Realistic Human Figure Synthesis and Animation for VR applications

By

Karansher Singh, Ph.D.

The Ohio State University, 2002

Dr. Richard Parent, Adviser

Human figure animation is a widely researched area with many applications. This thesis addresses issues specific to the synthesis, animation and environmental interaction of human figures in complex virtual worlds. The techniques developed are aimed at applications such as virtual space teleconferencing, where realistic real-time animation of the human figures is desired. Visual realism of the characters dominates over the physical accuracy of the generated motion.

A layered representation of the human figure is adopted. Physical modeling and animation techniques, though effective, are too complex to achieve real time results. Implicit functions show promise in elegantly handling many issues involved in articulated figure animation that can be cumbersome for existing B-rep (boundary representation) based techniques. Real-time animation requirements, however, currently make the use of polygon-based models imperative. Approaches to improving

the display efficiency of implicit functions are therefore discussed. Further, general implicit shape definitions for objects modeled using B-reps are provided. This allows an integration of B-rep and implicit function based techniques. To facilitate efficient environmental interaction, an existing collision detection and deformation model is extended and combined with constraint satisfaction for deformable objects. A realistic human figure model, allowing the application of implicit function and B-rep based animation techniques, is proposed. The implicit function muscle model and skeletal posture control a polygon based skin structure. Implicit functions detect and handle collisions in the environment homogeneously and efficiently. Precise collision contact surfaces are generated and physical characteristics of muscles in systems employing force feedback are simulated. Clothes are modeled as textures mapped on a geometric skin. Characteristic wrinkled textures for garments are synthesized and wrinkle formation is carried out by texture morphing and wireframe displacement controlled by the posture of the animated human. The effectiveness of the developed concepts and proposed models are illustrated by their implementation within a virtual space teleconferencing system.

This thesis contributes to object modeling and animation by successfully marrying implicit function and B-rep based methodologies. The human figure models developed are elegant, efficient and showcase this hybrid approach. These solutions can be easily applied to existing real-time human figure animation applications where visual realism is more important than an anatomically and physically precise model.

Realistic Human Figure Synthesis and Animation for VR applications

DISSERTATION

Presented in Partial Fulfillment of the Requirements for

the Degree Doctor of Philosophy in the Graduate

School of The Ohio State University

By

Karansher Singh, B.Tech., M.S.

* * * * *

The Ohio State University

2002

Dissertation Committee:

Dr. Richard Parent

Dr. Wayne Carlson

Dr. Roni Yagel

Approved by

Adviser

Department of Computer & Information Science

...to the ubiquitous Taramani Nair

No one told you when to run, you jumped the starting gun!

Acknowledgements

...a blast with a boldface capital 'b' this pachaddee has been. thanks to rick for being one hellova role model and the mellowest budweiser i have most ever known. rick said a thesis was like art... thanks to wayne and roni for helping me make this more than a quick caricature. maido okini to ohya-san and minna-sama at atr-csrl for the opportunity to work on their virtual space teleconferencing project, their monster machines and their futuristic toys. honto ni o sewa ni narimashita.

...to all the rocking folk who graced 2405 east ave. while it was my abode,

muchas gracias and go on mistaking paradise for that home across the road.

a salute to the accad machaans who prove everyday, without the soft shadow of a doubt, that a pretty picture is worth more than a thousand words using all seven letters at a time on a triple word score. thanks to my koshinokambai poting monk buddies for their spiritual guidance in the eternal quest for the buddhas monkey and subsequent realization that baaki diyan galan chhaddo, praaji... saru mo ki kara ochiru. ack's are also in order to maasta's taberuna gumbal and the grooviest gaijingle bells who shared with me the experience that was nihon. finally i just want to thank the system, for silently manouvering me to where...

less is more, more or less; yes is no and no is yes.

win or lose; confucian blues.

Vita

December 16, 1969	Born - Delhi, India
1991	B.Tech Computer Science and Engineering Indian Institute of Technology, Madras, India
1992	M.S. Computer and Information Science The Ohio State University, Columbus, Ohio
1995	Ph.D. Computer and Information Science The Ohio State University, Columbus, Ohio

Publications

Patents

Motion synthesis equipment using 3D models, *Tokuganhei 7-42120 (Japanese patent no. 1995 - 42120).*

3D image synthesis equipment for enabling wrinkle formation *Tokuganhei 7-105012 (Japanese patent no. 1995 - 105012).*

Research Publications

K. Singh & R. Parent. "Fast scanline processing of soft objects: spheres, rounded polygons, and cone-spheres", *Ohio State Univ. Tech. Rep.* OSU-CISRC-5/94-TR26, 1994.

K. Singh & R. Parent. "Polyhedral shapes as implicit surface primitives", submitted to *ACM Transactions on Graphics*, 1994.

K. Singh, J. Ohya & F. Kishino. "Realistic modeling and animation of a muscle and skin layer for human figures using implicit function techniques", *IPS Japan, CG & CAD Conference*, 49-56, 1994.

K. Singh & R. Parent. "Implicit function based deformations of polyhedral objects", *Eurographics workshop on Implicit Surfaces*, Grenoble, France, 113-128, 1995.

K. Singh & R. Parent. "Constraint satisfaction for deformable objects using implicit functions", submitted to *Visual Computer*, 1995.

K. Singh, J. Ohya & R. Parent. "Human figure synthesis and animation for virtual space teleconferencing", *IEEE Virtual Reality Annual International Symposium*, 118-126, 1995.

K. Singh, J. Ohya & F. Kishino. "Real-time cloth animation effects for virtual space teleconferencing using texture morphing", *Proc. IEICI, Japan*, IE94-88, 1–8, 1994.

K. Nariyama, K. Singh, J. Ohya & F. Kishino. "Realistic 3D synthesis of human body movements for virtual space teleconferencing", *IASTED International Conference on modeling and simulation*, Pittsburg, 1995.

K. Singh, L. Moubaraki, J. Ohya & F. Kishino. "A texture based wrinkle model for human figures in VR applications" submitted to *PRESENCE*, 1995.

R. Srikant, R. Sundaram, K. Singh & C.P. Rangan. "Optimal path cover problem on block graphs and bipartite permutation graphs", *Theoretical Computer Science*, 115:2, 351-357, 1993.

K. Singh & K. Fujimura. "Map making by cooperating mobile robots", *IEEE International Conference on Robotics and Automation*, Vol.2, 254-259, 1993.

R. Sundaram, K. Singh & C.P. Rangan. "Treewidth of circular-arc graphs", *SIAM J. of Discrete Mathematics*, 7:4, 647-655, 1994.

Fields of Study

Major Field: Computer and Information Science

Studies in:

Computer Graphics	Dr. Richard Parent
Theoretical Computer Science	Dr. Kenneth Supowit
Parallel and Distributed Computing	Dr. Anish Arora

Table of Contents

List of Tables

List of Figures

CHAPTER I

Introduction

Virtual Reality (VR) is a fast emerging field with a number of applications in prototyping, training, operator assistance, telecommunication, telepresence and entertainment. The user is typically immersed in a virtual world that can exceed the limitations of physical reality and where the illusion of reality is complete in all senses. Augmented reality applications aim to integrate the virtual world with the real world using devices such as see through displays. Simulated humans are an integral part of a large class of applications varying widely from *avatars* representing real humans in virtual space teleconferencing [44] to computer generated armies in military training simulators [21]. Most virtual environment applications do not require anatomical and physical accuracy of the character models. It is the real-time visual realism of the generated human figures that creates the illusion of reality. While anatomically and physically accurate models provide good visual results [15][37], their computational complexity makes them unsuitable for real-time applications. On the other hand, purely geometric models are becoming less desirable with the onset of haptic devices [13]. This thesis is thus targeted at developing a visually realistic human figure model that is computationally efficient and has a physical interpretation,

allowing its incorporation in virtual environments with force feedback.

1.1 Character animation

Human figure animation is a widely researched area in itself. Current photorealistic rendering capabilities make the main problem one of character modeling and animation. A layered approach [14] to the modeling and animation of articulated figures is currently a widely adopted methodology. With respect to human figures the layers may be broadly classified into:

1. Skeletal
2. Muscle, Skin and underlying tissue
3. Hair, Nails, Blemishes and other such features
4. Clothes and Accessories

The above layers are not necessarily mutually exclusive and are often omitted, collapsed together, or further subdivided depending on the sophistication and thrust of the application.

Given a human figure representation, motion control mechanisms (MCM) specify the appearance of the figure during animation [36]. These can be roughly classified into *Geometric*, *Physical* and *Behavioral*, based on the emphasis and the manner in which the problems related to animating a synthetic actor in a virtual world are addressed. *Geometric MCM's* treat the actor in the environment from a purely geometric perspective. Keyframe techniques, kinematics, geometric deformations, collision

processing and obstacle avoidance are examples of this category.

Physical MCM's incorporate the physical aspects of the actor by using techniques like dynamics, physical collision and deformation models, computation of environmental forces, and force feedback models.

Behavioral MCM's capture the emotional aspect of the human figure. Models involving individualized attributes, quirks and disabilities, sensory interaction with the environment and emotional communication between actors are examples in this class.

Building such levels of abstraction and automation of the animation process over the basic object representation of the actor is presented by Thalmann and Thalmann [36]. We would like the lower level representations to be modeled with enough generality so that a number of high level techniques may be applicable.

There is an outward dependency across the layers. The skeleton affects the shape of the muscle and skin, which in turn shape the apparel. Thus for physically accurate cloth model results on a human figure, the models of the underlying layers should be accurate. On the other hand, if visual realism is the major concern, visually reasonable models for the outer layers can hide problems in the models for the inner layers and may even obviate modeling them explicitly.

1.1.1 Skeletal layer

Most character animation systems model skeletons as articulated rigid bodies. As the skeleton is not directly visible, often the skeleton comprises only virtually linked joints with no explicit geometric skeletal shape. Human bones have limited flexibility and are almost always modeled as rigid bodies. Skeletal joints have over 200 degrees

of freedom. Typically, however, the number of bones and joints modeled are greatly reduced and only about 50 degrees of freedom are explicitly modeled.

A number of robotics algorithms for motion control such as kinematics, dynamics and obstacle avoidance can be applied to the skeleton modeled as an articulated rigid body. A number of problems result from the application of these techniques to the skeleton due to the many approximations:

- Human bones often indirectly shape the visual appearance of the skin in thin bony figures and around joints (such as the elbow and knee). The proximity of the bones to the skin surface determines the extent to which they deform the skin. This distance between bones and the surface of the skin is subject to change during animation of the figure. The influence of bones is thus hard to model for human figures where the skeletal model has no explicit geometry.

- For robotics algorithms such as inverse kinematics, often solutions for an under constrained skeleton are sought that avoid singularities and stay away from joint limits. While this is an acceptable solution for machines, the human skeleton is quite comfortable with many joints extended to their limits. This problem is worsened by the fact that these limits change based on the overall skeletal posture. Robotics solutions for these extreme cases thus appear unnatural.

- Human joints are not perfectly rotational joints at a point about a fixed axis. As an example, the center of the shoulder joint rotates away from the skin surface inside the body as the arm is raised. For joints fixed near the surface, vertices

tend to penetrate the body for large angles. Similarly, if the joint is fixed away from the surface, the arm is like a flexible pipe for small angles.

Despite the problems arising from such approximations [36], various robotics techniques adapted to the human figure have provided realistic results. Additionally, for many applications such as virtual space teleconferencing, the skeletal model is driven by tracking the posture of a real human figure. For such applications, motion control of the skeletal layer reduces to one of tracking the posture of the real human using equipment such as magnetic sensors [42] or by image processing live action. This thesis thus develops human figure synthesis and animation techniques based on reasonably accurate motion of a skeletal structure modeled as an articulated rigid body.

1.1.2 Muscle and Skin layer

Modeling and animation of the **muscle** and **skin** layer has almost entirely dealt with figures modeled by a boundary representation [14][37][60]. Polygon based structures are popular due to their simplicity, generality and hardware support. Prototype models, typically of the real figure in a relaxed pose, are used to represent the geometric skin. Muscle models then control the deformation of the skin during animation.

The deformable nature of human muscle, fatty tissue and skin is described in [60]. Physical tissue characteristics are modeled as spring and damper meshes attaching skin to the underlying skeleton. Forces are applied iteratively and the stabilized network shapes the skin. The paper deals specifically with facial animation. The use of the finite element method (FEM) is illustrated in animating a human hand in a grasping situation [37] and in simulating a biomechanical muscle model [15]. Physically

based models such as the above can handle interaction with the environment but are computationally too intense for real-time response in complex virtual environments.

Free form deformations are used to empirically deform the skin layer in [14]. There is no explicit underlying muscle model. An empirical correspondence between joint angles and the deformation is made. A human skin model based on Bezier surfaces in [30] controls deformations like [14] by manipulating the surface control points. Position of a wireframe skin around joints in terms of a function that is specific and local to the joint skeletal area is presented in [37]. Geometric approaches like those just described are efficient and show realistic results in many cases. They do not, however, address environmental interaction or issues relating to the dynamics of the figures.

The muscle and skin layer is very important for visual realism and its effect on subsequent layers. Further, the complexity of modeling facial detail, hair and clothing is often reduced to textures mapped on the skin surface making skin the quintessential layer.

Facial Animation is very important for many virtual world applications, such as teleconferencing, not only for the realism of the animated figures but for communication through facial expressions. Facial animation techniques provide a higher level of control, typically built on a muscle and skin layer, for behavioral control over the animation of facial expressions. The analysis and classification of different human facial expressions in relation to expressed emotions has been extensively researched and several facial animation models have been proposed. Most approaches for the

analysis and reconstruction of human facial expressions are physically based [52][64] or parameterized [49].

Ekman and Friesen [16] have identified six primary expressions and catalogued thousands of different expressions; they formulated the Facial Action Coding System (FACS) which is a parameterization of muscles in relation with the corresponding emotions. Simplified muscles are attached to a skin mesh and manipulated to model human facial expressions based on FACS by Platt and Badler [52]. Terzopoulos and Waters [64] have used a simple form of FACS to derive a 3D model motivated by tissue biomechanics; their physically-based model for facial tissues has three layers: skin, sub-cutaneous fatty tissue, and muscles. The underlying layers of these muscle-based approaches should be sufficiently precise to provide physically accurate results for motion. Parke [49] introduced parameterized facial models in which facial parameters are empirically specified. Such approaches can be efficient but the quality of realism is usually closely related to the resolution of the skin geometry, as surface geometry is used to model even the finest of wrinkles. Beier and Neely [6] have developed a technique in 2D for the metamorphosis of one image into another. The morph is controlled by manually specifying corresponding features on the images using line segments and defining a geometric transformation between them. This method may be used effectively to animate a face by morphing one facial expression into another.

While physically based techniques are computationally complex, most parametric models require dense wireframe meshes to represent fine surface detail like wrinkles. This makes them inefficient and hard to control. Facial animation is thus treated in

a hybrid fashion in our model. The muscle and skin model control large deformations on the face. Deformations resulting in small visual details are dealt with using texture morphing techniques.

1.1.3 Cloth Layer

Cloth animation has become an important area of computer animation, especially related to human figure animation [1][31][32][35][45][65]. The problems to be addressed may be separated into cloth shape and deformations due to environmental forces (gravity, wind) and cloth collisions with itself and synthetic actors. Techniques for cloth animation without environmental force deformations have been proposed [59][65]. Cloth animation with self-collisions and collision with almost rigid bodies is addressed by Lafleur [32]. Essential to animation of apparel on the human body is the process of wrinkle formation. Approaches to modeling wrinkle formation and propagation have been provided by Aono and Kunii [1][31].

Cloth models may be classified as geometric or physically based. Geometric models such as the one presented by Weil [65], while effective and robust for simple cloth pieces like a flag, cannot capture the complexity of clothes like a shirt on a human figure. Physical methods [1][24][59] animate the cloth shape by solving differential equations based on the physical properties of the piece of fabric. These models can provide realistic results for fairly complex clothes.

As described by Thalmann [35] a cloth animation system should have the following attributes:

- A cloth model that suitably addresses the requirements of the application. Factors to be considered are external forces, fabric properties, environmental collisions and self-collisions.

- An interactive interface for the synthesis of apparel on human figures.

- An interactive interface for adjusting parameters that affect the appearance of the cloth during animation.

Cloth animation of human figures in virtual worlds is important for enhancing their visual realism. It is impossible to realize physically based cloth animation in real-time with current computing resources for complex virtual worlds. For many applications the following are reasonable assumptions for tight fitting apparel:

- Shape deformations of clothes are governed by the shape and motion of the human body and not by wind, gravity or other external forces.

- Cloth animation only takes place during animation of the human figure.

- Cloth animation is local, in that animation of a part of the body only deforms a limited region of the apparel clothing it.

Tight fitting apparel, like facial detail, is modeled as color textures applied to the geometric skin. In this thesis we present simple interactive techniques by which cloth animation effects may be obtained based on the animation of the underlying skeleton. The approach makes use of the aforementioned assumptions. It is not physically accurate and thus unable to achieve the realism of physically based methods [1][59].

The advantages of our approach are twofold. The visual realism of the figure is enhanced and discrepancies in the behavior of the muscle and skin layer are hidden at a minimal computational overhead.

This thesis focuses on the muscle, skin and clothing layers which are the layers most directly responsible for the visual appearance of the human figure in a given skeletal posture. Implicit functions provide elegant solutions to many of the problems in the modeling and animation of these layers. They thus form the foundation over which the proposed models for these layers are built.

1.2 Implicit Functions

Implicit surfaces are a popular approach to object modeling and animation and are especially applicable to physically deformable objects [8][10][20][41][55][68][69]. An implicit surface is defined as the set of points P satisfying an implicit equation $F(P) = 0$.

1.2.1 Implicit Surface Primitive Shapes

A useful set of implicit surfaces can be generated as an algebraic combination of polynomial functions each of which is defined over a finite volume. For summed polynomial functions, $F(P) = \sum F_i(P) - T$, where i runs over the primitive polynomial functions F_i and T is a threshold value $\in [0, 1]$. Subsets of these surfaces are **distance surfaces** and **convolution surfaces** [11]. Typically, each primitive is defined by a skeleton S, a cutoff distance R, and a function f. An example of $f : [0, 1] \rightarrow [0, 1]$, called a density or blend function, with the desired properties [68] is shown on the

left in Figure 1. For a point P whose shortest euclidean distance from P to S (which is a point Q on S) is smaller than R, $F(P) = f(|P-Q|/R)$[1]. $|P-Q|/R$ is referred to as the *distance-ratio*(P) for the primitive. $F(P) = 0$ for points P with distances greater than R (outside the realm of influence of the primitive). Such a primitive is a distance or offset surface [11]. Convolution surfaces require the integration of function values obtained with Q varying over every point on S, rather than just the closest. The difference between the two surfaces lies in the function value computed for P. Convolution surfaces result in surfaces with properties such as seamlessly blended concavities and a lack of unwanted bulges on blends.

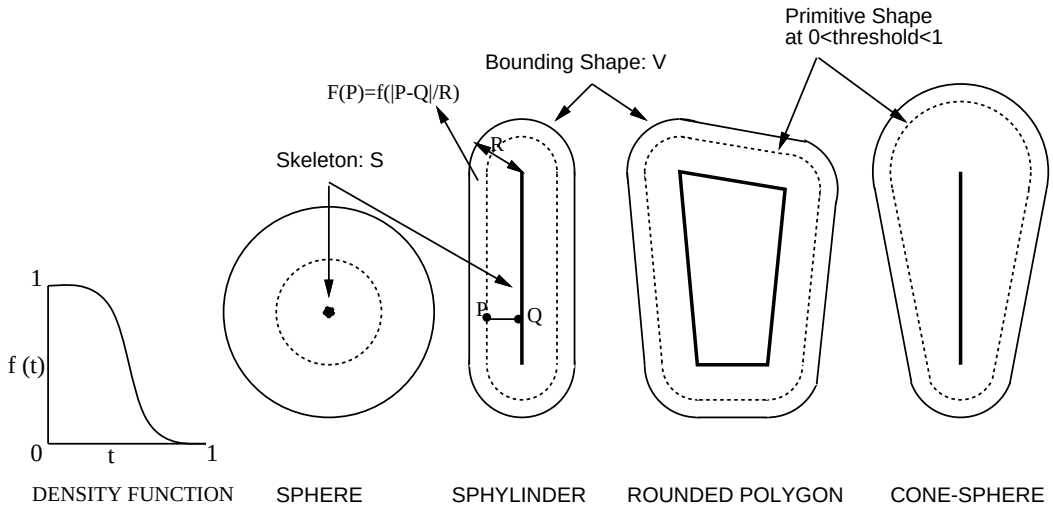


Figure 1: Implicit Primitive Shapes

The simplest implicit shape used as a modeling primitive is a sphere, which is

[1]**Terminology**: In this thesis implicit functions for points in space F are often based on a single variable function f. The procedure for mapping a point P to a value in the domain of f is generally fixed for given classes of primitive shapes. We use lower case to denote single variable functions and upper case for their corresponding function of points in space for implicit primitive shapes.

an offset surface around a central point S. The terms *metaball*, *blob* and *soft objects* have been used in previous work [8][18][41][68] to refer to this shape and, by extension, to any such primitive used in constructing an implicit surface. For the purpose of articulated figure animation we introduce the following primitives (see Figure 1):

- *Sphylinders* are finite extent cylinders with hemispheres capping both ends. They are offset surfaces whose skeleton, S, is a line segment. They are useful for modeling a variety of shapes, especially longitudinal ones, which can only be approximated by a number of linearly placed spheres.

- *Rounded polygons* are offset surfaces, where S is a planar polygon [11].

- *Cone-Spheres* are a generalization of sphylinders in which a truncated cone is capped on either end with spherical segments that maintain first order continuity at the junction of conical and spherical surfaces [38] [2].

More general implicit primitives are superquadric ellipsoids and toroids [4]. A powerful extension to these primitives is the overlay of modal deformations and displacement maps [55].

1.2.2 Implicit Surface Characteristics

Implicit surfaces are popular due to several reasons.

[2]The offset surface definition given above cannot define this shape. They may be defined by the more general implicit surface primitive definition we formulate in Chapter II.

- Implicit primitive shapes are intuitive for building more complex shapes [8], which can be deformed easily by manipulating the individual primitives [68]. Topological changes during animation are automatic.

- Continuity properties for the resulting surface can be ensured by appropriately combining continuous primitive functions, making implicit surfaces a modeling alternative to higher order surface patches.

- Various physical characteristics such as elasticity [20] and thermal properties of objects are easily modeled. Further, implicit functions can replace discrete spring models with stiffness fields that are continuous in space.

- Efficient implicit function evaluation greatly improves the efficiency of collision detection between objects [51]. Algebraic combinations of implicit functions may be used to model creases and collision contact surfaces [20].

- Implicit primitives like blobs [41] or superquadrics [55] can be fitted to real 3D data, such as obtained from laser scanners or range images.

A major drawback of implicit surfaces is the necessity to sample space in order to determine the implicitly defined surface [8][18]. Implicit surfaces can be rendered in one of two ways. The implicit surface representing the object can be reconstructed by polygonization [9][54] and subsequently displayed. Alternatively, one may display the implicit surface directly, by tracing rays through space, sampling the function as the rays progress away from the observer until the surface threshold is achieved [18]. Various techniques for fast and robust ray tracing of implicit surfaces have been

studied [23][26][70]. Blinn [8] addresses the issue of efficiently rendering surfaces defined by spherical density functions by employing scanline processing. While this fast scanline technique for rendering spheres exists, spheres alone have limited usefulness as a modeling primitive.

The bulk of modeling, animation and display techniques currently in use require object representations that are B-rep and often polygon based. Much work has therefore been done on generating a polymesh representation from other object representations [9][34][43][54].

Work on implicit surfaces thus is focussed on:

- Extending implicit surfaces into a general, user-friendly object modeling tool.
- Applying them to model and animate various physical models.
- Speeding up the implicit surface rendering process.

We formulate implicit shape definitions for polyhedral objects which unify implicit function and boundary representation based methods. Efficient display techniques for surfaces constructed as an algebraic combination of implicit surface primitives in general and polyhedral primitives in particular are addressed.

We further develop a method by which objects with a polyhedral definition are deformed by implicit functions. Such a method has many benefits of implicit function based techniques and the display efficiency of polygon based structures.

1.2.3 Collision Detection and Deformation

Collision detection between objects is a problem that has been widely researched [3][22][40][50][51][61]. While the problem can be analytically solved for rigid bodies [3][22], the dynamic simulation of deformable objects remains a difficult problem. Colliding deformable objects exhibit complex behavior. The objects remain in contact over a period of time and energy is typically lost as a result of the collision. Contact modeling is difficult and most approaches use penalty methods [40] to determine the reaction forces on interpenetrating objects.

A novel collision detection and deformation model for objects defined using implicit functions has been proposed by Gascuel [20]. Colliding objects impart each other with a deformation function. The deformed objects are represented by the implicit surface which is deformed as a result of the addition of the deformation function to the implicit function of the object. The deformation functions are negative in the region where the objects penetrate, causing the object surfaces to move inward and form a precise common contact surface. A small region around where interpenetration occurs is called the propagation zone. Here the deformation functions are positive causing the object surface to bulge outward. This is to maintain continuity of the deformation function and to model volume conservation of the objects. The simple case of two colliding objects is shown in Figure 2.

A physical interpretation may be attached to this model, allowing the implicit function gradient to model the stress-strain characteristics of the object. This model shows promise in modeling the environmental interaction of human muscle, fatty

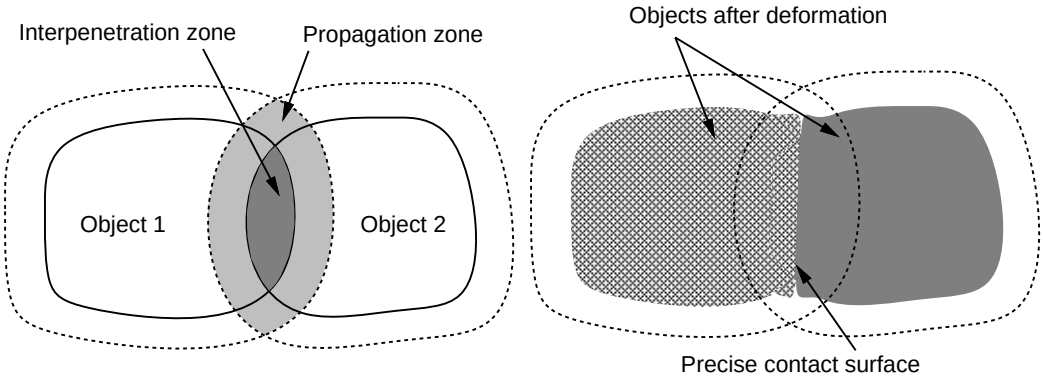


Figure 2: Implicit function based collision model

tissue and skin.

The separation of the dynamics of deformable bodies into *rigid* and *deformable* components [62] is computationally more efficient than considering the general Lagrange equations of motion [59]. This separation is based on the approximation that the mass distribution of the deformable object is constant and corresponds to that of the object's undeformed shape. Such an approach is well suited to human figure models, where the deformations are localized and are small relative to the size of the figure. The collision deformation approach [20] is such a deformable layer model.

In Chapter II we extend this collision model to incorporate time varying elasticity, to vary the relative rigidity of two colliding objects, and to facilitate constraint satisfaction between deformable objects. The constraint and collision models are homogeneously applied to the objects.

1.2.4 Constraint Satisfaction

Constraint satisfaction and collision deformations are crucial to the visual realism of grasped objects and in providing operator assistance to users while interacting with inanimate objects in a virtual world.

Satisfaction of constraints between objects has been well addressed. Featherstone [17] provides an efficient solution to articulated chains with 1 DOF links, which has been extended to articulated chains [33]. Wilhelms and Barsky [66] allow rotations and translations at the joints of objects linked in tree like structures. An efficient approach to the satisfaction of constraints between objects linked by rotational hinges in a system without a closed loop of constraints is given by Armstrong [2]. General 6 DOF constraints between objects are allowed by Isaacs and Cohen [25], which provides a solution based on d'Alemberts virtual work principle. An inefficient but general method using constraint forces holding together an arbitrary system of objects and constraints is presented by Barzell [5]. Here the constraints manifest themselves as forces forming a system of coupled differential equations. Van Overveld [63] describes a simple and efficient method based on point dynamics. Constraint satisfaction by iterative displacements is presented by Gascuel and Gascuel [19].

The model of Gascuel and Gascuel [19] is comprised of solid objects connected with an arbitrary constraint graph. The satisfaction of constraints is decoupled from the rest of the animation. Constraint satisfaction here does not involve explicit calculation of constraining forces and is thus also applicable to purely kinematic animation frameworks. Realistic results at interactive rates are illustrated. In this thesis we use

a similar constraint model with the satisfaction of constraints decoupled from the rest of the animation. As our approach is targeted at deformable objects, we satisfy constraints by localized deformations of the objects rather than by rigid body displacements as in [19].

We formulate a generalized framework for deformable objects with implicit function definitions, into which collision deformations and constraint satisfaction are elegantly embodied.

1.3 Overview

This thesis is motivated by the desire to obtain visually realistic human figure animation and environmental interaction in real-time. Realistic animation of a deformable body with the complexity of a human figure and its interaction with the environment is inherently an interaction of object volumes. Simple boundary representations do not capture this notion making algorithms like collision detection, that are based on a boundary representation, inelegant and inefficient. Implicit functions, on the other hand, define volumes in space that can interact efficiently [20][51]. We thus develop an implicit function based virtual world model within which an implicit function based human figure model is proposed. Following the transformations resulting from animation and interaction in the virtual world, the objects need to be displayed. Explicit boundary representations of objects, especially polygon based structures, can be displayed in real-time by existing hardware. Implicit surfaces have an inherently

inefficient display characteristic. Aside from the ability to polygonize implicit surfaces [43], the two modeling approaches have so far been distinct and isolated. This thesis unifies these distinct representations in ways that allow hybrid models to avail of the advantages of both approaches. Finally, real-time techniques for the animation of clothing on these human figures is developed.

This thesis thus provides a comprehensive model for the realistic real-time animation and environmental interaction of human figures in complex virtual worlds. All the ideas and concepts presented in this thesis barring Gascuel's collision deformation approach [20], are original and new.

The contributions of this thesis are:

1. A generalized framework for modeling and animating objects and external forces in a virtual world using implicit functions. The model can be viewed purely kinematically or with a physical interpretation.

2. A method for constraint satisfaction between deformable objects built on previous work in collision detection and deformation [20].

3. A technique by which objects with a polyhedral definition can be embedded in a hierarchy of implicit functions. The implicit functions can then deform the polyhedral definition of the object.

4. Implicit surface primitive shape definitions for polyhedral objects. These definitions unify the two object representation methods.

5. Efficient display techniques for implicit surfaces, especially those constructed with polyhedral implicit primitives.

6. Efficient geometric computations for analytic primitive shapes like cone-spheres, which are useful for human figure animation.

7. A new human figure model, where implicit functions model bones, muscle and skin. The model fits into our implicit function based framework for the virtual world. The model is visually realistic and has physical and anatomical interpretations.

8. A technique for the synthesis of wrinkled textures. Wrinkles on skin and clothing are animated by texture morphing and wireframe displacement of the human figure in real-time.

9. Finally, the techniques developed above are integrated and illustrated by a real-time implementation of human figures in a virtual space teleconferencing system.

The rest of this thesis is organized as follows.

Chapter II describes research built on the work of Gascuel [20]. The model is extended to incorporate time varying elasticity and to facilitate constraint satisfaction between deformable objects, such that the constraint and collision models may be homogeneously applied to the objects. The implicit function based model for articulated deformable objects is then presented. Deforming objects with a polymesh representation using implicit functions is also described.

Chapter III discusses implicit function based modeling and animation. Implicit primitive shape definitions, for objects modeled as B-reps, are presented. Methods to improve the display efficiency of these implicit surfaces are then presented.

Chapter IV proposes a human figure model that aims at high visual realism, efficient environmental interaction, and the capability of a physical interpretation for dynamics based systems.

Chapter V presents a texture based wrinkle model for the formation and propagation of wrinkles on skin and clothing.

Chapter VI describes the implementation of the proposed models and concepts within a virtual space teleconferencing system.

Chapter VII provides a discussion of the results obtained, presents conclusions and discusses the scope for future work.

Appendix A provides exhaustive detail for computation that is required for the rendering of some useful implicit surface primitives.

Appendix B shows effective approaches to shape transformation using polyhedral implicit surface primitives.

Appendix C lists example blend and ghost functions to achieve different kinds of implicit primitive interaction.

CHAPTER II

Implicit Function based Virtual World Model

This chapter proposes a model for the animation and efficient interaction of objects and other forces in a complex virtual environment. The central theme of this thesis are human figures in the virtual world and so the scope of objects in our model should include articulated deformable bodies. The model, being targeted at VR applications, should be computationally efficient and general. It should be applicable to open loop systems (which are typically kinematic) as well as those with environmental force feedback.

In our model, all objects are represented by implicit functions. Objects interact and deform each other by imparting additional implicit functions to each other. During animation, all the implicit primitives in the environment are first individually manipulated. This may be done kinematically or based on environmental forces. The objects then modify each other's implicit functions to model all interactions like collisions and constraints. System forces as a result of these interactions can be computed.

We also address the issue of incorporating objects with a polymesh representation into this virtual world model. The conversion of polyhedral shapes to implicit surface

primitives, as described in Chapter III, is one solution. In this chapter, however, we show how polymesh objects can be incorporated into the model while preserving their polymesh representation. Each polymesh object is embedded in a hierarchy of implicit functions. These functions act as the implicit function definition of the object, while the visual appearance of the object is governed by the polymesh. The functions interact with one another and with other implicit functions in the world, resulting in deformations to the underlying polymesh. This gives us the advantages of the computationally efficient implicit function model and the display efficient polygon based structure of the object.

Section 2.1 describes the notion of objects interacting by imparting additional functions to each other and lays the foundation for the implicit function based virtual world model. The collision detection and deformation approach of Gascuel [20] is shown in this light. Section 2.2 presents a new approach to satisfaction of point constraints between deformable objects, which is coherently integrated with the collision detection and deformation model of Gascuel. Section 2.3 presents the embedding of the polymesh object in a hierarchy of implicit functions and its animation as a rigid body with a subsequent implicit function based deformation. Section 2.4 extends the above model to incorporate articulated deformable objects, in particular human figures [58]. Section 2.5 describes the implementation and Section 2.6 discusses the results.

2.1 Implicit Functions: Ghosts

Different objects have their own implicit functions that determine the object surface. These objects may interact by imparting additional implicit functions to one another [20]. Here, during collision of two or more implicit objects, each object imparts an implicit function to each colliding (interacting) object so as to model collision deformations. Examples are the colliding eyeballs in Figure 33 and the elbow crease in Figure 43.

We extend this notion to allow an object to freely impart and control various implicit functions that deform other objects based on the interaction between the objects. These functions are temporal and are controlled by the imparting object. The temporal parameter allows us to model various temporal properties of objects like viscoelasticity. Additionally certain environmental deformation forces such as gravity and pressure in a fluid may be modeled as force field functions imparted to the objects in the environment. Such implicit functions that manifest themselves by influencing the shape of objects in the environment will be referred to as **ghost functions**.

Each ghost function also has a unique priority value to determine the order in which the ghost functions for an object combine with the implicit function for the object. Further, the influence of a ghost function is only considered within the realm of influence of the object's implicit function (The implicit functions dealt with in this paper are polynomial and have a positive value defined over finite volume in space, unlike the exponential functions used by Blinn [8]). For the most part, ghost functions

are simply additive, where the combination is both associative and commutative. The priority number is included here for the sake of completeness, but has not been used in practice. The collision deformation function [20] is such a ghost function. This and other examples of ghost functions, modeling various properties, are shown in Appendix C.

The deformable model separates the physical characteristics of objects into **rigid** and **deformable** components that may be applied successively [20]. The rigid component deals with the animation of rigid objects. It also involves animation of the implicit function primitives and ghost functions which define deformable objects independent of each other. The deformable objects are then deformed appropriately as a consequence of the interaction of their implicit function and imparted ghost functions (see Figure 3).

Ghost functions require that the deformable object has an implicit function of its own that they can interact with. The framework described so far thus applies directly to objects modeled using implicit functions. The embedding of polymesh objects in implicit functions to incorporate such objects into the above framework is presented in Section 2.3.

2.1.1 Gascuel's Collision Deformation Model

Gascuel [20] describes a technique by which objects with implicit function definitions can be deformed appropriately during collisions with each other. A collision between two implicitly defined objects causes each object to impart the other with a ghost function. These ghost function have a negative contribution in the region where

Figure 3: Rigid, Deformable Component Transformation

the objects penetrate each other (refered to as the interpenetration zone). In a finite neighboring region around the the interpenetration zone (refered to as the propagation zone) the ghost function values are positive. The negative ghost function values cause the object surfaces in the interpenetration zone to move inward causing the objects to abut along a common contact surface (See Figure 2). The positive ghost function values in the propagation zone cause the objects to bulge outwards. The provides the appearance of preserved object volume and maintains surface continuity properties between the inwardly deformed surface in the interpenetration zone and the undeformed part of the object surface. The ghost function in the propagation zone for an object is formulated such that the outward bulge of the object surface does not cause new interpenetrations with the colliding object.

Formalizing the above idea, let object i be modeled using implicit function primitives with blend function f_i. Objects are represented as the implicit surface at some threshold T (points P where $F_i(P) = T$) as described in Section 1.2.1. The ghost function g_i imparted by object i to object j on a collision is characterized by:

- An interpenetration zone (where $f_i \geq T$): $g_i = T - f_i$
- A propagation zone (where $f_i < T$): $g_i = prop_i$

$prop_i$ is a function with the properties in Figure 4.

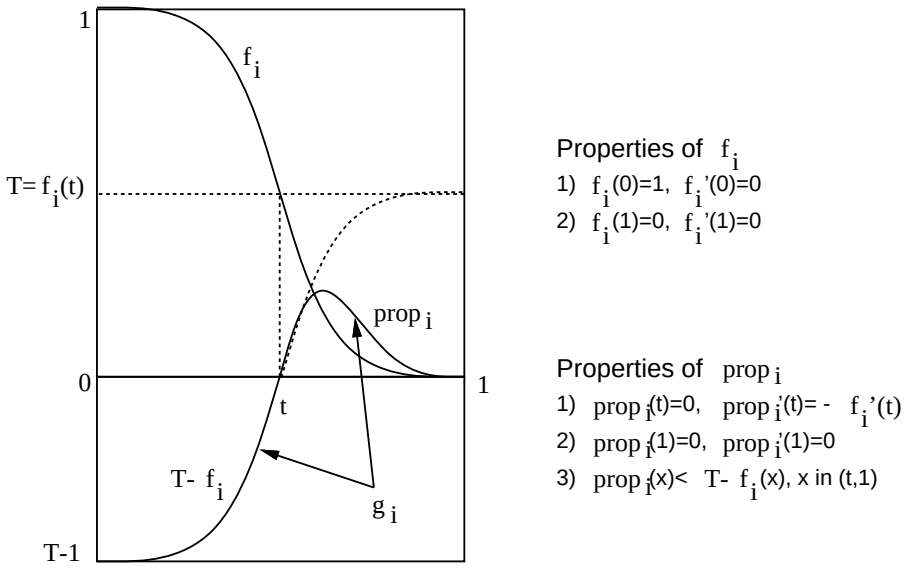


Figure 4: Collision Ghost Function

In the region where interpenetration occurs (see Figure 2) the ghost function object i imparts to object j is $g_i = T - f_i$. Thus the implicit surface of object j in this region shifts to points where $F_j(P) + (T - F_i(P)) = T$ or the median surface

where $F_j(P) = F_i(P)$. This symmetrically happens to the implicit surface of object i causing a common contact surface at points in the interpenetration zone where $F_i(P) = F_j(P)$ (see Figure 5).

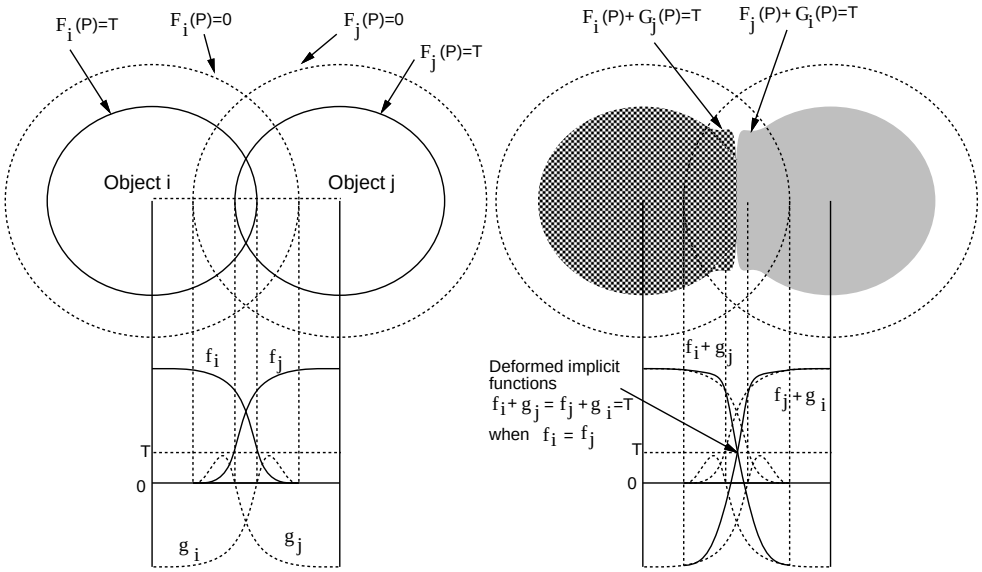


Figure 5: Collision Deformations

Stiffness along radial directions at points in the realm of influence of the implicit function of an object are modeled directly as the corresponding blend function gradient for distance surfaces. Enhanced ghost functions that model the rigidity of objects relative to each other are given in Appendix C.1.

2.2 Constraint Satisfaction

Satisfaction of constraints between objects is important for many VR applications. They are also important for human figures performing object grasping tasks in virtual

environments.

Most constraint satisfaction methods [2][17][19] do not address collisions between objects in the environment. The separation of collision and constraint processing into two stages poses a big problem. None of the existing constraint satisfaction approaches preserve a collision free environment. Satisfaction of constraints after collision processing can therefore cause new interpenetrations between objects. Similarly, application of current collision handling models after constraint satisfaction can cause a violation of the satisfied constraints. Another problem with most constraint satisfaction methods is their inability to handle constraint graphs with loops robustly. This is due to global changes to the objects in the presence of circular dependencies.

We aim at satisfying constraints between objects by local deformations that do not affect the object globally. One may envision a drop of honey dripping off the lid of a jar or a fly trying to escape from fly-paper as examples of such an approach. Local deformations make the changes to the object due to each constraint essentially independent of each other. The action of the constraints, just like collision deformations, is decoupled from the rest of the motion of the objects [19][20]. This new approach is built on the collision deformation method making collision detection deformation and constraint satisfaction a single integrated process.

2.2.1 Single Point-Point Constraint between Spheres

Let us start with the simplest case of a point-point constraint between two deformable implicit objects (for example, the spheres in Figure 6a). The objects are modeled as

implicit spherical primitives with a blend function f[3] as shown in Figure 1. The object i being defined as $f(D_i(P)) = f(t) = T$. D_i is called the *distance-ratio* function for distance surfaces. It is defined as $D_i(P) = |P - C_i|/r_i$, where C_i is the center and r_i the radius of influence of the spherical primitive i. The specified threshold value defining the object surface is T. Let $t = f^{-1}(T)$. There is a point constraint between points P_1, P_2 on the line joining C_1, C_2.

The ghost function corresponding to the interpenetration zone, $T - f$ deforms the objects inward such that the colliding object surfaces abut at a common surface. By using the same function for a part of the propagation zone we can cause objects to deform outward and abut at a common surface to meet constraints. This concept is formalized by the ghost function *con*. As can be seen from Figure 6a the function *con* is very similar to the collision ghost function in Figure 4. The difference lies in the region $x \in [t, t']$, where $con(x) = T - f(x)$ and $prop(x) < T - f(x)$. It is in this region that *con* ghost functions cause separated objects to abut.

The *con* ghost function value at points where the *distance-ratio* is t is zero. These points lie on the surface that separates the interpenetration and propagation zones. The value t' essentially determines the sub-region of the propagation zone around the interpenetration zone, where the positive ghost function values cause the outward deformation of the object surfaces to make contact. This results in satisfaction of constraints between objects. The constraint between two objects is satisfied only if the sub-regions of the ghost function defined by t' overlap. This imposes a cutoff

[3]The function f for each primitive can be different. We use the same function for both objects here to keep the presentation simple.

distance between the objects after which the approach will be unable to satisfy the constraint.

We define a function $con : [0,1] \rightarrow [T-1, T]$ with the following properties for the prespecified values t and t', $t' > t$:

- $con(x) = T - f(x)$, $x \in [0, t']$.
- $con(x)$ is a polynomial for $x \in [t', 1]$ with the constraints $con(t') = T - f(t')$, $con'(t') = -f'(t')$, $con(1) = 0$, $con'(1) = 0$ and $con(x) < T - f(x), \forall x \in (t', 1]$.

The negative region of the ghost function $T - f$ is used to prevent interpenetration between objects by causing the deformed objects to abut at a common surface. To satisfy point-point constraints, we exploit the positive region of this function in the region $[t, t']$, as in Figure 6a.

Let us use Q to refer to the mid-point of the line joining P_1 and P_2 as shown in Figure 6a. Q is also the point at which the spheres corresponding to $f(D_i(P)) = f(t')$, abut. Each object imparts a constraint ghost function *con*. $Object1$ is thus defined as $f(D_1(P)) + con(D_2(P)) = T$, $Object2$ is symmetrically $f(D_2(P)) + con(D_1(P)) = T$. The resulting function for points along the line joining the centers for $Object1$ and $Object2$ is shown in Figure 6b. The solid curves indicate $f(D_1(P)) + con(D_2(P))$ on the left and $f(D_1(P)) + con(D_2(P))$ on the right. The dashed curves correspond to functions $f(D_1(P)), con(D_1(P))$ and the dotted curves to $f(D_2(P)), con(D_2(P))$. It can be verified from the properties of *con* and Figure 6a,6b that the surface for $Object1$ is pushed out due to the positive contribution of $con(D_2(P))$ from where the

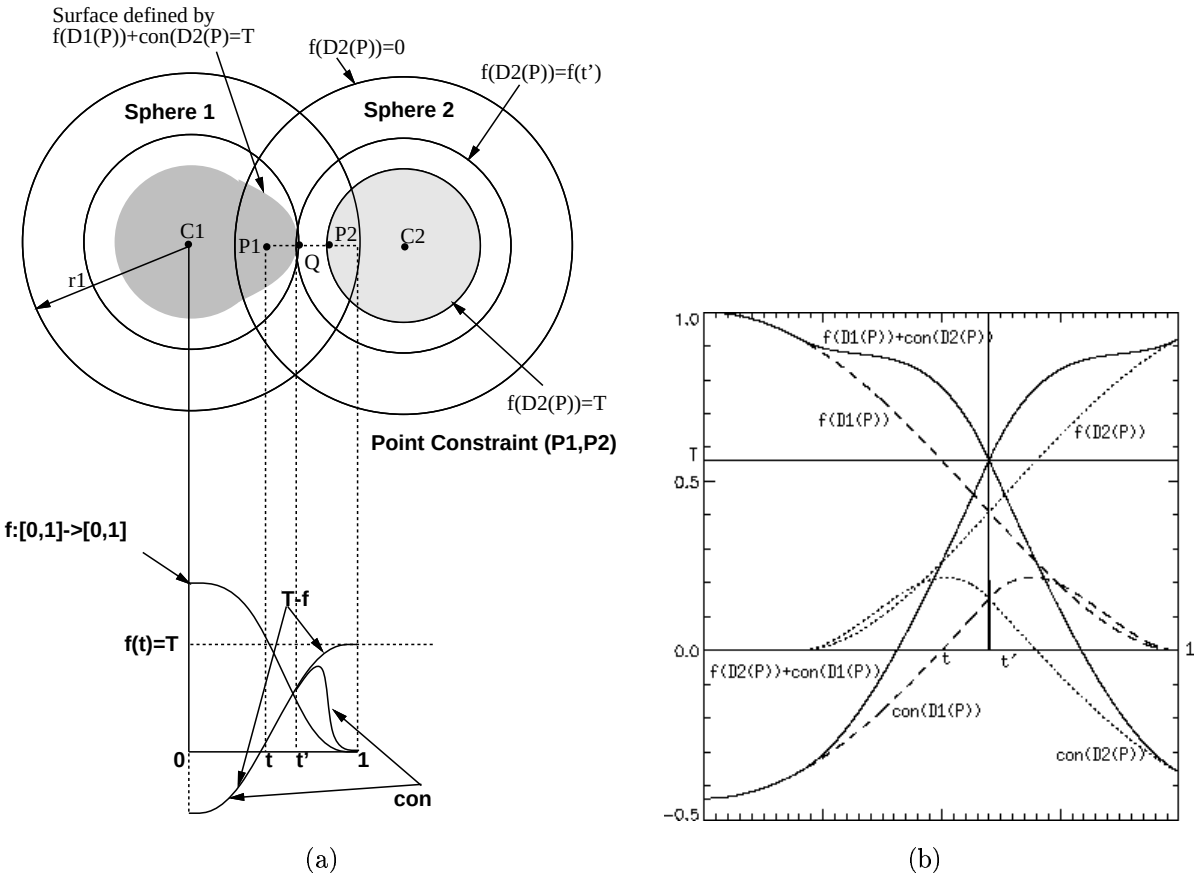


Figure 6: Point-Point constraint between spheres

second spherical primitive begins. The value at point Q on $Object1$ is $f(t')+T-f(t')$. Thus P_1 gets pushed out to Q on $Object1$, as does P_2 on $Object2$.

The objects deform appropriately to meet the constraint precisely. Moving the objects closer creates a contact surface and this continues to be the case even if the objects are colliding identical to the collision deformation method. Also note the failure to meet the constraint if the objects move further apart.

2.2.2 Implicit Objects with a single Point-Point Constraint

We now wish to adapt the above idea to general implicitly modeled objects with a single point-point constraint. We must, therefore, do away with many of the assumptions of the previous subsection:

- The implicitly modeled objects need not be spheres any more.
- The point constraint may be between any two points on the objects. The only assumption maintained is that the points be reasonably close. This is fixed by the value of t'.
- The objects involved need not be equally flexible.

Consider two arbitrary implicit objects with a point constraint P_1, P_2 as shown in Figure 7. The object i is still modeled by $f(D_i(P)) = f(t) = T$, $D_i(P)$ in this case being the distance-ratio function for the implicitly defined object.

If the objects penetrate each other at points P_1, P_2, the collision contact surface idea [20] satisfies the constraint. Suppose then that the objects do not penetrate each

other in some neighborhood of P_1, P_2, on their respective objects. Consider the line joining constrained points P_1, P_2. Let us call it the *deformation axis*. We will attempt to meet the constraint at some point Q, along the segment between P_1 and P_2. The position of Q models the relative rigidity between the objects.

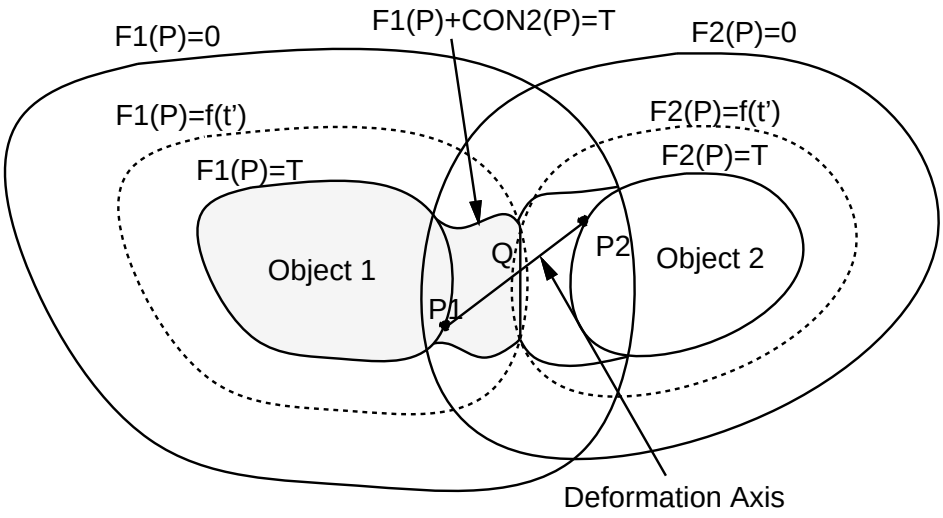


Figure 7: Constraint satisfaction without sphylinders

The following issues need to be addressed:

1. **Minimality of deformation**: The deformation should be minimal, in that it is just enough to maintain contact between the objects at some point along the deformation axis.

2. **Locality of deformation**: We would like to be able to localize the deformation to some controllable region around P_1 and P_2 on their respective objects.

3. **Directionality of deformation**: The deformation should be directed along the deformation axis.

Looking at these requirements with respect to Figure 6a, the placement of the spheres and the constrained points were such that the deformations were just enough to make the objects abut at Q. The deformation axis being radially aligned with respect to the two spheres made the deformations symmetrically oriented around the axis. Localization of the deformation was not addressed. Using the simple approach of the previous subsection with a deformation axis that is askew to the radial directions of the objects as in Figure 7 would violate all three stated requirements.

The following approach to satisfying the above requirements is proposed. Let j be the index of the constraint P_1, P_2. Define a realm of deformation with a parameter rd_{1j} and rd_{2j} for the each of the two objects for the given constraint. This defines two semi-infinite sphylindrical realms formed by $(P_1, < P_1, P_2 >, rd_{2j})$ and $(P_2, < P_2, P_1 > , rd_{1j})$, where each realm is defined by the point on the object, the deformation axis in the direction of the point on the other object and the realm of deformation of the other object as the radius.

A finite volume V_{1j} is now defined by the intersection of the volume $(f(D_1(P)) \geq 0) - (f(D_1(P)) \geq T)$ and the semi-infinite sphylinder $(P_1, < P_1, P_2 >, rd_{2j})$. V_{2j} is defined symmetrically (see Figure 8). These volumes define the regions for the imparted constraint ghost functions.

We now need to define the implicit functions within these volumes that realize the deformations. The ghost function *con* must be modified so that the resulting ghost

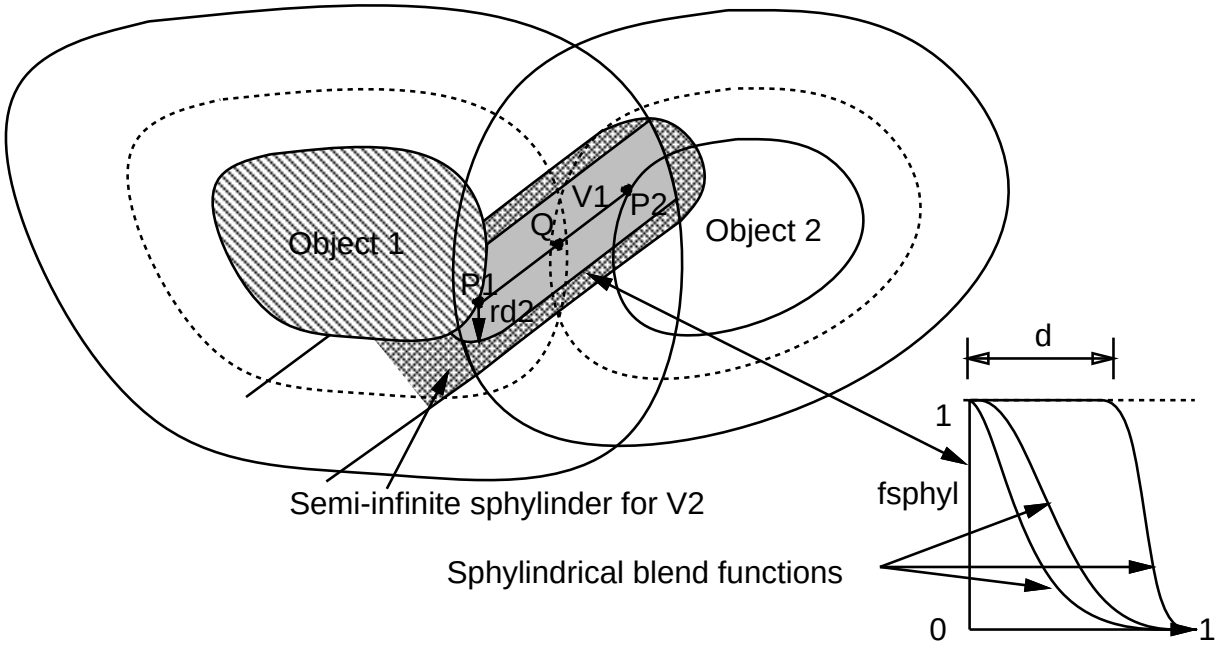


Figure 8: Definition of sphylindrical ghost functions

function is continuous at the boundaries of its realm of influence.

Define an implicit function $CONSTRAINT_{ij}(P)$ as follows:

1. If P lies outside $(f(D_i(P)) \geq T)UV_{ij}$, $CONSTRAINT_{ij}(P) = 0$, else

2. If P lies on or inside $f(D_i(P)) \geq T$, $CONSTRAINT_{ij}(P) = CON_i(P)$, else

3. If P lies on or inside V_{ij}, $CONSTRAINT_{ij}(P) = F_{sphyl_{ij}}(P) * CON_i(P)$, where $F_{sphyl_{ij}}(P)$, is the function value due to the sphylindrical primitive defined by V_{ij} (see Figure 8,Appendix A.2).

The deformed $Object1$ is now simply $F_1(P) + CONSTRAINT_{2j}(P) = T$. Similarly $Object2$ is $F_2(P) + CONSTRAINT_{1j}(P) = T$ (see Figure 9).

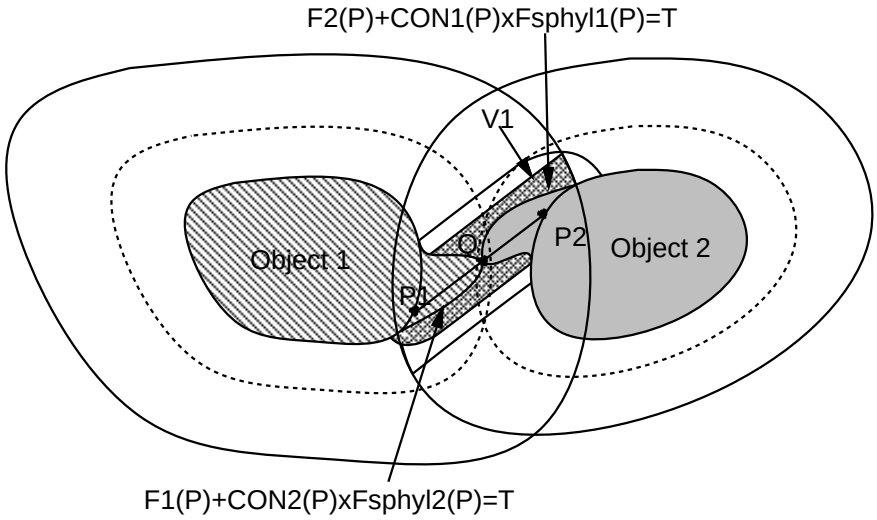


Figure 9: Constraint satisfaction with sphylinders

The relative rigidity of objects here can be modeled in two ways. The first is a direct extension of the approach for collision deformations. This approach can be found in Appendix C.1. Alternatively a simple solution for constraint satisfaction is as follows:

1. Let Q be the point on the deformation axis at which the constraint is to be met. The position of Q along the deformation axis can be arbitrarily fixed based on the relative rigidity of the two objects. Let $w = F_1(Q)/F_2(Q)$.

2. Now $con_1(x) = T - f_1(x)/w$ and $con_2(x) = T - wf_2(x)$ for $x \in [0, t']$. The continuity constraints for the remaining polynomial functions for $con(x), x \in [t', 1]$ are suitably altered.

It can be easily verified that the constraint ghost function *con* given above causes the constraint to be met at point Q on the deformation axis (see Figure 10). The

function in the previous subsection is a special case of this function with $w = 1$[4].

We will now verify the desired properties due to the proposed formulation.

1. **Continuity Properties**

   Let us first look at the continuity properties of the ghost function $CONSTRAINT_{ij}(P)$. Being an algebraic combination of continuous functions, it is continuous within each of the sub-volumes within which it is defined. We must, however, verify continuity at the sub-volume boundaries.

   - At the boundary of V_{ij} and the outside: $CONSTRAINT_{ij}(P) = F_{sphyl_{ij}}(P) * CON_i(P) = 0$, either due to the sphylinder boundary $F_{sphyl_{ij}}(P) = 0$ or the object boundary, where $CON_i(P) = 0$. Similarly $\nabla CONSTRAINT_{ij}(P) = (\nabla F_{sphyl_{ij}}(P)) * CON_i(P) + F_{sphyl_{ij}}(P) * (\nabla CON_i(P)) = 0$ on the boundary. This ensures continuity with the outside.

   - We only have zero order continuity at the boundary of $F_i(P) = T$ and V_{ij}. We use the assumption that objects do not interpenetrate in the sphylindrical neighborhood around the deformation axis. The deformation caused by $CONSTRAINT_{ij}(p)$ thus always lies within V_{ij}. It only touches the $F_i(P) = T$ boundary at its intersection with the deformation axis, when the Object i is rigid. At this point we have functional continuity as the function along the deformation axis is $CON_i(P)$.

[4] This approach may also be used for collision deformations that exclude rigid bodies. Rigid bodies pose a problem because the contact surface is fixed by the shape of the rigid object. The calculated value w is unlikely to be the same for all points on this predetermined contact surface.

This proves that the deformed object surface will possess the same continuity properties of the functions from which it was obtained (C_0, C_1 continuity for the functions we use).

2. **Constraint satisfaction**

   We must verify the satisfaction of the constraint at some point Q on the deformation axis between P_1 and P_2 controlled by w.

   Along the deformation axis in V_{ij}, $F_{sphyl_{ij}}(P) = 1$. Thus for $Object1$ we solve for $F_1(Q) + CON_2(Q) = T$, and for $Object2$ we solve for $F_2(Q) + CON_1(Q) = T$. We now use the precondition that the constrained points are close to each other. At point Q, $CON_1(P) = T - F_1(P)/w$ and $CON_2(P) = T - w * F_2(P)$ for the objects. The point at which the constraint is met on the deformation axis, thus has $F_1(P) = w * F_2(P)$, which by definition is point Q.

   If the assumption of proximity fails the deformation functions are smaller than those just mentioned. The objects in this case do not deform enough to satisfy the constraint at a common point. This may be useful in modeling a gradual breakaway of objects. Further, the deformations reduce and finally disappear as the constrained points get further apart. This can be clearly seen for many of the constraints in Figure 10.

3. **Minimality of deformation**

   One may expect a common contact surface area around Q, comprising points where $F_1(P) = w * F_2(P)$, like in Figure 8. The deformation ghost function is

$F_{sphyl_{ij}}(P) * CON_i(P)$. $f_{sphyl}(D_{ij}(Q)) = 1$ and $f_{sphyl}(D_{ij}(P)) < 1$ for all other points P on the hypothetical contact surface, as they are a finite distance away from the sphylindrical deformation axis. Thus the deformation term added would be less than that required for these points to lie on either of the two deformed objects. This proves the minimality of the deformation.

4. **Predetermined area of contact**

   By the same reasoning as above we can generate controlled contact surfaces. Suppose that the constrained points were close enough that the ghost function $CON_i(P)$ creates a larger contact surface than is desired. Define f_{sphyl} for the sphylinders to be 1 up to some distance d away from the sphylindrical axis, after which the function drops off satisfying its other requirements (see Figure 9). This generates an elliptical (due to the sphylindrical shape) contact area with a predetermined minor axis $d * min(rd_{1j}, rd_{2j})$, around the deformation axis. There is no sanctity in the use of sphylinders. Any semi-infinite implicit function may defined around the deformation axis such as a prism or even a cone-sphere. These can be used to generate contact surfaces of various non-elliptical shapes.

5. **Localised deformation around the deformation axis**

   The deformation of $Object1$ is essentially controlled by the volume V_{2j} which is controlled by the user parameter rd_{1j}. Thus even though $F_2(P) \geq 0$ may largely overlap $Object1$, the region of deformation on the surface of $Object1$ is restricted to the surface lying within the sphylindrical region. This can be

controlled individually for each object for each constraint. The sphylindrical functions also add a measure of directionality to the deformation.

2.2.3 Multiple Constraints

Arbitrary constraint graphs pose no problem. Each constraint is met via a local deformation without globally affecting the object, as can be seen in Figure 10,11. Local deformation ghost function values due to all constraints may be added up for each object. A problem does occur when there are deformation functions due to different constraints overlapping along deformation axes. This occurs either due to the deformation radii being large or if the set of constrained points are very close. This may be avoided by reducing the size of the deformation radii or, if this is unsatisfactory, by using an area constraint. An important observation is that this approach smoothly facilitates the satisfaction of multiple points of attachment between the same two objects, which is hard to do without singularities [20].

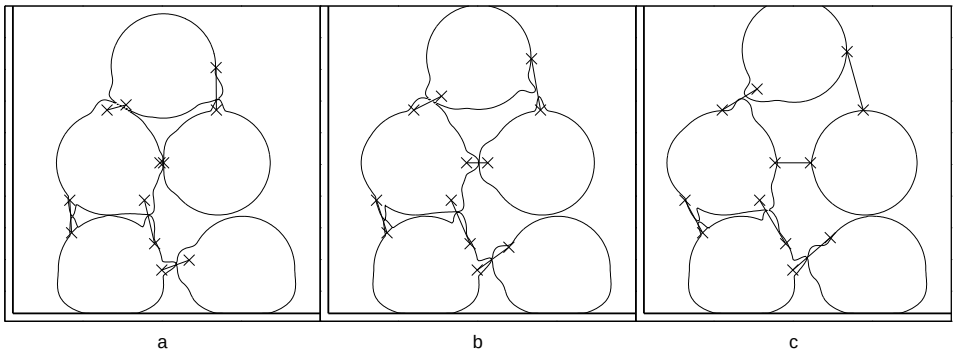


Figure 10: Animated Spherical Objects with Point-Point Constraints

2.2.4 Physical Properties

The same physical connotations as in the collision deformation method [20] may be attributed to the objects. Similar forces can be calculated. They may be used similarly in the next animation step. Equal and opposite reaction forces are applied at the constraint. Approaches that separate the characteristics of an object into rigid and deformable components [20][62] assume invariant object properties like mass and moment of inertia. Using this assumption and the fact that only balanced internal forces are added to the system, the first order momentum of the system is conserved [19].

2.2.5 Characteristics of the Approach

We have thus proposed and formulated an extension to Gascuel's [20] approach that facilitates constraint satisfaction between implicitly defined deformable objects. The chief characteristics of this technique are as follows:

- An integrated collision handling and constraint satisfaction model is provided.
- Physical forces can be calculated if required. First order momentum is conserved.
- There is good user control over the deformations.
- Each constraint independently deforms the object locally, allowing arbitrary constraint graphs.

- Precise constraint satisfaction precludes the need for iterations or convergence to a satisfying state.
- Point-Point constraints can be extended to Area-Area constraints, with a predetermined contact surface shape and area.
- Multiple points of attachment between the same pair of objects are easily handled.
- The approach may be integrated with a displacement based approach [19] for better results.

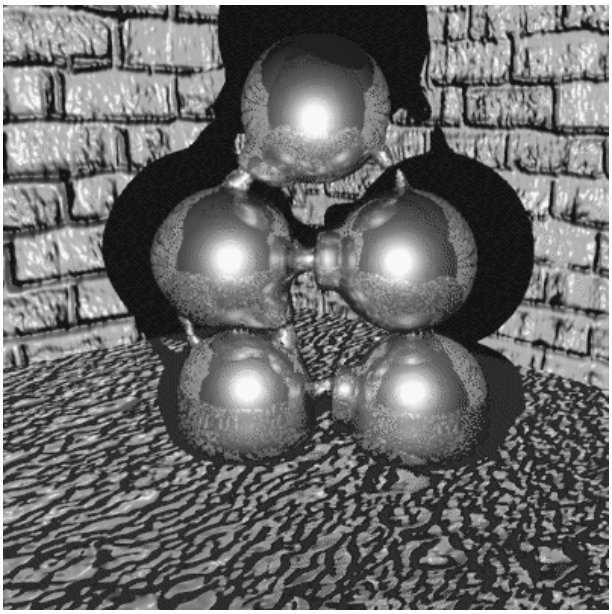


Figure 11: Constraint Satisfaction between Spherical Objects

2.3 Deformable Model applied to Polymesh Objects

2.3.1 Synthesis

Implicit functions for polymesh objects are constructed by immersing the polymesh model in a number of implicit surface primitives [68] that approximate the general shape of the object. The vertices of the polymesh object are then calibrated based on their implicit function values so that they all lie on some convenient implicit surface. During animation the vertices are transformed appropriately to stay on that implicit surface.

Implicit Primitive Selection

The shape of the implicit primitive [68] provides a bounded volume for the realm of influence of the associated function, as well as a mapping from a point within the volume to a value within the domain of the density function (to calculate the function at that point). A convenient threshold value T is chosen to define the implicit surface that approximates the polymesh object. The choice of T is influenced by the deformable characteristics of the object being modeled.

The choice of the implicit primitive shape for an object is influenced by the following requirements :

1. Implicit function computation for the primitive should be efficient.

2. Bounding volume intersection computation between the primitives should be efficient.

3. The primitive shape at some threshold value should fit the embedded region of the polymesh well [41]. The behavior of the polymesh region then closely follows that of the surface defined by the implicit primitive at that threshold.

4. The primitive bounding volume around the embedded polymesh region should be reasonably tight, so as to avoid wasted polymesh deformation computation during interaction with the environment.

Simple offset primitives (spheres, sphylinders, offset polygons) and cone-spheres (see Figure 1) meet the first two criteria well, which are independent of the object under consideration. The primitives for a particular object may be selected and positioned by the user as part of the object modeling stage (see Figure 12).

To avoid local bulges in the functional definition across the entire object these primitives should be treated as convolution surfaces [11]. For the sake of efficiency and the typically small number of primitives required, we adhere to the distance surface formulation. This problem will be further reduced by the calibration of polymesh vertices within the space of its implicit function definition, which takes this formulation into account.

Implicit Primitive Synthesis

Once appropriate primitive shapes for an object are selected, they must be fitted to the underlying polymesh data. For user selected offset primitives, the skeleton S may be positioned using a combination of user interaction and simple bounding box based techniques. Each polymesh vertex then contributes to the synthesis of the implicit primitive(s) whose skeleton it is closest to. Two radii R_{out} and R_{in} are then

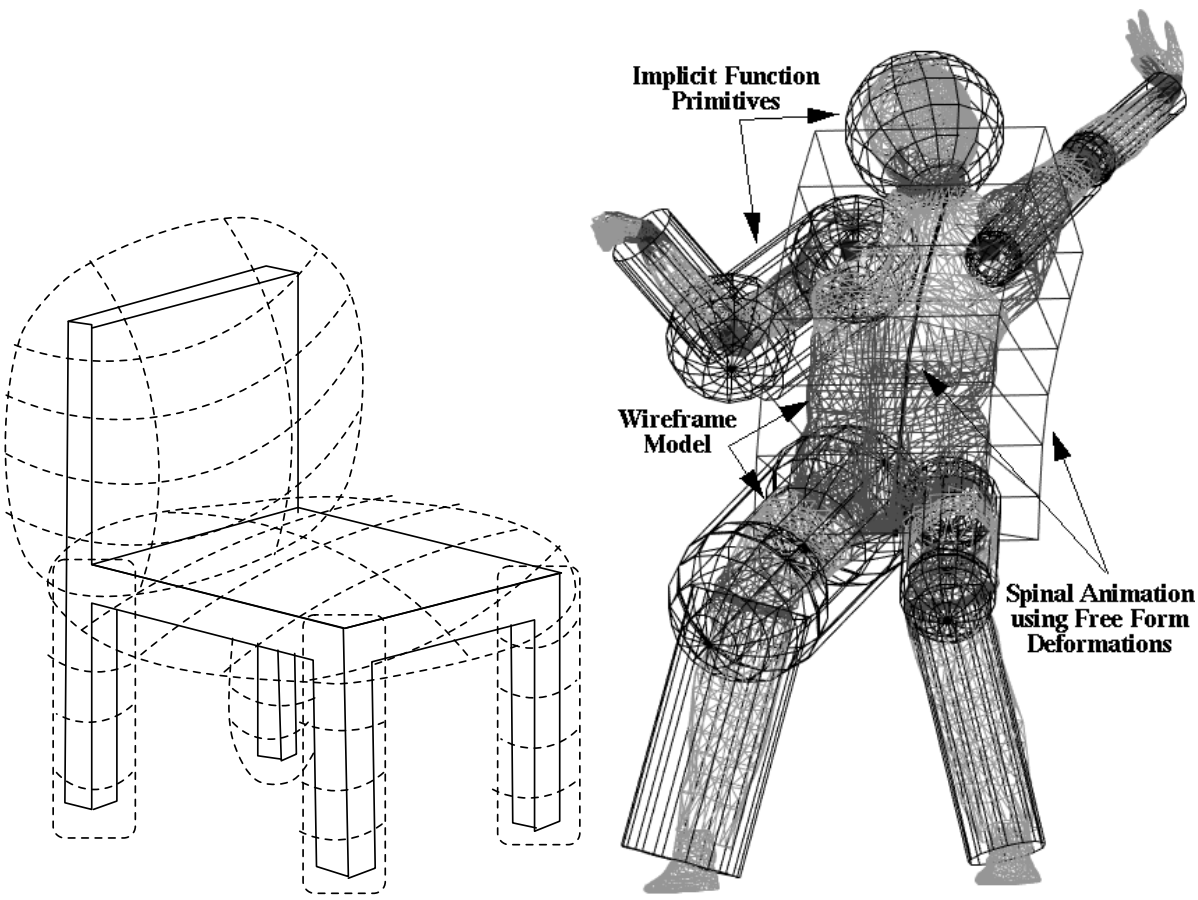
calculated for each offset primitive. These are based on the distances of contributing polymesh vertices from the primitive skeleton S. R_{out} is the maximum and R_{in} the average of distances of contributing vertices from S. R_{out} is the bounding radius of the primitive and R_{in} the surface of best fit. From the definition of R_{out} and the fact that every vertex contributes to some primitive, we are ensured that the implicit function definition completely envelops the polymesh object. The primitives within whose realm of influence a polymesh vertex lies, are now called the defining primitives for that vertex.

$W = T/f(R_{in}/R_{out})$ is the shape weight for the primitive, where f is the associated density function (see Figure 1). The function value for primitive i at a point P is then $F_i(P) = W_i * f_i(\textit{distance-ratio}(P))$, where *distance-ratio* is defined as in Section 1.2.1. Thus the shape at threshold T of the primitive is the offset surface with radius R_{in}.

Figure 12 shows primitives fitted to a human polymesh model. Bounding volumes for the right half of the body and primitives of best fit for the left half are shown. Complex primitives such as cone-spheres or superquadrics may be fitted using existing techniques [41][55].

Once the implicit primitives have been synthesized, the polymesh model needs to be calibrated. Here a weight attribute for each vertex is calculated, ensuring that the implicit function model produces no deformation to the polymesh object in the absence of any environmental interaction.

Polymesh Vertex Calibration



(a) Chair (b) Human Figure

Figure 12: Implicit Primitive Synthesis on Chair, Human figure

For a vertex v on the prototype polymesh, let $DF_v(P)$ be the implicit function computed based on its defining primitive(s) at a point P in space. The implicit function for a vertex v at point P, based on its defining primitive(s), $DF_v(P)$, is computed as a sum of $F_i(P)$, where i runs over the defining primitives. v is calibrated by assigning a weight $w_v = T/DF_v(P)$, where P is the spatial position of the vertex on the undeformed polymesh. In the absence of environmental interaction the implicit function for the vertex at P is, $F(P) = w_v * DF_v(P) = T$, ensuring that the polymesh lies on the implicit surface determined by the implicit model at threshold T. Note that the unwanted bulge problem occurring due to overlap of multiple defining primitives [11] is attenuated for vertex v by the use of shape weight w_v in the functional calculation.

This completes the construction of the implicit model and calibration for polymesh object.

2.3.2 Animation

Rigid bodies are well represented by polymesh models and can be animated efficiently and robustly. For the rigid component it suffices to subject the polymesh model, its enveloping implicit primitives and the primitives for the implicit ghost functions to the rigid body transformations specified by the animation. The result of this rigid component transformation forms the input to the deformable component model. The results before and after the deformable component transformation can be seen in Figure 3.

The individually manipulated implicit primitives and ghost functions for an object now interact to appropriately deform the embedded polymesh model. Deformation of the polymesh models is carried out by deforming the position of each vertex of the model from its current position P to a point P', such that $F(P') = T$.

Function Computation at Polymesh Vertices

Ghost functions for the implicit object at a point P are of the form $G(P,t)$, where t is a temporal parameter. We modify these functions for polymesh objects to permit additional control for each polymesh vertex. Thus every ghost function is of the form $G_v(P,t)$ for the vertex v. The implicit function for v at P at time t is then $F(P) = w_v * DF_v(P) + \sum_i G_{i,v}(P,t)$, where i runs over all ghost functions G for the object. To illustrate this, the ghost function that models collision deformations [20] $G_{i,v}(P,t) = COLL_{i,v}(P)$, where i runs over all interacting objects (whose $F(P) > 0$). $COLL_{i,v}(P)$ is the collision deformation function [20] imparted by object i with a minor difference (appropriate multiplication by the vertex weight w_v):

Penetration zone: $COLL_{i,v}(P) = T - w_v * F_i(P)$.

Propagation zone: $COLL_{i,v}(P) = w_v * PROP_i(P)$, where $PROP_i(P)$ is the propagation function. The introduction of t into the above formulation to model temporal elastic effects is discussed in Appendix C.

The vertex v is constrained to a point P, where $F(P) = T$, or for example in the presence of a colliding primitive i, the common collision contact surface where $DF_v(P) = F_i(P)$. Proof of polymeshes deforming to generate common collision contact surfaces is straightforward.

Polymesh Deformation

Having laid the theoretical foundation that deforms the polymesh model, we address the algorithmic aspects for a practical implementation. The algorithm for deformable component transformation of objects is carried out in 3 steps as follows:

1. A list of interacting objects is constructed for each object in the environment. Interacting objects impart ghost functions to each other. The ghost functions are controlled by the imparting objects and a temporal parameter.

2. The vertices of each polymesh object are then deformed based on function values computed using its defining primitive(s) and the objects ghost functions.

3. Further processing using the deformed polymeshes takes place.

Step 1 benefits from efficient intersection computation of the bounding volumes of primitive shapes. A simple analytic solution (see Appendix A) exists for intersection testing between simple shapes like offset surfaces, which may be exhaustively intersected with each other. Spatial subdivision techniques such as octrees may also be used.

Step 2 is the crux of the algorithm. Every vertex of a polymesh object must now be deformed based on an implicit function F. This function is specific to the vertex and is calculated as described in Section 2.3.1. The position of this vertex must be deformed from its position P computed by the rigid component transformation, to a point P', such that $F(P') = T$. As the implicit function defines a continuous implicit surface in the neighborhood of P, the deformation mapping of P to P' for the vertex

is ill-defined. This is an artifact of using a discrete polymesh representation to follow the deformations of a continuous implicit representation. The solution proposed is to deform P along its vertex normal. Alternatively, P maybe deformed along $\nabla F(P)$ i.e. the normal to the implicit function at that point (see Figure 13).

For offset primitives, the distance ratio function for any point along a ray can be represented in terms of its parametric distance along the ray (see 3.3.1). Thus P' may be obtained analytically along any ray, by solving a sequence of quartic equations by using a quartic density function. Alternatively, a hybrid Newton-Raphson, Regula Falsi iterative technique may be used which is efficient in this case, as the deformations, being incremental, are small.

Step 3 deals with the computation of surface normals or other spatial attributes of the polymesh or any desired parameters based on the deformed polymesh. Additionally, for systems with force feedback, integration of forces at individual vertices of the object and collision contact area computation may be done in this step.

The worst case complexity of the above algorithm is $O(m^2 + mn)$, where m is the number of implicit primitives (including those used to model the ghost functions) and n the number of polymesh vertices in the environment ($n >> m$). Despite fast octree based techniques for coarse collision detection, precise detection and handling using conventional polygon methods is $O(n^2)$ in the worst case [22][51], making this approach superior as the complexity of virtual worlds increase.

2.3.3 Dynamics

A physical muscle interpretation, which in conjunction with the skeleton controls the animation of the geometric model, is presented in Chapter IV. Variations in tissue characteristics [60] are modeled by piecewise smooth density polynomials whose gradient reflects the change in stiffness, making reaction force computation for a vertex simple [58]. Area computation of polymesh faces whose vertices are deformed to lie on a collision contact surface [20] may be done in *Step 3* of the algorithm in Section 2.3.2. These areas can then be used in the computation of area dependent reaction and friction forces completing a force feedback loop.

2.4 Articulated Deformable Objects

Here we address extensions required to coherently integrate articulated deformable objects like human figures into the proposed framework.

The synthesis of the implicit function embedding the articulated figure involves the implicit function synthesis for each articulation of the body as an independent polymesh object described above. Additionally a primitive (typically a sphere) is synthesized around each joint. The joint primitives are responsible for maintaining smooth connectivity across the joints. The realm of influence of the joint primitives is influenced by the polymesh vertices around the joint. The polymesh vertices are thus refered to as either joint vertices or limb vertices (see Figure 13).

The implicit function for a vertex v at point P, based on its defining primitive(s), $DF_v(P)$, is computed as follows :

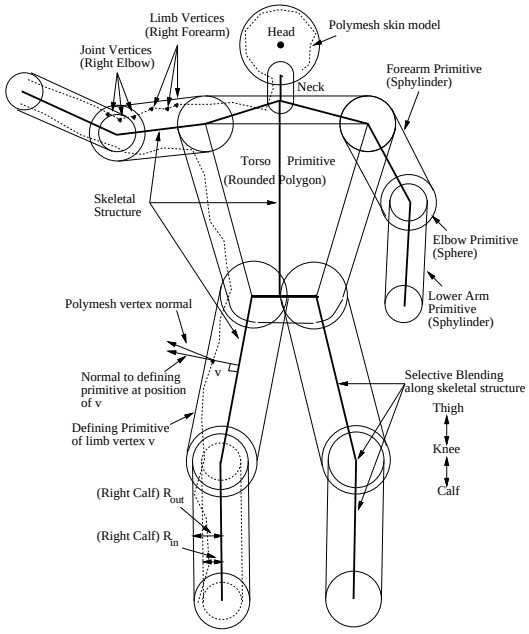


Figure 13: Implicit Primitive Hierarchy

Limb Vertex: $DF_v(P) = F_i(P)$, where i is the defining limb primitive.

Joint Vertex: $DF_v(P) = F_i(P) + DIFF(F_j(P), F_k(P))$, where i is the joint primitive and j, k the limb primitives. The $DIFF$ function, like $COLL$ [20] models the formation of creases at joints (see Figure 3). An example of $DIFF$ is $DIFF(a, b) = |a^n - b^n|^{1/n}$.

During animation the rigid component is determined by a skeletal structure; the deformable component is determined by the interaction with other objects in the environment. This model is particularly well suited to human figure animation. For the rigid component it suffices to subject the polymesh model to the rigid body transformations specified by the underlying skeleton. Care is taken to preserve connectivity and continuity around joints. This may be done using free form deformations [14].

Alternatively the quarternion based rotation around a joint is interpolated for vertices around a joint [58]. This maintains smooth connectivity around joints but does not provide much shape control or address collisions in the joint region.

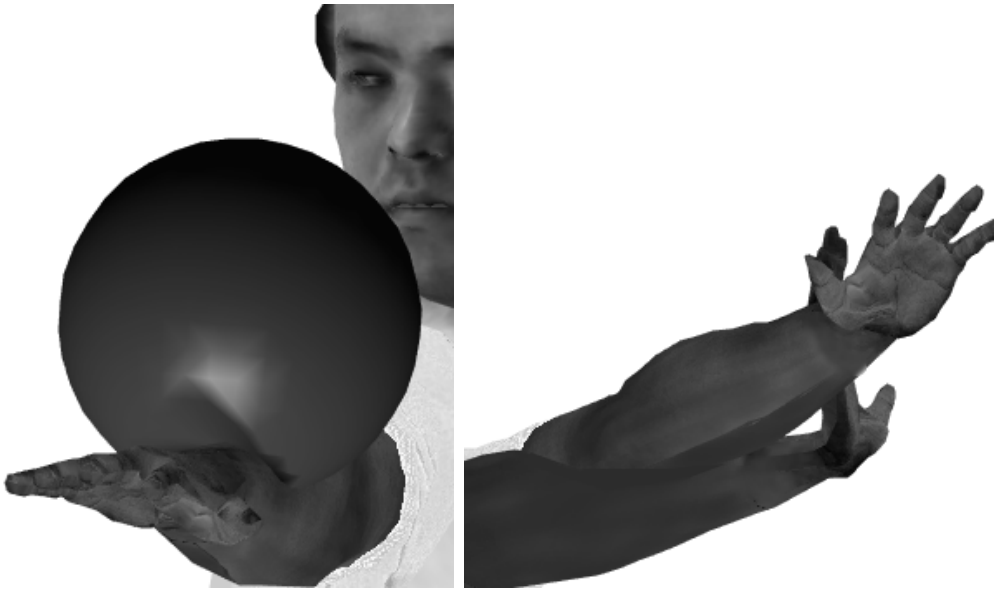
Articulations are treated as different implicit objects in the environment that interact with each other for collision detection and deformation. Thus self-collisions between different parts of the body are homogeneously handled alongside collisions with any other objects in the environment. Joint regions, where both blending and collision deformations occur, are handled by one or more joint primitives that blend the two limb primitives together (see Figure 3).

2.5 Implementation

This model enveloped human figures and other objects for a real time virtual world simulation on a SGI (Crimson, Reality Engine) machine.

The human figure polymesh ($\approx$ 7500 vertices) is embedded in 23 implicit primitives. The implicit model is constructed hierarchically, in an object oriented fashion, making the fitting of any polymesh object as well as the introduction of new implicit primitive shapes a simple task.

Figure 3 shows the elbow region after rigid component transformation and its subsequent deformable component transformation. Spherical primitives modeling skeletal elbows cause the elbow to protrude in the bent arm and precise crease formation may be seen. The deformable component computation using a Regula Falsi-Newton Raphson approach typically takes 2-3 iterations per vertex.



(a) Arm-Ball (b) Arm-Arm

Figure 14: Collision Deformations:1

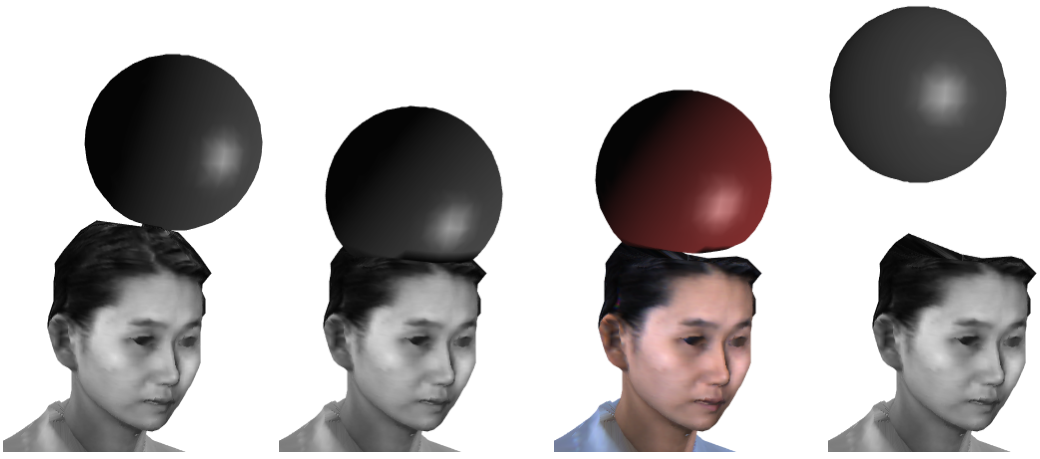


Figure 15: Collision Deformations:2

Figure 14a shows deformations of an arm and a ball on collision. Figure 14b illustrates the homogeneity of collision deformation on two arms belonging to the same articulated figure. Figure 15 shows a ball bouncing off a head. The head is given a plastic attribute and the ball a viscoelastic one so as to clearly show the deformations that result.

Existing polymesh based techniques may be integrated with the rigid component in our implementation. As an example, free form deformations [14] on the spine are used to animate the torso of the human figure (see Figure 12). The spatial positions of the torso vertices prior to the free form deformation are used in the torso implicit primitive function calculations.

2.6 Discussion of Results

A model for the synthesis and animation of objects with a polymesh representation is presented. The physical characteristics of the object are separated into rigid and deformable components. The implicit function based deformable component performs collision detection and handling with a linear time complexity in terms of number of object vertices, which is important when dealing with complex virtual worlds [51].

The implementation of the presented concepts shows their effectiveness both in terms of computation speed and the degree of realism obtained. The separation of the physical characteristics of objects into rigid and deformable components works particularly well for human figures. The model handles self-collisions of the body and skeletally based deformations (crease in Figure 3) elegantly. Physical characteristics

can be overlaid on the model for dynamic simulations [58].

The ability to apply implicit function techniques in general to existing polygon based data is an important advantage of our approach. It can unify and be integrated with existing polygon based or implicit surface based modeling and animation systems.

The under-constrained nature of the deformable component mapping of polymesh vertices may cause surface consistency problems. A dense well fitted polymesh such as the human figure, causes simple displacements along vertex normals to give good results without vertices bunching together or diverging abnormally. Incorporation of techniques [67] that adaptively subdivide and coalesce the polymesh in real time is a possible solution.

There is scope for future work on construction and fitting of primitives to the polymesh. Poor fitting primitives may result in very close objects being deformed to abut at the implicit model contact surface. For applications where visual realism dominates over spatial accuracy, the above artifact does not pose a problem. Using a greater number of primitives and more complex primitives improves the fit but degrades implicit function and bounding volume intersection computation efficiency. An empirical tradeoff between a better fit and computation efficiency should, therefore, be taken into consideration.

CHAPTER III

Polyhedral Shapes as Implicit Surface Primitives

A useful set of implicit surfaces can be generated as an algebraic combination of polynomial functions each of which is defined over a finite volume. The behavior of each polynomial function in space is intuitively governed by a geometric volume known as an implicit primitive shape. Examples of primitive shapes introduced in 1.2.1 are spheres, sphylinders, cone-spheres and rounded polygons.

Object modeling and animation with existing implicit surface primitives has certain limitations. Realistic objects, such as bone structures and plants, often exhibit a smooth amorphous nature, combined with sharp features which can be hard to model precisely with the aforementioned primitives. Though a superquadric primitive can be fitted to a 3D point data set [55], a precisely defined B-rep data model, can only be approximated by existing implicit surface primitives. Very thin objects that are best represented by surfaces, like paper or leaves may be difficult to model robustly as thin volume primitives. Often a large number of simple primitives may be needed to model certain objects realistically. An energy minimization technique was presented by Muraki [41], where summed implicit spherical primitives fit data points. The order of hundreds of primitives are required in general for a good fit.

This makes the rendering of the object computationally expensive [8], and its animation by manipulating primitives, cumbersome. Rendering techniques, special effects, anti-aliasing techniques and such issues have to be specially addressed for analytically defined shapes, often independently for each primitive.

On the other hand many of the advantages of implicit functions such as properties of smoothness, changes in topology, precise collision contact surfaces are hard to achieve with B-reps in a unified elegant fashion. The ability to integrate B-rep models with simple analytic shapes (which have complex B-rep approximations), would also be useful.

Many modeling and animation techniques individually developed for solving particular problems using B-rep models are easily and elegantly accomplished using implicit functions. As examples of such techniques, Ricci [53], converts CSG operations of union and intersection into algebraic functions that can blend objects together with a varying degree of smoothness. A technique for generating a controllable fillet surface between two planar facets of an object was presented by Middleditch [39]. Spring and damper techniques, [59] physically model deformable B-rep based objects. Free Form Deformations [14][56] are a very popular technique for modeling the deformable nature of objects modeled as B-reps.

We thus argue the case for a closer link between B-rep and implicit function modeling and animation paradigms. In this chapter we propose definitions for B-rep objects as implicit modeling primitives. This facilitates the direct application of implicit function techniques to B-rep models. Within an implicit surface modeling

system, B-rep objects become a powerful user definable primitive. Polygonization of general implicit surfaces [9][54] allows the user to easily switch between B-rep and implicit surface models. This toggling between models is effective in incremental and hierarchical object modeling and animation. The use of polyhedral implicit primitives is discussed in detail. We generalize in two ways on the idea of a skeleton and a radius of influence defining an implicit primitive. Existing implicit primitive shapes are special cases of this generalized idea.

The display of objects modeled and animated using implicit surface primitives is then addressed. Techniques for scanline display, ray tracing and polygonization of polyhedral primitives are presented.

The above approach makes B-rep and implicit surfaces representations completely interchangeable. The inherent display inefficiency due the implicit surface representation unfortunately remains, and is possibly increased. This problem is overcome by the indirect deformation approach of Chapter II. The methods in this and the previous chapter thus provide two alternatives. For cases where display efficiency is the foremost requirement, the indirect deformation approach of Chapter II should be used. In other cases the techniques in this chapter are more general and likely to produce better looking results.

Chapter II presents a technique by which implicit functions deform polyhedral structures. Further work needs to be done before this approach can be applied to complex deformations such as those involving a change in object topology. The approach also has other limitations resulting from the indirect nature of the deformations.

The algorithms of this chapter are easy to duplicate and may be incorporated into any existing B-rep or implicit surface based modeling and animation system.

The rest of this chapter is organized as follows:

Section 1 develops the generalized notions of the implicit primitive shape. Section 2 describes the usage of polyhedral shapes as primitives. Section 3 addresses the display of implicit surfaces using scanline rendering, ray tracing and polygonization. The emphasis is on the rendering of polyhedral primitives. Section 4 gives implementation details and presents the results. Section 5 presents conclusions and a discussion of results obtained from the approach.

3.1 General Implicit Surface Primitive Shape

The implicit primitive definition in 1.2.1 was based on a skeleton which along with a cutoff radius, fixed a finite volume as the realm of influence of the primitive.

We define each primitive shape as $< V, S, f >$. V is a finite volume, S is a skeleton within the volume, and f is a blend function. The primitive only contributes to the surface within V. For a point P within V, the function value is determined by first computing a value $\in [0, 1]$ called the *distance-ratio*. $f(\textit{distance-ratio})$ is then the function value. If Q is the closest point on S to P, then the distance-ratio is computed by taking the ratio of $|P - Q|$ (from P to a point Q on S), and a distance $|P' - Q|$, where P' is determined by the shape of V. Point P' is the intersection of the ray $\overrightarrow{QP}$ with the boundary of volume V. We assume that the intersection of $\overrightarrow{QP}$

with the boundary of volume V is unique. For points P outside V, $F(P) = 0$. This is a more general formulation than the currently accepted one in Chapter II. This formulation reduces to the previous definition in the special case that the boundary of V is an offset surface at distance R from S.

The shape of an implicit primitive V thus provides a bounded volume, the interior of which is the realm of influence of the associated blend function. In addition, V provides a mapping from a point within the realm of influence to a value within the domain of the blend function, which is used to calculate the function at that point.

We use this formulation to provide implicit primitive definitions to B-rep shapes which typically represent the surface of V.

Scaled volumes result from the isolated primitive shape at various thresholds when S is a single skeletal point contained within V. In general, isolated primitive shapes for various thresholds are an intermediate shape between the boundary of V and S. Typically, we would like S to be a simple surface such as a point-set, sphere or polygon. The closest point calculation given a query point is computationally efficient for such skeletons and the intermediate isolated shape reasonably predictable.

The restriction imposed by the assumption of a unique intersection point is equivalent in statement to the following:
Let V_P be the subvolume of V, comprising the points whose closest point on S is the same point P. Then V_P must be star-shaped with respect to P. This is illustrated in Figure 16.

Most existing primitives are special instances of V and S.

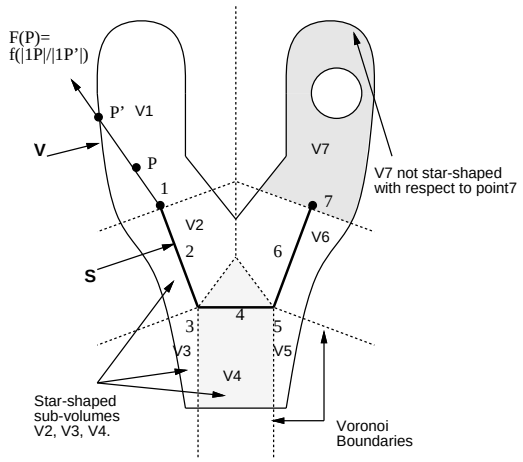


Figure 16: Restriction on shape of V,S

- For spheres, V is the bounding sphere defined by a radius r around a center C. S is just the point C. Similarly, superquadric ellipsoids are the superellipsoid volume V around the center C.

- Sphylinders (rounded polygons) have as V, the bounding sphylindrical (rounded polygon) shape, defined by radius r around S which is the line segment (polygon) of key points. For cone-spheres, the bounding volume V is the cone-spherical shape [38] around S, the line segment formed by the key points.

- Superquadric toroids [4] have as V, the bounding toroidal shape and as S, the circle defined by the hole parameter α.

3.2 Polyhedral Implicit Primitives

The complexity of modeling, animating and display of objects built from implicit primitives depends on:

1. The number of primitives.

2. The complexity of the primitive determined by the complexity of evaluating the parameters of the primitive at a point, such as the implicit function value, gradient or color value.

3. The combination complexity of the structure that combines all the primitives to obtain the implicit object.

Polyhedral primitives aim at shifting the complexity of the implicit object from a large number of simple primitives to a small number of primitives of higher complexity. This also reduces the combination complexity as the number of primitives to be combined decreases.

A general polyhedral primitive (referred to as a **polyblob**) is simply the generalized implicit primitive shape, where V, S are defined using polyhedral B-reps. An instance of the primitive thus has the polymeshes, V and S, appropriately positioned and f, its associated blend function.

For the polyblob to be a useful and intuitive primitive it is imperative that its shape as an isolated primitive at various thresholds be fairly predicatable. Thus for most of the thesis, S for a polyblob is assumed to be a point contained within V, unless otherwise specified. Such a polyblob is referred to as a **pointblob** (see Figure 17).

3.2.1 Pointblobs

The pointblob as stated above is thus defined as $< V, S, f >$, where V is a non empty bounded volume, S is a skeletal point contained within V and f is a blend function. The realm of influence of f is the interior of V. Given a point P within V, let the ray from S through P intersect the boundary of V at point P'. Let us restrict V, S to be such that the intersection P' is unique for all points P. The polygon of V containing P' is called the **defining polygon** of P (see Figure 17). Define $g : \mathcal{R}^3 \rightarrow [0, 1]$. For a point P, $g(P) = min(1, \frac{|P-S|}{|P'-S|})$. Clearly $g(P)$ is a value that ranges from 0 at S to 1 for points on or outside the boundary of volume V. Now the function value at P is $F(P) = f(g(P))$. $F(P)$ thus varies from a value of 1 at S, to 0 for points on and outside the boundary of volume V. Scaled volumes result for the isolated primitive shape at various thresholds (see Figure 17). Given a threshold T, let W be this isolated shape. We can equivalently define the pointblob in terms of W, T instead of V (since V can be uniquely obtained from W, T, f). In fact modeling with the shape W is more intuitive than with V as it gives a better indication of where the implicit surface at the threshold T lies.

The computation of g associates a unique point P' on V, with every point P within the realm of influence of the primitive. This point P' can be used in determining the normal, texture and other object attributes for the primitive at point P. The attributes for all contributing primitives are weighted appropriately to obtain the attribute at P for the implicit surface.

There may be cases, such as Figure 20, where clearly a single point within the

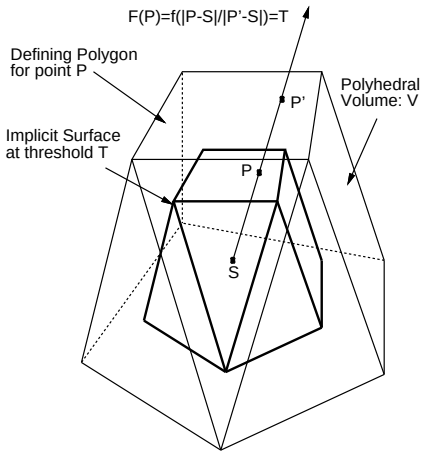


Figure 17: Pointblob

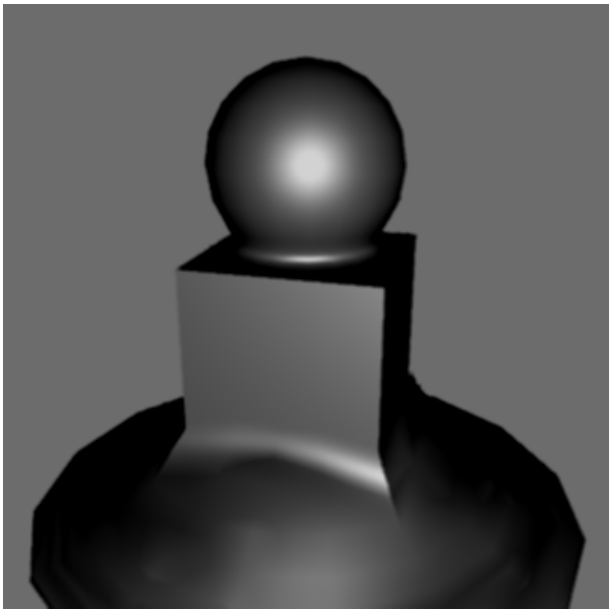


Figure 18: Blended Pointblobs: 1

polyhedron cannot satisfy the shape restriction for pointblobs. This is also evident in Figure 16 where a single point would not be sufficient to allow for functional interaction at both prongs of the fork. Further, with many different regions of interaction in a primitive, a measure of local control over the interaction in different regions of the primitive is useful. As defined, a pointblob lacks this property. The above two issues are addressed by point-setblobs. Point-setblobs extend the pointblob definition. They are quite different, however, from general polyblobs.

3.2.2 Point-setblobs

Point-setblobs may be motivated by the example of articulated figure modeling and animation, where each limb is an implicit primitive (pointblob) and the figure a blended result. It may be impossible to satisfy the star-shape restriction on V by the placement of a single skeletal point within arbitrarily shaped limbs. We would also like to localize the blending centered around joints (for example the upper arm may blend differently at the shoulder and elbow). This could be handled by a polyblob, where S could be the set of joint centers. The isolated shape of the polyblob, however, is unpredicatable for an arbitrary S, which is unacceptable as we wish to preserve the basic shape of the outer volume V.

Point-setblobs rely on the assumption that for many applications such as articulated figure animation, the threshold value of an implicit object remains constant. We define a point-setblob as $< W, S, \mathcal{F}, T >$. W is the shape of the isolated primitive desired at a surface threshold T. $S = S_1..S_n$ is a discrete set of points within W. Typically S would represent the various blend centers. $\mathcal{F} = f_1..f_n$ is a set of blending functions

corresponding to each point in S, each of which possess the properties of Figure 22. Construct volumes $V_1..V_n$, where V_i is $Trans(S_i)Scale(1/f_i^{-1}(T))Trans(-S_i)W$. Put simply, V_i is the appropriately scaled up volume, which treated as an isolated point-blob $< V_i, S_i, f_i >$, would give the shape W for a surface threshold value of T (see V_1, V_2 in Figure 19a).

The function value at a point P for point-setblob $< W, S, \mathcal{F}, T >$ is computed as follows:

1. Let $S_i = min_{j \in 1..n}(|S_j - P|)$.

2. The function value at P is the function value returned by pointblob $< V_i, S_i, f_i >$ at point P.

Hence the isolated shape of the point-setblob for the given threshold T, is W. Further, blending in a region near a blend center is locally controlled by its corresponding function due to spatial proximity.

This provides a satisfactory solution as long as the point-setblob has no functional interaction around a Voronoi boundary of the set S. For points on the Voronoi boundary, the functions are only guaranteed to return the same value for points on W (where all functions evaluate to T). Thus if there is no interaction with other primitives on a boundary, the surface at T around the boundary will be continuous and precisely W. At function values other than T, however, the values on either side of the Voronoi boundary can be very different. The present formulation is thus likely to cause discontinuities on the surface if there is interaction with other primitives in the region of a Voronoi boundary.

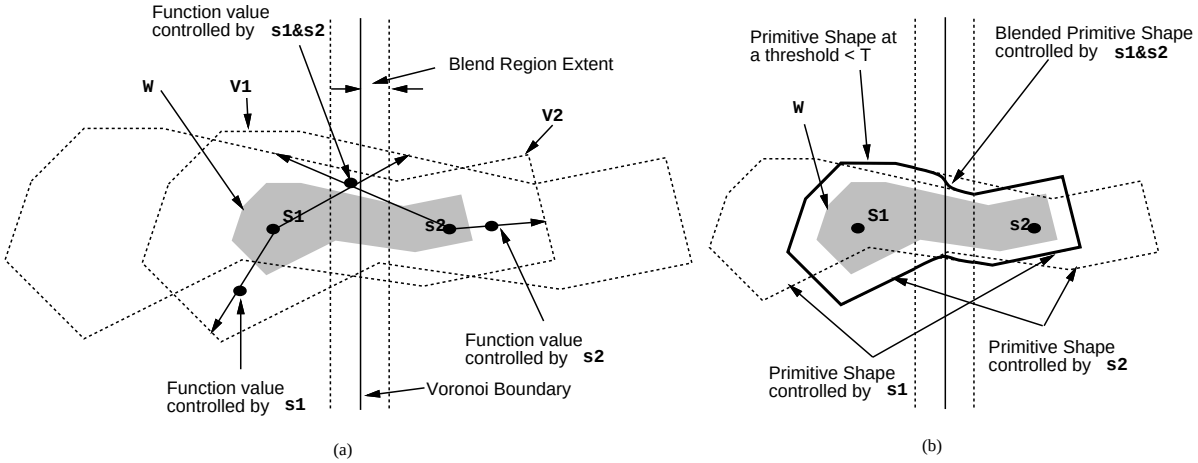


Figure 19: 2 Dimensional Point-setblob

A robust solution that would always maintain continuity across the Voronoi boundaries and preserve the previously achieved properties is as follows:

1. Find the closest point S_i in S for some query point P.

2. Let $\forall k \in 1..n$ $(d_k = |S_k - P| - |S_i - P|)$

3. Define a smoothing constant d, the size of which controls the size of a region around Voronoi boundaries, where function values are smoothed out to avoid singularities across the voronoi boundary.

4. Let $\forall k \in 1..n (w_k = f(min(1, d_k/d)))$ where f is a blending function with properties as in Figure 22.

5. Let $value_k$ be the function value at point P due to simple polyblob $< V_k, S_k, f_k >$, for $k \in 1..n$.

6. The function of the point-setblob at P is then obtained as a weighted average $\sum_{k=1}^{n} w_k * value_k / \sum_{k=1}^{n} w_k$.

The function values calculated due to more than one blend center near Voronoi boundaries are blended together with first order continuity; the extent of the blend is a user controlled parameter. In Figure 19a, the function value at the point shown within the blend region should be controlled by S_1 according to the closest blend center formulation. The function value at this point (and all points within the blend region around the Voronoi boundary), however, is a blend of the values with respect to S_1 and S_2. This maintains first order continuity of the primitive shape.

Note that none of the properties of local functional control and an isolated primitive shape of W at threshold T are compromised as a result of this blending (see Figure 19b). In Figure 20, a point-setblob torus that blends seamlessly and differently at 4 centers with identical spherical pointblobs is shown.

Note that the notion of multiple skeletal centers within a volume is not restricted to polyhedral structures but may be applied to an outer volume of any representation. An analytic sphere for instance, could have multiple skeletal centers within it defining the implicit function instead of the center of the sphere.

Point-setblobs thus form a useful object modeling and animation primitive that can be easily incorporated into any polygon or implicit function based modeling, animation or rendering system. Local functional control over different parts of the primitive make it well-suited to the modeling of articulated figures.

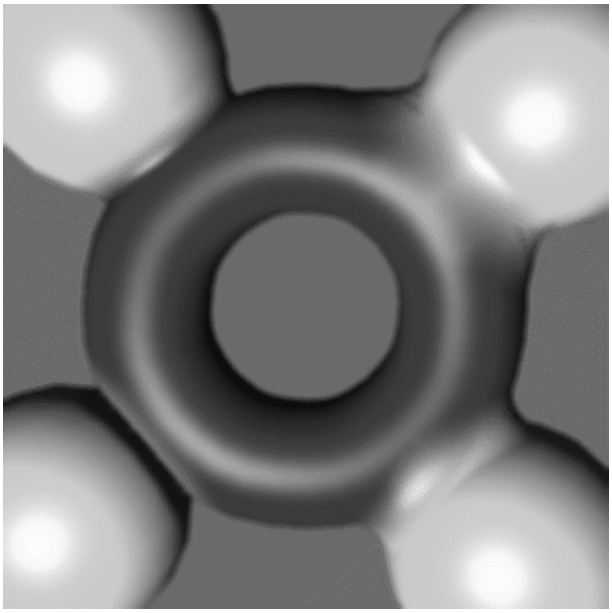


Figure 20: Point-Setblob

3.2.3 Local Control of Implicit Functions

The notions of local control and directional blending [70] for an implicit surface primitive deal with varying the blend function used over different regions of the primitive or in varying the way that the primitive's function values combine with other primitives. The problem lies in intuitively specifying that a particular region of a primitive blends in a different way from another. The local control afforded by different blend centers of point-setblobs provides variations in blending for different blend centers. Interaction of the kind seen in Figure 40, where the upper and lower arm primitives blend smoothly on one side of the elbow and collide on the other is both hard to specify and control. Using two closely placed blend centers for each limb primitive at the elbow (one for either side, with functions for blending and collision respectively) is

one solution. Alternatively, the blending can be intuitively controlled with polyhedral primitives using the following approach.

The polyhedral vertices are assigned blend functions, f_V, in the manner that current geometric modelers assign vertex normal/color values. Given the shape W, threshold T and a blend center S, a cutoff R_V for each vertex V is calculated to be $\frac{|V-S|}{f_V^{-1}(T)}$ (see Figure 21).

Now for any point P:

- Let $POLY$ be the polygon of W that is the defining polygon for P.
- Let $POLY'$ be the defining polygon $POLY$, scaled with respect to S so that point P lies on it.
- Calculate function values due to every vertex V of $POLY$, as $f_V(|V'-S|/R_V)$, where V' is the corresponding vertex on $POLY'$[5].
- $F(P)$ is then obtained by interpolating the function values at the vertices of $POLY$ bilinearly or in some another fashion based on the position of P in $POLY'$.

In this manner every vertex of the shape W has a measure of functional control around it, allowing different forms of interaction to manifest themself locally in different parts of the primitive. It is easy to see how this definition can be extended to account for the presence of multiple blend centers in point-setblobs. Note that

[5]This is equivalent to the computationally efficient form $f_V(f_V^{-1}(T) * ((P-S).\hat{N})/d)$, where d is the distance to the plane of $POLY$ from S and N the normal to the plane of $POLY$.

the above formulation preserves the shape W at threshold T in the absence of any interaction. Further, in the special case that the functions at all vertices of W are the same, this formulation reduces to the original definition of a point-setblob.

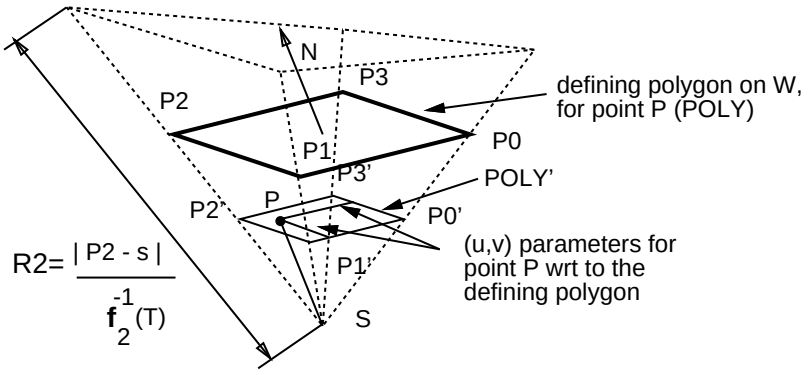


Figure 21: Functional calculation for Localized Blending

In Figure 40 functional attributes are assigned to the polyhedral vertices of the point-setblob upper and lower arm primitives. The vertices around the crease region have functions that interact to generate the collision contact surface. The function values of vertices on the other side of the elbow simply blend together. Functional calculation at any point is a smooth functional interpolation of the attributes at the vertices of the defining polygon.

Polyhedral primitives (*polyblob*, *pointblob* and *point-setblob*) can be easily incorporated into any modeling, animation or rendering system that treats objects and implicit primitive shapes in an object oriented fashion. Further all the concepts developed may be applied to B-rep primitives modeled using higher order surface patches. Optimizations specific to polyhedral primitives need not, however, generalize.

3.3 Display of Polyhedral Implicit Primitives

Implicit surface rendering may be done by tracing rays through space and determining the first ray surface intersection by solving the implicit equation along the ray using numerical or analytic techniques [8][18][23][26][70]. Alternatively, a surface reconstruction algorithm may be applied to construct a boundary representation of the surface [9][34][43][54]. The polygonized surface may be subsequently rendered.

Consider the definition of an implicit object comprised of implicit primitives. The object has a list of component primitives, a combination graph for the primitives and a surface threshold value T. Each primitive has attributes of :

Shape definition : such as $< V, S, f >$ for a polyblob, $< W, S, \mathcal{F}, T >$ for a pointsetblob, or center, radius and blend function for an analytic sphere. The shape definition specifies all the parameters required for calculating the functional value of the primitive at a point.

Attribute weights : such as a shape weight *sweight*, that provide additional control over the influence of the primitive to each object attribute independently.

The above attribute descriptions are those specific to the implicitly defined object. Display, physical, and other general attributes may also be defined, not only for the entire object but independently for each primitive. The primitive list is a generic list of various different implicit surface primitives, including existing analytically defined shapes. The combination graph model used is a directed acyclic graph with the primitive function values as source nodes, internal nodes as algebraic combinations, and a single sink node representing the functional value of the implicit object.

3.3.1 Blend Functions

The desired properties of a blend function f are: f is a function of the normalized distance (*distance-ratio*) between 0 and 1, $f(0) = 1$, $f(1) = 0$, $f'(0) = 0$, $f'(1) = 0$, f is monotonically decreasing. As developed by Wyvill [68], a simple Hermite cubic solution is well suited to the above. However, to avoid taking square roots in *distance-ratio* calculations, a sixth degree polynomial with the added constraint $f(\frac{1}{2}) = \frac{1}{2}$ is used.

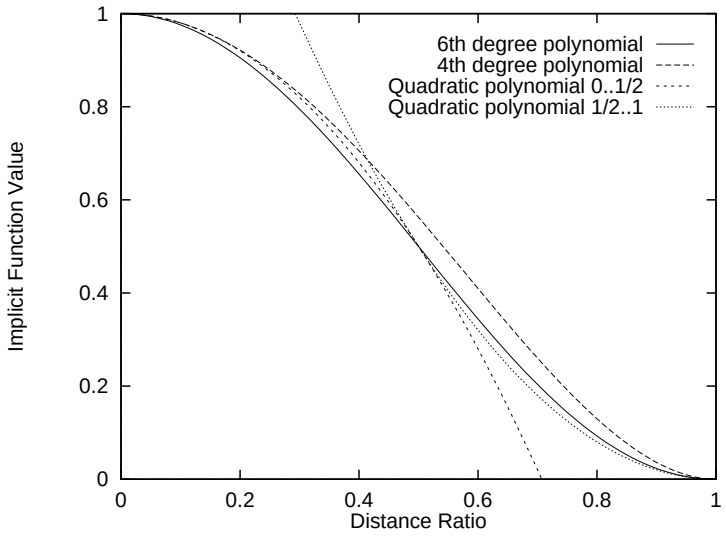


Figure 22: Polynomial Blend Functions

A close approximation to this blend function is provided by the quartic function $f(x) = (x^2 - 1)^2$ (see Figure 22). It has all the enumerated properties except $f(\frac{1}{2}) = \frac{9}{16}$; the visual difference between the blending obtained by the two functions is negligible. For primitives where explicit distance to the skeleton can be computed

rather than the square of the distance using geometric arguments, a combination of two quadratics $f(x) = 1 - 2 * x^2$ for $x \in [0, 0.5]$ and $f(x) = 2 * (1 - x)^2$ for $x \in [0.5, 1]$ provides similar blending properties.

The quartic and quadratic polynomials facilitate precise ray-surface intersection calculations for primitives whose geometry is defined as an offset surface. For example, the formulation of the vector to the closest point on the skeletal polygon in a rounded polygon falls under one of three cases for a point $P = A + B * t$ parametrically defined along a ray.

- The vector to one of the vertices Q is $(P - Q)$.
- The perpendicular vector to one of the normalized edge vectors E with an adjacent vertex Q is $P - (P \odot E)E - (Q - (Q \odot E)E)$[6].
- The perpendicular vector to polygon plane with unit normal N and a vertex Q is $((P - Q) \odot N)N$.

The vector is thus of the form C+D*t, fragmented over intervals defined by the case that determines the vector. Sphylinders and spheres are degenerate cases of the above. Thus the parameter t can be plugged into a quartic or quadratic implicit equation and solved for explicitly over a non-overlapping interval fragment defined by all contributing primitives. This approach has the advantage of being robust and theoretically precise. There are two major drawbacks of this approach. Firstly, it is inapplicable to more complex primitive shapes such as the cone-sphere where

[6] $\odot$ is used to denote the dot product of two vectors.

the implicit function cannot be represented precisely as a function of the interval parameter t. For cone-spheres we obtain a ratio of two polynomials in t which can only be approximated by quartic or quadratic polynomials in t. Additionally, the ray is fragmented into intervals with different polynomials due to three reasons:

1. Due to the blend function domain if piecewise blend functions like the quadratic in Figure 22 is used.

2. Due to changes in the formula for the *distance-ratio* within a primitive.

3. Due to the overlap and combination of the functions of multiple primitives.

These fragmented intervals along the ray are then solved in order for a surface intersection. For complex scenes the level of interval fragmentation may make the analytical solution inefficient.

3.3.2 Ray Tracing

Most ray tracers are designed in an object oriented fashion to allow for easy incorporation of new object representations. Typically, initialization, ray-object intersection, normal, and color computation procedures for any new primitive need to be specified. These are elaborated on in detail with respect to polyhedral implicit surface primitives.

Initialize(object)

Initialization usually entails preprocessing and transformation of the *object* into the space where the rendering process is carried out (typically world space). In the context

of polyblobs, the polyhedra, V and S, are transformed, typically into world space and preprocessed. Preprocessing involves the computation and building of various structures, such as bounding volumes, spatial subdivision schemes, and priority lists. These are employed by techniques that exploit different types of coherence to improve rendering efficiency. For point-setblobs, it further involves the construction of volumes V_i for each skeletal point S_i. Further, preprocessing such as the construction of voronoi regions to speed up the closest point computation in the context of point-setblobs and polyblobs may be done.

Intersect(RAY,object)

Given a point in space, P, the value of the implicit function can be determined by calculating the function values for each component primitive and then combining them as specified by the combination graph.

The *RAY* intersection with the *object* is the first point along the ray where the combined contributions of the primitives achieve the threshold value of the surface. For some applications such as CSG construction of objects, all intersections of the ray with the object may be required. A simple intersection procedure can be given based on function evaluation at a point. Points along the ray are evaluated by stepping along it at a fixed or adaptive resolution and the first point (or all points) where the function value achieves the threshold is returned.

Values such as the *defining-polygon* of V for point P, the point P' on the *defining-polygon*, corresponding to P and the functional value, $value = F(P)$ are likely to be calculated as part of the intersection procedure. They should be saved in an auxillary

structure as these values are also needed for the computation of surface normals and other object attributes.

The intersection computation can be optimized in a number of ways.

Primitive function computation at P

For polyblobs, efficient and accurate computation of the closest point on S to P is crucial. Given a point in space P and a polygon, the closest point on the polygon to P may be computed as follows:

1. Project P perpendicularly onto the plane of the polygon. If the projected point lies inside the polygon (in the plane), return the projected point.

2. Project P perpendicularly onto the line of each edge of the polygon. The minimum distance from P to an edge is either the projected point, if it lies within the edge, else it is the closer of the two end-points of the edge. Return the minimum distance from P to an edge over all the edges of the polygon.

A simple algorithm to determine the closest point on a polyhedron for a given P is to minimize on the point with the shortest distance to P for all polygons of S. For closed polyhedrons, back faces with respect to P may be culled from the list of faces to be tested. Further, S should be preprocessed into an edge based structure such as winged edge. This allows for easy access so that computations for edges and vertices of the polyhedron need be done only once. Spatial subdivision techniques such as octrees can also be employed to limit the number of faces that need to be tested for a given P, using the following heuristic for closed polyhedrons:

The voronoi space of a face is the polygon extruded outwards along its normal. The

voronoi space of an edge is the wedge shape emanating from it and bounded by the 2 normals of its adjacent faces. The voronoi space of a vertex is the pyramidal shape emanating from it and bounded by the planes normal to its adjacent edges. This is strictly true only for convex objects. A spatial subdivision based on this heuristic would typically locate the P to lie in at most a few of the above regions, all of whose corresponding elements would then be exhaustively processed for the closest point.

For the computation of the *distance-ratio* function g, the restriction that the ray through P uniquely intersects V may be used. The intersection routine can be sped up by intersecting polygons until it comes across any intersection, not the closest. This should be coupled with a priority list for the polygons, with the most recently intersected having the highest priority. The intersection routine's overall performance is now sped up due to the spatial coherence of rays through proximal points where the function is sampled.

Isolating regions where primitives interact would also help speed the functional calculation of point-setblobs immensely. *Intersection boxes*, which are a set of disjoint boxes resulting from the intersection of bounding boxes of all interacting primitives are precomputed during initialization. Each skeletal point of a primitive defines one or more semi-infinite pyramidal regions with the corners of the intersection box (see Figure 23). Any candidate defining polygon of the primitive for any point within the intersection box must intersect one of these semi-infinite regions. Thus for each primitive a subset of polygons may be tagged as candidate defining polygons for the primitive for all points within the intersection box. Often this preprocessing step can

reduce the number of candidate defining polygons by many orders of magnitude. (see Figure 23, Table 1,3).

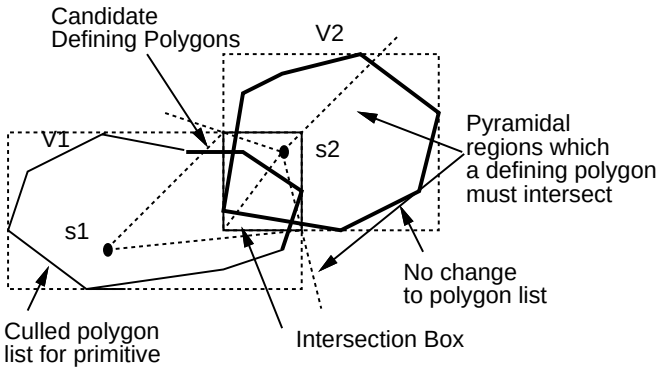


Figure 23: Candidate defining polygons for 2D Pointblobs

Computing interval bounds along *RAY*:

Intersecting the ray with volume V of each primitive generates a span or interval over which it is active (contributes to the object function value). The set of intervals for the primitives also provides us with bounds along the ray between which ray-object intersection points lie. A bounding volume hierarchy of the object and its component primitives can also be used to limit the number of active primitives at any point along the ray. Spatial subdivision is also useful in this regard.

For pointblobs and point-setblobs, W, representing the isolated shape of the primitive at threshold T, should be maintained. Now in the case of a single active primitive in some interval over a ray, the exact surface intersection is given by the ray intersection with the B-rep W.

Numerical Technique with Heuristics:

Blinn [8] uses a heuristically driven numerical technique to find the surface intersection for spherical primitives. The algorithm involves evaluating the function at a number of guess points, one for each primitive, where the threshold is likely to be achieved. The guess points are sorted by distance along the ray and the first interval of consecutive guess points that straddles the surface is selected and the function evaluated at points in the interval using a hybrid newton raphson, regula falsi approach to obtain the intersection point. The heuristic works well in practice for the symmetric and well behaved function generated by the span of a spherical primitive along a ray. Arbitrary polymeshes, however, can generate multiple intervals along the ray with highly varying functions, resulting in missed intersections. These problems are magnified if this approach is used to obtain all intersection points of the surface with the ray. The heuristic may still be applied with a few modifications. A guess point must be generated for each interval spanned by a polyblob along the ray. The guess point is determined in a manner similar to Blinn as follows:

1. Intersect the ray with W. The first intersection point corresponding to a span becomes its guess point.

2. If no intersection exists, we would like to find as guess points the points where the value contributed by the primitive attains a maxima. For spherical primitive spans it is precisely the mid-point of the span [8]. In this case to avoid complex maxima computation for what is only a heuristic, the first intersection with the isolated primitive shape at a threshold $T/2$ is used as the guess point; failing that, a fixed number of equally spaced guess points along the span are used.

Analytic Technique:

The analytical approach described in 3.3.1 can be applied to point-setblobs. For combination operators such as a summation, difference and union, the function value for the object along the ray can be obtained as a number of polynomials in t, the parametric distance of point P along the RAY. These equations can then be solved analytically in order along the ray until the first intersection is found.

The analytical technique above motivates the following observation for point-setblobs. The interval along a ray over which the point-setblob is active can be partitioned into a sequence of intervals by adjacent defining polygons (see Figure 24).

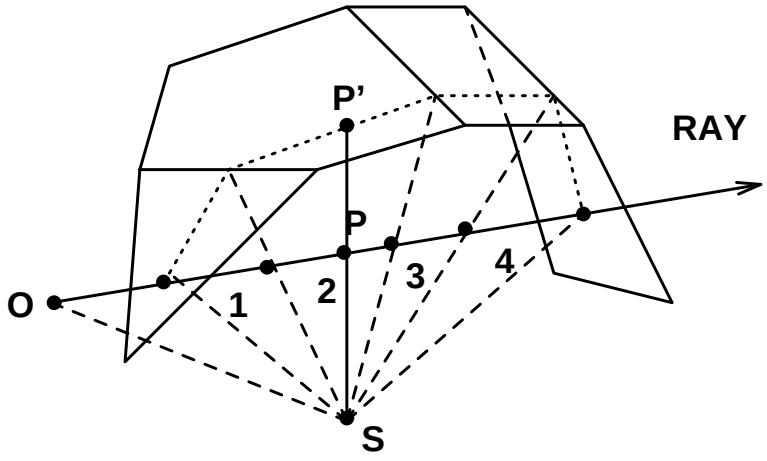


Figure 24: Defining Polygons along a Ray

Let a given polygon define the *distance-ratio* for points over an interval along the ray, the *distance-ratio* over the interval can be represented as a linear function of t, the parametric distance along the ray. Let the normalized ray vector be RAY, originating at O. The normal to the plane of the defining polygon is N and the perpendicular

distance from the skeletal point to the plane, r. Let a point $P = O + RAY * t$ along the ray. Then for all points pnt with this defining polygon, $distance - ratio = |P \odot N|/r$, or $distance - ratio = (O \odot N/r) + (RAY \odot N/r) * t$. This is shown in Figure 25.

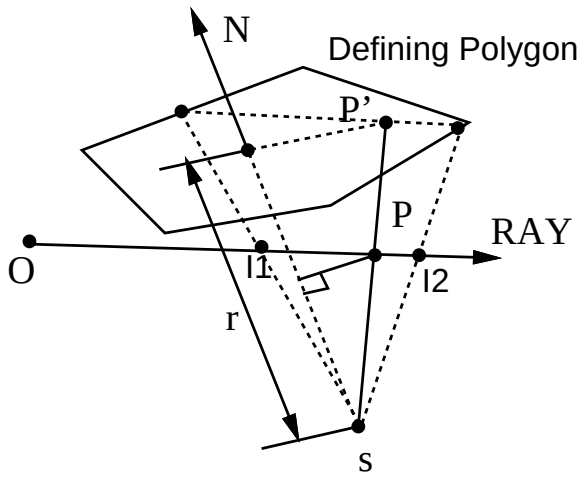


Figure 25: *distance-ratio* over a Defining Polygon

Computation of the sequence of defining polygons is a simple incremental process of determining the next defining polygon along the ray given the current one, by using an edge based data structure. The spans of a point-setblob can thus be partitioned into a sequence of intervals, the function over each being a quadratic in t. The intervals for all the component point-setblobs are then fragmented into nonoverlapping intervals and combined to obtain a number of quadratic functions. These are analytically solved in order for the first intersection point. The number of intervals generated may be upto twice the sum of defining polygons of all active interacting primitives along the ray. Additionally, due the piecewise nature of the quadratic blend function,

the number of intervals may be doubled.

This technique is robust and very efficient when the number of defining polygons for a ray is small. The complexity grows linearly with the number of defining polygons. Its locality of application and robust nature makes for a useful coupling with the optimizations and heuristics described earlier. The ray can be pruned using numerical and heuristic techniques to intervals in the neighborhood of a solution. The analytical procedure can then be applied. An intersection procedure with user control over the robustness-speed tradeoff is thus proposed.

Normal(P,object)

The normal vector to an implicit surface $F(x,y,z)$ at $P =< x,y,z >$ is given by $\overline{< \frac{\delta F}{\delta x}, \frac{\delta F}{\delta y}, \frac{\delta F}{\delta z} >}$ [7]. Thus, for functions f_i summed over n primitives, the normal vector is $\overline{< \sum_{i=1}^{n} \frac{\delta f_i}{\delta x}, \sum_{i=1}^{Npb} \frac{\delta f_i}{\delta y}, \sum_{i=1}^{Npb} \frac{\delta f_i}{\delta z} >}$, which is $\overline{\sum_{i=1}^{Npb} < \frac{\delta f_i}{\delta x}, \frac{\delta f_i}{\delta y}, \frac{\delta f_i}{\delta z} >}$. Consider an implicit primitive defined with a constant offset distance r from a skeleton [11] (spheres, sphylinders and rounded polygons) with an associated function of distance ratio f. The normal vector contribution for the primitive is $\frac{sweight * f'(d) * \overline{N}}{r}$, where d is the *distance-ratio* at P, and N is the normal vector to the primitive shape at P (see Appendix A).

For pointblobs, *distance-ratio* d is $((P - S) \odot \overline{N})/r$ where S is the skeletal point, $\overline{N}$ is the normal to the defining polgon and r is the perpendicular distance from S to the plane of the polygon. Thus $\nabla f(d) = f'(d) * \nabla d$. $\nabla d = \overline{N}/r$ making the normal vector contribution for the primitive $\frac{sweight * f'(d) * \overline{N}}{r}$ identical to the form for offset

[7] $\overline{N} = N/|N|$ denotes a normalized vector.

surfaces above. The above form represents the surface normal in terms of normals to individual primitives appropriately weighted and algebraically combined and thus allows for a simple integration of normal vectors computed using any primitive.

For pointblobs, the normal vector $\overline{N}$ for the polygon at P is the same as the normal at P' as normal vectors are invariant under uniform scaling. We can thus apply polyhedral techniques such as interpolated smooth shading or bump texture mapping to the primitives simply by using the normal value calculated for the polyhedron at P'. For smooth interpolation of normal vector contribution, r should also be smoothly interpolated. This may be done for instance by calculating r values for vertices of the pointblob averaged from the r values of their adjacent polygons. The r value at P could then be given by a bilinear interpolation of r values of the vertices of the defining polygon at P'.

For point-setblobs, the normal is precisely the appropriately weighted average of pointblob normals of the contributing skeletal points. For general polyblobs, the normal at P can be obtained as a weighted blend of the normal to V at P' and the normal to S at the euclidean closest point, based on the threshold value.

Color(P,object)

Color computation uses a weighted average of individual pointblob colors. These may be obtained from arbitrary shaders for each primitive and applied to the primitive at P'. The colors are then weighted by $value * cweight$ and combined to return a color value for the object, where *cweight* is the color weight of the primitive. Similarly any other desired object attribute can be obtained by computing the attribute for

a polyhedral B-rep at P' as the attribute for the primitive and then combining the primitive attributes appropriately.

Point-setblobs attributes are obtained from the attributes of the contributing skeletal points. The general polyblob case is handled similar to normal computation by weight averaging according to the threshold value, the attribute values obtained from corresponding points on V and S.

Other Issues

In the above procedures the primitive combination considered is a simple blend or summation. For other combining operators the relative contributions of the polyblobs would change accordingly. For a union or **max** operator, contributions of primitives that define the surface of the object at a point would be equally blended with the remaining contributions being nil.

From the description of intersection, normal and color computation, it is evident that display attributes such as texturing techniques can be applied at two levels. They may be applied independently to each primitive to influence the normal, intersection point or color of the primitive and then blended appropriately to a global value. Attributes obtained after blending may then be subjected to further techniques, such as illumination effects or texture mapping applied to the resultant surface.

Any technique applicable to polymeshes can be applied to a polyhedral implicit primitive. For pointblobs and point-setblobs all attributes may be determined from the outer bounding volumes. It is, however, advisable to scale P', the point on V corresponding to P, so that it lies on W and then apply the technique. The scaled

polymesh better represents the actual size of the object in the locality of P and would give more consistent results for any scale variant technique such as solid texturing and for antialiasing procedures.

Often a faster direct display approach is desired at the cost of visual realism. Scanline or Z-buffer rendering is such a method. The scanline rendering of implicit surfaces is, therefore, discussed.

3.3.3 Scanline Rendering

The approach taken by us is an extension of Blinn's work but with a marked geometric flavor [8]. While the organization of the processing is similar to that used in scanline algorithms, the system is still technically a ray caster. The ray casting, however, takes place in *eye-space*. In eye-space, the objects are defined relative to a left handed coordinate system centered at the observer with the positive Z-axis aligned with the line of sight. Object geometries are transformed to eye-space with minimal computation. Processing in eye-space simplifies the scanline calculations. Processing in a perspective space (image space or screen space) involves dealing with geometries and functions distorted by the perspective transform. Computation of normals to the implicit surface is also problematic in a perspective space.

Functions that calculate scanline ranges, spans across a scanline, and depth ranges at a pixel for every object primitive are required. This information is used to maintain an *active object list* which can be incrementally updated similar to standard scanline algorithms and produce a *span-segment* from the object for each scanline. While the scanline-planes are not parallel to the Z-axis, a z value can be computed based on

the viewing parameters such that, at that z value, each integer x value in the span segment corresponds to a pixel on the screen.

For each such pixel covered by the span-segment a *z-span* (depth range of the primitive at the pixel) is generated and added, in sorted order by closest z value, to a bucket for that x position in the scanline. For a given scanline, once this is done for all objects we have, for each pixel, all of the ray segments that intersect any of the primitives. These must then be searched for the nearest z value solution to the implicit surface equation.

In the case of incrementally building a desired surface or during animation, the intermediate z-span descriptions can be retained or updated to accelerate the rendering process for subsequent versions or frames. The z-spans contain all of the information necessary for evaluating the object blend functions.

Adding other objects to the display processing is easily handled. The implicit surfaces can be rendered into a Z-buffer and used to initialize its contents, or the other objects can be rendered in a scanline fashion along with the implicit surface primitives (the ubiquitous space-time tradeoff).

Other than functions for evaluating the implicit function and other surface attributes at a point for a primitive, the following functions are required:

- **calcYExtent**(*primitive*): returns the range of scanlines covered by the primitive.

- **calcXExtent**(*primitive*, *scanline*): returns the primitive span for a scanline.

- **calcZExtent**(*primitive*, *scanline*, *pixel*): returns the primitive depth for a pixel on the scanline.

The values returned by these functions are the finite extents over which the primitive contributes a function value. This additional information and utilization of coherence for effective scanline rendering can be easily extended from existing techniques for dealing with polymeshes. For polyhedral primitives, extent in Y, extent in X for a given scanline, and extent in Z for the pixel are simply that of its bounding polymesh and can be computed incrementally. The same may be done for S and W. Further, the list of polygons defining the extent in Z can be easily maintained. Now the analytic technique that specifies the function value as a quadratic in terms of the defining polygon over an interval can be computed with great efficiency. Interval coherence can be used when going from one pixel to another for a powerful heuristic for point-setblobs.

The necessary calculations for other primitives (spheres, sphylinders, rounded polygons and cone-spheres) are exhaustively enumerated in Appendix A.

The overall structure of the renderer is:

1. Transform all primitives to eye space.
2. Y-bucket sort the primitives.
3. Maintain an active primitive list and for every scanline compute extent in X for the active primitives.
4. For every pixel compute depth Z spans for all primitives with an active X-extent.

5. Fragment the Z spans into intervals as desired. Solve for the intersection point using a combination of the numerical techniques, and an incremental analytic technique.

6. Compute the surface normal and other attributes at the intersection point.

3.3.4 Surface Reconstruction

Polygonization of general implicit surfaces is described by Bloomenthal [9][43] using convergence and surface tracking techniques using an octree structure. Schmidt [54] extends the convergence technique [9] to allow for a more general class of surfaces with singularities.

A motivating factor of polyhedral primitives was to reduce the number of modeling primitives used. In such cases there are likely to be regions where a single primitive defines the surface. Here, for pointblobs and point-setblobs, the appropriately scaled bounding polymesh defines a polygonal representation of the surface. This provides not only a partial polygonization of the resultant object but also provides information of the behavior of the surface at the point where the primitive starts interacting with other primitives.

The polygonization algorithm adopted, therefore, first finds regions where primitives do not interact. These are the regions outside the *intersection boxes* (see Section 3.3.2) for the primitives. The existing primitive polyhedral shape is used to represent the surface there by clipping the primitive to the *intersection boxes* and retaining the outside. The inside of the *intersection boxes* are then polygonized using an algorithm

such as [9]. Finally the polyhedral surface generated inside is seamlessly connected to the clipped and retained polyhedral fragments outside the intersection boxes.

The polygonization of pointblobs and point-setblobs is as follows:

1. Compute the *intersection boxes* of all interacting primitives. Compute lists of candidate defining polygons (see Figure 23).

2. Clip the polyhedral models of each isolated primitive to the the *intersection boxes*. Retain the outside as part of the final polyhedral structure.

3. The clipped edges of the primitives lying on the face of an *intersection box* determine the quadtree structure on that face of the *intersection box*. We set a threshold on the number of clipped edges that lie completely within a single quadtree square on the face. The squares are subdivided until the threshold criterion is met.

4. The clipped edges also provide a number of seed cubes. Surface tracking algorithms require a seed cube, through which the object surface passes. The algorithm tracks a connected object surface by processing neighbor cubes of cubes through which the surface passes until all cubes through which the surface passes have been processed. The clipped edges are the intersection of the implicit object surface and a face of the *intersection box*. The octree cubes corresponding to the quadtree squares through which the clipped edges pass are seed cubes.

5. The seed cubes are used by an adaptive surface tracker [9] for polygonization confined within each *intersection box*. Primitives other than pointblobs and point-setblobs can be polygonized using any desired approach as long as connectivity with any adjacent *intersection box* is maintained [54].

An approach to exploiting the parallelism potential of polygonization is described in [71]. This approach further caters to parallelism because of the polygonization of various disjoint *intersection boxes*.

We need to ensure a seamless connection without cracks of the retained polyhedral structure and the polygonized surface within an *intersection box*. For each edge generated in a face of the *intersection box* due to clipping, the following action as shown in Figure 27 is taken, the results of which are shown in Figure 28a,28b,29.

- The edge is fragmented by inserting vertices along the edge at points where it enters and exits the edges of the clipping plane quadtree. The clipping plane quadtree is formed by the edges of the octree cubes in the clipping plane to which the edge to be fragmented belongs. The vertex insertion is easily accomplished by using a DDA type line algorithm.

- The edges of retained polygons adjacent to the clipped edge are beveled to triangles.

- The retained polygon of a clipped edge that lies entirely within a single cube must be connected to the surface intersections of the cube. A sequence of

clipped edges all lying within the same cube cause it to be partitioned until at most a single clipped edge lies within a cube.

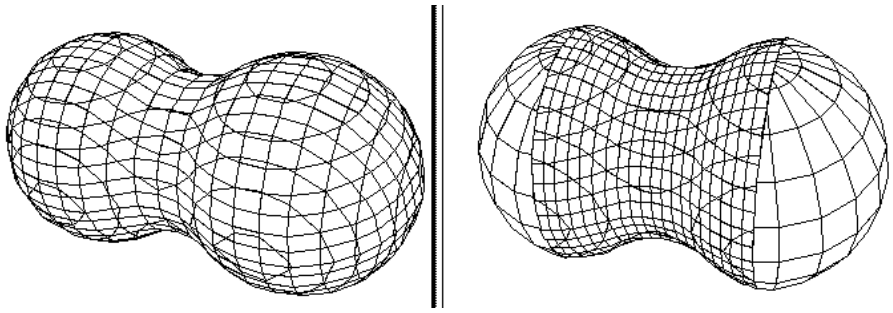


Figure 26: Polygonized Spherical Pointblobs (Without and With Clipping)

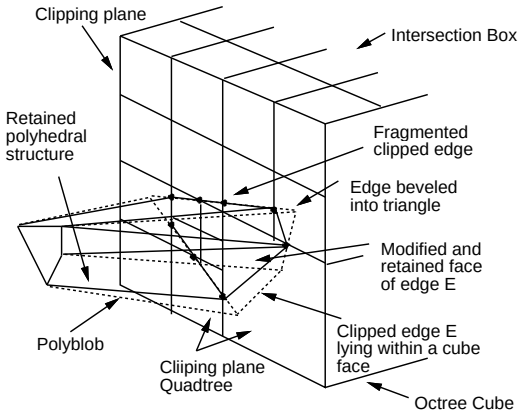


Figure 27: Clipping Isolated Point-setblobs

3.4 Implementation and Results

The implementation ray traces the object using a hierarchy of bounding volumes for each component primitive. The implicit surface equation is solved using a hybrid

approach that using heuristics, obtains the locality of a solution numerically, followed by an analytic calculation of the intersection. The speed/accuracy tradeoff is user controllable. A priority list of polygons of V for each primitive based on the most recently intersected is maintained to exploit spatial coherence of defining polygons. Further, intersection box preprocessing is done to obtain a possibly reduced set of candidate defining polygons for point-setblobs. Voronoi space precomputation for point-setblobs and polyblobs was not implemented.

Table 1: Ray trace timings (min:secs) and statistics

Figure	*# primitives*	*total # polygons*	*# candidate polygons*	*combination complexity*	*render time as polyhedra*	*render time*
18	3	447	187	summation	5:14	5:59
30	17	2236	1218	hierarchy	15:24	17:47
31	3	2484	503	summation	10:40	11:31
32	2	12100	650	summation	13:27	15:33
20	5	2096	678	summation	11:40	12:58
33	7	2008	733	hierarchy	21:13	24:44
42	3	1062	288	hierarchy	4:29	5:03
69	3	751	751	summation	10:01	12:22

The 400X400 images were ray traced with a recursive depth of 3 with shadows and adaptive antialiasing. Timings for the images given were taken on a SUN Sparc2.

Figures 18,30, 31 show objects comprised of pointblobs blended together. Figure 32 is a 450X300 image showing 2 very complex pointblobs (foot bone and muscle model), blended at the heel.

Figure 20 shows a point-setblob torus with four blend centers that interact differently with four identical spherical pointblobs. Figure 33 shows a point-setblob hand.

Local control is illustrated by a softer blend at the forefinger than at the thumb. This was accomplished by specifying different blend functions to skeletal points in the thumb and forefinger. Interacting implicit primitives to model collision deformations [20] is also indicated by two pointblob eyeballs. Further a conical pointblob causes a spike in the floor which is a large polygon modeled as a pointblob.

Figure 42 shows the application of point-setblobs to character animation. The arm is modeled as two point-setblob limbs hierarchically blended together with an analytic spherical primitive. The point-setblob limbs are laser scanned polyhedra. The arm is shown outstretched as well as bent, where collision deformation interaction between the point-setblob limbs causes the formation of a precise crease, while the primitives remain smoothly blended together due to the analytic sphere.

Figure 69 illustrates a shape transformation using shape weight interpolation between appropriately placed polyhedral primitives. The figure shows a CSG modeled pointblob transform to a pointblob sphere which transforms to a polyblob pillar. The transformation of various color and texture attributes is also illustrated. Details on this useful application of polyhedral primitives can be found in Appendix B.

The results of the scanline technique are shown in Figure 34,35,36,37.

The timings given in Table 2 are from a SUN IPX. In each case the object occupies about a 1x1x1 area in world space, centered at the origin with the observer at (5,5,5) and an angle of view of 10 degrees. The pixel resolution is 400x400. The composition of each image in terms of spheres, sphylinders and cone-spheres and rounded polygons is given. This is followed by the rendering time using scanline processing (referred to

as *scan*) and the rendering time using a ray caster. The ray caster is identical to the scanline renderer without the scanline related optimizations.

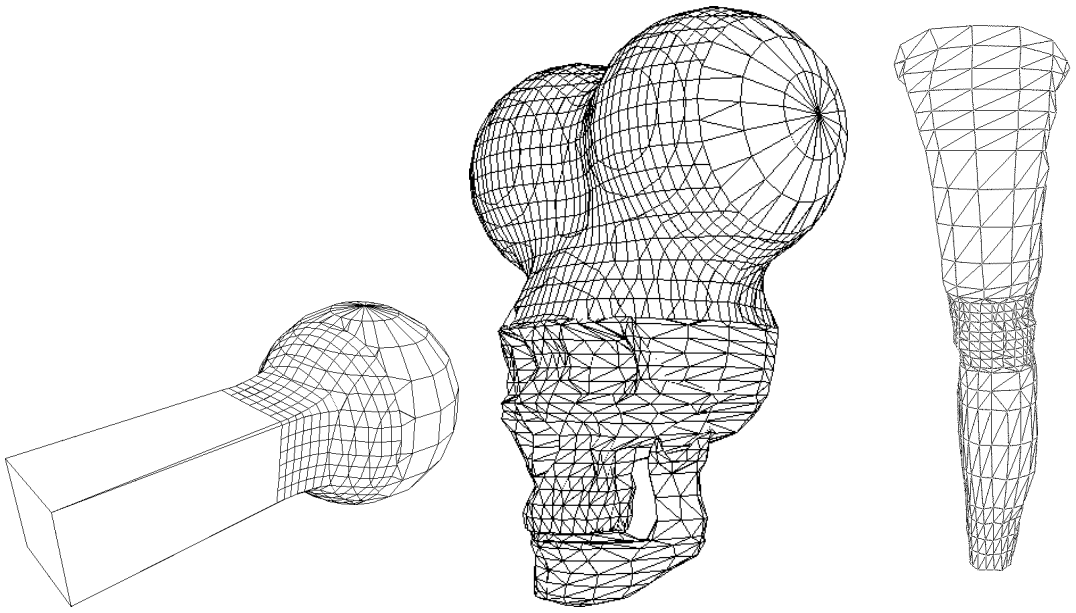
Table 2: Timings (in secs) for scanline rendering

Image	*Sphere*	*Sphyl*	*C-sphere*	*R-polygon*	*Scan*	*Ray cast*
35	16	20	0	0	58	387
34	106	0	0	0	75	428
34'	13	30	0	0	66	415
31	2	20	4	0	45	304
37	4	3	2	3	28	275

The implemented polygonization algorithms seamlessly integrate the retained polyhedral structure and the one generated by a variant of the surface tracking algorithm [9]. This algorithm described in Section 3.3.4 uses clipped edges on the faces of the intersection box to determine a fixed level of subdivision after which a number of seed cubes are spawned to the surface tracker that operates within each intersection box. Additionally the robust convergence based adaptive polygonization implementation [54] has been modified to polygonize polyhedral primitives.

Table 3: Polygonization timings (secs) and statistics

Figure	*# primitives*	*# total polygons*	*# candidate polygons*	*# polygons retained*	*# polygons generated*	*preprocessing, clipping time*	*polyg*
28a	2	206	98	189	536	0.1	
28b	3	2484	503	2159	1841	0.2	
28c	3	1062	288	774	243	0.2	
29	6	36	30	-	6827	-	
70	1	70	6	-	4290	-	



(a) (b) (c)

Figure 28: Polygonized pointblobs

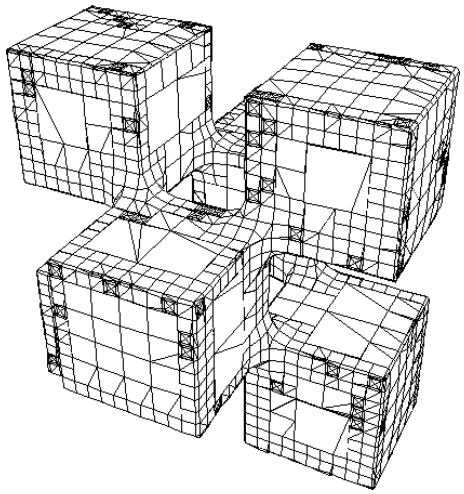


Figure 29: Polygonized cubes

A seamless integration of the clipped and polygonized structure can be clearly seen in Figure 28a, where a cuboid pointblob blends with a spherical pointblob. The beveling of edges to prevent cracks has been accentuated to be clearly visible.

Figure 28b shows the polygonization of the Mickey Mouse skull in Figure 31. Figure 29 shows pointblob cubes polygonized using the convergence based technique and Figure 28c shows the polygonization of the arm from Figure 42. Figure 70 shows a single polyblob with a polyhedral cube transforming to a polyhedral torus due to threshold value interpolation. The objects at various thresholds were polygonized using the convergence technique and subsequently rendered. The timings in Table 3 are shown for the polyblob at a threshold of 0.5.

3.5 Discussion of results

As can be seen from the resulting images, polyhedral primitives in general behave well and in a manner similar to their analytic counterparts. This gives the user an intuitive notion of the results whilst modeling. Blending is best made use of in polymesh regions that closely approximate a first degree continuous surface. Here the behavior of the implicit surface is very predicatable. It is useful for blending sharp corners [39][53].

Unacceptable bulges can arise from blending just as is the case with other primitives. These can be avoided by applying Bloomenthal's technique of convolution surfaces [11]. This would involve the integration of the g function defined in Section 3.2 over every point on S for the primitive.

Techniques that deal with variations in the blend functions such as directional blending are still applicable. Point-setblobs further provide local control allowing differential interaction over different regions of the primitive in a fashion that is intuitive and easy to specify. Good local control is afforded by polyhedral primitives. Individual blend functions may be specified as vertex attributes for the polyhedra. The functional value at a point is then obtained by a smooth interpolation of vertex functional attributes of its defining polygon.

The rendering efficiency obtained from the implementation is reasonable. This is illustrated from a comparison with the time taken to render the same image in which each primitive is considered an individual B-rep object (see Table 1). The timings were taken on the same ray tracer with all other parameters unchanged. As can be seen from Table 1, for 400X400 images of the quality shown, every 100 candidate polygons in the scene added roughly 15 seconds to the rendering time as B-rep objects. Other factors like the size of regions where primitives interact with respect to the viewport are also important.

As can be seen from Table 2, scanline processing of implicit surfaces offers a distinct advantage over standard ray tracing. In addition, intermediate structures can be retained if a single image was being worked on repeatedly.

The power of the introduced primitives (sphylinders, cone-spheres and rounded polygons) can also be seen [38]. These are especially useful for modeling articulated characters and will be extensively used for our human figure model in Chapter IV. In Table 2 *face'* generates a visually similar image to *face* (Figure 34), using sphylinders

to reduce the number of primitives as well as the rendering time.

Polygonization efficiency using the discussed algorithm is good (see Table 3). The timings shown indicate the effective use of polyhedral primitives for interactive incremental object modeling. Additionally, due to retaining parts of the original polyhedral structure, the number of facets added per primitive in regions of interaction is of the same order as the the number of original polygons in many cases, as can be seen from Table 3. This is important to keep the number of polygons generated during incremental modeling within a reasonable limit. The maximum time is spent in actual polygonization of intersection boxes which are likely to be small at any point of time if the object is built incrementally.

The star-shaped restrictions on the primitives can be relaxed, as long as they are satisfied in any region that it is interacting with another primitive. In non-interacting regions appropriately scaled polyhedra may be used to represent the surface. This is illustrated by the teapot lid pointblob in Figure 18 , the skull pointblob in Figure 28b,31 and the foot pointblob in Figure 32. This relaxation allows a very large class of polyhedra to be used even as pointblobs. Primitives such as the skull in Figure 28b, 31 and the foot in Figure 32 have a large genus and possibly disjoint pieces. They cannot be fitted by a superquadric primitive [55] and would probably require thousands of primitives to fit the complex topology, using Muraki's technique [41]. Further, the requirements of bounding polyhedral volumes may be relaxed to the existence of a unique defining polygon for any point where the primitive interacts with other primitives. Figure 33 shows a single polygon being used as a pointblob for

the floor. In Figure 33, the pointblob can be thought of as a square pyramid centered at the skeletal point with the floor polygon as the base. For all points of interaction with the conical pointblob the floor polygon is the unique defining polygon.

Exploiting the parallelization potential [71] of polyhedral primitives, and further reducing the search space for defining polygons are open to future work.

Summarizing, polyhedral implicit primitives have the following attributes:

- The primitives make modeling and animation of objects that comprise of parts with complex detail, smoothly blended together very simple. Digitized parts of a human body for example, could be blended together as polyhedral primitives (see Figure 42). This has the advantage of the realistic detail of polymeshes and the animation flexibility of implicit surfaces. Pointblobs are useful for animating blemishes on complex objects like a bump on a head (see Figure 31).

- Any polygon based technique is directly applicable to polyhedral primitives making them very powerful implicit modeling primitives.

- Polyhedral primitives are useful for incrementally building objects. An object constructed from any implicit primitives can be polygonized and reused as a single polyhedral primitive. This allows a degree of local control lacking with other primitives. It also keeps the number of primitives below a reasonable limit, which drastically improves display efficiency, making interactive modeling feasible.

- Polyhedral primitives show promise as a technique for shape transformation between polyhedrons, where they may be employed effectively in two different ways (see Appendix B).

- The implicit primitive definitions can be easily adapted to use higher order surface patches in place of polygons.

- A disadvantage of polyhedral primitives is the loss of a closed functional form for their definition. Such a form exists for every defining polygon of the bounding polyhedron. The functional computation at a point, however, involves a search for its defining polygon, before the functional form can be applied.

This chapter thus formulates a method by which object modeling and animation techniques based on polyhedral B-reps and implicit functions maybe combined in a simple and coherent manner.

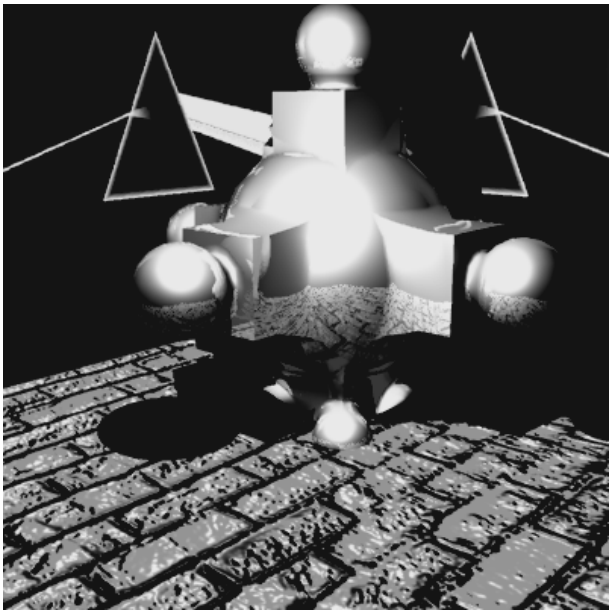


Figure 30: Blended Pointblobs: 2

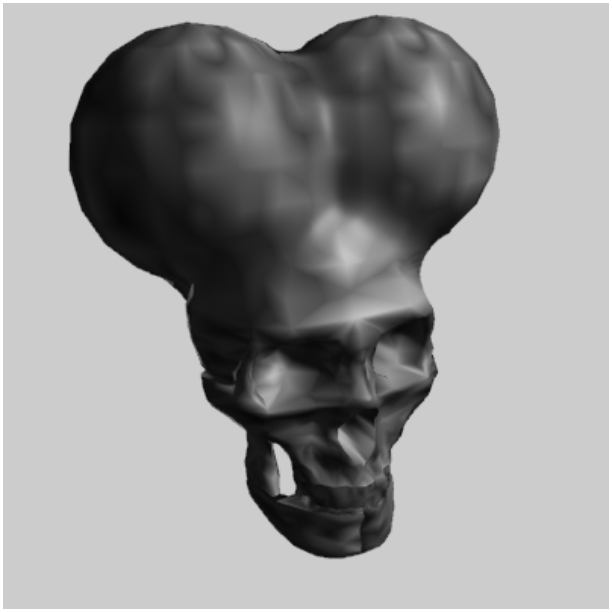


Figure 31: Blended Pointblobs: 3

Figure 32: Blended Pointblobs: 4

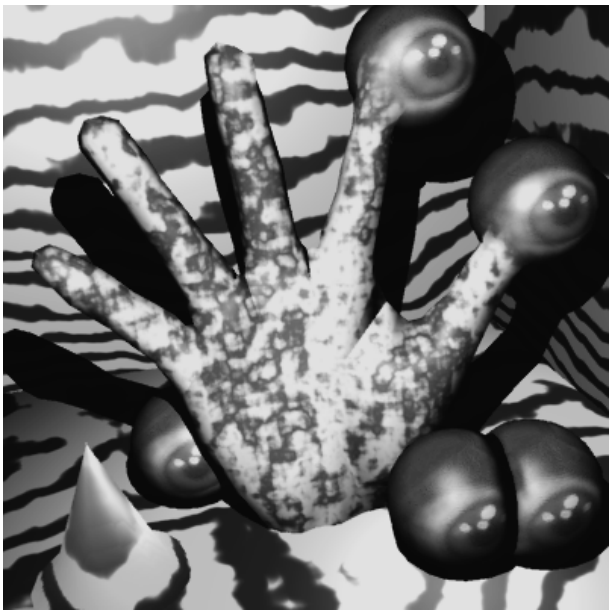


Figure 33: Point-setblob, Collision Deformations

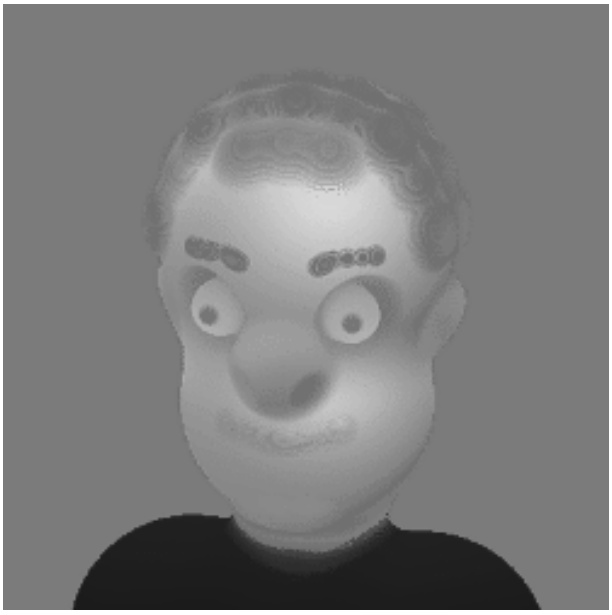


Figure 34: Scanline 1

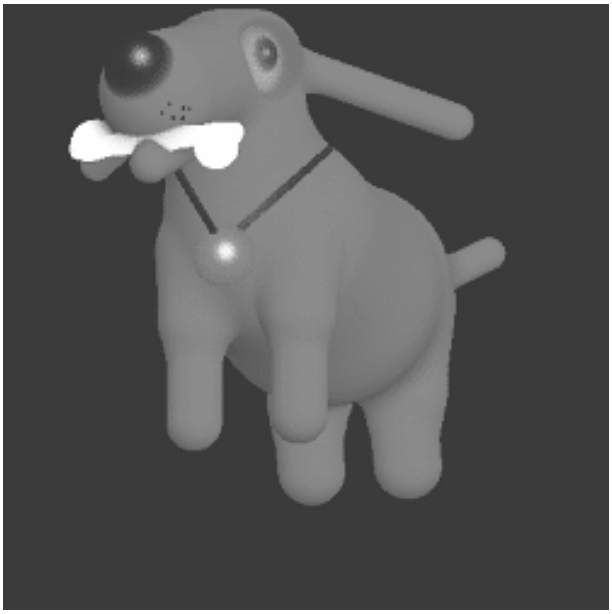


Figure 35: Scanline 2

Figure 36: Scanline 3

Figure 37: Scanline 4

CHAPTER IV

Implicit Function Based Human Figure Model

Human figures are an integral aspect of many VR applications. Varying degrees of realism are required of the human figures in different applications. Their animation and interaction with the virtual world in real-time, however, is quintessential. For applications like VR games or military training simulators [21], human figures are little more than passive objects in the environment. Figures that resemble robots or cartoon characters that possess little or no environmental interaction are often acceptable and even desirable. In such cases, simple geometric deformation approaches based on the skeletal posture [14][37] can be quite effective. The motion of the human figures is almost entirely scripted for most entertainment applications. This allows the off-line processing of motion using computationally expensive methods that provide a high degree of realism. The real-time aspect then simply involves a context dependent playback of precomputed postures or images from a database [21]. In an application like virtual space teleconferencing, every human figure is an active participant. A level of interpersonal interaction and communication that would make the illusion of reality complete is desired. It is evident that the aforementioned methods would not be suitable for such an application.

While dealing with human figures for a real-time application, decisions regarding the degree of realism need to be made with respect to modeling as well as animation.

Modeling at a level currently accepted as a realistic human figure (which is stationary for the most part) requires of the order of 10,000 polygonal facets [44]. This number needs to be increased in regions of large surface changes, if the figure is to be animated. Animation of very detailed object representations is both computationally expensive and hard to control. Textures are very useful for compensating for a lack of geometric detail. In Chapter V we describe a technique for the animation of facial wrinkles using textures, instead of modeling them geometrically. Such approaches, however, are limited to very fine surface detail. Other object representations like implicit surfaces are better suited to modeling human figures. They, however, lack the necessary hardware support for real-time display.

Given a virtual world application, we must first choose the level of detail at which to represent the human figures. The choice is largely determined by:

- The degree of realism required.
- The type and amount of mobility desired.
- The number of human figures involved.

For some applications like training simulators the size of objects relative to the observer is in constant flux. Human figure models with multiple levels of detail can be

used in such scenarios. For an application such as virtual space teleconferencing detailed human figures are used because the number of figures involved is few and a high level of realism is imperative. This thesis is primarily targeted at such applications.

Given a human figure representation, motion control mechanisms need to be prescribed. For most VR applications, behavioral and skeletal motion for the human figures is invariably scripted beforehand or driven by a real human using VR devices. Thus, we focus on generating the visual appearance of the human figure given a set of skeletal and behavioral parameters. Specifically, the issues to be addressed by the human figure model are:

1. Allowing customizable human models. We would like to generate or modify existing human figure geometries to closely resemble specific people.

2. Ensuring smooth connectivity of the geometric human figure during animation. This is a non-trivial problem especially around joints [36].

3. Modeling the formation of bone and muscle bulges, creases and wrinkles which add to the visual realism of the animated figure.

4. Modeling deformable contact surfaces on collisions. This may be considered as a global issue for the virtual world. We wish to address this issue from within the human figure model itself to better handle self-collisions. This also lays a solid foundation for hair and cloth modeling techniques.

5. Modeling the physical properties of the human figure for virtual worlds that involve force feedback. Issues such as muscle actuation, reaction to forces, and the deformable properties of human tissue need to be addressed.

Existing geometric approaches [14][37], while efficient, are not as realistic as physically-based animation. In particular, they do not address requirements 4 and 5. While manipulation of bezier surfaces [30] is well suited to requirements 2 and 3, they lack customizability and once again do not address interaction with the environment. The blobby model as developed by Opalach [46] caters to Disney type animation and lacks both realism and display efficiency. Physically-based approaches, such as biomechanical models [15], spring and damper methods [60] or the finite element method [37], are capable of addressing the above requirements well. They, however, lack the efficiency necessary for a real-time implementation within a complex virtual world.

This chapter presents a model for the animation of human figures using implicit functions. The techniques presented in Chapter II and Chapter III allow this model to be applied to both implicit surface and polygon based representations. The model proposed here is not anatomically correct. Instead it attempts to capture the basic dynamics of the human skeleton, muscles and fatty tissue and their visual impact on the shape of the skin. The model is based on implicit function techniques and lets us address both geometric and physical issues involved at various levels of sophistication.

The hierarchical model of the human figure, comprised of implicit primitives is presented in Section 4.1, and techniques for handling collisions, and other physical deformations during animation are discussed in Section 4.2.

4.1 Geometric Model

The human figure is essentially built up from a hierarchical model of implicit surface primitives. Unlike [46] where simple primitives build blobby human figures, a visually realistic appearance is required here, often of a specific human figure, such as for *avatars* of participants in virtual space teleconferencing. We thus require that the model allow some degree of customization based on data obtained from real humans[8].

In our approach a number of implicit surface primitives combined in a prespecified manner are used to define the skin surface. These may be obtained from real data (typically laser scanned) as polyhedra (point-setblobs) or superquadrics [55] (see Figure 39,55a). These primitives are fitted on a point-linked virtual skeleton (see Figure 38a). The blended **skin** primitives in themselves result in a realistic representation of the scanned human in some relaxed posture.

Similarly **muscles** are modeled as implicit function primitives that blend with and thus deform the implicit surface skin that envelops them. Muscle primitives are usually simple offset surface shapes (see Figure 39). From a purely geometric point of view, the change in shape of the muscle primitive based on the skeletal posture influences the shape of the skin (see Figure 40). The next section describes physical interpretations of the geometric model for muscles.

Bones often come close to the surface and actually define the shape of the skin during animation [36]. The implicit model handles this elegantly. Primitives for the skeletal structure are specified and blended with the geometric skin primitives. During

[8]Note that our model will work equally well for cartoonish characters.

animation, the skeletal primitives contribute to the shape of the skin automatically only when the primitive is close to the surface of the geometric skin primitive.

Veins may be modeled effectively using a technique similar to that presented by Bloomenthal [12]. Analytic offset surfaces around free form curves may be used to model a network of veins blended with the geometric skin primitives (see Figure 43).

Additional primitives may be used for control in special situations like animating bumps, scars, blemishes.

Point-setblob skin primitives allow the use of a number of blend centers to localize and control the blending in different parts of the primitive. Placed along the skeletal structure (see Figure 38a,39), the blend centers are analogous to points of muscle attachment to bones.

The primitives are selectively blended based on the hierarchical structure in Figure 38a,39. We use the term **selective blending** to indicate that primitives are combined in a manner specified by a combination graph (see Figure 39b). Primitives that do not blend mutually are treated as different implicit objects. All implicit objects in the environment interact with each other for collision detection and deformation. Thus self-collisions between different parts of the body and other environmental collisions are handled homogeneously. Joint regions such as the elbow where both blending and collision deformations occur, may be handled using the localized blending (see Figure 40) described in Section 3.2.3 or by introduction of an intermediate primitive that blends the two arm primitives together (see Figure 39,43). The use of an intermediate primitive is more general in that it is direction independent and can therefore handle

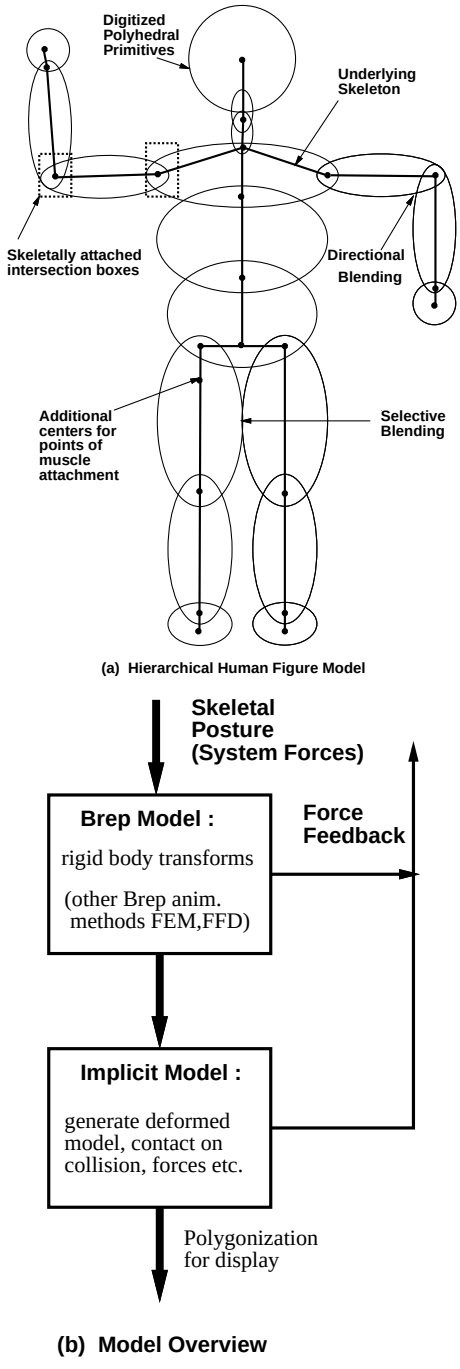


Figure 38: Hierarchical Implicit Primitive Human Model

3 DOF joints like the shoulder. On the other hand the localized approach allows finer control over the blending.

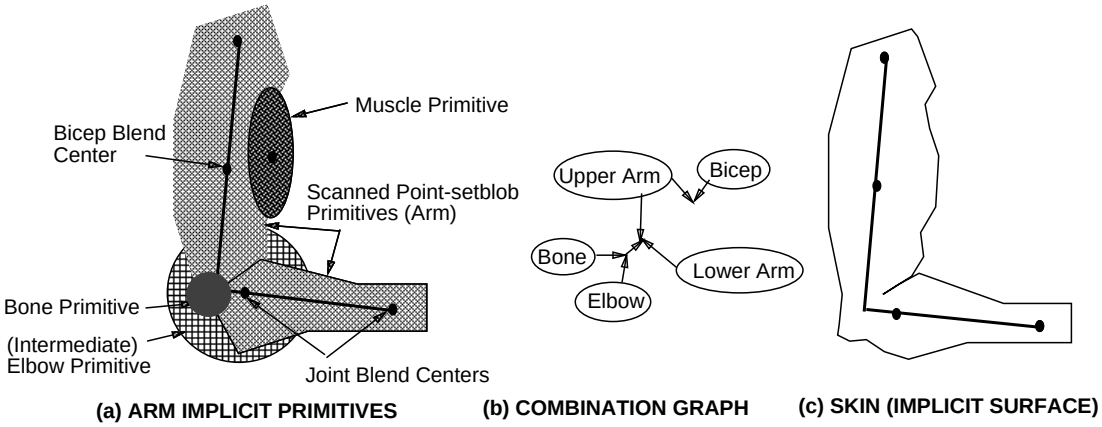


Figure 39: Implicit Function Model for the Arm

This model is animated by individual manipulation of the primitives based on the underlying skeletal model. The skin primitives are transformed as rigid bodies based on the skeltal posture. Muscle primitives undergo a similar rigid body transform. They are also scaled appropriately based on the relevant joint angles to simulate flexing and relaxing of muscles. Similarly other primitives are manipulated as functions of the skeletal posture. An overview of the model is shown in Figure 38b. Other implicit objects in the environment are treated similarly as described in [57]. The implicit functions automatically maintain a smoothly blended body as well as collision deformations.

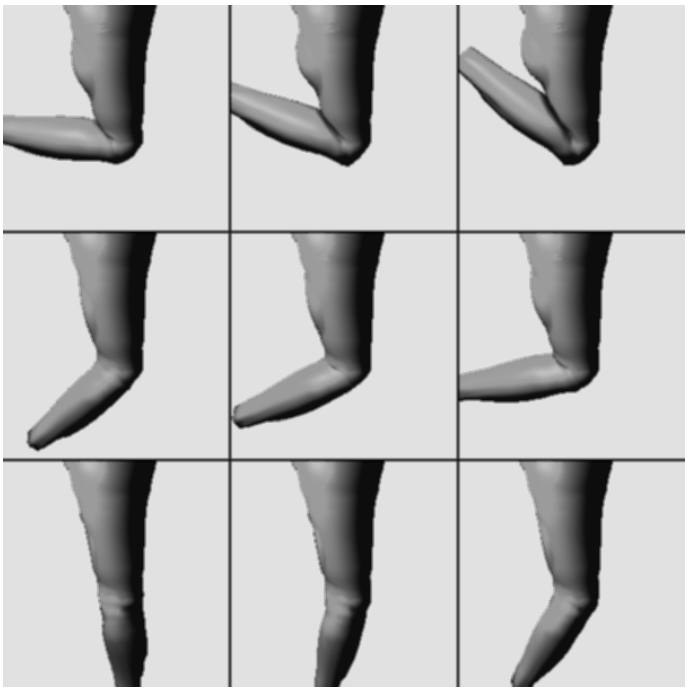


Figure 40: Animated Arm with Localized blending

4.2 Physical Model

The above model is sufficient for a purely kinematic implementation. So far all implicit primitive deformations are empirical functions of the animated skeletal posture. Further, the issue of interaction of the human figure with system forces is yet to be addressed. We now describe physical interpretations of the above model, which allow the human figure to be part of a dynamics driven environment. Issues that need to be addressed are the deformable nature of the tissue, modeling of forces generated by the figure due to muscle actuations and reactions (both with the self and with external objects/forces) The results should be manifested in terms of collision contact, creases and bulges.

- The general deformable model techniques proposed by Terzopoulos and Fleischer [59] may be applied directly in the context of point-setblobs by working on the primitives as polyhedra and then converting to the implicit function model.

- The blend function gradient is used to model radial spring stiffness by Gascuel [20]. Objects are considered as rigid with a superimposed deformable layer. Such a model is well suited to articulated figures that are supported by more or less rigid skeletal components [57].

- Variations in tissue characteristics [60] are modeled by piecewise smooth polynomial blend functions whose gradient reflects the change in stiffness. Localized variations of tissue stress/strain characteristics are easily handled by using different blend functions for different vertices of the skin point-setblob primitives (see Figure 21,22).

- Muscle actuations may be modeled by changes in the shape of the blend function instead of a scaling of the primitive. The change in shape of the blend function causes the *distance-ratio* value at which the fixed threshold value is attained, to change. This causes a displacement of the implicit surface. This displacement also causes an internal force. The force is obtained as the product of the displacement and the gradient of the blend function for a point on the implicit surface. The force computed in this fashion can then manifest itself on the skeletal posture (see Figure 40).

- The point-setblobs, in addition to joint blending centers, also have skeletal points that serve as points of attachment of muscles where forces resulting from muscle actuation can be applied. (see Figure 39).

- Wrinkles are handled well by local functional control. They may be generated by changing the blend function shape of the polyhedral vertices of the skin primitives along crease lines (see Figure 41).

Various levels of sophistication loosely based on the physical and anatomical characteristics of real humans are described above. More importantly, however, they capture the visual manifestations of human figure animation.

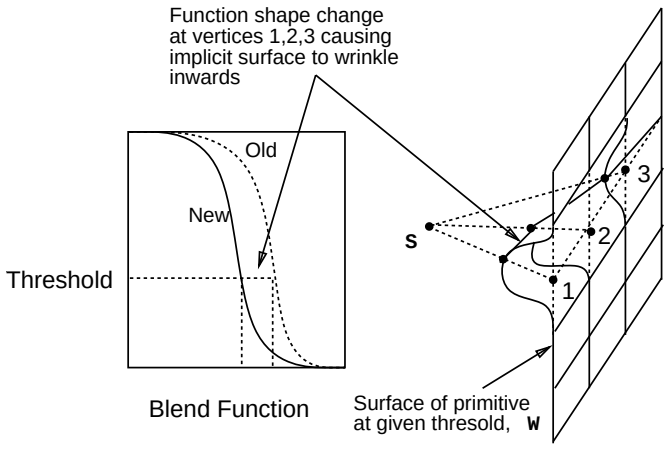


Figure 41: Wrinkles using local function control

The computational loop is like that described by Gascuel [20] and Singh [57]. The given forces and torques in the system are applied to the implicit function primitives and other objects of the system. Based on these forces, rigid body transformations,

such as updation of the skeletal posture, are carried out. The implicit function based deformable model is then applied. The implicit functions in the environment are updated and the implicit surfaces representing the human figure and other objects are displayed. New forces and torques in the system are then generated from the new function values (like muscle actuation) as well as forces such as reaction and friction calculated from collision contact surfaces.

To summarize then, the physical characteristics of objects are separated into two components (see Figure 38b). The first involves the animation of the figure as an articulated rigid body based on the skeletal posture. The implicit function primitives are individually manipulated using simple affine transformations. Further, techniques such as FFDs [14] or JLDs [37] that can deform individual polyhedral primitives are applied at this stage. This stage thus includes all transformations that are not implicit function based. The implicit functions primitives then interact with one another and with other implicit functions in the environment, spawning ghost functions to model blends and collision deformations and other physically based deformations.

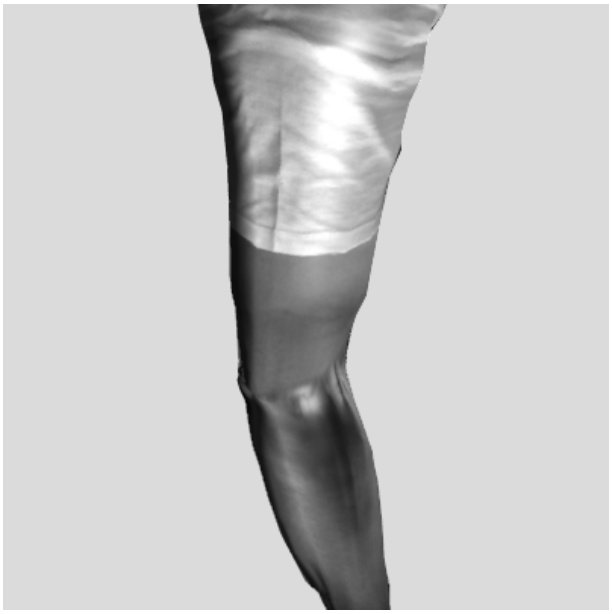


Figure 42: Animated Arm animated with Selective Blending:1

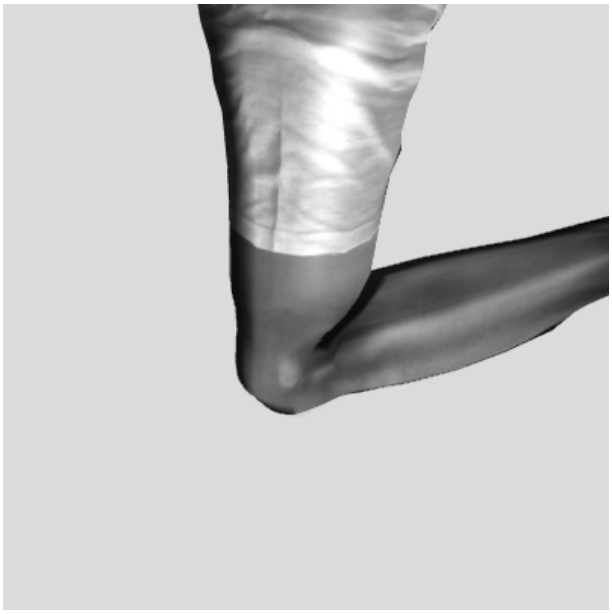


Figure 43: Animated Arm with Selective Blending:2

CHAPTER V

Texture Based Wrinkle Model for Skin and Clothing

Wrinkles and creases on skin and clothing go a long way in enhancing the realism of animated human figure models in virtual environments. Physical modeling and animation techniques for the wrinkling process, though effective, are too complex to achieve the real-time requirements of most VR applications. In this chapter we address the detection, formation and propagation of wrinkles for computer generated human figures in virtual environments.

Our wrinkle model allows the synthesis of wrinkled textures for skin and tight fitting clothing in a semiautomatic and interactive fashion. The wrinkles in a region are controlled by parameters extracted from the real human such as the skeletal posture or facial cues. Wrinkle formation is carried out by a combination of texture morphing and displacement of the underlying wireframe. Textures representing wrinkled and unwrinkled extremes are registered on the geometric wireframe and pixel level blending is employed on the textured polygons. The texture vertices of polygons on the two textures need not be spatially identical, justifying the use of the term texture morphing instead of simple texture blending. This model contributes simple techniques for synthesizing and animating skin and cloth textures for virtual humans.

We first propose a general model for the synthesis of wrinkles on textures. The animation of these wrinkles is accomplished in real-time by a combination of texture morphing and geometric wireframe displacement. Specific assumptions are made to tailor this model separately for wrinkles on the face, skin and tight fitting apparel.

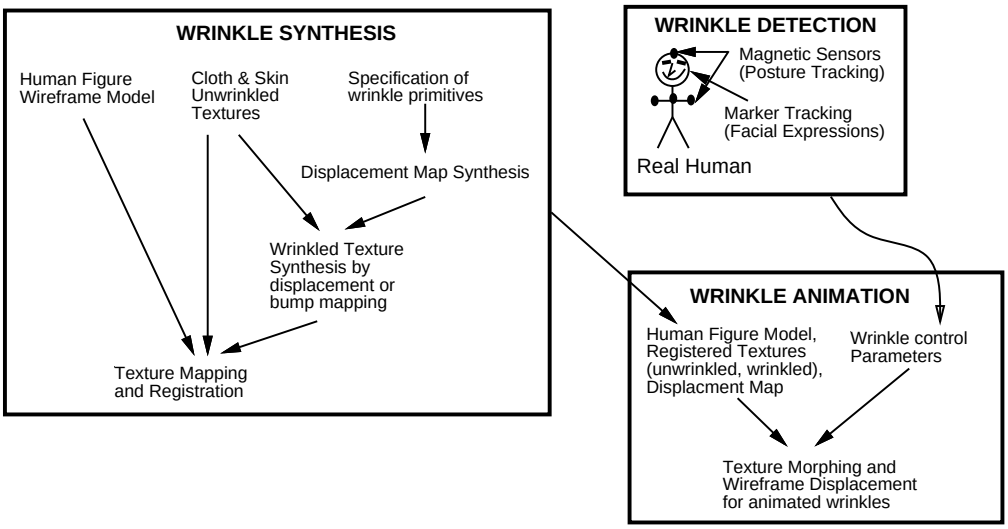


Figure 44: Wrinkle Synthesis and Animation Overview

Color textures for skin and garments are obtained in pieces and registered on the geometric wireframe. Wrinkled cloth textures may be similarly obtained and registered. This process of correlating separately obtained cloth textures representing extremes of deformation of the cloth or skin is a cumbersome task. Further the user would like more control over deformation parameters of wrinkles and creases. For this purpose we present a wrinkle model that captures its essential characteristics. Wrinkles described by the user in this fashion can be interactively used to generate a displacement map that synthesizes wrinkles on unwrinkled textures with intuitive

control over its characteristics. This makes the registering of textures with desired deformed shapes simple and efficient. Once the wireframe model and the textures have been obtained and correspondence established, an empirical correlation is interactively obtained between the human figure parameters and the resultant texture. The resultant texture is a morph between the synthesized wrinkled and unwrinkled textures. Further the displacement map that is responsible for wrinkled texture synthesis is used to displace the vertices of the wireframe model to enhance the realism of wrinkles. An overview of this system is shown in Figure 44.

Section 1 emphasizes the aspects of the human figure model our wrinkle model is built on. Section 2 presents a model for the synthesis of wrinkles on textures. Special assumptions are then made to tailor the model for facial and cloth wrinkles. Section 3 describes the interactive correlation of the morph between textures and skeletal animation in the case of clothing and facial parameters in the case of facial lines.

5.1 Synthesis of the Human Figure Wireframe Model

As described in Chapter IV, wireframe parts corresponding to various limbs are obtained, as may be conveniently sought using a Cyberware scanner [44]. These parts are then fitted together using implicit function techniques. Results of this approach are shown on a polygonized elbow (see Figure 55a). Laser scanned faces yield a wireframe structure that is adapted to the somatic characteristics of faces. Regions of high resolution correspond to parts that move a lot during facial animation (see Figure 55b).

Textures for skin and clothing are also obtained and registered by the Cyberware scanner (see Figure 50). Texture for the polygonized region may be obtained by blending between two textures as shown in Figure 55a. This provides a wireframe prototype representing the human in some prespecified skeletal posture.

Obtaining wrinkled clothing textures resulting from bent limbs using a Cyberware scanner is a difficult task. This is due to the deviation from a generally cylindrical shape and occlusion due to the folds of the garment. Artificially constructing realistic wrinkles on the cloth to be digitized is both time consuming and difficult to control. The wrinkles are unstable and thus have to be pinned or sewn together. Further, there is a lack of local control making small changes in the wrinkled appearance difficult. Another technique for photographing wrinkled textures is to place the wrinkled cloth on a flat surface and photograph it using a camera mounted overhead. This approach still suffers from the above problems, but to a lesser extent. Further, clothing like sleeves may have to be taken apart at the seams in order to lay it out flat and the approach is not efficient, if one wishes to construct a number of clothes from different textures.

Synthesizing wrinkles on textures using a computer dispenses with the large problem of correspondence between textures that are independently obtained. The print, seams, buttons, zippers and such make the problem of correspondence both essential and cumbersome for clothing. Equally important is the positioning of the eyes, nose, mouth, distinguishing marks and other such features on a facial texture on which facial wrinkles are desired.

The next section, therefore, addresses the problem of user controllable wrinkle synthesis on unwrinkled cloth textures.

5.2 Synthesis of Wrinkles

Kunii addresses wrinkle formation on garments [31]. The advantages of using properties of wrinkle formation when animating complex garments, as opposed to a general cloth animation approach are stressed. Singularity theory is used to analyze the shape of garment wrinkles. Modeling primitives comprise characteristic branching and vanishing points with associated contours. We use similar characteristic primitive shapes. Wrinkle formation in Kunii's approach is carried out by solving an energy minimization problem while preserving metric invariance. Environmental effects can be incorporated into the solution of this problem. In our approach a static wrinkle specification is used to synthesize a wrinkled texture. Animation is achieved by morphing between the original and the wrinkled texture. The wrinkle specification is also used to displace the geometric wireframe skin during animation.

5.2.1 Wrinkle Specification

We first specify a wrinkle or crease in clothing by defining a characteristic shape of the wrinkle. The shape is represented as a parametric 2D curve specified by a number of control points as in Figure 45. Each control point also has values that specify attributes of the wrinkle, such as intensity, in the proximity of that control point. Attribute values and the size of the wrinkle are normalized to represent a characteristic shape (see Figure 45). A taxonomy of characteristic wrinkle shapes is built. A

characteristic shape may be created interactively using a graphic editor. Alternatively, we can isolate the skeletal shape of a photographed wrinkle by edge detection and assign intensity weights based on wrinkle gradient values (see Figure 49). The shape is normalized automatically.

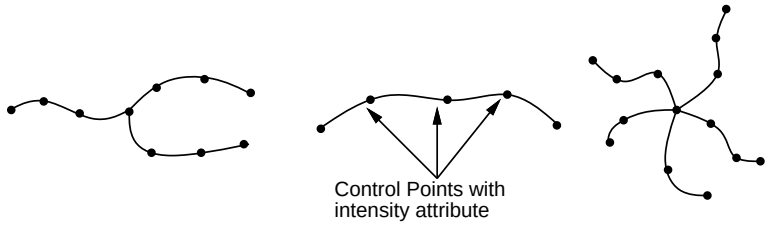


Figure 45: Characteristic Wrinkle Shapes

A synthesized wrinkle is then an instance of a characteristic shape with the following parameters:

- Wrinkledness: Determines the pseudo-random deviation from the characteristic shape.
- Thickness: Defines the extent of deformation of the texture around the characteristic wrinkle shape.
- Intensity: Defines the depth of the furrows and ridges of the wrinkle shape.
- Spatial Transform: Transforms the normalized characteristic shape to a common space where the wrinkled texture is synthesized.

A number of wrinkle instances, specified interactively, are then used to generate a displacement map. Implicit surface techniques are used in the generation of the displacement map.

5.2.2 Wrinkled Texture Synthesis

For the purpose of displacement map synthesis we construct an **offset surface** density map around each wrinkle with the transformed characteristic shape as the skeleton S. The offset primitive corresponding to the wrinkle is constructed as follows:

1. The normalized characteristic shape is first deformed based on the *wrinkledness* factor. This is done by suitably jittering the control points of the articulated skeletal shape. This skeletal shape S is then spatially transformed to a common space where the displacement map is synthesized.

2. The *thickness* parameter controls the radius r of the offset surface.

3. The *intensity* parameter scales the intensity (implicit function) values appropriately.

A 2D displacement map is then calculated by evaluating the summed displacement contributions due to the wrinkles at each pixel of the map. For any pixel P of the map, the displacement contribution of a wrinkle is computed as follows:

1. The point Q on S with the shortest euclidean distance to P is computed.

2. The wrinkle contributes to the displacement at P only if it is within radius R of the wrinkle, $|P - Q| < r$.

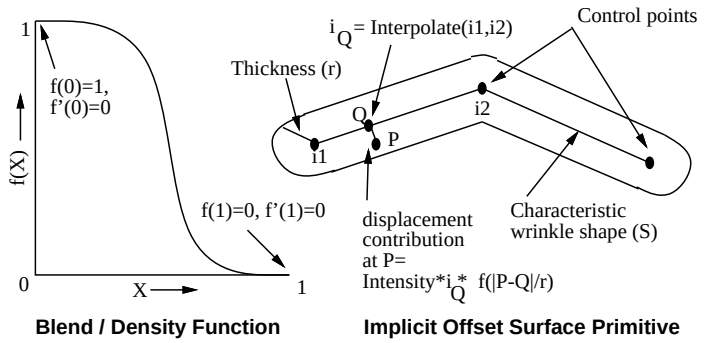


Figure 46: Implicit Function Primitive

3. The intensity attribute value of the characteristic shape at Q is obtained as an interpolation of the attribute values of the two closest control points between which it lies. Let this value be i_Q.

4. The displacement contribution of the wrinkle is then $intensity * i_Q * f(|P-Q|/r)$, where *intensity* is the user controlled parameter of the wrinkle instance and f a density function as in Figure 46.

This primitive shape formulation is well suited to facial wrinkles. Cloth wrinkles often form in a series of ridges and valleys connected together [31]. The offset primitive above models the beginning of a ridge well. Valleys may be modeled by wrinkle instances with negative intensities.

Alternatively, the semicircular cap at the end of a wrinkle shape can be replaced by the curve shapes as in Figure 48. The intensity computation procedure for a point where the closest point to the skeleton is an end point can be performed as follows:

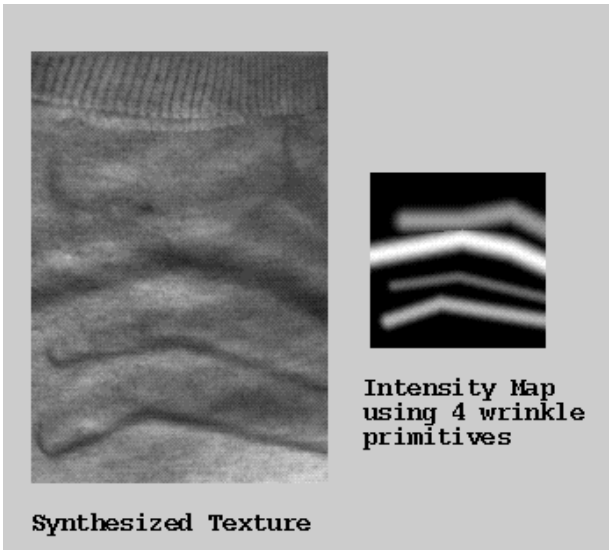


Figure 47: Displacement (Intensity) Map Synthesis

1. Compute the intercepts of the point on the two corresponding curves modeled as explicit cubic polynomials (see Figure 48).

2. The displacement contribution is then $intensity * i_Q * f(d_x) * f(d_y)$, where d_x and d_y are corresponding distance ratios of the point from the axes to the curve intercepts.

First order continuity at the axes where the computation procedure differs is ensured by the properties of the function f. This provides more intuitive control over the shape of a valley.

The unwrinkled texture is then mapped onto a flat surface. The displacement map is used to displace the surface appropriately. The displaced (wrinkled) surface subsequently rendered provides a wrinkled texture. The lighting conditions are kept

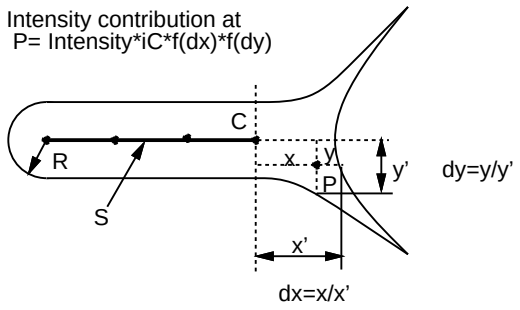


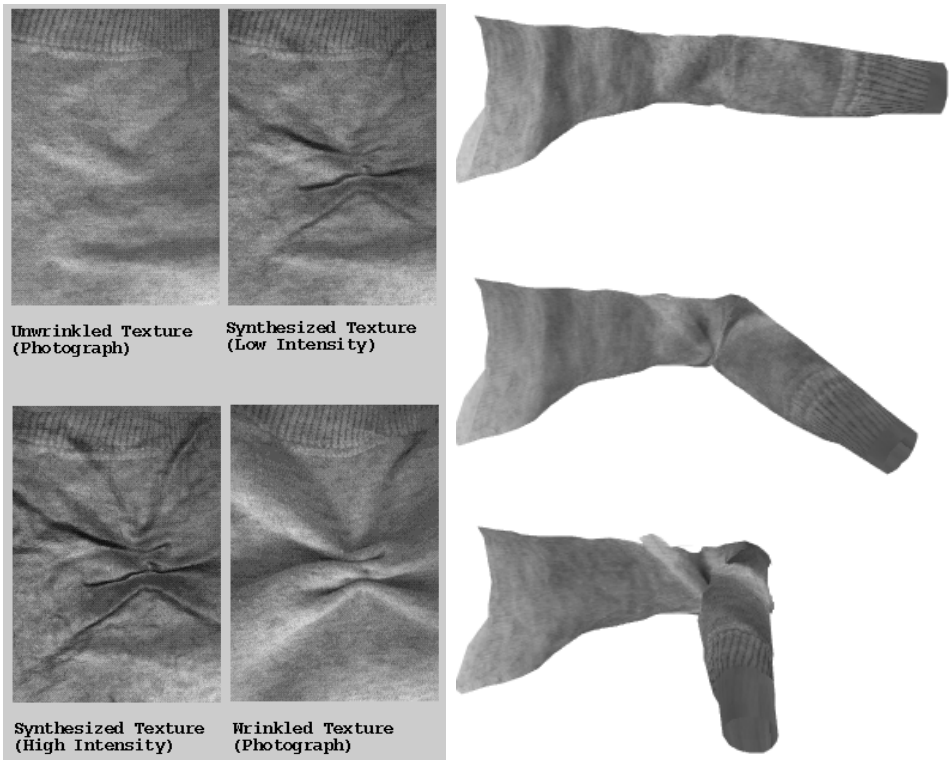
Figure 48: Augmented Implicit Primitive

close to those expected in the virtual world application. The user has precise and intuitive control over the shape, size, number and extent of wrinkles (see Figure 47,49).

Wrinkled cloth textures often exhibit occlusion as a result of large and deep wrinkles. A high intensity displacement map can cause enough motion of the texture to lose precise correspondence with the unwrinkled texture. Thus, there is a tradeoff between perfectly matched textures and the extent to which the wireframe can be displaced. The surface detail of facial wrinkles is much finer. In this case the gradient of the displacement map may be used as a bump map to perturb the surface normals across the polygons of the mesh [7]. The textures in this case remain in perfect correspondence as the geometric surface of the mesh remains the same (see Figure 50).

5.3 Animation of Wrinkles

Animation of wrinkles is a two step process. The first step deals with control of the wrinkling process, which generates wrinkle specifications that lead to the display of the wrinkled figures in the second step.



(a) (b)

Figure 49: Wrinkled Cloth Texture Synthesis and Animation

Figure 50: Wrinkled Facial Texture Synthesis

5.3.1 Wrinkle Detection

In virtual environments, control may be computer generated or based on the motion of a real human as in the case of virtual space teleconferencing. We assume the formation and disappearance of wrinkles is based solely on motion local to the region being wrinkled. Thus motion cues for different textures are specific to the region of the human figure to which they are applied.

Joint angle values of the skeletal posture are used to control the wrinkling of clothing. Posture computation in a virtual space teleconferencing system is in itself a nontrivial task [44]. The wrinkle model assumes correct posture computation of the underlying skeletal structure to some reasonable level of accuracy.

Facial wrinkles may be parametrically controlled for computer generated figures. Tracking a real human face is a much harder problem. In *VISTEL* all facial parameters are derived from simple real-time image processing of the real human face. Wrinkles on the forehead for instance are controlled well by the relative positions of the two eyebrows and the hairline. Wrinkles around the eyes are controlled by the opening and closing of eyelids.

5.3.2 Wrinkle Generation

Animation of the wireframe model based on motion of the underlying skeletal structure is carried out using a hybrid model [58]. The wireframe skin is deformed using an implicit function model. Additionally Free Form Deformations animate the torso, joint local deformation operators are used for the hands [58] and marker tracking is

used for facial animation [44].

The quarternion based rotation angle across joints is used to control a color texture morph between two extremes of cloth textures. This is coupled with additional displacements to the geometric wireframe around the joints and regions where cloth animation occurs based on the corresponding displacement map. The vertices of the wireframe in these regions are displaced along their vertex normal. The extent of displacement is controlled by both the displacement map and the joint angle used to control the texture morph (see Figure 49). Correspondences between angle value, wireframe displacements and morph are interactively manipulated to realize satisfactory realism during animation.

Rationale for employing both a texture morph and wireframe displacement is the following:

- Clothes being only finitely elastic, shift in position on the body on wrinkle formation. This is hard to capture by a simple displacement of the geometric position of the wireframe. This may, however, be accomplished by varying the texture vertices during animation of the body. This can be seen in Figure 49b, where the sleeve is pulled up on the bent arm.

- The clothes are not explicitly modeled geometrically. The displacements from the displacement map, therefore, need to be attenuated so as to maintain continuity with the skin and to prevent wireframe consistency problems due to self-collisions. Thus one needs to couple the displacements with a morphed wrinkled texture so as to accentuate the appearance of the wrinkles.

- Synthesizing wrinkled textures may be done in a more realistic fashion as it is a precomputation step. It is also independent of the resolution of the human figure wireframe, as opposed to displacement of the wireframe itself.

- The displacements to the actual wireframe are carried out so as to prevent wrinkle silhouettes from looking unnatural (see Figure 49).

The facial wrinkles are similarly animated as texture morphs between two extreme textures. These wrinkles are fine surface detail with respect to the geometry of the face. The morph, therefore, need not be accompanied by surface displacement (as shown by the realistic profile in Figure 51). In this way we can limit the resolution of the facial wireframe.

Figure 51: Facial Animation and Profile

This chapter has thus provided techniques for the synthesis and propagation of

wrinkles on skin and clothing. The approach enhances the visual realism of the human figure in virtual environments at a minimal computational overhead.

CHAPTER VI

Virtual Space Teleconferencing System: Implementation

This chapter describes the application of the techniques developed in the previous chapters to the synthesis and animation of human figures in a virtual space teleconferencing system.

In recent years, communication between humans at distant locations has increased in importance due to the trend toward more collaboration and the increasing time, energy, and expense necessary for transportation. The concept of Virtual Space Teleconferencing (*VISTEL*) aims at an environment where teleconference participants at different sites can feel as if they are all at one site, allowing them to hold meetings and work cooperatively. In Virtual Space Teleconferencing, models of the participants at each site are created in a computer generated 3D image of a virtual world. The participants' movements are reproduced by deforming the human models according to the detected movements of the actual participants. The virtual world is rendered on a 3D display at each site so that the participants experience the feeling of meeting each other within a common space. The principal advantages of this concept over current video conferencing systems are improved interpersonal communication and

the capability of co-operative manipulation of the virtual world. Further, the volume of information transmitted between sites is small, which has a significant impact on the time lag due to data transmission.

To achieve natural, smooth communication among the participants, it is necessary to realistically reproduce their movements in real-time. In Virtual Space Teleconferencing, real-time reproduction of human motion is carried out in three steps:

1. detecting human motion at the originating site,
2. sending data regarding the detected motion through transmission channels to all of the other sites, and
3. animating the human models using the transmitted data at each receiving site.

Section 6.1 provides an overview of the *VISTEL* setup. Section 6.2 deals with the detection of the skeletal motion of the real humans using magnetic sensors and data gloves. Section 6.3 describes the tracking of facial expressions using cameras and markers. Section 6.3 provides details on the implementation of the implicit function based human figure model described in Chapter IV. Section 6.4 discusses animation of facial and cloth wrinkles based on the model presented in Chapter V. Section 6.5 applies the concepts developed in Chapter II for realistic grasping of virtual objects by human figures.

6.1 System Overview

An experimental system for Virtual Space Teleconferencing for three participants has been constructed at ATR Communication Systems Research Laboratories. Figure 52

shows a two person teleconference scene, where persons A and B are at Sites A and B, respectively. At Site A, for example, the 3D image of person B is placed in an artificially created virtual space (cooperative work space). The virtual space, including the image of person B, is rendered on the 3D display at Site A so that person A can work cooperatively with person B. The system is currently configured for 3 people.

Figure 52: Virtual Space Teleconferencing

SGI (Onyx, Reality Engine) machines perform graphics processing and display at each site. the reality engines have hardware support for color texture mapping and pixel level α-blending. The 3D human image in a synthesized virtual space is rendered on a V-shaped 70x2-inch stereoscopic display. The participants wear LCD shutter glasses for stereo viewing.

The facial expressions of a participant are detected by a workstation (Iris 4D340/VGX) at that site. Images are acquired by a color video camera attached to a helmet worn by the participant. These images are thresholded by a chroma keyer to detect green tape marks. Images from another video camera trained on the eye are thresholded by a level keyer for gaze and blink detection. The tracking process based on the processed images is then carried out by the workstation. Posture computation employs 4 Fastrak magnetic sensors (head, chest and wrists) and cyber gloves on each participant.

Motion and other facial information acquired at each site is sent to the others through the EtherNet. The skeletal parameters and virtual world information are used to animate the computer generated human figures using the implicit function deformable model. The skeletal and facial parameters also drive the wrinkle model that determine the mapping of color textures on the wireframe geometry of each human figure. An overview of this can be seen in Figure 53.

Microphones and speakers are present at the sites so that participants can communicate by voice. To synchronize the voices with the images, delay equipment is used on the acoustic line between the two sites with the time delay being set at 0.2 in the current implementation.

To illustrate the power of collaborative work, participants can grasp and manipulate computer generated objects in the virtual world. A speech recognition program allows participants to position and deform these objects based on a vocabulary of object transformation commands in Japanese.

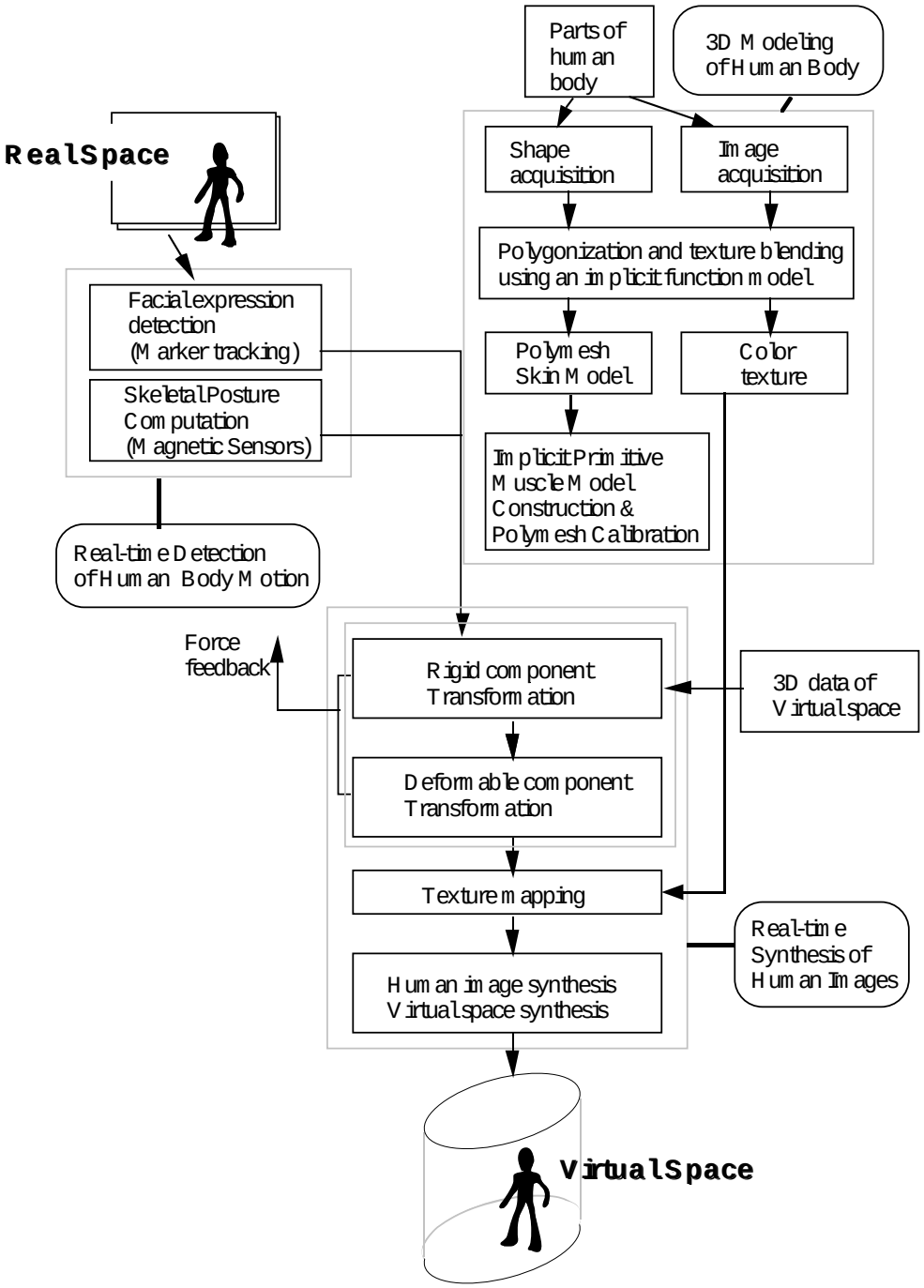


Figure 53: Human Figures in *VISTEL*

The virtual world is currently refreshed at 8 frames per second. The implicit function based human figure model running as an application at an isolated site, runs in real time.

6.2 Skeletal Posture Computation

The skeleton of human figures in *VISTEL* is approximated as an articulated rigid body. The skeleton as such has no explicit geometry. Bones that are important in their contribution to the visual appearance of the skin surface (see Section 4.1), are modeled using implicit functions. The elbow and knee cap, for example, are modeled as analytic spheres. An exception to the articulated rigid body approximation is made at the shoulder, where the collar bone is capable of expanding and contracting. This is the solution proposed by the Thalmanns [36] to account for the shift in the center of the shoulder joint for different angles of the arm at the joint.

We use a number of assumptions and heuristics to greatly reduce the number of degrees of freedom (*DOFs*) possessed by our skeletal model. Typical teleconference scenarios deal with the participants sitting around a conference table. We thus assume that only a sitting/standing motion needs to be reconstructed below the waist. We further assume that the feet remain fixed to the floor. The skeletal model adopted is shown in Figure 54.

The spine is modeled using a number of segments (currently 8). Experimental measurements and heuristics are used to distribute motion over the spinal segments. This allows us to model the entire spine using 6 *DOFs* at the top of the spine with

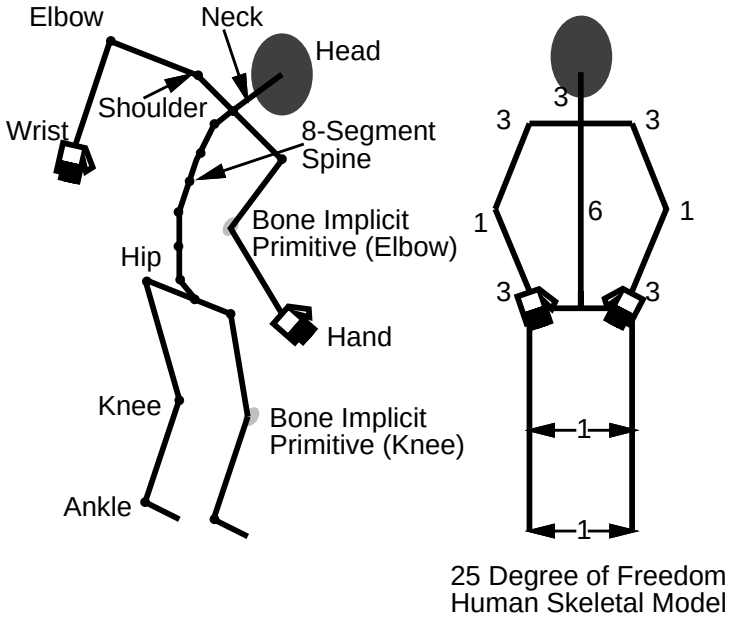


Figure 54: Skeletal Model of the Human Figure

respect to the base of the spine at the waist. The type of motion (sitting/standing) below the waist is simple and the same for both legs. The rotation of the hip joint is incorporated into the base of the spine. We also assume that the orientation base of the hip remains constant with respect to the virtual world. Motion below the waist is thus reduced to 2 *DOFs*, one each at the knee and ankle. Each arm is modeled with 7 *DOFs*. The shoulder is a 3 *DOF* joint, the elbow a 1 *DOF* joint and the wrist is modeled with 3 *DOFs*. The neck has 3 *DOFs* with respect to the top of the spine, which ends at the base of the neck. The position of the shoulder joint relative to the base of the neck is determined by the collar bone. This bone varies in length based on the orientation of the shoulder with respect to the torso. An experimental correlation between the orientation of the shoulder and the length of the bone is

made for realistic results [36]. Each hand is modeled as an articulated rigid body with every finger possessing 3 digits with 1 *DOF* each attached to a rigid palm. The human figure, excluding the hands, is thus skeletally modeled using 25 *DOFs*.

Image processing could be useful for motion detection in this system due to its passive nature. It is, unfortunately, difficult to achieve real-time detection of human body motion using existing computers and current image processing algorithms. In addition, most image processing methods are not sufficiently robust for teleconferencing applications. Therefore, alternatives using sensors and Cyber-gloves are used in the current implementation.

Four Fastrak sensors are attached to top of the head, top of the torso and to the two wrists. Each sensor returns 6 *DOF* position and orientation information at a rate of 60Hz.

The positioning of the trackers in this manner makes posture tracking from head to toe a simple inverse kinematic task using the 12 *DOFs* obtained from the sensors on the torso and head. Each arm has one *DOF* more than the 6 *DOFs* obtained from the wrist sensor. An energy minimization approach is adopted to solve this underconstrained system [42]. A more robust solution that involves tracking the skeletal posture rather than recomputing it is being studied. This would make use of the high rate at which the sensors return position and orientation information. The hand gesture is computed directly from values returned by the cyber gloves worn by the participant.

6.3 Facial Expression Tracking

For real-time detection of facial expressions, thirteen tape marks are pasted to a participant's face. The positions of the tape marks correspond to the facial muscles that strongly affect changes in facial expressions. The marks are then tracked in the images acquired by a video camera positioned in front of the face. To facilitate tracking, the color green, which is very different from faces' natural color, is used for the marks. Green pixels are extracted from the face images by a chroma keyer. The position of a mark is obtained from the centroid of the candidate pixels for the marks. In the current implementation, the participant wears a helmet with mounted cameras to enable stable tracking.

Another video camera is attached to the helmet in order to detect gaze directions and blinks of the eye. In the current implementation, only one eye is observed. The images are processed in a manner similar to the tape marks. The gaze and blinks of the unobserved eye are assumed to be symmetric.

The results of the tracked markers are used to move the nodes of the wire-frame model of the face. The wire-frame deformation is based on motion rules which are defined in advance. These rules are based on the actions of facial muscles described in the Facial Action Coding System [16].

The results of detecting gaze and blinks of the eye determine the orientation of the eyeballs and the movement of the eyelids, respectively. Facial wrinkles on the forehead are animated using the technique based on texture morphing in Chapter V. Figure 51a,b show wrinkles of increasing intensity on the forehead in correspondence

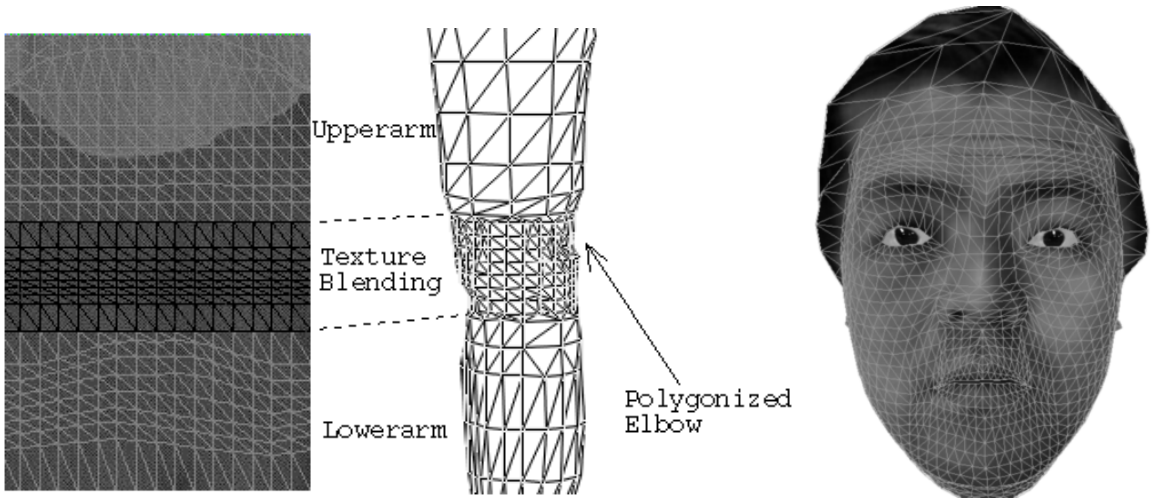
with the eyebrows being raised.

6.4 Polymesh Human Figures and the Implicit function Based Model

Synthesis of polymesh models of real humans is a nontrivial problem. Various reconstruction methods, using sculpted models, range data, photographic images have been proposed [47]. In our approach a number of polymesh parts corresponding to various limbs are obtained using a Cyberware Color 3D digitizer [44]. These parts are then fitted together, which may be done by lofting between the end contours of segments. We choose, however, to blend the parts using the polyhedral implicit function primitive definitions of Chapter III. This better preserves the overall length and shape of the limbs. Further, control over the region of the blend can help automatically attenuate glitches and noise in the scanned data that often results at the fringes.

Results are shown on a polygonized elbow in Figure 55a. The upper and lower digitized limbs are treated as point-setblob primitives and blended together at the elbow. A user controllable region is then defined within which the blended point-setblobs are repolygonized using the algorithm developed in Section 3.3.4. The color texture (skin and clothing) for the polygonized region is an appropriate blend of the textures for the two limbs as shown in Figure 55a. This provides a polymesh prototype representing the human in some prespecified skeletal posture. Complexity of the human figures (see Figure 12a) is approximately 10000 vertices.

The implementation of the implicit function based human figure model described



(a) Arm (b) Head

Figure 55: Human Figure Wireframe Synthesis

in Chapter IV, treating the limbs as point-setblobs, would require regions of primitive interaction to be re-polygonized every frame. For models of the complexity above, the computational efficiency currently falls short of the real-time requirements. Chapter II, however, provides us with a technique by which implicit function based models can be applied to polymesh objects. The human figures are, therefore, embedded in 23 simple implicit primitive shapes such as spheres and sphylinders (see Figure 12a). The implicit model is constructed hierarchically, in an object oriented fashion, making the fitting of other polymesh objects as well as the introduction of new implicit primitive shapes, a simple task. Additional analytically defined implicit primitives model bones and muscles as described in Chapter IV. These implicit primitives deform the underlying polymesh model in **real-time** by causing the polymesh vertices to track an implicit surface defined by the implicit function primitives. The model

is computationally efficient and performs collision detection and all physical deformations with a worst case time complexity that is linear in the number of polymesh vertices of the environment (see Section 2.4.2).

Figure 3 shows the elbow region after rigid component transformation and its subsequent deformable component transformation. Spherical primitives modeling skeletal elbows cause the elbow to protrude in the bent arm and precise crease formation may be seen. The deformable component computation using a Regula Falsi-Newton Raphson approach typically takes 2-3 iterations per vertex.

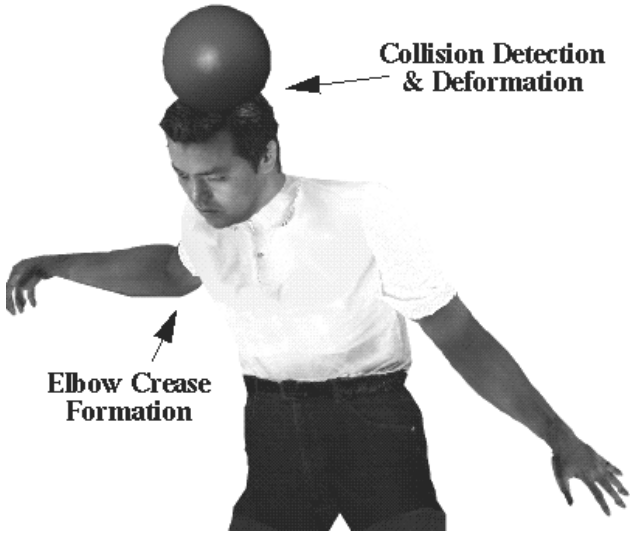


Figure 56: Human Figure Animation

Existing polymesh based muscle and skin modeling techniques may easily be integrated with the rigid component in our implementation. As an example, FFDs [14] on the spine animate the torso (see Figure 12b,56).

6.5 Cloth Animation

The overview of the system for cloth and facial animation effects is shown in Figure 44. Unwrinkled textures are photographed as in Figure 8a or laser scanned as in Figure 50. Wrinkled cloth textures may be either photographed or synthesized on the unwrinkled cloth textures (See Figure 49a). Wrinkled texture synthesis requires the synthesis of a displacement map (See Figure 47). Wrinkle primitives for the construction of this map can be either interactively drawn as in Figure 47, or obtained by image processing a wrinkled photograph as in Figure 49a. The unwrinkled and wrinkled textures are then automatically mapped and interactively manipulated on the wireframe model as in Figure 57. Figure 57 also shows the mapping of a wrinkled shirt texture (synthesized on an unwrinkled cyberscanned texture) on the torso wireframe of a human figure.

The results of the mapping during animation of the wireframe may also be viewed simultaneously, making texture registration a simple process. Texture and displacement interpolation can also be interactively controlled here. Texture morphing is carried out by using the pixel blending hardware supported by the SGI (Reality Engine) machines. Figure 49b shows wrinkle formation on a bending arm. Note the upward motion of the sleeve as the arm bends exposing the wrist and the wrinkled appearance on the silhouette resulting from wireframe displacement.

The order in which the wireframe deformations on the human figure due to the implicit function based model and due to the wrinkle model are applied is important. The wrinkle model deformations are independent of the human figures interaction with the virtual world. It must, therefore, be applied to the figure before the implicit

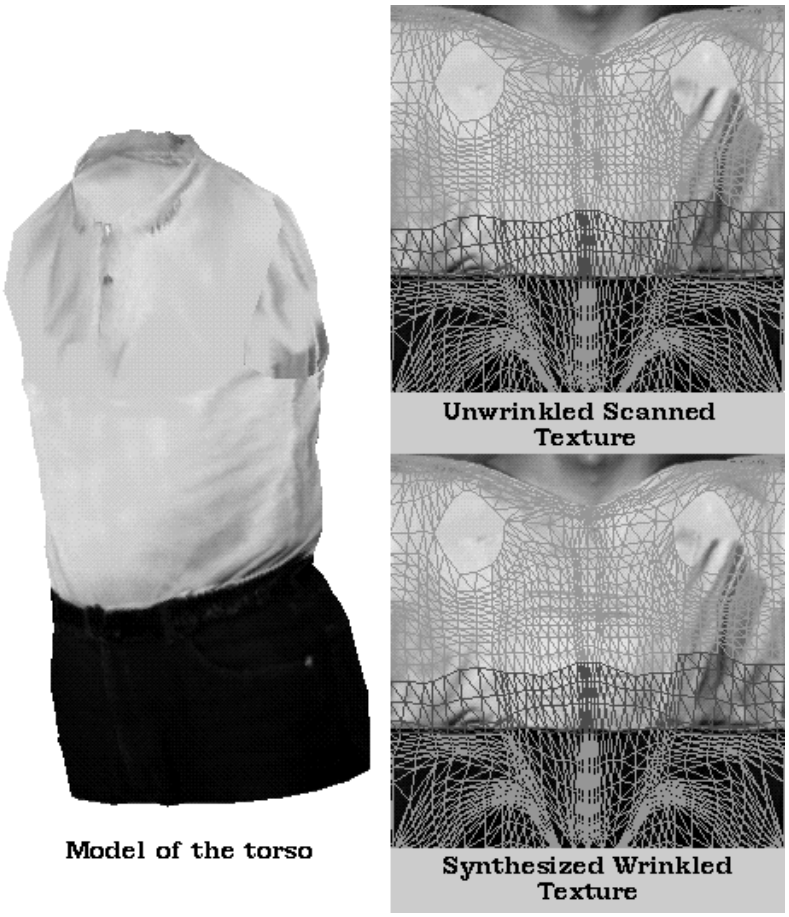


Figure 57: Interactive Texture Mapping

function based deformation model, which addresses the interaction of the figure with the environment. Applying the wrinkle model before the implicit function based model, however, introduces a new problem. The displaced wireframe vertices due to the wrinkles are at different spatial positions from the ones at which they were calibrated (see Section 2.3.1). Thus the implicit function based model will tend to flatten out the wrinkles by driving the displaced vertices to the positions at which they were calibrated. This can be solved be recalibrating the wireframe after the wrinkle model has been applied.

A computationally cheaper solution is based on the following observation. The problem of wrinkles flattening out is only present in regions where the human figure does not interact with the environment. In other regions, such as where an arm collides with another object, the wrinkles on the sleeve should be deformed anyway, as dictated by the implicit functions. Only wrinkled regions where environmental interaction is absent need to be left untouched. In the implementation in Section 2.5, vertices of the wireframe whose defining implicit primitives do not interact with other implicit function primitives are not deformed. This is perfectly in accordance with the deformable model. We simply do the same for the vertices after they are displaced by the wrinkle model. Note that now some vertices that are displaced by the wrinkle model will not lie on the implicit surface at threshold T for the human figure. We do, however, get all the visual and environmentally related properties of the human figure that we desire. This is the solution adopted in the current implementation.

6.6 Object Grasping: Constraint Satisfaction

The virtual space teleconferencing system allows participants to grasp and manipulate objects in the virtual world. The distance computations between the implicit primitives embedding the objects and the participant's hand are used to determine proximity to the object. Proximity and the degree of bend in the fingers of the hand determine whether an object has been grasped. To make the grasp realistic the virtual hand's position and orientation is adjusted with respect to the object being grasped. This is done by aligning the hand parallel to an axis at a precomputed distance for a given object. A number of axes are constructed as potential directions along which an object can be grasped. For a given axis an average distance of the palm from the axis is calculated based on the size of the object. As an example a typical axis for an object like a glass is the cylindrical axis and the radius of the glass is the precomputed distance. The hand is aligned with the axis which requires the minimum change is position and orientation of the hand.

The constraint satisfaction and collision model of Chapter II between the hand and the object then cause the hand to deform so that it precisely grasps the object without penetrating it. In this scenario a constraint contact surface between the hand and object is desired. This implementation, therefore, does not use the sphylinders that limit the deformations along an axis.

CHAPTER VII

Conclusion

To summarize, this thesis develops an implicit function based human figure model. The model addresses issues involved in the modeling and animation of human figures in VR applications. Methods that allow an integrated application of implicit function and B-rep based techniques are presented. These solutions are applied to the realistic real-time animation of human figures in a virtual space teleconferencing system.

7.1 Contributions

7.1.1 Implicit Function based Virtual World Model

Firstly, we provided a generalized framework for modeling and animating deformable objects and external forces in a virtual world using implicit functions. Implicit functions are an elegant an efficient way to model interaction between objects and forces in an environment. Ghost functions allow the modeling of complex dynamic behavior efficiently and intuitively. The model can also be treated kinematically, the dynamics simulated by simple user defined parameters. A method for constraint satisfaction built on previous work in collision detection and deformation was fitted into the general framework. The constraint satisfaction is novel and has many advantages over

existing approaches. It is an integrated collision and constraint satisfaction model for deformable objects. Like the implicit function deformable model it is analytic and the deformations are theoretically precise. The deformations are also local allowing arbitrary constraint graphs between objects and even multiple constraints between the same pair of objects. The implicit function model for deformable objects thus provides us with a new framework catered to the modeling and interaction of objects and forces in complex virtual worlds. Collisions and constraints, which are quintessential to many virtual world applications are handled exceptionally well within this model.

7.1.2 Merging B-rep and Implicit Function Technology

On studying possible representations for computer generated human figures, we found that B-rep and implicit surface representations have advantages that are complementary. We thus developed techniques that allow complete flexibility in using B-rep or implicit surface representations for objects. This was done in two ways.

The first allows implicit functions to approximate an object modeled as a B-rep. Changes to the implicit functions manifest themselves by deforming the boundary representation of the object. In this approach the object surface has an explicit boundary representation at all times. Such an object can be efficiently displayed. Implicit function techniques can be applied to the object by modifying the implicit function approximating the object surface. A shortcoming of this apporach is that the quality of the results are dependent on the accuracy with which the implicit function approximates the B-rep object. Further, transmitting topology changes of the implicit function to the B-rep object surface is non-trivial.

The second technique provides an analytic implicit function definition for an object modeled as a B-rep. As an example implicit function primitive definitions are specified for polyhedral objects. We provide ways by which local control over parts of the primitive is easy to specify. Tesselation of implicit surfaces makes it possible to go back and forth between implicit surface and B-rep representations for an object. This is an excellent technique for incrementally sculpting objects. As the object surface representation in this technique is no longer explicit it suffers from the display inefficiences of general implicit surfaces. Efficient display techniques for implicit surfaces are therefore, presented, especially those constructed with polyhedral implicit primitives.

The user can use either of the above two approaches to employ both B-rep and implicit function techniques. The choice depends on the degree of precision of modeling and animation, and the rendering efficiency desired.

7.1.3 Human Figure Model

We present a new human figure model, where implicit functions model bones, muscle and skin. The model fits into our implicit function based framework for the virtual world. The model is visually realistic and has physical and anatomical interpretations.

7.1.4 Wrinkle Synthesis and Animation

We also describe a technique for the synthesis of wrinkled textures. Wrinkles on skin and clothing are animated by texture morphing and wireframe displacement of the human figure in real-time.

7.1.5 Virtual Space Teleconferencing

Finally, the techniques developed above were integrated and illustrated by a real-time implementation of human figures in a virtual space teleconferencing system. Virtual space teleconferencing in itself is a novel concept. Its primary advantages over traditional video teleconferencing are enhanced interaction between participants, the capability to collaboratively manipulate the virtual environment and the greatly reduced bandwidth requirements.

7.1.6 Summary of Contributions

Briefly the salient features and contributions of this thesis are as follows:

1. A generalized framework for modeling and animating objects and external forces in a virtual world using implicit functions. The model is general, computationally efficient and can be viewed purely kinematically or with a physical interpretation.

2. A method for constraint satisfaction between deformable objects built on previous work in collision detection and deformation [20]. The constraints are satisfied as local deformations to the objects. The user has excellent control over the extent and nature of the deformations. Arbitrary constraint graphs are trivially supported.

3. A technique by which objects with a polyhedral definition can be embedded in a hierarchy of implicit functions. The implicit functions can then deform the polyhedral definition of the object.

4. Implicit surface primitive shape definitions for polyhedral objects. These definitions unify the two object representation methods.

5. Efficient display techniques for implicit surfaces, especially those constructed with polyhedral implicit primitives.

6. Efficient geometric computations for analytic primitive shapes like cone-spheres, which are useful for human figure animation.

7. A new human figure model, where implicit functions model bones, muscle and skin. The model fits into our implicit function based framework for the virtual world. It is the only model that currently handles self-collisions of the human figurely precisely.

8. A technique for the synthesis of wrinkled textures. Wrinkles on skin and clothing are animated by texture morphing and wireframe displacement of the human figure in real-time. The approach provides visually realistic results for complex garments and is the only technique that currently supports animation of tight-fitting clothing in real-time.

9. Finally, the techniques developed above are integrated and illustrated by a real-time implementation of human figures in a futuristic virtual space teleconferencing system.

7.2 Evaluation of results

The implementation of the presented concepts shows their effectiveness both in terms of computation speed and the degree of realism obtained. The separation of the physical characteristics of objects into rigid and deformable components, works particularly well for human figures. The model handles self-collisions of the body and skeletally based deformations elegantly. The ability to apply implicit function techniques in general to existing polygon based data is an important advantage of our approach. It can unify and be integrated with existing polygon based or implicit surface based modeling and animation systems.

The approach achieves linear time complexity in terms of number of object vertices for collision detection and handling, which is important when dealing with complex virtual worlds [51].

The synthesis of static wrinkled textures using our approach produce reasonably realistic results. This can be seen by comparing the photographed wrinkled textures in Figure 49a and their synthesized counterparts. As a result a number of cloth textures can be quickly synthesized with far better control over wrinkles and creases than can be achieved by actually manipulating the fabric. The simple synthesis of perfectly corresponding textures for different facial expressions as in Figure 50 obviates the manual labor of establishing a correspondence between these textures.

The results of color texture morphing coupled with wireframe displacement are shown to improve the realism of human figure models in a virtual space teleconferencing system. While the results are by no means physically accurate they do enhance

realism with minimal computational expense.

7.3 Future Research

This thesis leaves scope for furture work in a number of directions.

The implicit function based virtual world model, though well defined, is only a loose specification of how objects and forces in a virtual world interact. A collision handling and constraint satisfaction approach was built on this model. There is great scope for modeling and simulating physical phenomena using the implicit function model. Ghost functions also show great promise as sculpting tools for incremental object modeling. Intuitive shape transformations can easily be effected and controlled using ghost functions.

Work may also be done on improving the manner by which implicit functions fit and deform underlying polymeshes. The underconstrained nature of the deformable component mapping of polymesh vertices may cause surface consistency problems. A dense human figure polymesh and the radial nature of the limbs and primitives causes simple displacements along vertex normals to give good results without vertices bunching together or diverging abnormally. Incorporation of techniques such as that of Witkin [67], that adaptively subdivide and coalesce the polymesh in real time, are subject to current research. There is scope for future work on construction and fitting of primitives to the polymesh. Poor fitting primitives may result in very close objects being deformed to abut at the implicit model contact surface. For *VISTEL* and other applications where visual realism dominates over spatial accuracy (we assume

a tolerable inaccuracy in the transition from real to virtual space), the above artifact does not pose a problem. Using a greater number and more complex primitives improves the fit but degrades implicit function and bounding volume intersection computation efficiency. An empirical tradeoff between a better fit and computation efficiency should therefore, be taken into consideration.

Many techniques for improving the efficiency of rendering implicit surfaces in general and polyhedral implicit primitives in particular have been presented in this thesis. Future work in this area can generalize these techniques to higher order surface patch primitives. Tesselation of polyhedral primitives is a problem with great scope for parallelization.

Further work can be done to incorporate dynamic aspects [31] into our wrinkle model so as to be able to better control the texture morphing process by possibly incorporating intermediate synthesized wrinkle textures. The wrinkle model thus far has only been applied to fine facial detail. Further work needs to be done before the wrinkle model can be successfully applied to parts of the face where the deformations are large. The model can also be applied in the image processing domain to simulate aging on existing facial images.

The implicit function based human figure model presented does not address the layers of hair and clothing explicitly. In this thesis they are simply color textures that are mapped onto the skin surface. Modeling and animating realistic hair is still a largely unsolved problem and most approaches involve complex physical methods. Future research on the dynamic uses of textures to animate close cropped hair may be

profitable. While the wrinkle model addresses tight-fitting cloth animation well, loose-fitting garments such as skirts need an explicit geometric model. An integrated cloth animation system that involves aspects of our wrinkle model and other approaches [35][65] that handle loose-fitting garments well, is envisioned.

The overall goal of this thesis was to build a model for interactive virtual worlds in which realistic human figures as synthetic actors are an integral part. We have but taken a step in this direction.

APPENDIX A

Useful Implicit Primitives

Some comments concerning notation are in order. In the calculations that follow, a bar over a vector variable indicates normalization of the vector. For example, $\overline{X}$ is the unit vector in the direction of X. Upper case letters are used to denote vectors; lower case letters are used to denote scalars. The *dot product* operation is denoted by '$\cdot$'; the *cross product* operation is denoted by '$\otimes$'.

A.1 Spheres

Spheres are defined by a center point C and a radius r. The methods for computing the object functions which are dependent on the geometry of the primitive are given below.

A.1.1 Calculate Distance Ratio and Normal

Given the object information and a point P, the distance ratio is the ratio of the distance between the point and the center to the radius of the sphere. $distanceratio = \frac{|P-C|}{r}$.

Given the object information and a point P, the normal is the normalized vector N from the center of the sphere to the point: $N = \overline{P - C}$ (see Figure 58).

A.1.2 Calculate Y-Extent

Given the object information, the Y-Extent can be calculated by forming the highest and lowest planes whose normals are in the YZ plane and which pass through the origin. In order to find the plane, the YZ projection of the plane (a line) and the sphere (a circle) is used. Points $P1$ and $P2$ are found on the lines through the origin tangent to the circle such that the line joining $P1$, C and $P2$ is perpendicular to the line from the origin to C. The y coordinates of $P1, P2$ give the scanline extent when projected onto the picture plane (see Figure 58).

1. $V = \overline{< 1, 0, 0 > \otimes C}$
2. $k = \frac{r|C|}{sqrt(|C|^2 - r^2)}$
3. $P1 = C + kV, P2 = C - kV$

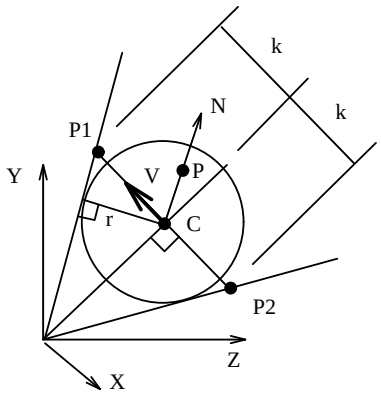


Figure 58: SPHERE: Normal, Y-Extent Calculation

A.1.3 Calculate X-Extent

For spheres, calculating the X-extent is similar to the Y-extent. A given scanline defines a plane in eye-space. The intersection of the plane, with unit normal N, and the sphere in eye-space is a circle of center C' and radius r' (see Figure 59).

1. $s = C \cdot N$
2. $C' = C - sN$
3. $r' = sqrt(r^2 - s^2)$
4. calculate X-extent using C', r'

A.1.4 Calculate Z-Extent

For each pixel of each scanline covered by the sphere, the points of intersection of the ray with the sphere, $I1$ and $I2$, can be computed, given a unit ray through the pixel, RAY. Calculation of a midpoint, $MIDP$, allows for breaking the extent into two monotonic depth spans (see Figure 59).

1. $d = C' \cdot RAY$
2. $MIDP = dRAY$
3. $middist = |C - MIDP|$

   (used in calculating monotonic z-spans)
4. $m = sqrt(r'^2 - |C' - MIDP|^2)$
5. $I1 = (d - m)RAY, I2 = (d + m)RAY$

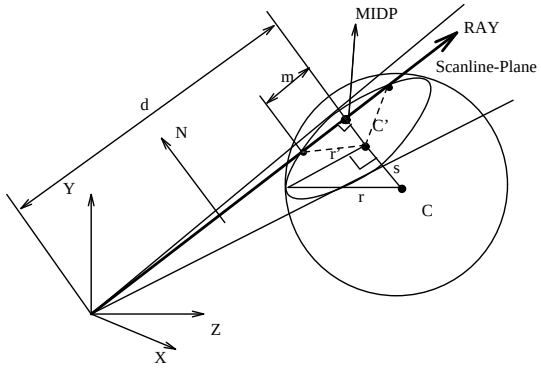


Figure 59: SPHERE: X-Extent, Z-Extent Calculation

A.2 Sphylinders

Sphylinders are finite extent cylinders with hemispheres capping both ends. They are useful for modeling a variety of shapes, especially longitudinal ones. A sphylinder is defined by two center points, $C1$ and $C2$, and a radius r (see Figure 60). The line segment between $C1$ and $C2$ will be referred to as the *center axis*.

A.2.1 Calculate Distance Ratio and Normal

Given the object information and a point P, the distance ratio is computed as in the case of the sphere, if the closest point on the center axis to P is one of the end points. If the closest point to P, call it P', is interior to the center axis, then the distance ratio is calculated as the ratio of the distance between P and P' to the radius of the sphylinder. The normal is the normalized vector N from P' to P (see Figure 60):

1. $P' = C1 + (P - C1) \cdot \overline{(C2 - C1)} * \overline{(C2 - C1)}$.

2. $distanceratio = \frac{|P-P'|}{r}$.

3. $N = \overline{P - P'}$.

A.2.2 Calculate Y-Extent

The Y-extent is calculated by computing the Y-extent of its end spheres and returning their combined Y-extent (see Figure 60).

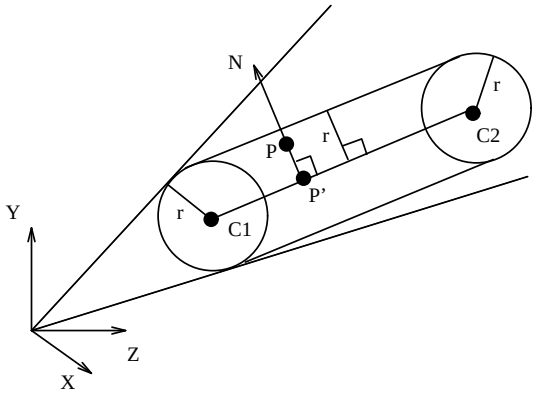


Figure 60: SPHYLINDER: Normal, Y-Extent Calculation

A.2.3 Calculate X-Extent

The X-extent, unlike the Y-extent, can either be delimited by the spheres or by the cylinder. The X-extents of the scanline-plane with the capping spheres are calculated. If the plane does not intersect the spheres, the X-extent of the scanline-plane (with normal N) and the infinite cylinder is computed as follows. Note that in *eye-space* a scanline-plane passes through the origin. It is assumed that $C1$ is below the scanline-plane with no loss of generality.

1. $Y' = \overline{N \otimes (C2 - C1)}$

2. $X' = \overline{Y' \otimes N}$

3. $majoraxis = \frac{r}{|\overline{(C2-C1)} \cdot N|}$

4. $minoraxis = r$

5. $C' = C1 + \frac{-N \cdot C1}{N \cdot (C2-C1)} * (C2 - C1)$ (see Figure 61)

6. The intersection ellipse is defined by center, C', major and minor axes of symmetry, X' and Y', and their magnitudes,$majoraxis$ and $minoraxis$. Transform the origin to the local frame defined by axes of symmetry of the intersection ellipse centered at C' and solve the quadratic equation for the tangent points to the ellipse. The tangent points transformed back form the X-extent with the cylinder (see Figure 62).

A.2.4 Calculate Z-Extent

As is the case with the sphere, the Z-extents of the sphylinder can be broken into monotonic segments. The closest point on the ray to the sphylinder's defining line is determined and the Z-extent is broken into the two segments at that point. The ray need only be intersected with the components of the sphylinder that define the surface of intersection in the scanline-plane. Thus the following check on X-extent definition can be made (see Figure 63):

- Single defining component: Intersect ray with that component.

- Sphere and cylinder definitions: Intersect ray with that sphere and the cylinder and take the extremes.

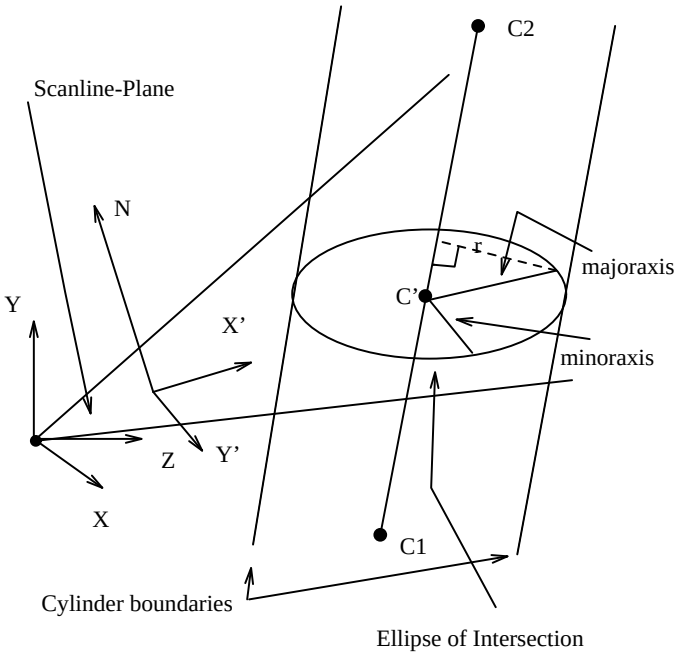


Figure 61: SPHYLINDER: X-Extent Calculation

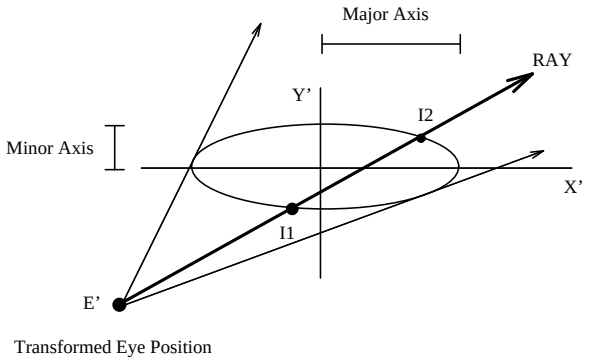


Figure 62: 2D Ellipse Tangent, Ray Intersection Calculation

- Spheres' definitions: Intersect ray with both spheres and the cylinder and take the extremes.

The ray infinite cylinder intersection can be computed efficiently by using the 2D intersection ellipse information above to transform the ray and calculate the 2D ray ellipse intersection. A special case where the scanline plane is parallel to the sphylinder axis is handled as follows.

1. $d = C1 \cdot Y'$.
2. $d' = \sqrt{r^2 - (C1 \cdot N)^2}$.
3. $\frac{(d \pm d')}{RAY \cdot Y'}$ are distances of intersection along RAY.

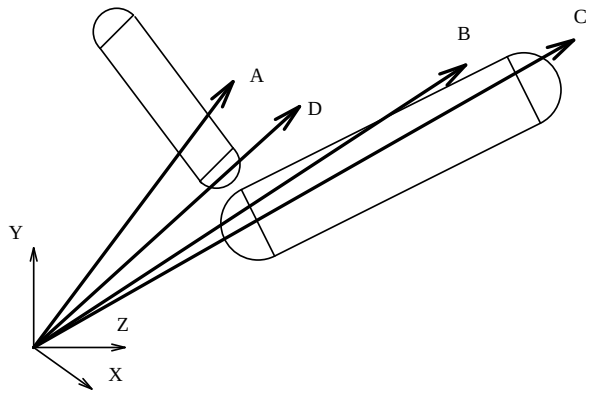


Figure 63: Ray Intersections: A) Cylinder-Cylinder B) Sphere-Cylinder C,D) Sphere-Sphere

A.3 Cone-Sphere

Cone-Spheres are a generalization of sphylinders in which the spherical segments on either end have unequal radii and the connecting surface is a truncated cone instead of a cylinder; interior radii are linear interpolations of the end radii. The cone is defined so that the surface has first order continuity at the junction between the cone and spherical segments. This means that the spherical segments at either end are only hemispheres in the case that degenerates to a sphylinder. Cone-spheres are also useful for modeling a variety of shapes, especially articulated animate figures.

Cone-Spheres are defined by two center points, $C1$ and $C2$, and two corresponding radii, $r1$ and $r2$. Using the above information, the following is computed (see Figure 64):

- $\theta = cos^{-1}(\frac{r2-r1}{|C2-C1|})$
- $C21 = \overline{(C2 - C1)}$
- $C0 = C1 - C21 * \frac{r1}{cos\theta}$

A.3.1 Calculate Distance Ratio and Normal

Given the object information and a point P, the distance ratio, *dist-ratio*, and normal, N, computation is as follows (see Figure 64):

1. $P' = P - C0$
2. $t' = (\sqrt{|P'|^2 - (P' \cdot C21)^2})cotan\theta$

3. $t = P' \cdot C21 + t'$

4. $N = \overline{P' - t * C21}$

5. *dist-ratio*= $\frac{t' * sec^2\theta}{t}$

A.3.2 Calculate Y-Extent

The Y-extent, as in the case of the sphylinder, is the combined extent of the end spheres.

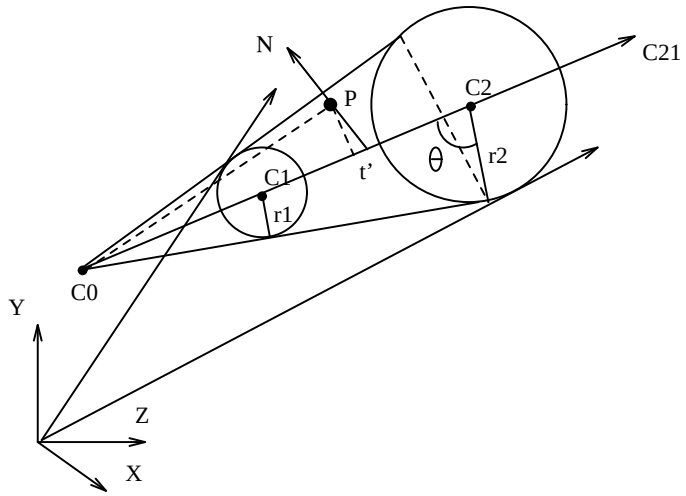


Figure 64: CONE-SPHERE: Normal, Y-Extent Calculation

A.3.3 Calculate X-Extent

The X-extent can either be delimited by the spheres or by the cone. The X-extents of the scanline-plane with the capping spheres are calculated. If the plane does not intersect the spheres the X-extent of the scanline-plane with the semi-infinite cone is computed as follows:

1. $\phi = cos^{-1}(X' \cdot C21)$ (see Figure 65)

2. The conic of intersection is determined as follows:

   - Ellipse : $\phi > 90^o - \theta$
   - Hyperbola : $0 < \phi <= 90^o - \theta$
   - Parabola : $\phi = 0^o$

3. Axes of symmetry ,X' and Y', and, in the case of ellipse and hyperbola, intersection point, C', are calculated as in the case of the sphylinder.

The 3 cases are now treated individually for clarity, though some of the computation may overlap. The computations are similar to those used for sphylinders.

- **ELLIPSE**
  - $t = |C' - C0| * cos\theta$
  - $distC'P1 = \frac{t}{cos(\theta - \phi)}$
  - $distC'P2 = \frac{-t}{cos(\phi + \theta)}$
  - $P1 = C' - distC'P1 * X'$
  - $P2 = C' + distC'P2 * X'$
  - $MIDP = \frac{P1+P2}{2}$
  - $majoraxis = |MIDP - P1|$
  - $t' = (MIDP - C') \cdot C21$
  - $MIDP' = C' + t' * C21$

- $rMIDP' = |MIDP' - C0| * cotan\theta$
- $d = sqrt(|MIDP - C'|^2 - t'^2)$
- $minoraxis = sqrt(rMIDP'^2 - d^2)$

Solve for X-extent of the ellipse of intersection defined by the axes of symmetry centered at $MIDP'$ as in the case of the sphylinder.

- **HYPERBOLA**
  - Calculate P1 as for ellipse
  - The general equation of a hyperbola symmetric about positive X-axis, passing through the origin is :

    $y^2 = a^2x^2 - 2abx$. The defining constants a, b are obtained by computing points $(1, y1)$ and $(2, y2)$ on the hyperbola and solving the linear equations to get

    $a^2 = \frac{y2^2 - 2*y1^2}{2}$

    $ab = \frac{y2^2 - 4*y1^2}{4}$

  - $y1^2, y2^2$ are computed as follows (see Figure 65):
    - $Q = P1 + 1 * X'$
    - $Q' = C0 + ((Q - C0) \cdot C21)C21$
    - $rQ' = |Q' - C0|cotan\theta$
    - $y1^2 = rQ'^2 - |Q - Q'|^2$

  - $y2^2$ is calculated similarly by a distance of 2 along X'

- We thus obtain the definition of the intersection hyperbola with axes X', Y', constants a^2, ab and local origin $P1$.

- Tangent calculations after transforming the origin to local space results in a quadratic equation. One of the solutions of this equation will lie within the bounds of the truncated cone (see Figure 66).

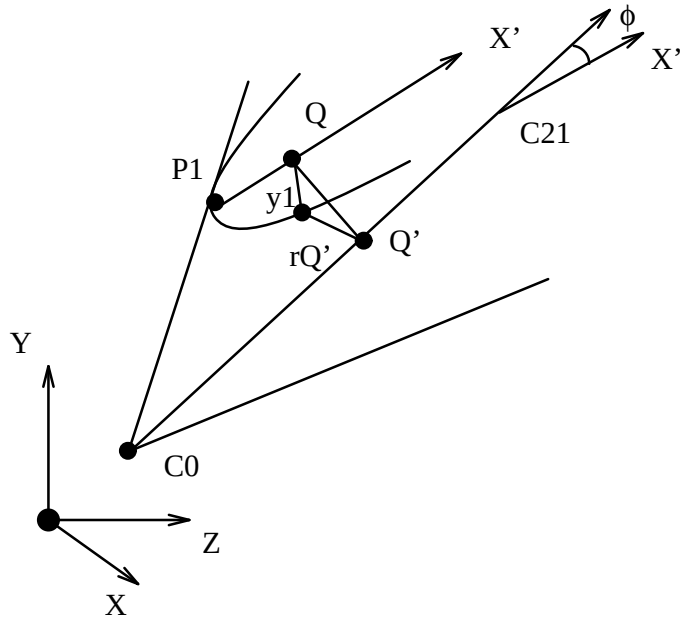


Figure 65: Hyperbolic Intersection of Cone/Plane

- **PARABOLA**

  - $d = (N \cdot C0)tan\theta$

  - $P1 = C0 + d * C21 + (N \cdot C0)N$

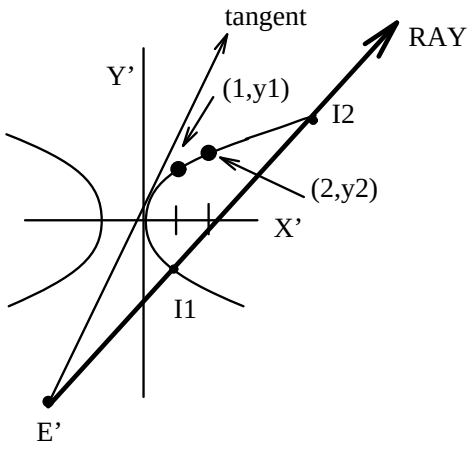


Figure 66: 2D Hyperbola Tangent, Ray Intersection Calculation

- The general equation of a parabola symmetric about positive X-axis, passing through the origin is :

  $y^2 = a^2x$. The defining constant a is obtained by computing point $(1, y1)$ on the parabola giving $a^2 = y1^2$.

- $y1^2 = ((d+1) * cotan\theta)^2 - (N \cdot C0)^2$

- We thus obtain the definition of the intersection parabola with axes X', Y', constant a^2 and local origin $P1$.

- Tangent calculations after transforming the origin to local space results in a quadratic equation of whose solutions one will lie within the bounds of the truncated cone.

A.3.4 Calculate Z-Extent

Z-Extent calculation is similar to that of the sphylinder. The ray intersection for the infinite cylinder component (accomplished by a 2D ray, intersection ellipse, intersection), however, must be replaced for the infinite cone component by 2D intersection with the appropriate conic of intersection the definition of which is made once for the scan-line, while computing X-Extents.

A.4 Rounded Polygons

Rounded Polygons are defined by a list of vertices $C_1, .., C_n$ defining a planar polygon and a radius r (see Figure 67). For the most part all the computation is done by treating the rounded polygon as n sphylinders with consecutive vertices forming the two centers and a radius r, within appropriate bounds. Care should be taken not to repeat the computation for the region common to two adjacent sphylinders.

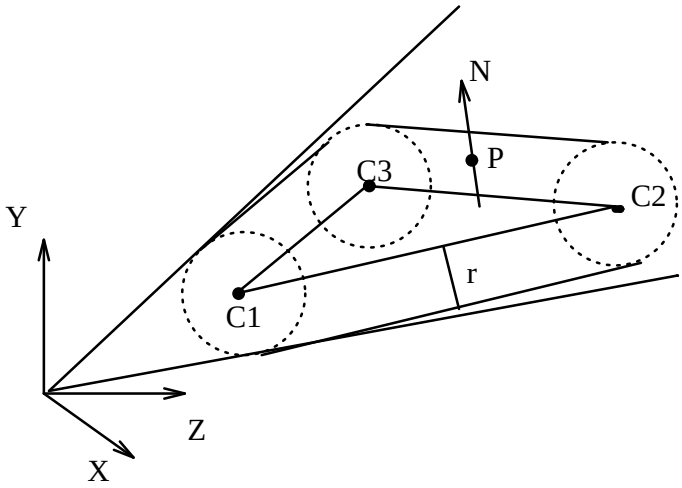


Figure 67: ROUNDED POLYGON: Normal, Y-Extent Calculation

Further computation is required for the case when the surface normal and Z spans are specified by the polygon plane. Let N be the normal to the plane. Given a point P for which the plane defines the normal and distance ratio, the normal is N and *dist-ratio* $= \frac{(P-C_1)\cdot N}{r}$. For a RAY whose Z spans are defined by the plane the distances of intersection along RAY are $\frac{(C_1\cdot N)\pm r}{RAY\cdot N}$.

APPENDIX B

Shape transformation using Polyhedral implicit primitives

A simple way of effecting shape change between implicit primitives in general is to interpolate the shape weights of two or more appropriately superposed implicit primitives. This has been shown for existing analytically defined primitives. The same technique gives good results with polyhedral primitives (See Figure 69).

The definition of a polyblob also provides a measure of shape change. Surfaces that are generated by a threshold value varying from 1 to 0 display a shape transformation from S to V for the primitive. Thus we can effect a transformation between two polymeshes by making one S and the other V for a polyblob, positioning them appropriately and interpolating the threshold value (See Figure 68,70).

Because the interior polymesh can be arbitrary in edge-vertex topology as well as conventional surface topology, the polyhedral-based implicit surface provides a way to interpolate between a star-shaped polyhedron (that is, star-shaped relative to the internal polymesh) and a polyhedron with arbitrary edge-vertex connectivity, arbitrary genus and even an arbitrary number of component polyhedrons. This is illustrated in two dimensions in Figure 68, with a square transforming into 3 circles.

Most polygon-based methods for shape-change cannot handle objects of arbitrary genus [29],[48]. Approaches which in some way utilize the volume of the objects, such as that being proposed here and PIPs [27], tend to be able to handle a wider variety of objects elegantly.

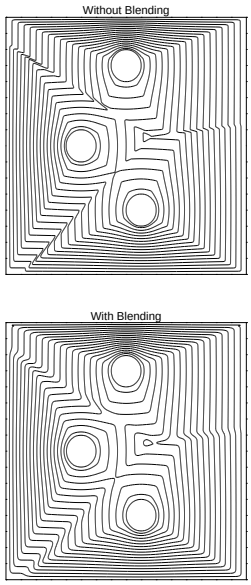


Figure 68: Shape Transformation by threshold interpolation

Some observations regarding these approaches are the following:

- The approach using a single polyblob requires that one polymesh S, lies entirely within the other, V. This would appear to impose a relative size restriction on the polyhedra. However, given two polyhedra of roughly the same size, we can scale one to lie entirely within the other. Then for varying threshold values the composite primitive can be scaled appropriately so that the threshold extremes

result in the original size of the polyhedra. Intermediate surfaces will be at scale values between the two extremes. Suppose S is scaled by a factor of sc to lie within V. Scaling the composed implicit primitive by $(\frac{1}{sc} - 1) * threshold + 1$ achieves the desired effect.

- The shape transformation generated by replacing the roles of S and V between two polymeshes, do not in general yield the same results. This can be viewed as a shortcoming or a bonus, in that two different transformations may be chosen from. Note, however, that instances of the same object yields an identity transform.

- Sharp edges resulting from functional discontinuities at Voronoi boundaries for polyblobs may be softened if desired using the same blend solution presented for point-setblobs. This is shown in Figure 68.

- The shape transformation is observed to have an ease in ease out nature. This is due to the shape of the density function used. Control over the interpolation may be achieved by modifying the density function or better still by spline interpolation of the shape weight or threshold value, for the two approaches respectively.

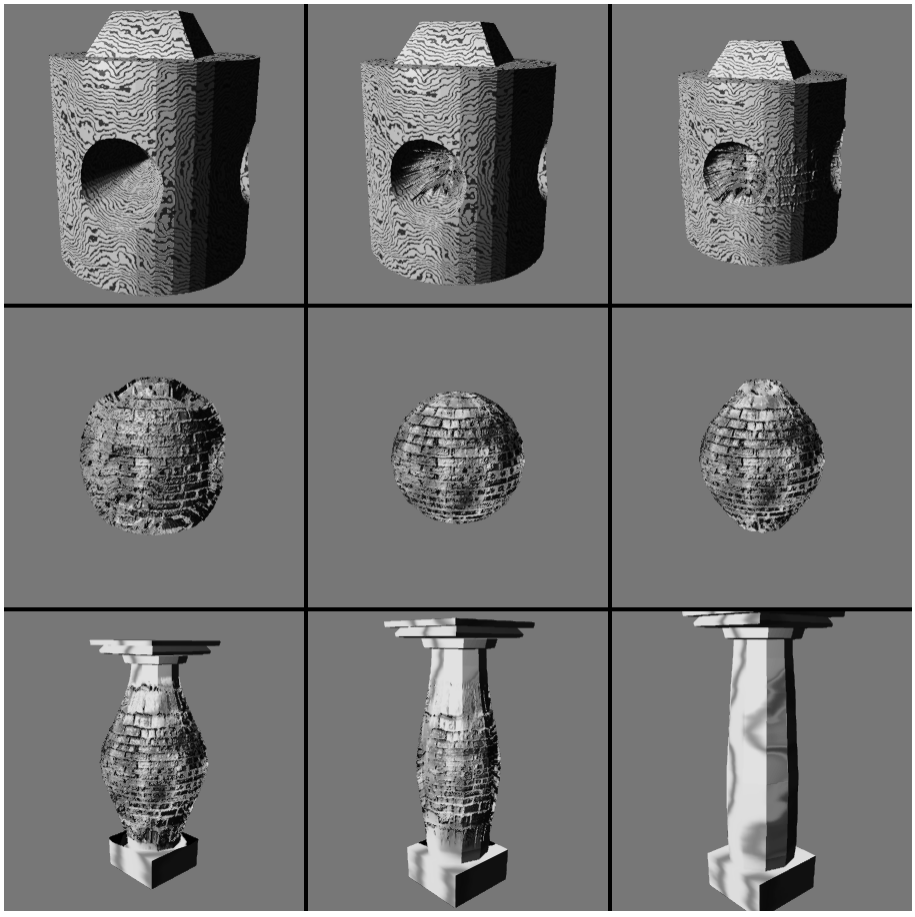


Figure 69: Shape Change by shape weight interpolation

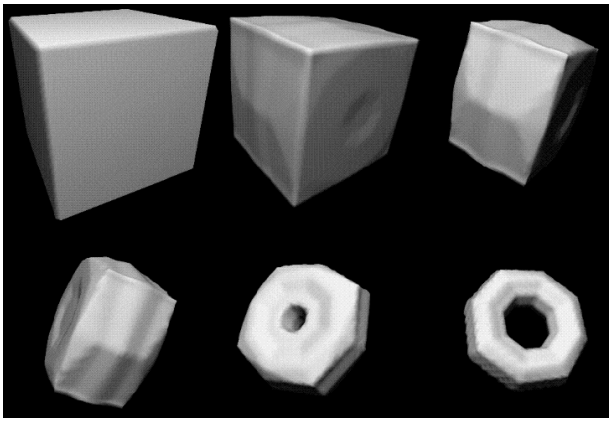


Figure 70: Shape Change by threshold value interpolation

APPENDIX C

Example Ghost functions

C.1 Modeling the relative rigidity of objects

In Gascuel's formulation both colliding objects are considered equally flexible. The special case of collision of a flexible body with a rigid one is also discussed [20]. For a flexible object j colliding with a rigid object i this is accomplished by imparting j with a ghost function $G_i(P) = 2T - F_i(P) - F_j(P)$ in the interpenetration zone. A similar function is used for $prop_i$. The gradient property in Figure 4 must be changed to $\nabla PROP_i(P) = -\nabla(F_i(P) + F_j(P))$, so as to maintain first order continuity of the implicit surface of object j across the interpenetration-propagation zonal boundary.

For equally flexible objects the gradient constraint is $\nabla PROP_i(P) = -\nabla F_i(P)$, which translates easily to the single variable functions as $prop_i'(t) = -f_i'(t)$, where $f_i(t) = T$. The case with a rigid object does not follow in the same fashion due to the presence of $F_j(P)$)

We first extend these two cases to model the continuum of rigidity of colliding objects, relative to each other.

Let $w \in [0, 1]$ be the rigidity of object i with respect to object j. The value $w = 0$ corresponds to a rigid object j and $w = 1$ to a rigid object i. $w = 0.5$ is the case of

equally flexible objects. Without loss of generality let $w \in [0, 0.5]$ as the roles of i and j may be interchanged. The functions in the interpenetration zone are:

- $G_i(P) = 2w(T - F_i(P))$
- $G_j(P) = (T - F_j(P)) + (1 - 2w) * (T - f_i(P))$

The ghost functions above reduce to those presented in Gascuel for the special cases $w = 0$ and $w = 0.5$. We now address the functions for the propagation zone. $g_i = 2w * prop_i$ in the propagation zone where $prop_i$ has the properties in Figure 4. For G_j we would like to satisfy $\nabla PROP_j(P) = -\nabla(F_j(P) + (1 - 2w) * F_i(P))$. This can be simplified for distance surfaces to $\frac{prop'_j(t)\hat{N}_j(P)}{r_j} = \frac{-f'_j(t)\hat{N}_j(P)}{r_j} + \frac{(1-2w)f'_i(t)\hat{N}_i(P)}{r_i}$[9]. Simplified $prop'_j(t) = -f'_j(t) + (1 - 2w)f'_i(t)\frac{r_j}{r_i}(\hat{N}_i(P).\hat{N}_j(P))$

The angle between the two normal vectors depends on the surface geometry of the objects and the extent of interpenetration of the colliding objects th (See Figure 71). For the configuration shown in Figure 71, $\hat{N}_i(P).\hat{N}_j(P) = \frac{r_i^2+r_j^2-(r_i+r_j-th)^2}{r_i r_j}$.

C.2 Temporal elastic effects

We make use of the temporal parameter to modify the collision deformation ghost function imparted [20] to model viscoelastic and plastic objects in Figure 14c. While the objects collide the ghost function is as described in [20]. The ghost function remains static at the point of maximum penetration for a plastic object. Viscoelasticity can be modeled by a ghost function that lags behind its imparting object on

[9] $\hat{N}_i(P)$ denotes the normalized normal vector to the isosurface of implicit function F_i on which point P lies.

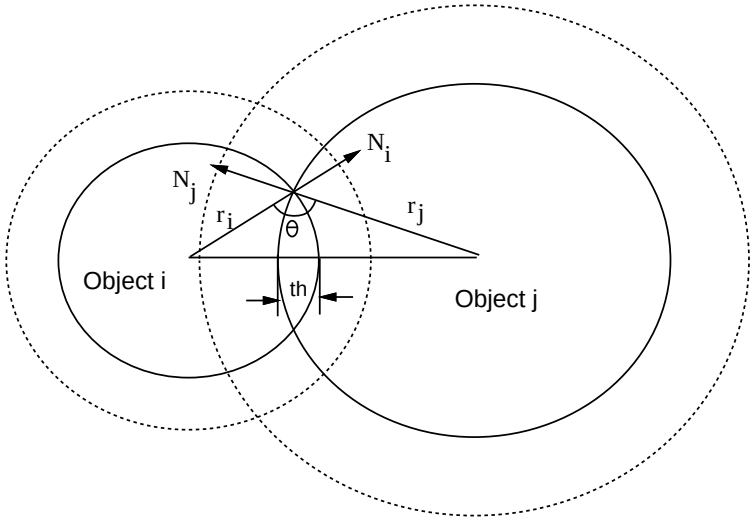


Figure 71: Computing the gradient value for the $PROP$ function

separation. Further multiplication by various decay functions of time can simulate arbitrary stress-strain characteristics of deformable objects.

C.3 Directional Force Fields

Environmental forces such as gravity, winds, shearing and compressive forces maybe visually simulated by constructing ghost functions for deformable objects. Directional forces like winds can be modeled as a ghost function $G(P) = F(P) * f((P \cdot \overline{N}) - d)$, where f is a polynomial function and $\overline{N}, d$ define a plane perpendicular to the direction of the force passing through the deformable object. $F(P)$ is the implicit function for the object. As an example, for $f(u) = c * u$, where c is a constant, the function causes the object to deform inward on one side and outward on the other side of the plane. Multiplying f by F takes care of continuity properties and the object boundary. The result is an empirical deformation in the direction of the force. This can be seen in

Figure 72 where the solid lines indicate the implicit objects. The realm of influence of the spherical implicit function is shown dotted in Figure 72a and the dashed lines show an implicit surface of the spherical object at a threshold value greater than that of the object. In Figure 72a a deformable sphere is perfectly juxtaposed to a rigid corner. In Figure 72b the sphere is deformed based on two ghost functions simulating wind (against the wall) and gravity (acting downward), causing the object to penetrate the floor and wall. Collision ghost functions imparted by the floor and the wall are now added to the sphere to get the result shown in Figure 72c.

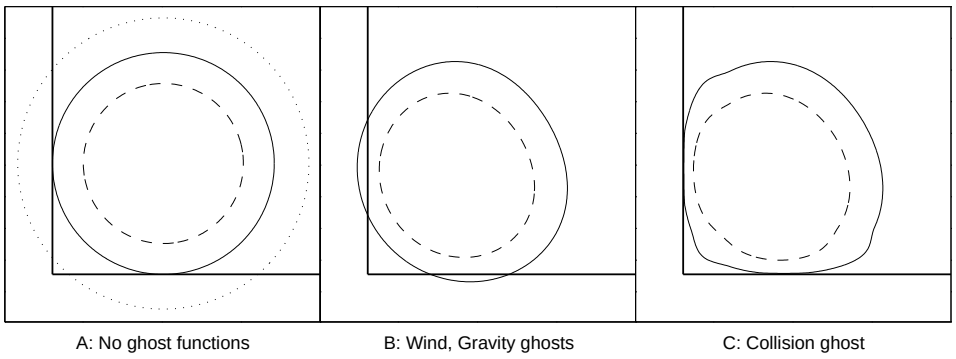


Figure 72: Ghost Function Deformations

Compressive forces acting inward on a deformable object can be modeled as a ghost function as above, defining $G(P) = -F(P) * f(|P - P'|/r)$, where f is a density function as in Figure 1 and P', r define an attractor and region of influence of the force. Expansive forces, like gas pressure in a balloon can be modeled as the negative of the above compressive force.

Bibliography

[1] M. Aono. A wrinkle propagation model for cloth. *Proc. Computer Graphics International*, 96–115, 1990.

[2] W. Armstrong and M. Green. The dynamics of articulated rigid bodies for purposes of animation. *Graphics Interface*, 407–415, 1985.

[3] D. Baraff. Dynamic simulation of non-penetrating rigid bodies. *PhD Thesis, Cornell University*, May 1992.

[4] A. Barr. Superquadrics and angle-preserving transformations. *IEEE Computer Graphics and Applications*, 1:1–20, 1981.

[5] R. Barzel and A. Barr. A modeling system based on dynamic constraints. *Computer Graphics*, 22:179–188, 1988.

[6] T. Beier and S. Neely. Feature-Based Image Metamorphosis. *Computer Graphics*, 26:2 , July 1992.

[7] J. Blinn. Simulation of wrinkled surfaces. *Proc. SIGGRAPH'78*, 286–292, 1978.

[8] J. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):235–256, 1982.

[9] J. Bloomenthal. Polygonization of implicit surfaces. *Computer Aided Geometric Design*, 5:341–355, 1988.

[10] J. Bloomenthal and B. Wyvill. Interactive techniques for implicit modeling. *Computer Graphics*, 24(4):109–116, 1990.

[11] J. Bloomenthal and K. Shoemake. Convolution surfaces. *Computer Graphics*, 25(4):251–256, 1991.

[12] J. Bloomenthal. Hand Crafted. *SIGGRAPH Implicit Surface Course Notes*, 11.1–11.3, 1993

[13] P. Buttolo and B. Hannaford. Pen based force display for precision manipulation in virtual environments. *IEEE VRAIS*, 217–224, 1995.

[14] J. Chadwick, D. Haumann and R. Parent. Layered construction for deformable animated characters. *Computer Graphics*, 23(3):234–243, 1989.

[15] D. Chen and D. Zeltzer. Pump it up: Computer animation of a biomechanically based model of muscle using the finite element method. *Computer Graphics*, 26:89–98, 1992.

[16] P. Ekman and W. Friesen Facial Action Coding System. *Consulting Psychologists Press*, Palo Alto , 1978.

[17] R. Featherstone The calculation of robot motion using articulated-body inertias. *International Journal of Robotics Research*, 2:13–30.

[18] A. Fujimoto T. Tanaka and K. Iwata. Arts: Accelerated ray tracing system. *Computer Graphics and Applications*, 6(4):16–26, 1986.

[19] J. Gascuel and M. Gascuel. Displacement constraints for interactive modeling and animation of articulated structures. *Visual Computer*, 10:191–204, 1994.

[20] M. Gascuel. An implicit formulation for precise contact modeling between flexible solids. *Proc. of SIGGRAPH*, pages 313–320, 1993.

[21] J. Granieri, J. Crabtree and N. Badler. Production and playback of human figure motion for 3D virtual environments. *IEEE VRAIS*, 127–135, 1995.

[22] J. Hahn. Realistic animation of rigid bodies. *Computer Graphics*, 22(4):299–308, 1988.

[23] P. Hanrahan. Ray tracing algebraic surfaces. *Computer Graphics*, 17(3):83–90, 1983.

[24] D. Haumann. Modeling the physical behavior of flexible objects. *SIGGRAPH*, Course Notes vol 17, 1987.

[25] P. Isaacs and U. Cohen. Mixed method for complex kinematics constraints in dynamic figure animation. *Visual Computer*, 2:296–305, 1988.

[26] D. Kalra and A. Barr. Guaranteed ray intersections with implicit surfaces. *Computer Graphics*, 23(3):297–306, 1989.

[27] A. Kaul and J. Rossignac. Solid-interpolating deformations: Construction and Animation of PIPs. *Technical Report 77458, IBM, 1992.*

[28] T. Kay and J. Kajiya. Ray Tracing Complex Scenes. *Proc. of SIGGRAPH*, 269–278, 1986.

[29] J. Kent, W. Carlson and R. Parent. Shape Transformation for Polyhedral Objects. *Proc. of SIGGRAPH*, 313–320, 1993.

[30] K. Komatsu. Human skin model capable of natural shape variation. *Visual Computer*, 3:265–271, 1988.

[31] T. Kunii and H. Gotoda. Modleing and animation of garment wrinkle formation process. *Proc. Computer Graphics International*, 131–147, 1990.

[32] B. Lafleur, N. Magnetat-Thalmann and D. Thalmann. Cloth animation with self-collision detection. *Modeling in Computer Graphics*, Springer Verlag, Tokyo, 1991.

[33] R. Lathrop. Constrined robot simulation by local constraint propagation. *IEEE Int. Conf. on Robotics and Automation*, 689–694, 1986.

[34] W. Lorenson and H. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *Computer Graphics*, 21(4):163–169, 1987.

[35] N. Magnetat-Thalmann and Y. Yang. Techniques for cloth animation. *SIGGRAPH Course Notes C20*, 151–163, 1991.

[36] N. Magnetat-Thalmann, D. Thalmann. Complex models for visualising humans, *SIGGRAPH Course Notes C20*, 3–10, 1991.

[37] N. Magnetat-Thalmann, D. Thalmann. Human body deformations using Joint Dependent Local Operators and Finite Element Theory. *Making Them Move*, Morgan Kaufmann, 243–262.

[38] N. Max. Cone-spheres. *Computer Graphics*, 24(4):59–62, 1990.

[39] A. Middleditch and K. Sears Blend surfaces for set-theoretic volume modeling systems. *Computer Graphics*, 19(3):161–170, 1985.

[40] M. Moore and J. Wilhelms. Collision detection and response for computer animation. *Computer Graphics*, 22(4):289–298, 1988.

[41] S. Muraki. Volumetric shape description of range data using the blobby model. *Computer Graphics*, 23(3):234–243, 1991.

[42] K. Nariyama, K. Singh, J. Ohya and F. Kishino Realistic 3D synthesis of human body movements for virtual space teleconferencing. *IASTED International Conference on modeling and simulation*, Pittsburg, 1995.

[43] P. Ning and J. Bloomenthal. An evaluation of implicit surface tilers. *IEEE Computer Graphics and Applications*, 13(6):33–41, 1993.

[44] J. Ohya, Y. Kitamura, H. Takemura, F. Kishino and N. Terashima. Real-time reproduction of 3D human images in Virtual Space Teleconferencing. *VRAIS*, 408–414, 1993.

[45] H. Okabe, H. Imaokaa, T. Tomiha and H. Niwaya. 3D Apparel CAD system *Computer Graphics*, 26(2):105–110, 1992.

[46] A. Opalach and S. Maddock. High level control of implicit surfaces for character animation. *Eurographics workshop on Implicit Surfaces*, Grenoble, France, 223–232, 1995.

[47] A. Paouri, N. Magnetat-Thalmann and D. Thalmann. Creating realistic 3D human shape characters for computer generated films. *SIGGRAPH Course Notes C20*, 18–29, 1991.

[48] R. Parent. Shape transformation by boundary representation interpolation: a recursive approach to establishing face correspondences. *Visualization and Computer Animation*, 3:219–230, 1982.

[49] F. Parke Parameterized Models for facial animation. *IEEE Computer Graphics and Applications*, 2:9, 61–68, 1982.

[50] A. Pentland and J. Williams. Good vibrations: Modal dynamics for graphics and animation. *Computer Graphics*, 23(3):215–222, 1989.

[51] A. Pentland. Computational complexity versus virtual worlds. *Computer Graphics*, 24(2):185–192, 1990.

[52] S. Platt and N. Badler Animating Facial Expressions *Proc. Siggraph'81*, 245–252, 1981.

[53] A. Ricci. A constructive geometry for computer graphics. *Computer Journal*, 16(2):157–169, 1973.

[54] M. Schmidt. Cutting cubes- vizualizing implicit surfaces by adaptive polygonization. *Visual Computer*, 10:101–115, 1993.

[55] S. Sclaroff and A. Pentland. Generalized implicit functions for computer graphics. *Computer Graphics*, 25(4):247–250, 1991.

[56] T. Sederberg and S. Parry. Free-form deformation of solid geometric models. *Computer Graphics*, 20:151–160, 1986.

[57] K. Singh and R. Parent. Implicit function based deformations of polyhedral objects. *Eurographics workshop on Implicit Surfaces*, Grenoble, France, 113–128, 1995.

[58] K. Singh, J. Ohya and R. Parent. Human figure synthesis and animation for virtual space teleconferencing. *IEEE VRAIS*, 118–126, 1995.

[59] D. Terzopoulos and K. Fleischer. Deformable Models. *Visual Computer*, 4:306–331, 1988.

[60] D. Terzopoulos and K. Waters. Physically based facial modeling, analysis and animation. *Journal of Visualization and Computer Animation*, Vol.1, No.2, 73–79, 1990

[61] D. Terzopoulos and D. Metaxas. Dynamic 3D models with local and global deformations: deformable superquadrics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*,13(7):703–714, 1991.

[62] D. Terzopoulos and A. Witkin. Physically based modeling with rigid and deformable components. *IEEE Computer Graphics and Applications*,41–51, December 1988.

[63] C. Van Overveld. An iterative approach to dynamic simulation of 3-D rigid body motions for real-time interactive computer animation. *Visual Computer*, 7:29–38, 1991.

[64] K. Waters and D. Terzopoulos Analysis and Synthesis of Facial Image Sequences Using Physical and Anatomical Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:6, June 1993.

[65] J. Weil. The Synthesis of cloth objects. *Computer Graphics*, 20(4):49–54, 1986.

[66] J. Wilhelms and B. Barsky. Using dynamic analysis to animate articulated bodies as humans and robots. *Graphics Interface*, 97–104, 1985.

[67] A. Witkin and P. Heckbert. Using particles to sample and control implicit surfaces. *Proc. of SIGGRAPH*, 269–278, 1994.

[68] G. Wyvill C. McPheeters and B. Wyvill. Data structures for soft objects. *Visual Computer*, 2:227–234, 1986.

[69] G. Wyvill C. McPheeters and B. Wyvill. Animating soft objects. *Visual Computer*, 2:235–242, 1986.

[70] G. Wyvill and A. Trotman. Ray tracing soft objects. *Proc. of Computer Graphics International*, 1990.

[71] B. Wyvill, H. Zhang and A. Davis. Parallel implicit surface polygonization on a simulated parallel architecture. *Proc. of the Western Computer Graphics Symposium*, 35–46, 1992.