# Feature–Based Implicit Function Shaping with Isocontour Embedded Curves

Patrick Coleman

December 19, 2005

**Abstract**

Implicit functions are used for many applications, including shape modeling, shape editing, and character rigging. However, techniques for shaping implicit functions are primarily restricted to applying mathematical operations to primitive shapes. This report describes an investigation into an alternate methodology for shaping implicit functions. Users embed features into the implicit function domain, which alter the shape of the implicit function such that it has spatial features orthogonal to the function's local gradient. To facilitate this embedding, features are embedded into extracted isosurfaces. Such features can potentially take the form of points, curves, and surfaces. In this report, we focus on the design of curve based features. Implicit surface modeling is used as a demonstration application for visualization purposes, although the technique is amenable to other implicit function applications.

## 1 Introduction

Feature–based modeling and editing has become an increasingly accepted methodology for developing shape and form in computer graphics applications. Users have historically exploited the underlying mathematical structure of surface models to impart features (such as groups of control points)that they can manipulate and animate as a whole. More recently, technical approaches have been developed that allow users to specify a feature, which is then algorithmically selected and manipulated based on more of a conceptual modeling framework.

Feature based structure is also imparted for applications such as character rigging and shading. This is typically done by hand or with special–case

procedural methods by working with primitive functions and mathematical combination operations. More recently, as implicit functions have been adopted for physical simulation [8], user controllable function primitives have been used to control and animate the simulation parameters in time [15].

Current methodologies for specifying feature are based on selecting a set of function primitives and applying mathematical operations such as *union*, *min*, and *max* to achieve desired function shapes. This is analogous to CSG methods in shape modeling. As this approach to shape modeling has not proven popular, we believe more direct approaches to function manipulation might provide users with more expressive control.

This report presents an approach that takes a given implicit function representation, along with feature primitives embedded in isosurfaces, to produce a new implicit function. We build our formulation around implicit offset functions defined relative a skeleton shape [3], as they can be shaped using any underlying skeleton. Given such a function and the user–provided features, the new implicit function is created using the skeleton of the original function (known as the core), and evaluation is defined in terms of projections of points to the core, relative to the feature. From a conceptual standpoint, the function's core is extended into space along the gradient at the feature point.

## 2 Related Work

Implicit functions have a long history of development in computer graphics [2, 12, 18]. Originally presented as a surface modeling methodology [3], they have gained greater acceptance as methods for shape editing and deformation, including application to creating parameterized surfaces for character animation. As arbitrary surfaces can be converted to an implicit representation [17], implicit function–based editing can provide a powerful technique for editing shape.

In character rigging, surfaces are typically parameterized by transformations and deformations with local shape control [13]. This local shape control is often expressed as implicit weight functions that attenuate the deformation as distance to some central region increases [14]. As an example, Singh and Fiume presented an approach to shape modeling and deformation in which the underlying curve form of the implicit function defined the deformation and attenuation [16]. Such implicitly–defined attenuation functions have found use in other applications, including styled projection [5].

Explicit implicit function shaping is done by positioning multiple implicit

function primitives and applying mathematical operations such as *min*, *max*, *union*, and *intersection* [3]. These operations have various tradeoffs, however, especially in regard to function continuity. The approach presented in this report, in contrast, constructs entirely new functions from the given implicit function and a set of features.

Feature–based techniques have become increasingly predominant for modeling applications. Parametric surfaces can be manipulated with points and curves corresponding to control points and rows of control points. Extra knots can be inserted to provide controls at desired locations [7]. Feature based techniques are also becoming popular for mesh editing. Igarashi and colleagues incorporate user–sketched curves representing desired feature profiles to construct and edit meshes [10]. More recently, sketched curves have been used to re–mesh and edit meshes to create desired surface features [11]. Subdivision surfaces can be defined in terms of adaptive rules that allow users to embed curve based features [9, 6].

# 3 Implicit Function Construction from Isocontour Features

To add features to implicit functions, users position points, curves, or surfaces on isocontours of an initial implicit function, referred to as the core function. This core function is represented as an offset implicit function; such functions can be reconstructed from arbitrary surface representations [17]. For purposes of this report, we use curve based features. We reconstruct an isosurface of the core function using a tracking algorithm [3], and users position curves on this reconstructed surface using a modified version of the cords algorithm (modifications are described in Appendix B) [4].

## 3.1 Offset Implicit Functions

Offset implicit functions are defined using a skeleton shape, which can be a point, curve, or surface (Figure 1). Within the skeletal region, the function is defined to have a value of one. Outside the skeleton, the function is defined relative to the distance from the surface. Beyond some radius, this value is defined to be zero, and within that radius, it is defined as a smooth falloff with $G^1$ continuity (Appendix A). In terms of our framework, our core function $i(\mathbf{x})$ is defined in terms of the core skeleton $C$ and a radius $R_C$.

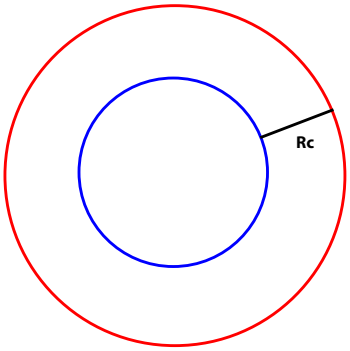



Figure 1: The mathematical form of offset implicit functions as used in this report. Within the blue *core* region $C$, $i(\mathbf{x}) = 1$. Outside the red *falloff* region, specified by a radius $R_c$, $i(\mathbf{x}) = 1$. In the falloff region, $i(\mathbf{x})$ is a smooth falloff function.

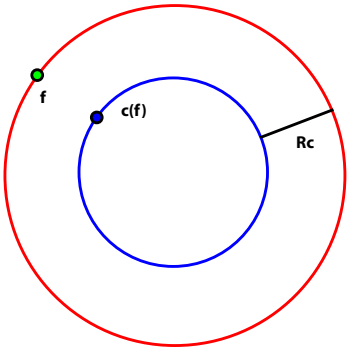
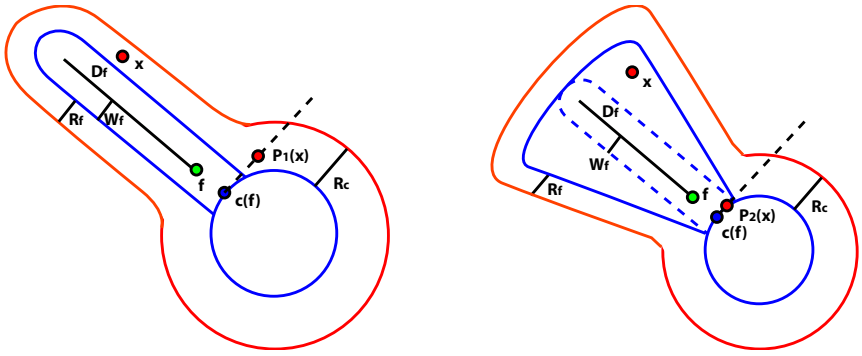



Figure 2: To specify the location of a feature, a connected set of points. This report describes techniques for point and curve features. In this example a point feature $\mathbf{f}$ is specified on some isocontour of the function $i(\mathbf{x})$. The feature point $\mathbf{f}$ has a corresponding *core feature point* $\mathbf{c}(\mathbf{f})$. $\mathbf{f}$ can lie anywhere within the falloff region.

### 3.2 Defining the New Function

To specify the new function, users position a *feature* on an isocontour. Figure 2 illustrates this with a feature point $\mathbf{f}$ embedded on the $i(\mathbf{x}) = 1$ isocontour. Given this feature, we defined the *core feature* $\mathbf{c}(\mathbf{f})$ such that every feature point maps to the closest point on the core skeleton.

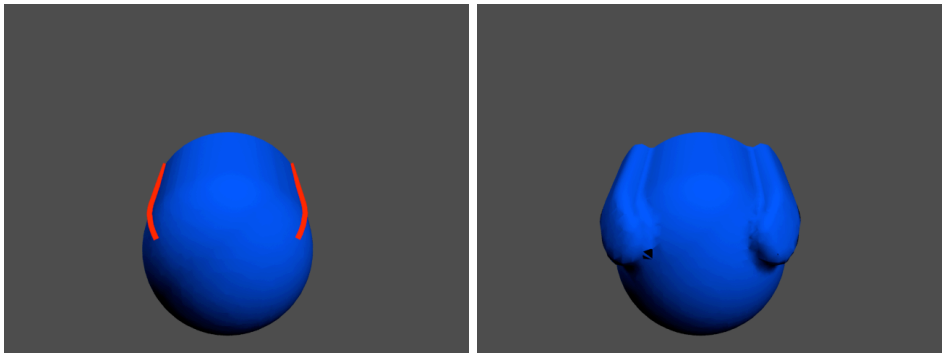



(a) (b)

Figure 3: The modified implicit function includes a modified core. The feature core is defined orthogonal to the surface at the feature point. This feature core is parameterized by a width $W_f$ and a depth $D_f$, as well as a separate radius $R_f$. Function evaluation includes a projection to the tangent plane through the core feature point $\mathbf{c}(\mathbf{f})$. If this projection is parallel to $\mathbf{f} - \mathbf{c}(\mathbf{f})$, the feature core edges are parallel (a). If the projection maps to the closest core point sufficiently close to $\mathbf{c}(\mathbf{f})$, the feature edges will be orthogonal to the original core (b). This quality if called *flare*.

Users have control over a number of feature parameters (Figure 3). The feature width $W_f$ represents the width of a region that evaluates to one; this width is defined at the feature point $\mathbf{f}$. The feature depth $D_f$ represents the depth of the region that evaluates to one. The feature radius $R_f$ represents the width of a function falloff region, which is independent of the core radius $R_C$. As such, the final form is not a true offset function. These parameters and their affect on the shape are illustrated in Figures 4 and 5.

Function evaluation is defined in terms of a projection to the core skeleton. We consider one of two projections. To create a feature whose sides are parallel to the gradient at the feature point, we define the projection $\mathbf{p}_1(\mathbf{x})$ to the plane normal to the function gradient at the core feature. To create a feature whose sides are parallel to the local gradient, we define the projection $\mathbf{p}_2(\mathbf{x})$ to a point on the core "near" the core feature.



(a) (b)

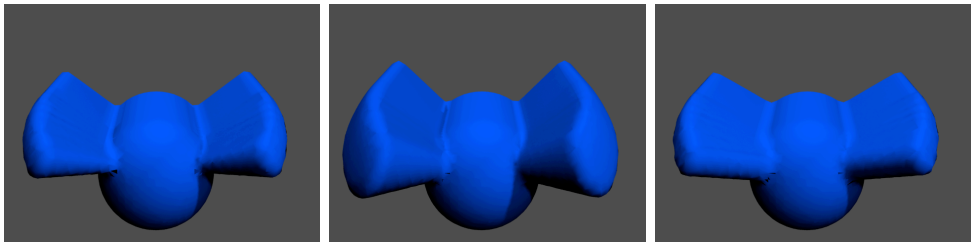
Figure 4: Example creation of implicit function features. Given an isosurface of the core function, users can embed curves to define features (a). The new implicit function, an isocontour of which is given in b, includes features that lie along the path off the curve.

We use a closest point evaluation from the point $\mathbf{x}$ to the core (this has some complications, described in Section 5). A feature distance value can then be computed, for either projection, as $||\mathbf{p}(\mathbf{x}) - \mathbf{c}(\mathbf{f})||$. Given this distance, an implicit function can be defined over the falloff region of the feature as $k(\mathbf{x}) = s(||\mathbf{p}(\mathbf{x}) - \mathbf{c}(\mathbf{f})||; W_f, W_f + R_f)$ (see Appendix A for details of the falloff function $s(x)$). We provide a user control called flare, which allows for a smooth transition between the two projections, and interpolate the resulting feature distances with this control. We then have $k(\mathbf{x}) = \mathit{flare} * ||\mathbf{p}_2(\mathbf{x}) - \mathbf{c}(\mathbf{f})|| + (1 - \mathit{flare}) * ||\mathbf{p}_1(\mathbf{x}) - \mathbf{c}(\mathbf{f})||$.

Given this function $k(\mathbf{x})$, we define our final function $f(\mathbf{x})$ as an interpolation between the core function $i(\mathbf{x})$ and the feature–distance function $j(\mathbf{x})$. $j(\mathbf{x})$ is defined as an offset function with a falloff region more distant to the core skeleton than the initial falloff region as $j(\mathbf{x}) = s(||\mathbf{x} - \mathbf{c}(\mathbf{x})||; R_C + D_f + W_f; R_c + D_f + W_f + R_f)$. For efficiency, we evaluate it to zero whenever $k(\mathbf{x}) = 0$. The final form of the function with the feature is $f(\mathbf{x}) = k(\mathbf{x}) * j(\mathbf{x}) + (1 - k(\mathbf{x})) * i(\mathbf{x})$, which results in a smoothly varying form near the edge of the feature.

## 4 Results

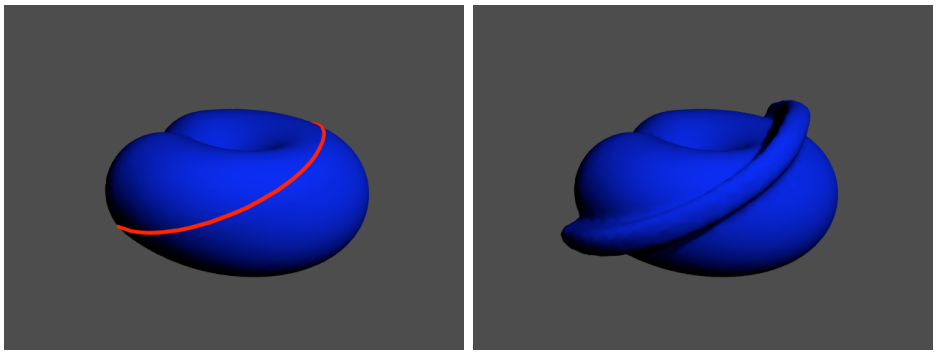
Figures 6 to 9 demonstrate how user controls affect the shape of the function, as well as how the function can be visualized as an interpolation between



(a) (b) (c)

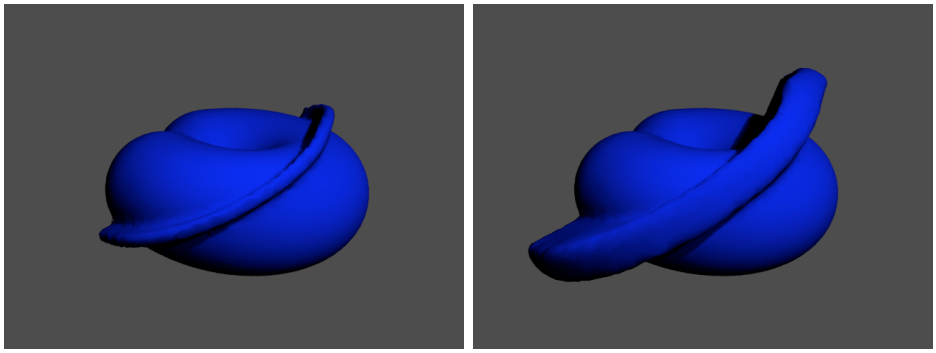
Figure 5: Users can edit the feature parameters to achieve particular cross–sectional shapes. Features can be made taller or deeper (a), wider (b), and the amount of flare can be controlled (c).

two functions by a third. In Figure 6a, a user has generated an isocontour to position a curve. This results in the new function $f(\mathbf{x})$ as shown in Figure 6b. By reducing $W_f$, the feature becomes narrower (Figure 7a). By increasing $D_f$, the feature becomes taller (Figure 7b). Flare control represents how much the feature expands or contracts with respect to distance to the core, as shown in Figure 8. Figure 9 illustrates the decomposition of $f(\mathbf{x})$ into $i(\mathbf{x})$ (blue), $j(\mathbf{x})$ (light blue), and $k(\mathbf{x})$ (yellow–orange). $j(\mathbf{x})$ appears to fall off sharply with distance, which is due to the efficiency culling of its evaluation when $k(\mathbf{x}) = 0$, although it does have a smooth definition.



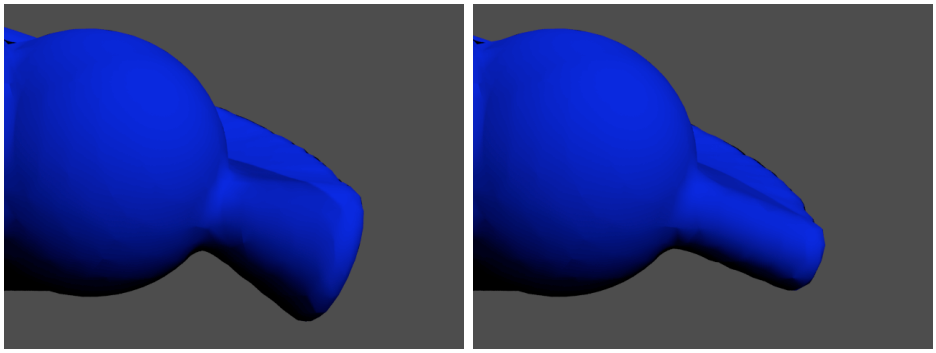
(a) (b)

Figure 6: Example curve on a core function isocontour (a) and the corresponding isocontour of the new implicit function (b).



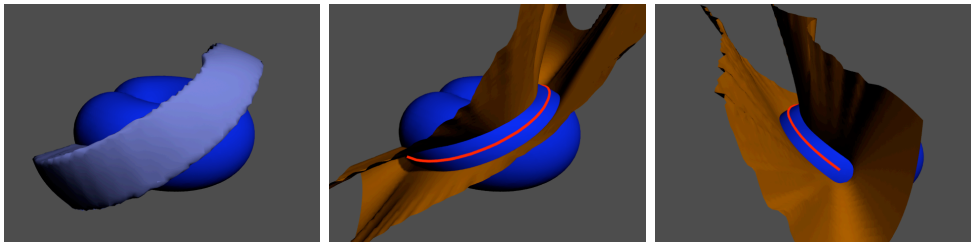
(a) (b)

Figure 7: the feature From Figure 6 is edited to be (a) narrower by decreasing the feature width $W_f$ and (b) taller by increasing the feature depth $D_f$.



(a) (b)

Figure 8: Flare can be continuously adjusted to result in feature side perpendicular to the surface (a) or parallel to the feature core (b). The interpolation is in the distance metric of the two projections of the evaluation point $\mathbf{x}$ to the feature core point $\mathbf{c}(\mathbf{f})$.



(a) (b) (c)

Figure 9: The definition of the feature–inclusive implicit function can be represented as an interpolation between (a) the core implicit function $i(\mathbf{x})$ (blue) and the feature implicit function $j(\mathbf{x})$ (light blue) by (b) an interpolating implicit function $k(\mathbf{x})$ (yellow–orange), which lies in the falloff region the feature core surrounding the feature curve (c). The core feature implicit function $j(\mathbf{x})$ incorrectly appears to have jagged edges due to an efficiency–motivated discontinuity outside the interpolation region, to which the isosurface extraction does not easily converge. The oscillatory pattern in $k(\mathbf{x})$ is due to numerical inaccuracies resulting from the piecewise linear core function isocontour used to generate the feature curve.

## 5 Discussion and Conclusion

This report presents a technique for defining new implicit functions in terms of an existing implicit offset function and user–positioned features that lie in an isocontour of the original function. While the approach allows users to creatively shape the form of an implicit function, it is the result of exploratory design, and some improvements could by made. For example, flare is defined such that it is dependent on the curvature of the core function isocontours. A curvature–independent form would also be useful and worth designing.

Currently, the projection $\mathbf{p}_2(\mathbf{x})$ is implemented with a closest point evaluation. This can be problematic for locally convex surfaces, as the closest point might not be near the feature when the evaluation point is in the region intended to be part of the feature. To avoid this problem, multiple evaluations on subsurfaces of the core are considered and the maximum evaluation value is retained. A more efficient approach would be worth investigating. The surface cord technique does not handle local isocontour concavities; an alternate technique that handles such concavities would be useful.

The current implementation only supports point and curve features; an

extension to surface features would be worthwhile. Various parameters could also be parameterized by the feature. For example, the curve–based feature could follow a contour by parameterizing $D_f$ in terms of the curve's arc length.

Finally, the practical usefulness of these implicit functions defined with features can be explored for applications ranging from deformation and character rigging to shading and even physical simulation control.

# A Piecewise Smooth Falloff Functions

Falloff functions for implicit functions are typically defined such that for some region $[a, b]$, the falloff function $s$ has the properties $s(a; a, b) = 1$ and $s'(a; a, b) = 0$. Similarly $s(b; a, b) = 0$ and $s'(b; a, b) = 0$. Outside the region $[a, b]$, $s$ is constant while maintaining tangent continuity at $x = a$ and $x = b$. This is typically defined using a piecewise cubic polynomial; a reversal of the smoothstep function $smoothstep(x, a, b)$ is used here [1]. This has the properties that $s(x; a, b) = smoothstep(-x, -b, -a)$.

# B Surface Cords

Surface cords are user–controllable curves that wrap around locally convex surfaces without requiring specification of a surface embedding. Users instead position a *guide curve* near a surface, and a *cord* is generated that interpolates the guide curve endpoints while wrapping around the surface [4]. Surface cords are a modified cord form, in which the cord originates at an arbitrary surface point near the guide curve start point $\mathbf{f}_0$ and ends at another surface point, near the end point $\mathbf{f}_n$. The closest surface point to the starting point of the guide curve is used as the starting point $\mathbf{p}_0$ of the surface cord. Surface cord generation follows the form of inelastic, non–stiff cords, but growth terminates when $(\mathbf{f}_n - \mathbf{p}_i) \cdot (\mathbf{p}_{i+1} - \mathbf{p}_i) < 0$, at which point the partial segment $\mathbf{p}_i + ((\mathbf{f}_n - \mathbf{p}_i) \cdot (\mathbf{p}_{i+1} - \mathbf{p}_i))/||\mathbf{p}_{i+1} - \mathbf{p}_i||$ is appended. This termination point is continuous with respect to changes of shape to the guide curve.

# References


[1] Anthony A. Apodaca and Larry Gritz. *Advanced Renderman: Creating CGI for Motion Pictures.* Morgan Kaufmann, 2000.

[2] James F. Blinn. A Generalization of Algebraic Surface Drawing. *ACM Trans. Graph.*, 1(3):235–256, 1982.

[3] Jules Bloomenthal. *Introduction to Implicit Surfaces*. Morgan Kaufmann, 1997.

[4] Patrick Coleman and Karan Singh. Cords: Keyframe Control of Curves with Physical Properties. In *SIGGRAPH 2004 Sketches*, 2004.

[5] Patrick Coleman and Karan Singh. RYAN: Rendering Your Animation Nonlinearly projected. In *NPAR '04: Proceedings of the 3rd International Symposium on Non–Photorealistic Animation and Rendering*, pages 129–156, New York, NY, USA, 2004. ACM Press.

[6] Tony DeRose, Michael Kass, and Tien Truong. Subdivision Surfaces in Character Animation. In *SIGGRAPH '98: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pages 85–94, New York, NY, USA, 1998. ACM Press.

[7] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide*. Morgan Kaufmann, 2001.

[8] Nick Foster and Ronald Fedkiw. Practical Animation of Liquids. In *SIGGRAPH '01: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 23–30, New York, NY, USA, 2001. ACM Press.

[9] Hugues Hoppe, Tony DeRose, Tom Duchamp, Mark Halstead, Hubert Jin, John McDonald, Jean Schweitzer, and Werner Stuetzle. Piecewise Smooth Surface Reconstruction. In *SIGGRAPH '94: Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, pages 295–302, New York, NY, USA, 1994. ACM Press.

[10] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A Sketching Interface for 3D Freeform Design. In *SIGGRAPH '99: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 409–416, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[11] Andrew Nealen, Olga Sorkine, Marc Alexa, and Daniel Cohen-Or. A Sketch–Based Interface for Detail–Preserving Mesh Editing. *ACM Trans. Graph.*, 24(3):1142–1147, 2005.

[12] H. Nishimura, M. Hirai, T. Kawai, T. Kawata, I. Shirakara, and K. Omura. Object Modeling by Distribution Functions. *Electronic Communications*, 68(4):718–725, 1985.

[13] Rick Parent. *Computer Animation: Algorithms and Techniques*. Morgan Kaufmann, 2001.

[14] Michael Pratscher, Patrick Coleman, Joe Laszlo, and Karan Singh. *Outside–In* Anatomy Based Character Rigging. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 329–338, New York, NY, USA, 2005. ACM Press.

[15] N. Rasmussen, D. Enright, D. Nguyen, S. Marino, N. Sumner, W. Geiger, S. Hoon, and R. Fedkiw. Directable Photorealistic Liquids. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 193–202, New York, NY, USA, 2004. ACM Press.

[16] Karan Singh and Eugene Fiume. Wires: A Geometric Deformation Technique. In *SIGGRAPH '98: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pages 405–414, New York, NY, USA, 1998. ACM Press.

[17] Karan Singh and Rick Parent. Joining Polyhedral Objects Using Implicitly Defined Primitives. *The Visual Computer*, 17:415–428, 2001.

[18] G. Wyvill, C. McPheeters, and B. Wyvill. Data Structures for Soft Objects. *The Visual Computer*, 2:227–234, 1986.