A^{(1-m)/2} \right) \int_{\underline{u}_i(P)}^{\overline{u}_i(P)} \frac{du}{(u^2+1)^{m/2}} \right]$$

$$\frac{d}{dy}\Phi(P) = ||\Delta P_i||^{m-1} \left[A^{(1-m)/2} \left(\frac{d}{dy} \int_{\underline{u}_i(P)}^{\overline{u}_i(P)} \frac{du}{(u^2+1)^{m/2}} \right) + \left(\frac{d}{dy} A^{(1-m)/2} \right) \int_{\underline{u}_i(P)}^{\overline{u}_i(P)} \frac{du}{(u^2+1)^{m/2}} \right]$$

Taking these terms componentwise, we have

$$\begin{aligned}
\frac{d}{dx}\int_{\underline{u}_i(P)}^{\overline{u}_i(P)} \frac{du}{(u^2+1)^{m/2}} &= \frac{du}{dx}\frac{d}{du}\int_{\underline{u}_i(P)}^{\overline{u}_i(P)} \frac{du}{(u^2+1)^{m/2}} \\
&= \frac{du}{dx}\left[\frac{1}{(1+u^2)^{m/2}}\right]\Bigg|_{\underline{u}_i}^{\overline{u}_i} \\
&= \frac{1}{(1+\overline{u}_i^2)^{m/2}}\frac{d\overline{u}_i}{dx} - \frac{1}{(1+\underline{u}_i^2)^{m/2}}\frac{d\underline{u}_i}{dx} \\
\frac{d}{dy}\int_{\underline{u}_i(P)}^{\overline{u}_i(P)} \frac{du}{(u^2+1)^{m/2}} &= \frac{du}{dy}\frac{d}{du}\int_{\underline{u}_i(P)}^{\overline{u}_i(P)} \frac{du}{(u^2+1)^{m/2}} \\
&= \frac{du}{dy}\left[\frac{1}{(1+u^2)^{m/2}}\right]\Bigg|_{\underline{u}_i}^{\overline{u}_i} \\
&= \frac{1}{(1+\overline{u}_i^2)^{m/2}}\frac{d\overline{u}_i}{dy} - \frac{1}{(1+\underline{u}_i^2)^{m/2}}\frac{d\underline{u}_i}{dy}
\end{aligned}$$

We can simplify A as well

$$\begin{aligned}
A &= ||P-P_i||^2||\Delta P_i||^2 - \langle P-P_i, \Delta P_i\rangle^2 \\
&= [(x-x_i)^2+(y-y_i)^2][\Delta x_i^2+\Delta y_i^2] - [(x-x_i)\Delta x_i + (y-y_i)\Delta y_i]^2 \\
&= (x-x_i)^2\Delta y_i^2 + (y-y_i)\Delta x_i^2 - 2(x-x_i)(y-y_i)\Delta x_i\Delta y_i \\
&= [(x-x_i)\Delta y_i - (y-y_i)\Delta x_i]^2 \\
&= [\langle (P-P_i), N_i\rangle]^2
\end{aligned}$$

The final statement gives some measure of geometric interpretation to A. Recalling the definitions of $\underline{u}_i$ (Equation (C.3)) and $\overline{u}_i$ (Equation (C.4)), substituting for A, we have

$$\begin{aligned}
\underline{u}_i &= -\frac{\langle P-P_i, \Delta P_i\rangle}{A^{1/2}} \\
&= -\frac{(x-x_i)\Delta x_i + (y-y_i)\Delta y_i}{|(x-x_i)\Delta y_i - (y-y_i)\Delta x_i|} \\
&= -\frac{\langle P-P_i, T_i\rangle}{|\langle P-P_i, N_i\rangle|} \\
&= -\cos\theta_i
\end{aligned}$$

and

$$\begin{aligned}
\overline{u}_i &= \frac{||\Delta P_i||^2}{|(x-x_i)\Delta y_i - (y-y_i)\Delta x_i|} + \underline{u}_i \\
&= \frac{\langle \Delta P_i, \Delta P_i \rangle - \langle P - P_i, \Delta P_i \rangle}{|(x-x_i)\Delta y_i - (y-y_i)\Delta x_i|} \\
&= \frac{\langle P_{i+1} - P, T_i \rangle}{|\langle P - P_i, N_i \rangle|} \\
&= -\cos\phi_{i+1}
\end{aligned}$$

If we define $B \equiv \langle P - P_i, N_i \rangle$, then $A^{1/2} = \mathrm{sgn}(B)B$, $\frac{dB}{dx} = \Delta y_i$, and $\frac{dB}{dy} = -\Delta x_i$. Taking derivatives of $\underline{u}_i$ and $\overline{u}_i$, we have

$$\begin{aligned}
\frac{d\underline{u}_i}{dx} &= \frac{d}{dx}\left[-\frac{(x-x_i)\Delta x_i + (y-y_i)\Delta y_i}{B\mathrm{sgn}(B)}\right] \\
&= -\frac{B\mathrm{sgn}(B)\Delta x_i - [(x-x_i)\Delta x_i + (y-y_i)\Delta y_i]\Delta y_i \mathrm{sgn}(B)}{B^2} \\
&= -\frac{\mathrm{sgn}(B)[(x-x_i)\Delta y_i \Delta x_i - (y-y_i)\Delta x_i^2 - (x-x_i)\Delta x_i \Delta y_i - (y-y_i)\Delta y_i^2]}{B^2} \\
&= \frac{\mathrm{sgn}(B)(y-y_i)(\Delta x_i^2 + \Delta y_i^2)}{B^2}
\end{aligned}$$

$$\begin{aligned}
\frac{d\underline{u}_i}{dy} &= \frac{d}{dy}\left[-\frac{(x-x_i)\Delta x_i + (y-y_i)\Delta y_i}{B\mathrm{sgn}(B)}\right] \\
&= -\frac{B\mathrm{sgn}(B)\Delta y_i - \frac{dB}{dy}\mathrm{sgn}(B)[(x-x_i)\Delta x_i + (y-y_i)\Delta y_i]}{B^2} \\
&= -\frac{[(x-x_i)\Delta y_i - (y-y_i)\Delta x_i]\mathrm{sgn}(B)\Delta y_i}{B^2} + \\
&\quad -\frac{[(x-x_i)\Delta x_i + (y-y_i)\Delta y_i]\mathrm{sgn}(B)\Delta x_i}{B^2} \\
&= -\frac{\mathrm{sgn}(B)[(x-x_i)\Delta y_i^2 - (y-y_i)\Delta x_i \Delta y_i + (x-x_i)\Delta x_i^2 + (y-y_i)\Delta x_i \Delta y_i]}{B^2} \\
&= \frac{\mathrm{sgn}(B)(x_i-x)(\Delta x_i^2 + \Delta y_i^2)}{B^2}
\end{aligned}$$

$$\begin{aligned}
\frac{d\overline{u}_i}{dx} &= \frac{d}{dx}\left(\frac{||\Delta P_i||^2}{B\text{sgn}(B)} + \underline{u}_i\right) \\
&= -\frac{\text{sgn}(B)\Delta y_i||\Delta P_i||^2}{B^2} + \frac{d\underline{u}_i}{dx} \\
&= \frac{\text{sgn}(B)[-\Delta y_i||\Delta P_i||^2 + (y-y_i)(\Delta x_i^2 + \Delta y_i^2)]}{B^2} \\
&= \frac{\text{sgn}(B)[(-y_{i+1}+y_i)(\Delta x_i^2 + \Delta y_i^2) + (y-y_i)(\Delta x_i^2 + \Delta y_i^2)]}{B^2} \\
&= \frac{\text{sgn}(B)(y-y_{i+1})(\Delta x_i^2 + \Delta y_i^2)}{B^2}
\end{aligned}$$

$$\begin{aligned}
\frac{d\overline{u}_i}{dy} &= \frac{d}{dy}\left(\frac{||\Delta P_i||^2}{B\text{sgn}(B)} + \underline{u}_i\right) \\
&= -\frac{||\Delta P_i||^2 \frac{dB}{dy}\text{sgn}(B)}{B^2} + \frac{d\underline{u}_i}{dy} \\
&= -\frac{||\Delta P_i||^2(-\Delta x_i)\text{sgn}(B)}{B^2} + \frac{\text{sgn}(B)(x_i-x)(\Delta x_i^2 + \Delta y_i^2)}{B^2} \\
&= \frac{\text{sgn}(B)[(x_{i+1}-x_i)(\Delta x_i^2 + \Delta y_i^2) + (x_i-x)(\Delta x_i^2 + \Delta y_i^2)]}{B^2} \\
&= \frac{\text{sgn}(B)(x_{i+1}-x)(\Delta x_i^2 + \Delta y_i^2)}{B^2}
\end{aligned}$$

$$\begin{aligned}
\frac{dA}{dx} &= 2[(x-x_i)\Delta y_i - (y-y_i)\Delta x_i]\Delta y_i \\
&= \text{sgn}(B)2\Delta y_i A^{1/2} \\
\frac{dA^m}{dx} &= mA^{m-1}\frac{dA}{dx} \\
&= mA^{m-1}\text{sgn}(B)2\Delta y_i A^{1/2} \\
&= \text{sgn}(B)2m\Delta y_i A^{m-1/2}
\end{aligned}$$

Putting it all together, we have

$$
\begin{aligned}
&\frac{d}{dx}\Phi(P) \\
&= ||\Delta P_i||^{m-1}\left[A^{(1-m)/2}\left(\frac{d}{dx}\int_{\underline{u}_i(P)}^{\overline{u}_i(P)}\frac{du}{(u^2+1)^{m/2}}\right)+\right. \\
&\quad \left.\left(\frac{d}{dx}A^{(1-m)/2}\right)\int_{\underline{u}_i(P)}^{\overline{u}_i(P)}\frac{du}{(u^2+1)^{m/2}}\right] \\
&= ||\Delta P_i||^{m-1}\left[A^{(1-m)/2}\left(\frac{1}{(1+\overline{u}_i^2)^{m/2}}\frac{d\overline{u}_i}{dx}-\frac{1}{(1+\underline{u}_i^2)^{m/2}}\frac{d\underline{u}_i}{dx}\right)+\right. \\
&\quad \left.\mathrm{sgn}(B)2\Delta y_i\left(\frac{1-m}{2}\right)A^{-m/2}\,F(P)|_{\underline{u}_i}^{\overline{u}_i}\right] \\
&= ||\Delta P_i||^{m-1}\left[A^{(1-m)/2}\left(\frac{1}{(1+\overline{u}_i^2)^{m/2}}\frac{\mathrm{sgn}(B)(y-y_{i+1})(\Delta x_i^2+\Delta y_i^2)}{B^2}-\right.\right. \\
&\quad \left.\left.\frac{1}{(1+\underline{u}_i^2)^{m/2}}\frac{\mathrm{sgn}(B)(y-y_i)(\Delta x_i^2+\Delta y_i^2)}{B^2}\right)+\right. \\
&\quad \left.\mathrm{sgn}(B)(1-m)\Delta y_i A^{-m/2}\,F(P)|_{\underline{u}_i}^{\overline{u}_i}\right] \\
&= \mathrm{sgn}(B)\left[\frac{||\Delta P_i||^{m+1}}{A^{(m-1)/2}A}\left(\frac{(y-y_{i+1})}{(1+\overline{u}_i^2)^{m/2}}+\frac{(y_i-y)}{(1+\underline{u}_i^2)^{m/2}}\right)+\right. \\
&\quad \left.(1-m)\Delta y_i\frac{||\Delta P_i||^{m-1}}{A^{m/2}}\,F(P)|_{\underline{u}_i}^{\overline{u}_i}\right] \\
&= \mathrm{sgn}(B)\left[\frac{||\Delta P_i||^{m+1}}{A^{(m+1)/2}}\left(\frac{(y-y_{i+1})}{(1+\overline{u}_i^2)^{m/2}}+\frac{(y_i-y)}{(1+\underline{u}_i^2)^{m/2}}\right)+(1-m)\Delta y_i A^{-1/2}\Phi(P)\right]
\end{aligned}
$$

$$
\begin{aligned}
&\frac{d}{dy}\Phi(P) \\
&= ||\Delta P_i||^{m-1}\left[A^{(1-m)/2}\left(\frac{d}{dy}\int_{\underline{u}_i(P)}^{\overline{u}_i(P)}\frac{du}{(u^2+1)^{m/2}}\right)+\right. \\
&\quad \left.\left(\frac{d}{dy}A^{(1-m)/2}\right)\int_{\underline{u}_i(P)}^{\overline{u}_i(P)}\frac{du}{(u^2+1)^{m/2}}\right] \\
&= ||\Delta P_i||^{m-1}\left[A^{(1-m)/2}\left(\frac{1}{(1+\overline{u}_i^2)^{m/2}}\frac{\mathrm{sgn}(B)(\Delta x_i^2+\Delta y_i^2)(x_{i+1}-x)}{B^2}-\right.\right. \\
&\quad \left.\left.\frac{1}{(1+\underline{u}_i^2)^{m/2}}\frac{\mathrm{sgn}(B)(\Delta x_i^2+\Delta y_i^2)(x_i-x)}{B^2}\right)-\right. \\
&\quad \left.\mathrm{sgn}(B)(1-m)\Delta x_i A^{-m/2}F(P)\Big|_{\underline{u}_i}^{\overline{u}_i}\right] \\
&= \mathrm{sgn}(B)\left[\frac{||\Delta P_i||^{m+1}}{A^{(m+1)/2}}\left(\frac{x_{i+1}-x}{(1+\overline{u}_i^2)^{m/2}}+\frac{x-x_i}{(1+\underline{u}_i^2)^{m/2}}\right)-\right. \\
&\quad \left.(1-m)\Delta x_i\frac{||\Delta P_i||^{m-1}}{A^{m/2}}F(P)\Big|_{\underline{u}_i}^{\overline{u}_i}\right]- \\
&= \mathrm{sgn}(B)\left[\frac{||\Delta P_i||^{m+1}}{A^{(m+1)/2}}\left(\frac{x-x_{i+1}}{(1+\overline{u}_i^2)^{m/2}}+\frac{x_i-x}{(1+\underline{u}_i^2)^{m/2}}\right)+(1-m)\Delta x_i A^{-1/2}\Phi(P)\right]
\end{aligned}
$$

The Jacobian of $\nabla\Phi$ is given by

$$
\begin{bmatrix} \frac{d}{dx}\frac{d}{dx}\Phi & \frac{d}{dy}\frac{d}{dx}\Phi \\ \frac{d}{dx}\frac{d}{dy}\Phi & \frac{d}{dy}\frac{d}{dy}\Phi \end{bmatrix}
$$

We need only to calculate $\frac{d}{dx}\frac{d}{dx}\Phi$ and $\frac{d}{dy}\frac{d}{dx}\Phi$, and $\frac{d}{dy}\frac{d}{dy}\Phi$.

$$
\begin{aligned}
\frac{d}{dx}\frac{d}{dx}\Phi &= \mathrm{sgn}(B)\left[||\Delta P_i||^{m+1}\left(\frac{d}{dx}A^{-(m+1)/2}\right)\left(\frac{y-y_{i+1}}{(1+\overline{u}_i^2)^{m/2}}+\frac{y_i-y}{(1+\underline{u}_i^2)^{m/2}}\right)+\right. \\
&\quad ||\Delta P_i||^{m+1}A^{-(m+1)/2}\frac{d}{dx}\left(\frac{y-y_{i+1}}{(1+\overline{u}_i^2)^{m/2}}+\frac{y_i-y}{(1+\underline{u}_i^2)^{m/2}}\right)+ \\
&\quad (1-m)\Delta y_i\left(\frac{d}{dx}A^{-1/2}\right)\Phi(P)+ \\
&\quad \left.(1-m)\Delta y_i A^{-1/2}\left(\frac{d}{dx}\Phi(P)\right)\right]
\end{aligned}
$$

We have that

$$\frac{d}{dx}[(y-y_{i+1})(1+\overline{u}_i^2)^{-m/2}+(y_i-y)(1+\underline{u}_i^2)^{-m/2}] = m(y_{i+1}-y)(1+\overline{u}_i^2)^{-(m+2)/2}\overline{u}_i\frac{d\overline{u}_i}{dx}+m(y-y_i)(1+\underline{u}_i^2)^{-(m+2)/2}\underline{u}_i\frac{d\underline{u}_i}{dx}$$

$$\begin{aligned}\frac{d}{dy}\frac{d}{dx}\Phi &= -\text{sgn}(B)\left[||\Delta P_i||^{m+1}\left(\frac{d}{dy}A^{-(m+1)/2}\right)\left(\frac{y-y_{i+1}}{(1+\overline{u}_i^2)^{m/2}}+\frac{y_i-y}{(1+\underline{u}_i^2)^{m/2}}\right)+\right.\\ &||\Delta P_i||^{m+1}A^{-(m+1)/2}\frac{d}{dy}\left(\frac{y-y_{i+1}}{(1+\overline{u}_i^2)^{m/2}}+\frac{y_i-y}{(1+\underline{u}_i^2)^{m/2}}\right)+\\ &(1-m)\Delta y_i\left(\frac{d}{dy}A^{-1/2}\right)\Phi(P)+\\ &\left.(1-m)\Delta y_i A^{-1/2}\left(\frac{d}{dy}\Phi(P)\right)\right]\end{aligned}$$

Here, we have

$$\begin{aligned}\frac{d}{dy}[(y-y_{i+1})(1+\overline{u}_i^2)^{-m/2}+(y_i-y)(1+\underline{u}_i^2)^{-m/2}] = &\\ (1+\overline{u}_i^2)^{-m/2}+m(y_{i+1}-y)\overline{u}_i(1+\overline{u}_i^2)^{-(m+2)/2}\frac{d\overline{u}_i}{dy}+&\\ (1+\underline{u}_i^2)^{-m/2}+m(y-y_i)\underline{u}_i(1+\underline{u}_i^2)^{-(m+2)/2}\frac{d\underline{u}_i}{dy}&\end{aligned}$$

Finally,

$$\begin{aligned}\frac{d}{dy}\frac{d}{dy}\Phi &= -\text{sgn}(B)\left[||\Delta P_i||^{m+1}\left(\frac{d}{dy}A^{-(m+1)/2}\right)\left(\frac{x-x_{i+1}}{(1+\overline{u}_i^2)^{m/2}}+\frac{x_i-x}{(1+\underline{u}_i^2)^{m/2}}\right)+\right.\\ &||\Delta P_i||^{m+1}A^{-(m+1)/2}\frac{d}{dy}\left(\frac{x-x_{i+1}}{(1+\overline{u}_i^2)^{m/2}}+\frac{x_i-x}{(1+\underline{u}_i^2)^{m/2}}\right)+\\ &(1-m)\Delta x_i\left(\frac{d}{dy}A^{-1/2}\right)\Phi(P)+\\ &\left.(1-m)\Delta x_i A^{-1/2}\left(\frac{d}{dy}\Phi(P)\right)\right]\end{aligned}$$

where

$$\frac{d}{dy}[(x - x_{i+1})(1 + \overline{u}_i^2)^{-m/2} + (x_i - x)(1 + \underline{u}_i^2)^{-m/2}] =$$
$$m(x_{i+1} - x)\overline{u}_i(1 + \overline{u}_i^2)^{-(m+2)/2}\frac{d\overline{u}_i}{dy} + m(x - x_i)\underline{u}_i(1 + \underline{u}_i^2)^{-(m+2)/2}\frac{d\underline{u}_i}{dy}$$

REFERENCES

[1] S. Parker, W. Martin, P.-P. J. Sloan, P. Shirley, B. Smits, and C. Hansen, "Interactive ray tracing," *1999 ACM Symposium on Interactive 3D Graphics*, pp. 119–126, April 1999. ISBN 1-58113-082-1.

[2] Integrated Graphics Modeling Design and Manufacturing Research Group, *Alpha1 Geometric Modeling System, User's Manual.* Department of Computer Science, University of Utah, November 1988.

[3] R. F. Riesenfeld, *Applications of B-Spline Approximation to Geometric Problems of Computer-Aided Design.* PhD thesis, Department of Computer Science, Syracuse University, 1972.

[4] J. Cottrell, T. Hughes, and Y. Bazilevs, *Isogeometric Analysis: Toward Integration of CAD and FEA*. John Wiley & Sons, 2009.

[5] E. Cohen, R. F. Riesenfeld, and G. Elber, *Geometric Modeling with Splines, An Introduction.* Natick, MA 01760: A K Peters, Ltd., 2001.

[6] S. Suresh, "Graded materials for resistance to contact deformation and damage," *Science*, vol. 292, pp. 2447–51, June 2001.

[7] E. M. Sachs, J. Haggerty, M. Cima, and P. Williams, "Three dimensional printing techniques." U. S. Patent No. 5,204,055, April 1993.

[8] M. do Carmo, *Differential geometry of curves and surfaces.* Prentice-Hall, 1976.

[9] R. L. Cook, L. Carpenter, and E. Catmull, "The Reyes image rendering architecture," *SIGGRAPH Comput. Graph.*, vol. 21, pp. 95–102, Aug. 1987.

[10] D. P. Greenberg, K. E. Torrance, P. Shirley, J. Arvo, E. Lafortune, J. A. Ferwerda, B. Walter, B. Trumbore, S. Pattanaik, and S.-C. Foo, "A framework for realistic image synthesis," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, (New York, NY, USA), pp. 477–494, ACM Press/Addison-Wesley Publishing Co., 1997.

[11] C. M. Goral, K. E. Torrance, D. P. Greenberg, and B. Battaile, "Modeling the interaction of light between diffuse surfaces," in *Proceedings of SIGGRAPH '84*, pp. 212–22, 1984.

[12] F. X. Sillion, J. R. Arvo, S. H. Westin, and D. P. Greenberg, "A global illumination solution for general reflectance distributions," in *Proceedings of SIGGRAPH '91*, pp. 187–196, 1991.

[13] B. Smits, J. Arvo, and D. Greenberg, "A clustering algorithm for radiosity in complex

environments," in *Proceedings of SIGGRAPH 94*, pp. 435–442, 1994.

[14] S. S. Abi-Ezzi and S. Subramaniam, "Fast dynamic tessellation of trimmed NURBS surfaces," in *Eurographics '94*, 1994.

[15] S. Kumar, D. Manocha, and A. Lastra, "Interactive display of large NURBS models," *IEEE Transactions on Visualization and Computer Graphics*, vol. 2, December 1996.

[16] J. W. Peterson, "Tessellation of NURBS surfaces," in *Graphics Gems IV* (P. Heckbert, ed.), pp. 286–320, Boston: Academic Press, 1994.

[17] A. Rockwood, K. Heaton, and T. Davis, "Real-time rendering of trimmed surfaces," in *SIGGRAPH '89*, 1989.

[18] L. A. Piegl and A. M. Richard, "Tessellating trimmed NURBS surfaces," *Computer-Aided Design*, vol. 27, no. 1, pp. 16–26, 1995.

[19] W. Barth and W. Stürzlinger, "Efficient ray tracing for Bézier and B-spline surfaces," *Computers and Graphics*, vol. 17, no. 4, 1993.

[20] S. Campagna, P. Slusallek, and H.-P. Seidel, "Ray tracing of spline surfaces: Bézier clipping, Chebyshev boxing, and bounding volume hierarchy – a critical comparison with new results," tech. rep., University of Erlangen, IMMD IX, Computer Graphics Group, Am Weichselgarten 9, D-91058 Erlangen, Germany, October 1996.

[21] A. Fournier and J. Buchanan, "Chebyshev polynomials for boxing and intersections of parametric curves and surfaces," in *Eurographics '94*, 1994.

[22] J. T. Kajiya, "Ray tracing parametric patches," *Computer Graphics (SIGGRAPH '82 Proceedings)*, vol. 16, pp. 245–254, July 1982.

[23] D. Lischinski and J. Gonczarowski, "Improved techniques for ray tracing parametric surfaces," *The Visual Computer*, vol. 6, no. 3, pp. 134–152, June 1990. ISSN 0178-2789.

[24] T. Nishita, T. W. Sederberg, and M. Kakimoto, "Ray tracing trimmed rational surface patches," in *SIGGRAPH '90*, 1990.

[25] M. A. Sweeney and R. H. Bartels, "Ray tracing free-form B-spline surfaces," *CG&A*, vol. 6, February 1986.

[26] D. L. Toth, "On ray tracing parametric surfaces," *Computer Graphics (Proceedings of SIGGRAPH 85)*, vol. 19, no. 3, pp. 171–179, July 1985. Held in San Francisco, California.

[27] C.-G. Yang, "On speeding up ray tracing of B-spline surfaces," *Computer Aided Design*, vol. 19, April 1987.

[28] R. Farouki and J. Hinds, "A hierarchy of geometric forms," *IEEE Computer Graphics & Applications*, vol. 5, no. 5, pp. 51–78, 1985.

[29] D. Lasser, "Bernstein-Bézier representation of volumes," *Computer Aided Geometric Design*, vol. 2, no. 1-3, pp. 145–150, 1985.

[30] K. L. Paik, "Trivariate splines," Master's thesis, Department of Computer Science, University of Utah, December 1992.

[31] C. Madrigal and K. I. Joy, "Generating the envelope of a swept trivariate solid," in *Proceedings of the IASTED Internation Conference on Computer Graphics and Imaging*, pp. 5–9, 1999.

[32] K. Joy and M. Duchaineau, "Boundary determination for trivariate solids," *Pacific Graphics '99*, October 1999. Held in Seoul, Korea.

[33] S. W. Wang and A. E. Kaufman, "Volume sculpting," *1995 Symposium on Interactive 3D Graphics*, pp. 151–156, April 1995. ISBN 0-89791-736-7.

[34] E. L. Stanton, L. M. Crain, and T. F. Neu, "A parametric cubic modelling system for general solids of composite material," *International Journal for Numerical Methods in Engineering*, vol. 11, pp. 653–670, 1977.

[35] M. S. Casale and E. L. Stanton, "An overview of analytic solid modeling," *IEEE Computer Graphics and Applications*, vol. 5, no. 2, pp. 45–56, 1985.

[36] W.-C. J. Yen, *On Representation and Discretization of Finite Element Analyses*. PhD thesis, Department of Computer Science, University of Utah, December 1985.

[37] R. R. Dickinson, R. H. Bartels, and A. H. Vermeulen, "The interactive editing and contouring of empirical fields," *IEEE Computer Graphics & Applications*, vol. 9, pp. 34–43, May 1989.

[38] A. Kaufman, "Efficient algorithms for 3D scan-conversion of parametric curves, surfaces, and volumes," *Computer Graphics (Proceedings of SIGGRAPH 87)*, vol. 21, pp. 171–179, July 1987. Held in Anaheim, California.

[39] K. S. Bonnell, D. R. Schikore, K. I. Joy, M. Duchaineau, and B. Hamann, "Constructing material interfaces from data sets with volume-fraction information," *IEEE Visualization 2000*, pp. 367–372, October 2000. ISBN 0-7803-6478-3.

[40] A. Raviv and G. Elber, "Three-dimensional freeform sculpting via zero sets of scalar trivariate functions," *Computer-Aided Design*, vol. 32, pp. 513–526, August 2000. ISSN 0010-4485.

[41] A. Raviv and G. Elber, "Interactive direct rendering of trivariate B-spline scalar functions," *IEEE Transactions on Visualization and Computer Graphics*, vol. 7, pp. 109–119, April - June 2001. ISSN 1077-2626.

[42] S. Park and K. Lee, "High-dimensional trivariate NURBS representation for analyzing and visualizing fluid flow data," *Computers & Graphics*, vol. 21, pp. 473–482, July 1997. ISSN 0097-8493.

[43] V. Kumar and D. Dutta, "An approach to modeling multi-material objects," in *Proceedings of the 1997 Solid Modeling Symposium*, 1997.

[44] C. M. Hoffmann, *Geometric and Solid Modeling*. Morgan Kaufmann Publishers, 1989.

[45] V. Kumar and D. Dutta, "An approach to modeling & representation of heterogeneous objects," *Journal of Mechanical Design*, vol. 120, pp. 659–667, December 1998.

[46] V. Kumar, D. Burns, D. Dutta, and C. Hoffman, "A framework for object modeling," *Computer-Aided Design*, vol. 31, pp. 541–556, 1999.

[47] A. Marsan and D. Dutta, "On the application of tensor product solids in heterogeneous solid modeling," in *Proceedings of the 1998 ASME Design Technical Conferences*, 1998.

[48] H. Liu, "Algorithms for design and interrogation of functionally graded material solids," Master's thesis, Department of Ocean Engineering and Department of Mechanical Engineering, Massachusetts Institute of Technology, June 2000.

[49] H. Liu, W. Cho, T. R. Jackson, and N. M. Patrikalakis, "Algorithms for design and interrogation of functionally gradient material objects," in *ASME Design and Automation Conference*, 2000.

[50] T. R. Jackson, *Analysis of Functionally Graded Material Object Representation Methods*. PhD thesis, Department of Ocean Engineering, Massachusetts Institute of Technology, June 2000.

[51] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *Computer Graphics (Proceedings of SIGGRAPH 87)*, vol. 21, pp. 163–169, July 1987. Held in Anaheim, California.

[52] R. A. Drebin, L. Carpenter, and P. Hanrahan, "Volume rendering," *Computer Graphics (Proceedings of SIGGRAPH 88)*, vol. 22, pp. 65–74, August 1988. Held in Atlanta, Georgia.

[53] P. Shirley and A. Tuchman, "A polygonal approximation to direct scalar volume rendering," *Computer Graphics (San Diego Workshop on Volume Visualization)*, vol. 24, pp. 63–70, November 1990.

[54] M. Levoy, "Efficient ray tracing of volume data," *ACM Transactions on Graphics*, vol. 9, pp. 245–261, July 1990. ISSN 0730-0301.

[55] N. Max, "Optical models for direct volume rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 1, pp. 99–108, June 1995. ISSN 1077-2626.

[56] Y.-K. Chang, A. P. Rockwood, and Q. He, "Direct rendering of freeform volumes," *Computer-aided Design*, vol. 27, no. 7, pp. 553–558, 1995.

[57] M. M. Madi and D. J. Walton, "Modeling and visualization of layered objects," *Computers & Graphics*, vol. 23, pp. 331–342, June 1999. ISSN 0097-8493.

[58] K. I. Joy and J. Conkey, "Using isosurface method for visualizing the envelope of a swept trivariate solid," *8th Pacific Conference on Computer Graphics and Applications*, pp. 272–280, October 2000. ISBN 0-7695-0868-5.

[59] T. Martin, E. Cohen, and R. M. Kirby, "Direct isosurface visualization of hex-based high-order geometry and attribute representations," *IEEE Transactions on Visualization and*

Computer Graphics, vol. 18, pp. 753–766, 2012.

[60] H. Blum, "A Transformation for Extracting New Descriptors of Shape," in *Models for the Perception of Speech and Visual Form* (W. Wathen-Dunn, ed.), pp. 362–380, Cambridge: MIT Press, 1967.

[61] H. Blum, "Biological shape and visual science (Part I)," *Journal of Theoretical Biology*, vol. 38, pp. 205–287, 1973.

[62] N. Ahuja and J.-H. Chuang, "Shape representation using a generalized potential field model," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, pp. 169–176, February 1997.

[63] J.-H. Chuang, "Potential-based modeling of three-dimensional workspace for obstacle avoidance," *IEEE Transactions on Robotics and Automation*, vol. 14, pp. 778–785, October 1998.

[64] J.-H. Chuang, C.-H. Tsai, and M.-C. Ko, "Skeletonization of three-dimensional object using generalized potential field," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, pp. 1241–1251, November 2000.

[65] T.-C. Lee, R. L. Kashyap, and C.-N. Chu, "Building skeleton models via 3-D medial surface/axis thinning algorithms," *CVGIP: Graph. Models Image Process.*, vol. 56, pp. 462–478, Nov. 1994.

[66] O. K.-C. Au, C.-L. Tai, H.-K. Chu, D. Cohen-Or, and T.-Y. Lee, "Skeleton extraction by mesh contraction," *ACM Transactions on Graphics*, vol. 27, pp. 44:1–44:10, Aug 2008.

[67] C. Yao and J. G. Rokne, "A straightforward algorithm for computing the medial axis of a simple polygon," *Internat. J. Comput. Math.*, vol. 39, pp. 51–60, 1991.

[68] F. Chin, J. Snoeyink, and C. A. Wang, "Finding the medial axis of a simple polygon in linear time," in *6th International Symposium on Algorithms and Computation*, 1995.

[69] F.-E. Wolter, "Cut locus and medial axis in global shape interrogation and representation," in *MIT Design Laboratory Memorandum 92-2 and MIT Sea Grant Report*, 1992.

[70] N. Amenta, S. Choi, and R. K. Kolluri, "The power crust," in *Proceedings of the Sixth ACM Symposium on Solid Modeling and Applications*, SMA '01, (New York, NY, USA), pp. 249–266, ACM, 2001.

[71] N. Amenta and R. K. Kolluri, "The medial axis of a union of balls," *Comput. Geom. Theory Appl*, vol. 20, pp. 25–37, 2001.

[72] N. Amenta, S. Choi, and R. K. Kolluri, "The power crust, unions of balls, and the medial axis transform," *Computational Geometry: Theory and Applications*, vol. 19, pp. 127–153, 2000.

[73] C. K. Chui and M.-J. Lai, "Filling polygonal holes using C^1 cubic triangular patches," *Computer-Aided Geometric Design*, vol. 17, pp. 297–307, 2000.

[74] S. A. Coons, "Surface patches and B-spline curves," in *Computer Aided Geometric Design*,

pp. 1–16, Academic Press, 1974.

[75] C. Loop and T. D. DeRose, "Generalized B-spline surfaces of arbitrary topology," in *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '90, (New York, NY, USA), pp. 347–356, ACM, 1990.

[76] M. A. Sabin, "Non-rectangular surfaces suitable for inclusion in a B-spline surface," in *Proceedings of Eurographics '83*, pp. 57–69, 1983.

[77] J. J. Zheng and A. A. Ball, "Control point surfaces over non-four-sided areas," *Computer-Aided Geometric Design*, vol. 14, pp. 807–821, 1997.

[78] T. Martin, E. Cohen, and M. Kirby, "Volumetric parameterization and trivariate B-spline fitting using harmonic functions," in *SPM '08: Proceedings of the 2008 ACM symposium on Solid and physical modeling*, (New York, NY, USA), pp. 269–280, ACM, 2008.

[79] T. Martin, E. Cohen, and R. M. Kirby, "Mixed-element volume completion from NURBS surfaces," *Computers & Graphics*, vol. 36, no. 5, 2012.

[80] M. S. Floater and K. Hormann, "Surface parameterization: a tutorial and survey," in *Advances in Multiresolution for Geometric Modelling* (N. A. Dodgson, M. S. Floater, and M. A. Sabin, eds.), pp. 157–186, Springer Verlag, 2005.

[81] A. Sheffer, E. Praun, and K. Rose, "Mesh parameterization methods and their applications," *Found. Trends. Comput. Graph. Vis.*, vol. 2, pp. 105–171, Jan. 2006.

[82] T. O. Binford, "Visual perception by computer," in *Proceedings of the IEEE Conference on Systems and Control (Miami, FL)*, 1971.

[83] J.-H. Chuang, N. Ahuja, C.-C. Lin, C.-H. Tsai, and C.-H. Chen, "Technical section: A potential-based generalized cylinder representation," *Comput. Graph.*, vol. 28, pp. 907–918, Dec. 2004.

[84] R. Troutman and N. L. Max, "Radiosity algorithms using higher order finite element methods," in *Proceedings of SIGGRAPH '93*, pp. 209–212, 1993.

[85] H. R. Zatz, "Galerkin radiosity: A higher order solution method for global illumination," in *Proceedings of SIGGRAPH '93*, pp. 213–220, 1993.

[86] G. J. Ward, F. M. Rubinstein, and R. D. Clear, "A ray tracing solution for diffuse interreflection," in *Proceedings of SIGGRAPH '88*, pp. 85–92, August 1988.

[87] G. Greger, P. Shirley, P. M. Hubbard, and D. P. Greenberg, "The irradiance volume," *IEEE Computer Graphics & Applications*, vol. 18, no. 2, pp. 32–43, 1998.

[88] B. Walter, P. M. Hubbard, P. Shirley, and D. F. Greenberg, "Global illumination using local linear density estimation," *ACM Transactions on Graphics*, vol. 16, pp. 217–259, July 1997.

[89] H. W. Jensen, *Realistic Image Synthesis Using Photon Mapping*. Natick, MA: A K Peters, 2001.

[90] B. Walter, G. Alppay, E. P. F. Lafortune, S. Fernandez, and D. P. Greenberg, "Fitting virtual lights for non-diffuse walkthroughs," in *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, pp. 45–48, August 1997.

[91] A. Fournier, "Separating reflection functions for linear radiosity," in *Eurographics Rendering Workshop 1995*, pp. 296–305, June 1995.

[92] M. D. McCool, J. Ang, and A. Ahmad, "Homomorphic factorization of BRDFs for high-performance rendering," in *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pp. 171–178, Aug 2001.

[93] L. Latta and A. Kolb, "Homomorphic factorization of brdf-based lighting computation," *ACM Transactions on Graphics*, vol. 21, pp. 509–516, July 2002.

[94] M. Stamminger, A. Scheel, X. Granier, F. Perez-Cazorla, G. Drettakis, and F. X. Sillion, "Efficient glossy global illumination with interactive viewing," *Computer Graphics Forum*, vol. 19, pp. 13–25, March 2000.

[95] B. Cabral, N. Max, and R. Springmeyer, "Bidirectional reflection functions from surface bump maps," in *Proceedings of SIGGRAPH '87*, pp. 273–281, July 1987.

[96] P.-P. Sloan, J. Kautz, and J. Snyder, "Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments," *ACM Transactions on Graphics*, vol. 21, pp. 527–536, July 2002.

[97] J. Kautz, P.-P. Sloan, and J. Snyder, "Fast arbitrary BRDF shading for low-frequency lighting using spherical harmonics," in *Rendering Techniques 2002*, Springer, 2002.

[98] R. Ramamoorthi and P. Hanrahan, "An efficient representation for irradiance environment maps," in *Proceedings of SIGGRAPH '01*, pp. 497–500, ACM Press, August 2001.

[99] D. N. Wood, D. I. Azuma, K. Aldinger, B. Curless, T. Duchamp, D. H. Salesin, and W. Stuetzle, "Surface light fields for 3D photography," in *Proceedings of ACM SIGGRAPH 2000*, pp. 287–296, ACM Press, July 2000.

[100] T. Malzbender, D. Gelb, and H. Wolters, "Polynomial texture maps," in *Proceedings of ACM SIGGRAPH 2001*, pp. 519–528, ACM Press, August 2001.

[101] P. Hanrahan, D. Salzman, and L. Aupperle, "A rapid hierarchical radiosity algorithm," in *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '91, (New York, NY, USA), pp. 197–206, ACM, 1991.

[102] L. Aupperle and P. Hanrahan, "A hierarchical illumination algorithm for surfaces with glossy reflection," in *Proceedings of SIGGRAPH 93*, Computer Graphics Proceedings, Annual Conference Series, pp. 155–162, August 1993.

[103] S. J. Gortler, P. Schröder, M. F. Cohen, and P. Hanrahan, "Wavelet radiosity," in *Proceedings of SIGGRAPH 93*, Computer Graphics Proceedings, Annual Conference Series, pp. 221–230, August 1993.

[104] Y. Yu and Q. Peng, "Multiresolution B-spline radiosity," *Computer Graphics Forum*,

vol. 14, no. 3, pp. 285–298, 1995.

[105] P. H. Christensen, E. J. Stollnitz, D. H. Salesin, and T. D. DeRose, "Wavelet radiance," in *Rendering Techniques '94*, pp. 287–302, Springer, 1994.

[106] K. Bala, J. Dorsey, and S. Teller, "Radiance interpolants for accelerated bounded-error ray tracing," *ACM Transactions on Graphics*, vol. 18, pp. 213–256, July 1999.

[107] R. A. Redner, M. E. Lee, and S. P. Uselton, "Smooth B-Spline illumination maps for bidirectional ray tracing," *ACM Trans. on Graphics*, vol. 14, no. 4, pp. 337–362, 1995.

[108] R. H. Bartels, J. C. Beatty, and B. A. Barsky, *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*. Los Altos, CA: Morgan Kauffman Publishers, Inc., 1987.

[109] G. E. Farin, *Curves and surfaces for computer aided geometric design: a practical guide, 4th ed.* San Diego, CA: Academic Press, Inc., 1996.

[110] J. Hoschek and D. Lasser, *Fundamentals of Computer Aided Geometric Design*. Wellesley, MA: A.K. Peters, 1993.

[111] L. A. Piegl and W. Tiller, *The NURBS Book*. New York, NY: Springer Verlag, 1997.

[112] W. Boehm, "Inserting new knots into B-spline curves," *Computer-Aided Design*, vol. 12, pp. 199–201, July 1980.

[113] E. Cohen, T. Lyche, and R. Riesenfeld, "Discrete B-splines and subdivision techniques in computer-aided geometric design and computer graphics," *Comput. Gr. Image Process.*, vol. 14, pp. 87–111, October 1980.

[114] T. Lyche, E. Cohen, and K. Morken, "Knot line refinement algorithms for tensor product B-spline surfaces," *Computer Aided Geometric Design*, vol. 2, no. 1-3, pp. 133–139, 1985.

[115] B. Smits, "Efficiency issues for ray tracing," *Journal of Graphics Tools*, vol. 3, no. 2, pp. 1–14, 1998. ISSN 1086-7651.

[116] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing (2nd ed.)*. Cambridge University Press, 1992. ISBN 0-521-43108-5. Held in Cambridge.

[117] A. Glassner, *An Introduction to Ray Tracing*. Morgan Kaufmann Series in Computer Graphics and Geometric Modeling, Academic Press, 1989.

[118] W. L. Luken and F. F. Cheng, "Comparison of surface and derivative evaluation methods for the rendering of NURB surfaces," *ACM Transactions on Graphics*, vol. 15, no. 2, pp. 153–178, April 1996. ISSN 0730-0301.

[119] T. V. Thompson II and E. Cohen, "Direct haptic rendering of complex NURBS models," in *Proceedings of Symposium on Haptic Interfaces*, 1999.

[120] J. Rokne, "The area of a simple polygon," in *Graphics Gems II* (J. R. Arvo, ed.), Academic Press, 1991.

[121] W. Martin, E. Cohen, R. Fish, and P. Shirley, "Practical ray tracing of trimmed NURBS surfaces," *Journal of Graphics Tools*, vol. 5, no. 1, pp. 27–52, 2000.

[122] B. O'Neill, *Elementary Differential Geometry*. Academic Press, 1966.

[123] T. Sederberg and R. Meyers, "Loop detection in surface patch intersections," *Computer Aided Geometric Design*, vol. 5, no. 2, pp. 161–171, 1988.

[124] G. Barequet and G. Elber, "Optimal bounding cones of vectors in three dimensions," *Inf Process. Lett.*, vol. 93, no. 2, pp. 83–89, 2005.

[125] J. A. Sethian, *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. New York: Cambridge University Press, 1999.

[126] R. Kimmel, D. Shaked, N. Kiryati, and A. M. Bruckstein, "Skeletonization via distance maps and level sets," *Computer Vision and Image Understanding*, vol. 62, pp. 382–391, 1995.

[127] D. Lutterkort and J. Peters, "Tight linear envelopes for splines," *Numerische Mathematik*, vol. 89, pp. 735–748, 2001.

[128] S. Mann and A. P. Rockwood, "Computing singularities of 3D vector fields with geometric algebra," in *IEEE Visualization*, 2002.

[129] D. Fontijne, "Gaigen 2: a geometric algebra implementation generator," in *Generative Programming and Component Engineering, 5th International Conference, GPCE 2006, Portland, Oregon, USA, October 22-26, 2006, Proceedings* (S. Jarzabek, D. C. Schmidt, and T. L. Veldhuizen, eds.), pp. 141–150, ACM, 2006.

[130] A. Narkhede and D. Manocha, "Fast polygon triangulation algorithm based on Seidel's Algorithm," tech. rep., Department of Computer Science, UNC-Chapel Hill, 1994.

[131] H. Si, "Tetgen: A quality tetrahedral mesh generator and three-dimensional Delaunay triangulator.."

[132] E. Cohen and L. L. Schumaker, "Rates of convergence of control polygons," *Comput. Aided Geom. Des.*, vol. 2, pp. 229–235, Sept. 1985.

[133] Y. Zhou and A. W. Toga, "Efficient skeletonization of volumetric objects," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, pp. 196–209, July - September 1999. ISSN 1077-2626.

[134] W. Schroeder, K. M. Martin, and W. E. Lorensen, *The visualization toolkit (2nd ed.): an object-oriented approach to 3D graphics*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1998.

[135] T. W. Sederberg, J. Zheng, A. Bakenov, and A. Nasri, "T-splines and T-NURCCs," *ACM Trans. Graph.*, vol. 22, pp. 477–484, July 2003.

[136] T. Dokken, T. Lyche, and K. F. Pettersen, "Locally refinable splines over box-partitions," Tech. Rep. SINTEF A22403, SINTEF, February 2012.

[137] D. R. Forsey and R. H. Bartels, "Hierarchical B-spline refinement," in *Proceedings of SIGGRAPH '88*, pp. 205–212, 1988.

[138] D. P. Mitchell and A. N. Netravali, "Reconstruction filters in computer-graphics," in *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '88, (New York, NY, USA), pp. 221–228, ACM, 1988.

[139] I. J. Schoenberg, "Contributions to the problem of approximation of equidistant data by analytic functions, part a: On the problem of smoothing or graduation, a first class of analytic approximation formulas," *Quart. Appl. Math.*, vol. 4, pp. 45–99, 112–141, 1946.

[140] C. de Boor and G. Fix, "Spline approximation by quasi-interpolants," *J. Approximation Theory*, vol. 8, pp. 19–45, 1973.

[141] B.-G. Lee, T. Lyche, and K. Mørken, "Some examples of quasi-interpolants constructed from local spline projectors," in *Mathematical Methods in CAGD: Oslo 2000*, (Nashville, TN), pp. 243–252, Vanderbilt University Press, 2000.

[142] T. Lyche and L. L. Schumaker, "Local spline approximation methods," *J. Approx. Theory*, vol. 15, pp. 294–325, 1975.

[143] P. Shirley and K. Chiu, "A low distortion map between disk and square," *J. Graphics Tools*, vol. 2, no. 3, pp. 45–52, 1997.

[144] T. Lyche, K. Mørken, and E. Quak, "Theory and algorithms for non-uniform spline wavelets," in *Multivariate Approximation and Applications*, pp. 152–187, Cambridge University Press, 2001.

[145] M. Ashikhmin and P. Shirley, "An anisotropic Phong BRDF model," *J. Graphics Tools*, vol. 5, no. 2, pp. 25–32, 2000.

[146] M. M. Stark, W. Martin, E. Cohen, T. Lyche, and R. F. Riesenfeld, "B-splines for physically-based rendering," Tech. Rep. UUCS-02-005, University of Utah, October 2002.

[147] P. H. Christensen, D. Lischinski, E. J. Stollnitz, and D. H. Salesin, "Clustering for glossy global illumination," *ACM Trans. on Graphics*, vol. 16, pp. 3–33, Jan 1997.

[148] F. Sillion, G. Drettakis, and C. Soler, "A clustering algorithm for radiance calculation in general environments," in *Rendering Techniques '95*, Springer, 1995.