Chapter 11
ShapeShop: Free-Form 3D Design with Implicit Solid Modeling

Ryan Schmidt and Brian Wyvill

11.1 Introduction

Implicit modeling has been used since the early 1980s as an alternative to mainstream solid-modeling techniques. Beyond Boolean composition, implicit modeling integrates blending and deformation operators which allow complex free-form solid models to be more easily described. Recent advances have alleviated some of the major technical problems, such as visualization speed and surface control. In this chapter we describe how to combine these new techniques with a sketch-based 3D modeling interface, resulting in a powerful tool for quickly creating solid models. Our system, called *ShapeShop*, has been used as a testbed to investigate a range of problems, from real-time visualization [36], sweep surfaces [35], surface parameterization [38], and implicit surface deformation [41], to higher-level design and usability issues such as structured visualization [39] and 3D manipulation [40]. Sketch-based 3D modeling is the common thread which ties all these various problems together [33, 34, 37, 46].

Much like the seminal Teddy system [18], ShapeShop is a tool for incrementally creating a 3D model using simple 2D sketches. From this starting point, many other sketch and pen-based interaction techniques have been adapted and integrated into the system. ShapeShop borrows liberally from work in sketching assistance [7, 17], gestural systems [48], crossing interfaces [2], and suggestive modeling [16]. One long-term aspect of the ShapeShop project is to evaluate how these non-traditional techniques can be effectively combined within a complex interface.

R. Schmidt (✉)
Dept. of Computer Science, University of Toronto, 10 King's College Road, Rm. 3302, Toronto, Ontario M5S 3G4, Canada
e-mail: rms@dgp.toronto.edu

B. Wyvill
Department of Computer Science, University of Victoria, Engineering/ Computer Science Building (ECS), Room 504, STN CSC, PO Box 3055, Victoria, BC, Canada V8W 3P6
e-mail: blob@cs.uvic.ca

J. Jorge, F. Samavati (eds.), *Sketch-based Interfaces and Modeling*,
DOI 10.1007/978-1-84882-812-4_11,

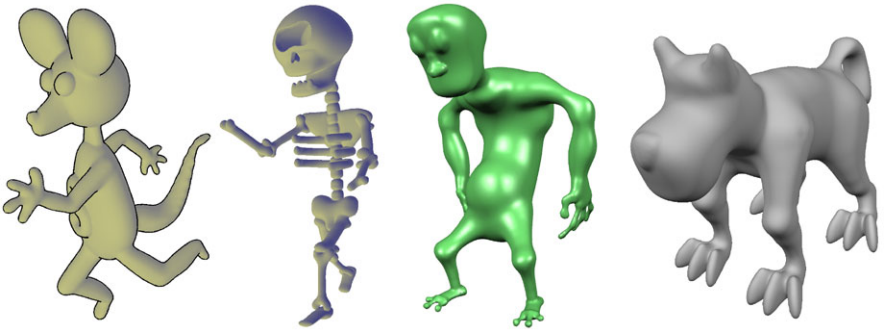

Fig. 11.1 Character models designed by an expert ShapeShop user (the first author), progressing from the earliest versions of the software (*left*) to the most recent releases (*right*)

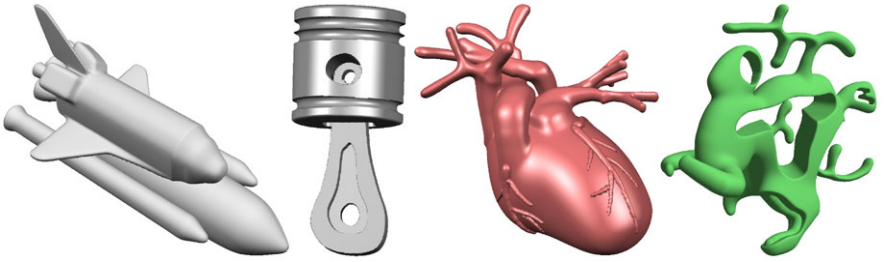

Fig. 11.2 ShapeShop provides an expressive set of sketch-based implicit modeling techniques which support a variety of modeling styles, from high-level conceptual design and CAD-style solid modeling to free-form biological modeling, and even "3D doodling"

The most fundamental difference between ShapeShop and its predecessors is in the use of procedural shape modeling techniques, particularly the hierarchical implicit volume representation known as the BlobTree [45]. As with many other problems in computer science, utilizing a structured, hierarchical framework allows designers to interact with complex models more effectively. In addition, the BlobTree combines traditional CAD-style solid modeling with organic free-form blending in a single interface, greatly enhancing the range of models which can be constructed via sketching (Figs. 11.1 and 11.2). ShapeShop also takes a non-purist approach to sketch-based interface design—functionality is exposed using traditional 2D widgets if suitable alternatives have not yet been developed. As a result, designers can express levels of model complexity not yet reachable using "pure" sketch-based tools. Examples of such models are sprinkled throughout the following chapter, in which ShapeShop's interface techniques, modeling tools, and implementation details are described.

Another notable aspect of the ShapeShop project is that due to extensive development carried out since 2004, the software is quite capable. Many of the results mentioned above are exposed in the current development versions, which are (ir)regularly released on the internet at http://shapeshop3d.com.

Although still very much "research software", ShapeShop has been extended to the point where working artists have found a use for it in their production pipelines.

Fig. 11.3 3D sculptures created by first modeling in ShapeShop, and then importing the surface mesh into Modo [23] for texturing and rendering. Images ©Corien Klapwijk

Some digital sculptors have also taken an interest in ShapeShop, and a few such works are shown in Fig. 11.3. The feedback we have received from this small but growing community of active users is highly informative. The chapter closes with insights gathered from this feedback, as well as issues encountered during the design and development of ShapeShop.

11.2 The ShapeShop Interface

A screenshot of the ShapeShop user interface is shown in Fig. 11.4. The system is implemented using a combination of Microsoft's MFC C++ application framework and custom OpenGL widgets. The main interface window is a single-pane model view. Two additional windows are used to manipulate the scene—a tree view for interacting with the current BlobTree hierarchy, and a list view for changing parameters of a selected tree node. Standard menus and toolbars are also used to control functionality which has not yet been exposed in the sketch-based interface.

Various interface components are embedded in a Heads-Up Display (HUD) rendered on top of the model view. The Expectation List dynamically responds to sketches drawn by the user, offering possible model interactions. The Parameter Toolbar offers interactive manipulation of the most commonly-used parameters of the selected node. Similarly, the Options Toolbar provides access to frequently changed scene parameters, and the View Toolbar provides camera control.

11.2.1 Pencil-based Interaction

ShapeShop has been designed primarily to support use on direct-input displays, such as the touch-sensitive SmartBoard (Fig. 11.5). These devices lack any sort of modal

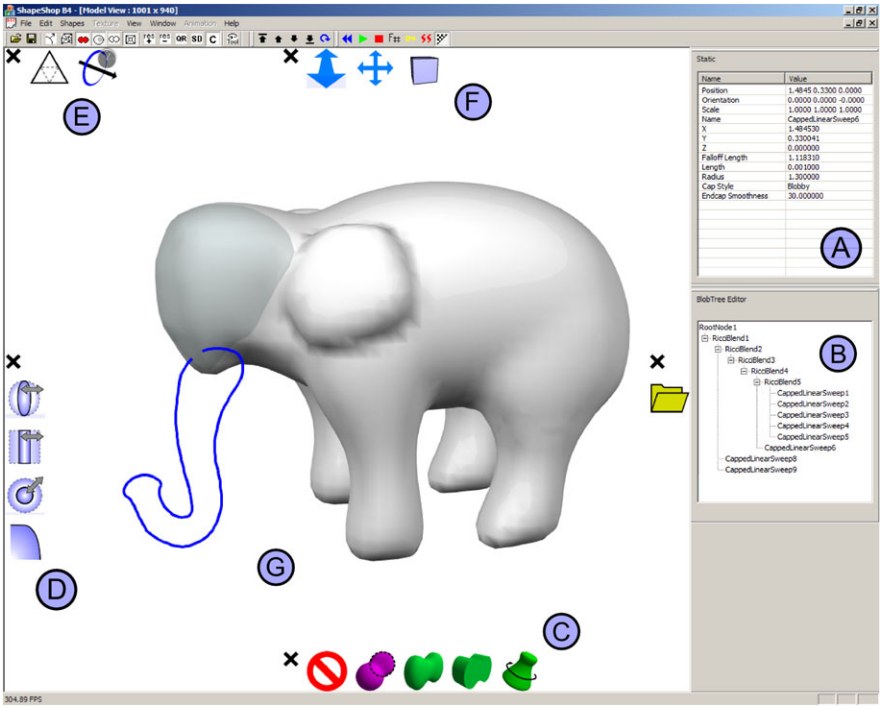

Fig. 11.4 The various major interface elements in ShapeShop include the (A) Parameter Editor, (B) BlobTree Editor, (C) Expectation List, (D) Parameter Toolbar, (E) Options Toolbar, (F) View Toolbar, and (G) Model View

switch (buttons). Hence, we think of the interaction style in ShapeShop as *pencil-based* rather than *pen-based*, as most pen-based systems incorporate physical mode switches such as buttons on the pen barrel. Restricting ourselves to pencil-based interaction does complicate the interface, as tasks commonly initiated with physical mode switches must be converted to alternate schemes. In addition, since traditional 2D interface widgets can be difficult to use with pen or touch input, we adopt the stroke-based widget interaction techniques of CrossY [2]. For example, a button is "pressed" by drawing a stroke across it.

An obvious drawback is that we have heavily overloaded the meaning of such strokes. For example, the user may intend to interact with a 2D widget, make a gestural command, or specify the 2D silhouette of many possible 3D shapes. To resolve this ambiguity, we apply three stages of interpretation. First, visible 2D and 3D widgets that take continuous input are given a chance to capture the current stroke as it is being drawn. Next, uncaptured strokes are tested against visible widgets for crossing actions, and then against the small set of gestures that the system understands.

If no widget or gesture interactions are detected, the system assumes that a 3D construction or editing operation is desired. These actions are presented to the user via the Expectation List, a dynamic toolbar utilized in several other sketch-

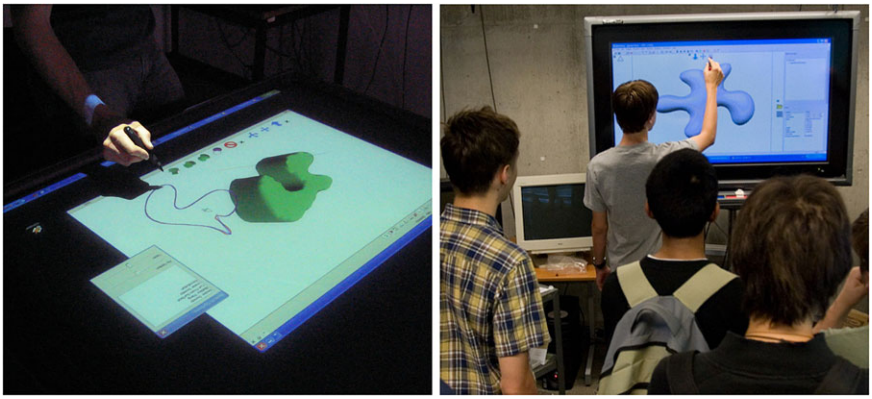

Fig. 11.5 Our *pencil-based* modeling interface is designed to support non-modal input devices, like these touch-sensitive horizontal tabletop and digital whiteboard displays

ing tools [3, 13, 16]. Context-dependent rules are used to populate the Expectation List, by comparing the current sketch with the underlying 3D model. For example, a closed contour generates several sweep-surface options, but hole-cutting options are only produced if the contour intersects the current surface. Note that Expectation Lists in previous systems have generally contained small images of what the updated surface would look like for each expectation list icon. For complex models the user may be required to carefully inspect each image to find the desired action. Instead, we use color-coded iconic representations which may be more easily recognized.

11.2.2 Sketching Assistance

Two-dimensional sketches form the basis for 3D shape creation in ShapeShop. To aid the user in the 2D drawing task, ShapeShop includes techniques that assist with the creation of smooth 2D contours. Our approach is inspired by Baudel's over-sketching techniques [7] and the *interactive beautification* tools found in the Pegasus system [17]. See [27] for a recent survey of these and related techniques.

A fundamental limitation of most standard input devices is that they provide only point samples to the operating system. These discrete data can be converted to a polyline by connecting temporally-adjacent point samples. However, in the case of curves the polyline is only an approximation to the smooth curve the user desires. In our system we do not create an approximate polyline, but instead fit a smooth 2D variational implicit curve [32, 43] to the discrete samples. Curve normals derived from the discrete polyline are used to generate the necessary off-curve constraint points [10]. Variational curves provide many benefits, such as automatic smoothing and gap-closing with minimal curvature (Fig. 11.6).

While this approach is most effective for closed curves, it can also be applied to open curves in some cases. If the fit variational curve extends beyond the sketching

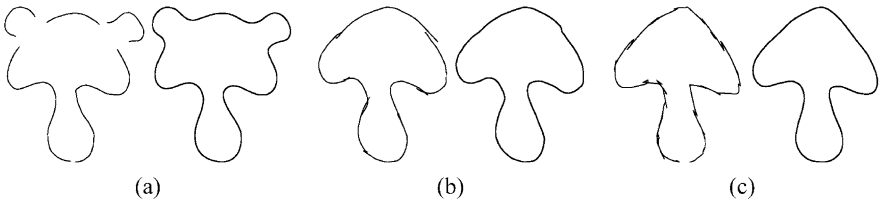

Fig. 11.6 The gap-filling and smoothing properties of variational curves simplify 2D curve sketching. In (**a**), multiple disjoint strokes are automatically connected by fitting a variational curve to the input samples. In (**b**), smoothing parameters are used to handle intersections between multiple strokes. Rough self-intersecting sketches can be automatically smoothed, as shown in (**c**)

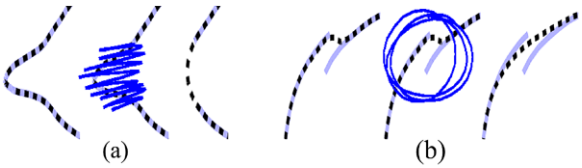

Fig. 11.7 Examples of the *eraser* gesture (**a**) and *smooth* gesture (**b**). These gestures manipulate the parameters used to compute the final variational curve (*dashed line*)

area, we assume the curve is open and clip it to lie within the endpoints of the sampled polyline. However, as ShapeShop is a volume modeling interface, closed contours are required to perform most of the creation and editing actions.

ShapeShop supports sketch-based editing of the set of point samples, but not the final variational curve. To simultaneously visualize these two different components, we render the current variational curve in black and the sketched polyline in transparent blue (Fig. 11.7). Three gestural commands are available to assist users when drawing 2D sketches. The first, *eraser*, is initiated with a "scribble", as shown in Fig. 11.7(a). An oriented bounding box is fit to the scribble vertices and used to remove point samples from the current 2D sketch. The variational curve is recomputed using the remaining samples.

The second gestural command is *smooth*, initiated by circling the desired smoothing region a minimum of two times. Each point sample has a smoothing parameter associated with it which is incremented if the point is contained in the circled region. The variational curve is then re-computed with the new smoothing parameters (Fig. 11.7(b)). This gesture can be applied multiple times to the same point samples to further smooth the 2D sketch. Finally, we include an *undo* gesture, input as a quick stroke straight to the left, which removes the most recent stroke.

We have found these techniques to be very effective for creating smooth 2D sketches. This in turn improves the efficiency of 3D modeling, since fewer corrections need to be made to the 3D shape. One current limitation is that sharp creases in the input sketch are lost, since the underlying variational curve is always C^2 continuous. A useful extension to our technique would be to automatically detect sharp edges, and re-introduce them into the smoothed variational curves.

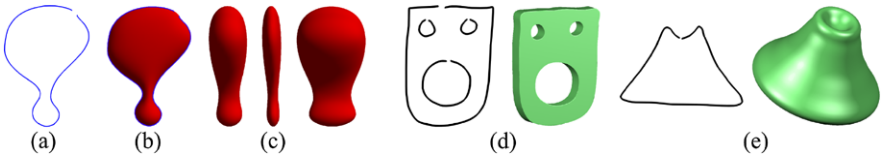

Fig. 11.8 Blobby inflation converts the 2D sketch shown in **a** into the 3D volume **b** such that the 2D sketch lies on the 3D silhouette. The width of the inflated volume can be manipulated interactively, shown in (**c**). Sketched 2D curves can also be used to create **d** linear sweeps and **e** surfaces of revolution

11.2.3 Sketch-based Modeling Operations

ShapeShop supports construction of three basic types of shapes derived from sketched 2D contours—"blobby" inflation in the style of Teddy [18], linear sweeps, and surfaces of revolution. Based on these three shapes, sketch-based cutting and blending operations are implemented using BlobTree composition operators.

A key benefit of utilizing the BlobTree shape representation is that the current volume is procedurally defined by an underlying model tree which represents both a scene graph and a full construction history. Single primitives, as well as entire portions of the tree, can be modified or removed at any time. Exposing this flexibility through a sketch-based interface can be quite difficult, and much research remains to be done on intuitive techniques for editing volumetric scene graphs. In ShapeShop, the designer can use gestural commands and 3D widgets to manipulate individual BlobTree nodes, however more complex operations like tree re-structuring require the use of a traditional tree-view widget (Fig. 11.4). In our experience, designers find it difficult to understand the link between this text-based tree view and the actual model structure. An abstraction which simplified this tree view while still preserving the considerable power it provides would be highly desirable.

11.2.3.1 Blobby Inflation

As in many other sketch-based modeling tools [3, 18, 20, 25, 28, 42], the primary shape-creation operation in ShapeShop is *inflation*, where a closed 2D contour is then inflated into a "blobby" 3D shape. This operation can be easily accomplished using implicit sweeps with the blobby endcap style, as described in Sect. 11.3.3. The 2D sketch (Fig. 11.8a) is projected onto a 3D plane parallel to the current view plane, and then inflated in both directions (Fig. 11.8b). After creation, the width of the primitive can be manipulated interactively with a 2D widget (Fig. 11.8c). The inflation width is functionally defined and could be manipulated to provide a larger difference between thick and thin sections. One advantage of the implicit representation is that holes and disjoint pieces can be handled transparently.

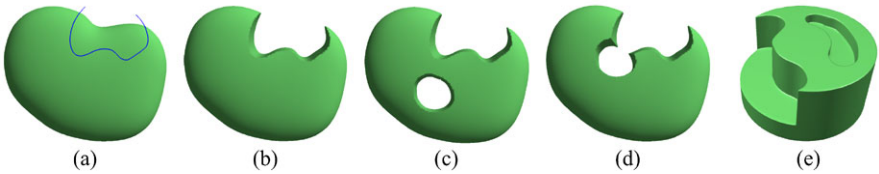

Fig. 11.9 Cutting can be performed **b** across the object silhouette or **c** through the object interior. Holes can be interactively translated and rotated. Intersection with other holes is automatically handled, as shown in (**d**). Hole depth can also be modified to create cut-out regions (**e**)

11.2.3.2 Sweep Surfaces

The sweep-surface representation underlying our blobby inflation scheme also supports linear sweeps (Fig. 11.8d) and surfaces of revolution (Fig. 11.8e). Linear sweeps are created in the same way as blobby shapes, with the sweep axis perpendicular to the view-parallel plane. The initial length of the sweep is proportional to the screen area covered by the bounding box of the 2D curve, but can be interactively manipulated with a 2D widget. Surfaces of revolution are created by revolving the sketch around an axis lying in the view-parallel plane. As in the case of linear sweeps, the revolution template can contain holes, and revolutions with both spherical and toroidal topology can be created.

Aside from Cherlin et al.'s advanced revolution technique [11], most sketch-based systems have not included these additional types of shapes. While blobby inflation is highly useful for many modeling tasks, linear sweeps and revolutions are invaluable in situations such as mechanical modeling. In particular, surfaces of revolution are a class of shape that cannot be approximated with blobby inflation.

11.2.3.3 Cutting

Since the underlying BlobTree is a true volumetric model, cutting operations can be easily implemented using CSG operators. Designers can either cut a hole through the object or remove volume by cutting across the object silhouette. Since the "hole" is internally represented as a linear sweep, no additional implementation is necessary to support cutting. In addition, the designer may interactively transform this subtracted sweep at any time, effectively dragging the hole around through the surface. Interactive controls are also available to modify the depth of cutting operations. An example is shown in Fig. 11.9. This CSG-based cutting operation is both more precise and less restrictive than in existing systems. Note that although only sharp edges are demonstrated in Fig. 11.9, ShapeShop includes various "soft" CSG difference operators which generate filleted edges of varying smoothness.

11.2.3.4 Blending

ShapeShop relies on the easy-to-implement implicit blending techniques supported by the BlobTree to allow the designer to increase the volume of the current object.

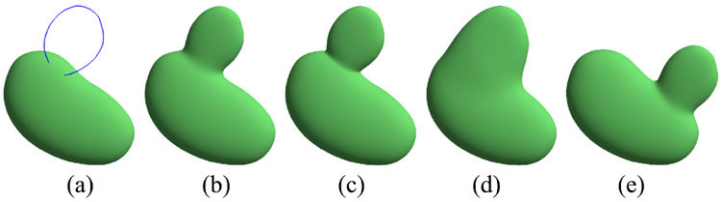

Fig. 11.10 The sketch-based blending operation **a** creates a new blobby inflation primitive and **b** blends it to the current volume. The blending strength can be interactively manipulated, the extreme settings are shown in (**c**) and (**d**). The blend region is re-computed automatically when the blended primitives move, as shown in (**e**)

As demonstrated in Fig. 11.10, this action is initiated by sketching a contour across the silhouette or interior of the current shape. Selecting the resulting suggestion creates a new blobby primitive which is blended with the current model. The width of the new primitive can be manipulated with a slider, as can the amount of blending. Again, as the blend is a dynamic composition of two implicit volumes, either can be transformed interactively (Fig. 11.10(e)).

Various other tools have explored implicit blending [3, 20] or discrete fairing [25] in a sketch-based context. However, the style of dynamic implicit blending available in ShapeShop is highly useful in practice, and one of the features that professional 3D artists find most desirable when first being introduced to the system.

One issue neglected thus far is how to fix the depth of the view-parallel 3D plane onto which a sketched contour is projected. Unlike Teddy's extrusions, we try to infer the correct depth from context, rather than require the user to mark the surface. Without any prior evidence, the plane is assumed to pass through the origin. However, if the sketched strokes overlap the 2D projection of the surface, we center the projection plane at the average depth value along the strokes. If a part is selected, only its surface is considered. This technique is reasonably effective in practice, and one can learn to manipulate the viewpoint and stroke to generate a good initial guess. However, minor errors are common and major errors sometime occur, requiring 3D manipulation. We are currently exploring efficient techniques for allowing the designer to more explicitly specify the depth of newly created primitives.

11.2.3.5 Surface Drawing

Perhaps the most straightforward type of sketch-based interaction is drawing curves directly on an existing 3D surface. Such techniques have long been used in traditional modeling systems [4], and are a basis for operations in many sketch-based systems [18, 24, 25]. ShapeShop supports such a "surface-drawing" technique, useful for adding detail and creating arbitrary 3D structures. The operation is very simple—rays through the 2D strokes are intersected with the current implicit volume, and a solid tube-like volume represented by a 3D implicit polyline

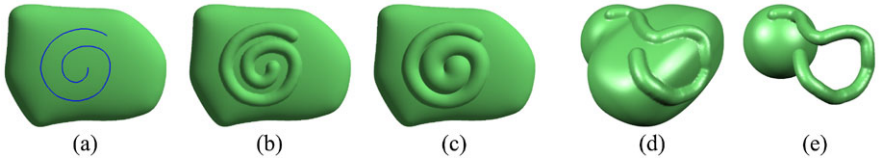

Fig. 11.11 Surface drawing is specified by a 2D sketch, as shown in (**a**). Blended skeletal implicit point primitives are placed along the line at intersection points with the model, shown in (**b**). In (**c**) the radius of the points is increased and then tapered along the length of the 2D curve. Temporary construction surfaces (**d**) can be used to create more complex 3D curves (**e**)

primitive is generated. Interactive controls are provided to manipulate both the surface radius and linear scaling (tapering) along the polyline. Results are shown in Fig. 11.11(a–c).

With Surface Drawing, any pair of implicit primitive and composition operator can be used as a type of "brush" to add detail to the current surface. Implementing these alternative tools within the BlobTree framework is very straightforward. In addition, since each surface-drawing stroke is represented independently in the model hierarchy, individual surface details can be modified or removed using the existing modeling interface. Of course, as the surface is dynamically polygonized, interactive visual fidelity must be limited at levels which often do not resolve fine details. This does unfortunately limit the use of surface drawing in practice.

Surface drawing also readily demonstrates another extremely useful property of utilizing the BlobTree as an underlying shape representation for sketch-based modeling. As shown in Fig. 11.11(d–e), surface drawing can be applied to a temporary *construction* surface, which is then erased, resulting in free-floating geometry which does not lie on a planar space curve. The same technique can be used to fix the depth of sketched primitives without excessive manual positioning. These temporary construction surfaces are a novel property of ShapeShop which was not designed into the system, but simply emerged out of the non-linear hierarchical editing capabilities of the BlobTree. One potentially fruitful area of future research would be to explore more explicit support for construction surfaces in sketching systems.

11.2.4 Selection and Transformation

Procedurally defined BlobTree volumes inherently support non-linear editing of internal tree nodes. However, before a primitive can be manipulated it must be selected. One option is to cast a ray into the set of primitives and select the first-hit primitive. This technique is problematic when dealing with blending surfaces, since the designer may click on the visible surface but no primitive is hit.

Picking in ShapeShop is implemented by intersecting a ray with the current volume, and then selecting the primitive which contributes most to the total field value at the intersection point. This algorithm selects the largest contributor in blending situations, and selects the subtracted primitive when the user clicks on the inside

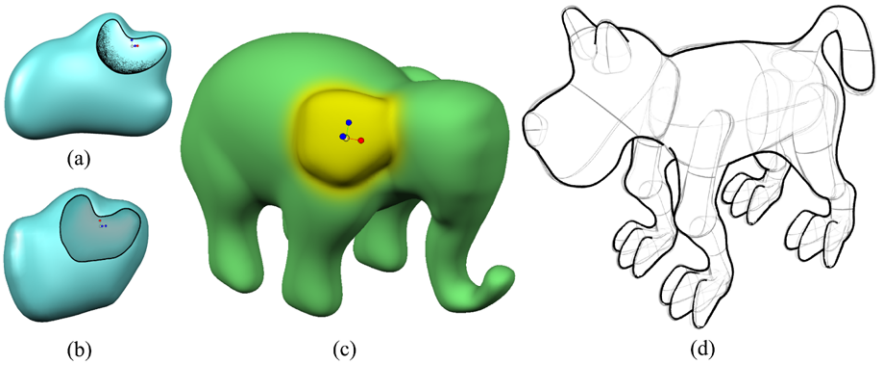

Fig. 11.12 Internal volumes can be directly rendered using pen-and-ink stippling (**a**) or transparency (**b**). Portions of the surface can also be highlighted to show the influence region of a selection (**c**). Visual Scaffolding techniques provide an integrated display of all the primitives making up a model (**d**), but do not convey the structure of the BlobTree

of a hole surface. However, selecting the maximum contributor can result in non-intuitive behavior in cases where a small primitive is blended with a larger one, as the larger primitive may contribute more to the field at all points on the surface. In this case, the user must use the BlobTree Editor tree view (Fig. 11.4) to select the desired node. An un-implemented but sensible alternative would be to cycle through the possible selections using multiple taps.

This selection system only allows for selection of primitives. To select composition nodes we implement a *parent* gesture, which selects the parent of the current node. The *parent* gesture is entered as a straight line towards the top of the screen. No similar child-selection gesture has been implemented because it is unclear how to disambiguate which child is desired in cases where a node has multiple children. A selected primitive or composition node can be removed using the *eraser* gesture described in Sect. 11.2.2. Removing a composition node is equivalent to cutting a branch from the model tree—all children are also removed. More complex tree traversal and manipulation, such as re-arranging nodes, currently require the use of the BlobTree Editor tree view.

We have experimented with several rendering modes to display the shape of selected primitives, which are often completely contained within the current volume (Fig. 11.12). These techniques effectively convey the shape of the selected volume, but the semantics of the local BlobTree structure are completely opaque. Visualization of structured hierarchical 3D models, in a manner suitable for intuitive direct manipulation, is a challenging and relatively open problem. Obvious approaches like transparency or cut-away views do not scale well to complex nested trees. One possible approach we have recently explored involves *visual scaffolding*, a rendering style which mimics the construction geometry sketched by artists to help produce correct proportions and perspective in pencil-and-paper drawing [39]. However, this technique only addresses visualization of the BlobTree primitives; no support for display or interaction with composition nodes was provided (Fig. 11.12(d)). This

is a key direction for future work, which impacts not only solid modeling, but any dataflow-based procedural modeling interface.

ShapeShop supports 3D manipulation using standard 3D translation and rotation widgets. Compared to the fluid gestural commands used elsewhere in ShapeShop, these 3D widgets are rather crude, and hence recent work has been directed towards exploring alternate 3D manipulation schemes [40]. These techniques still involve 3D widgets, but utilize context-sensitive gestural and suggestive methods which are more compatible with sketch-based interfaces.

11.3 Technical Details

The technical details underlying the various components of the ShapeShop system span many areas of computer graphics and human-computer interaction. In the following text we focus the discussion on the critical shape modeling aspects relating to hierarchical implicit volume modeling. Even that is quite a large subject, far too extensive to describe here in any depth. Hence, we limit ourselves to a very brief overview of the basics, and refer the interested reader to [33] for detailed information and discussion of open problems in this area.

11.3.1 Hierarchical Implicit Volume Modeling

Consider a function f that, when applied to a point $\mathbf{p} \in \mathbb{R}^3$, produces a scalar value $f(\mathbf{p}) \in \mathbb{R}$. A surface $\mathcal{S} \in \mathbb{R}^3$ can then be defined by the equality

$$f(\mathbf{p}) = v \tag{1}$$

where $v \in \mathbb{R}$ is a scalar value. This surface $\mathcal{S}$ is an *iso-contour* of the *scalar field* produced by $f(\mathbf{p})$, and v is the *iso-value* that produces $\mathcal{S}$. In computer graphics, $\mathcal{S}$ is commonly known as an *implicit surface*. One example is the *distance field*, defined with respect to some geometric entity T, such as a point or a curve:

$$d_{\mathsf{T}}(\mathbf{p}) = \min_{\mathbf{q} \in \mathsf{T}} |\mathbf{q} - \mathbf{p}|. \tag{2}$$

Intuitively, $d_{\mathsf{T}}(\mathbf{p})$ is the shortest distance from $\mathbf{p}$ to T. Hence, $d_{\mathsf{T}}(\mathbf{p}) = 0$ describes the set of points $\mathbf{p}$ lying on T, while non-zero iso-values define offset surfaces. By mapping values to grayscale, we can visualize a 2D slice of the field (Fig. 11.13(a)).

Distance fields can be used directly for 3D modeling, however they have several limitations—they do not necessarily define closed surfaces, may be discontinuous, and have infinite extent. As we shall soon see, these are problematic if we wish to functionally combine implicit surfaces. Instead, we can apply a second function g to the distance field, which is known as a *falloff* or *potential* function. We use

$$g(d) = \left(1 - d^2/r^2\right)^3. \tag{3}$$

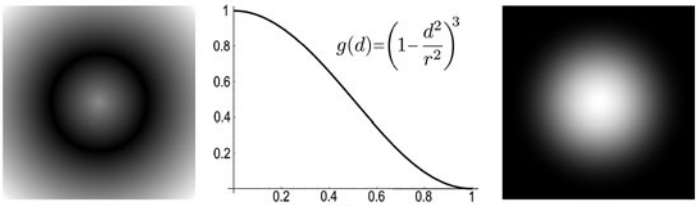

Fig. 11.13 The 2D distance field from a circle is shown in (**a**), with distance values mapped to grayscale—brighter values indicate larger distances. Skeletal primitives are created by applying a potential field (**b**) to a distance field, resulting in a bounded field such as in (**c**), which visualizes the value of $g \circ d(\mathbf{p})$ when the skeleton is a single point

As shown in Fig. 11.13(b), this function smoothly decreases from 1 to 0. When composed with a distance field, the resulting field $f(\mathbf{p}) = g \circ d_{\mathsf{T}}(\mathbf{p})$ is *bounded*, meaning that there is a finite region within which all non-zero values, as well as the iso-surface, are guaranteed to be contained (Fig. 11.13(c)). This type of implicit surface is known as a *skeletal primitive*, because T is the *skeleton* of the iso-surface.

Skeletal primitives provide other guarantees as well. Assuming T is convex, the field is necessarily continuous, and is closed by definition. Hence, given an *iso-value* v, skeletal primitives also define implicit *volumes*:

$$\mathcal{V} = \{\mathbf{p} : f(\mathbf{p}) \geq v\}. \tag{4}$$

The volumetric property is quite useful. For example, (4) provides a trivial point containment test. Implicit volumes can also be trivially composed via *Boolean operations*. The union of two implicit volumes f_1 and f_2 can be described by a new scalar field, generated by functional composition [31]:

$$(f_1 \cup f_2)(\mathbf{p}) = \max\big(f_1(\mathbf{p}), f_2(\mathbf{p})\big). \tag{5}$$

The power of this operation, and similar ones for *intersection* and *subtraction* or *difference*, is that they are *closed* under the space of all possible implicit volumes, meaning the can be applied repeatedly, each time producing another implicit volume (Fig. 11.14). Hence, implementing solid-modeling techniques such as *Constructive Solid Geometry* (CSG) is nearly trivial with implicit volumes. The CSG Tree is represented as a hierarchy of functional compositions such as (5), with skeletal primitives at the leaf nodes (see Fig. 11.19 for a simple example).

Solid modeling is not limited to CSG. Another useful class of operation is the construction of smooth transitions between two surfaces, often known as a *blend*. Functional blend operators can be defined for implicit volumes, such as Ricci's blend operator [31] (Fig. 11.14(c)):

$$(f_1 \uplus f_2)(\mathbf{p}) = \big(f_1(\mathbf{p})^s + f_2(\mathbf{p})^s\big)^{\frac{1}{s}} \tag{6}$$

which allows the user to control blend smoothness via the parameter s (as $s \to \infty$, $\uplus \to \cup$). Like the CSG operators, this blend operator is independent of the complexity of the implicit surface, and simply produces another implicit volume. We

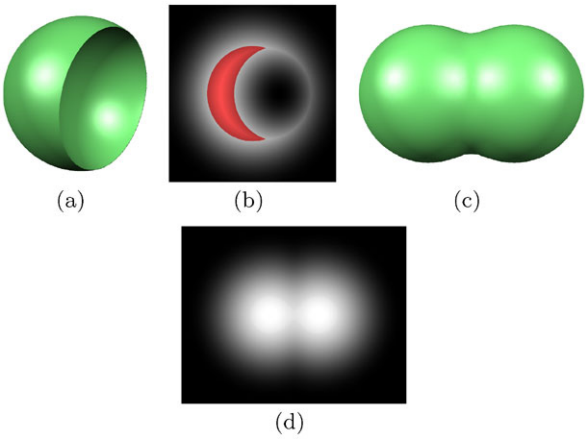

Fig. 11.14 An implicit sphere is subtracted from another using a CSG Difference operation (**a**), and blended in (**c**). 2D slices through the respective 3D scalar fields are shown in (**b**) and (**d**)

can now see why *bounded* fields such as those produced by skeletal primitives are so important—each input field to (6) can only affect the blended surface within its bounding region. This local influence preserves a "principle of least surprise" that greatly improves the usability of constructive implicit modeling.

The BlobTree hierarchical modeling framework is an extension of the traditional CSG Tree which encapsulates techniques for constructive solid modeling with skeletal primitives [45]. In addition to CSG and blending, the BlobTree system includes support for functional warping and deformation, texturing, and animation. See [15] for a thorough description of the full BlobTree system. The BlobTree implementation used in ShapeShop is relatively non-traditional, in that only functionality relating specifically to shape modeling has been implemented, largely to reduce computational costs. For example, even basic BlobTree color support would quadruple the memory requirements involved in the Hierarchical Spatial Caching described in Sect. 11.3.4.

11.3.2 BlobTree Visualization

As noted in the previous section, an implicit surface is defined as an iso-contour of a scalar field, $f(\mathbf{p}) = v$. Unlike a parametric definition, this equation does not directly provide points on the surface. Instead, visualization algorithms must search through space to determine where the surface lies.

Perhaps the simplest technique for visualizing implicit surfaces is polygonization. We use a continuation polygonization algorithm, which initially finds points on the surface by marchings outwards from internal *seed points* defined by the primitives. Space is then subdivided into small cubes, and the cubes intersecting the surface are incrementally enumerated using a stack and hash table [44]. This minimizes the number of field evaluations, and hence is more efficient for polygonizing functional surfaces than the popular *Marching Cubes* algorithm [22], which

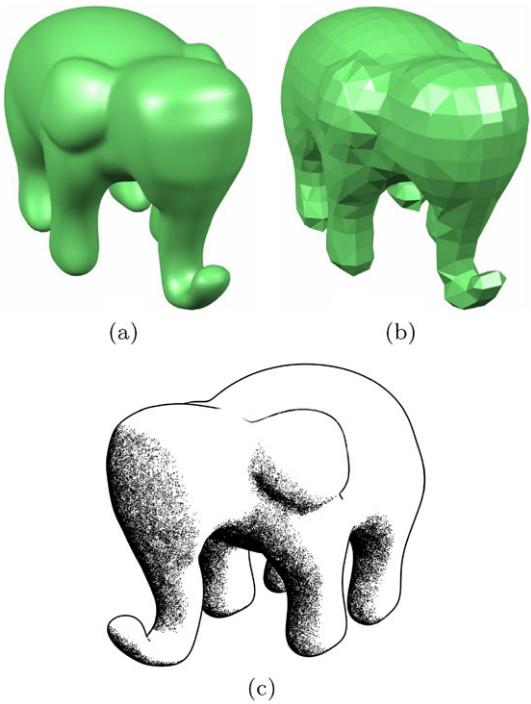

Fig. 11.15 Implicit surfaces can be visualized by dynamically tessellating the surface (**a**). However, to ensure real-time feedback, the mesh fidelity must be dramatically lowered (**b**). Based on this coarse mesh, we can generate high-fidelity pen-and-ink renderings at little extra cost (**c**)

performs a brute-force enumeration of all cubes inside a fixed volume. Ideally these algorithms produce the same mesh, although we cannot guarantee that a seed point exists inside each disconnected component, and hence the continuation approach can sometimes miss parts of the surface.

To interactively visualize BlobTree models in ShapeShop, the polygonization algorithm is performed in real-time, using an optimized version of Bloomenthal's code [8]. Two modified versions of this polygonizer are also available. The first simply adds the crease-finding techniques described in the Extended Marching Cubes work [21]. The second provides support for local updates, where mesh re-computation is limited to the regions in which the current model has changed. The extra contextual information that must be stored to support partial re-meshing does introduce significant overhead, however, smaller local updates are so much more efficient that the benefits largely outweigh this cost.

Despite expending significant effort in our attempts to optimize our polygonizer, we still must sacrifice visual fidelity to ensure interactive feedback rates, even for moderately simple models. Hence, we have begun to explore other visualization techniques. By combining a coarse mesh with local refinement techniques, we can provide real-time pen-and-ink-style rendering at a higher level of visual fidelity, but within the same computation budget as lower quality real-time polygonization (Fig. 11.15). See [39] for technical implementation details.

11.3.3 Sketchable Implicit Sweep Primitives

Traditionally, modeling with the BlobTree involved composition of fixed geometric primitives—spheres, cylinders, and so on [45]. However, in a sketch-based modeling tool, we would like to be able to create an inflated shape with a silhouette that closely matches an arbitrary 2D contour sketched by the designer. One approach is to use numerical optimization to find a set of simple primitives which, when blended, will produce an appropriate shape. This is computationally impractical [9], although a recent specialization for the inflation problem has made it more tractable [1]. Instead, we developed a new BlobTree primitive which supports direct manipulation of the silhouette contour, so that it can be matched directly to a given 2D sketch.

As described in Sect. 11.2.3, ShapeShop's creation tools allow the user to sketch closed 2D contours on a plane in space. This contour is then *inflated* into some 3D volume [18]. Essentially, these inflated shapes are a type of sweep surface or extrusion, where the planar contour is the *template* and the plane normal is the *trajectory*. While sweep surfaces are ubiquitous in surface modeling [30], implicit sweep representations have generally been limited to star-shaped templates [12] which are procedurally defined. To create an implicit primitive whose silhouette contour could be matched to an arbitrary 2D sketch, it was necessary to develop sweep primitives which allowed for arbitrary template curves.

In the implicit domain, we define the desired primitive by sweeping a bounded, continuous 2D template scalar field $f_{\mathcal{C}}$, whose iso-contour v approximates the sketched closed contour $\mathcal{C}$, along the trajectory $\mathcal{T}$. As with other BlobTree primitives, the general approach is to apply a falloff function to the distance field of the curve, hence $f_{\mathcal{C}} = g \circ d_{\mathcal{C}}$, where $d_{\mathcal{C}}$ is the 2D distance field defined by $\mathcal{C}$ and g is as in (3). Note that as the surface of other BlobTree primitives is defined at $v = 0.5$, the values of $d_{\mathcal{C}}$ must be shifted so that the v contour aligns with the 0-contour of the distance field. The shifted distance $d'_{\mathcal{C}}$ is then:

$$d'_{\mathcal{C}} = \min\left(g^{-1}(v) + d_{\mathcal{C}}, 0\right). \tag{7}$$

The bounded template field $f_{\mathcal{C}}$ is then defined as

$$f_{\mathcal{C}} = g \circ d'_{\mathcal{C}}. \tag{8}$$

This formulation produces a template with the desired iso-contour, but the distance field of a non-convex $\mathcal{C}$ contains C^1 discontinuities (Fig. 11.16(a)). These discontinuities will be swept in 3D and produce undesirable artifacts when the sweep primitive is blended with other shapes [33]. Hence, it is necessary to generate a "smoothed" distance field, which approximates $d_{\mathcal{C}}$ but remains continuous.

To create such a smooth distance field, we utilize variational interpolation, also known as *thin-plate splines* approximation. Essentially, a C^2 interpolating thin-plate spline is fit to a set of constraint points placed at samples of $\mathcal{C}$ [47]. To ensure that the solution passes through the sample points, inner and outer *normal constraints* are added at short distances along the normals to $\mathcal{C}$ at the on-curve samples (Fig. 11.17(a)).

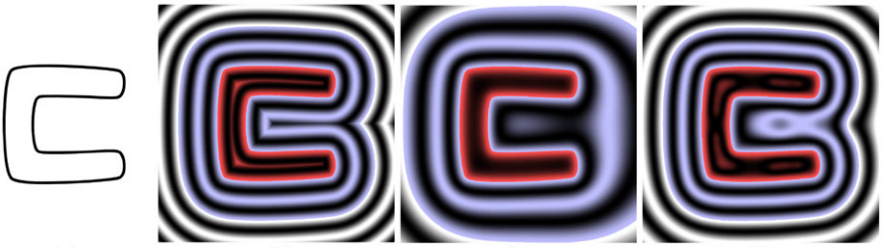

Fig. 11.16 Scalar fields generated using a non-convex curve (**a**). The exact distance field (**b**) has C^1 discontinuities inside and outside the curve. Standard variational interpolation with normal constraints provides a poor approximation in concave regions, and is difficult to bound (**c**). Our approach (**d**) smoothly approximates the distance field away from the surface

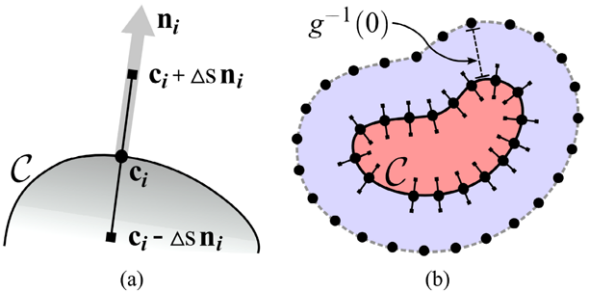

Fig. 11.17 *Normal constraints* (**a**) at a point $\mathbf{c}_i$ are added at short offset Δs from the curve $\mathcal{C}$, along the curve normal $\mathbf{n}_i$. *Boundary constraints* (**b**) are placed at a constant distance from $\mathcal{C}$ to improve the distance field approximation and ensure that the field $f_{\mathcal{C}}$ is bounded within a known distance

Normal constraints only constrain the solution near $\mathcal{C}$—the rest of the field is unconstrained, resulting in a poor approximation to the distance field (Fig. 11.16). This is problematic if the field is to be bounded by applying a falloff function, as a time-consuming spatial search is required to determine the non-zero region of the resulting field. With the true distance field, the bounding zero-contour lies along the distance contour $d_{\mathcal{C}} = g^{-1}(0)$, the bounds of which can be reliably computed. Hence, to predictably bound our approximate distance field, we add *boundary* constraints which force the variational field to approximate this outer contour. In addition, we add constraints along two interior contours in the distance field, one at $g^{-1}(0.5v)$, which is approximately half-way between $\mathcal{C}$ and the zero-contour, and another at $g^{-1}(1.5v)$, which lies inside $\mathcal{C}$. The purpose of these extra constraints is to further reduce error in the distance field approximation. To sample these contours, we compute the distance transform of $\mathcal{C}$ on a 512^2 pixel image, and trace the discrete iso-contours in the image. As shown in Fig. 11.16, these constraints greatly improve the distance field approximation, while maintaining global C^2 continuity.

One drawback of this approach is that the evaluation of the variational field which approximates the distance field is $O(N)$ in the number of constraint points, so evaluating (8) is quite expensive. However, since we are only interested in the final bounded field, we can pre-compute its values on a regular grid, or *field image*. From

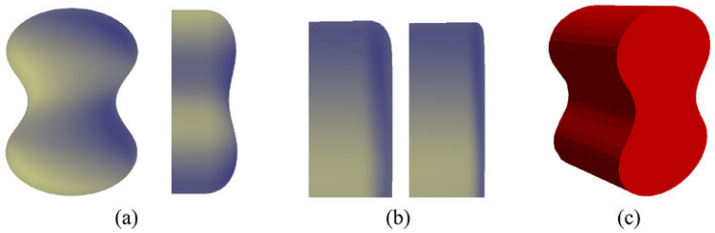

Fig. 11.18 ShapeShop's inflation algorithm is based on linear sweeps with a "blobby" endcap (**a**). Flat endcaps with filleted edges (**b**) or sharp creases (**c**) are also supported, increasing the range of shapes which can be modeled

this field image, the value of (8) can be approximated at any point in constant time using a C^1 bi-quadratic filter [35].

This 2D template scalar field can be easily swept along a line $L(t) = \mathbf{o} + t\mathbf{d}$ in space, generating an 3D scalar field. The value of this field is defined at a 3D point $\mathbf{p}$ by finding the nearest point $L(t_{\text{near}})$ on the line, transforming $\mathbf{p}$ into 2D coordinates $\mathbf{u}$ in the plane perpendicular to the line at $L(t_{\text{near}})$, and sampling the 2D field image. However, this field is infinite. To bound it, or "cap" the sweep, we multiply the infinite sweep values by a falloff function whose value ranges from 1 at $t = 0$ to 0 at t_{max}. Since the values of the infinite sweep vary inside the template, they reach 0 at different distances along the line, producing an endcap which is wider in regions further from the sketched contour, giving the impression of a shape which has been inflated. The falloff function can be modulated to vary the width of the shape, and also to produce different effects such as completely flat endcaps with smooth or sharp transitions (Fig. 11.18).

In a traditional surface or solid-modeling environment, limiting the available primitives to linear sweeps and surfaces of revolution would be highly restrictive. However, by the simple addition of implicit blending, ShapeShop is capable of expressing a wide range of complex shapes. While the description here has been necessarily brief, the interested reader is referred to [35] and [33] for a more thorough discussion, including details on creating implicit sweeps with circular and arbitrary trajectories. The latter seems particularly useful in a sketch-based tool, although it has yet to be integrated into ShapeShop. We also note that a variety of other approaches to implicit inflation have been explored, based on blending point primitives [1], 3D variational interpolation [3, 20], and convolution surfaces [1, 42]. The main limitation with all of these methods, including the technique described here, is that they produce continuous fields which cannot represent any sharp corners in the sketch. We have proposed one solution for restricted cases [35], but the general problem remains unaddressed.

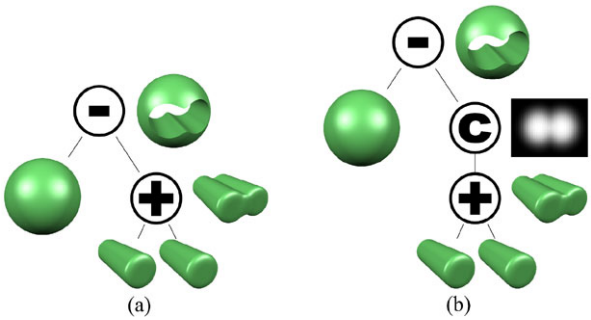

Fig. 11.19 In (**a**), two cylinder primitives are blended, and then subtracted from a sphere. In (**b**), a cache node is inserted above the blend node. Once filled, the cache short-circuits evaluation of the blend subtree, replacing an $O(m)$ traversal with an $O(1)$ tri-linear interpolation

11.3.4 Hierarchical Spatial Caching

One of the major limitations of hierarchical implicit modeling techniques like the BlobTree is that the complexity of the hierarchy grows with the complexity of the model. Hence, the recursive evaluations of the tree which are required to sample the scalar field defining the implicit volume become increasingly expensive. Unfortunately, implicit surface visualization methods rely on sampling the value and gradient of this field many times for each output mesh vertex. In profiling implicit surface polygonizers, we observed that for even moderately complex models, over 95% of the computation time is spent recursively evaluating the BlobTree. The cost of these evaluations must be reduced to ensure that the designer is not hampered by non-interactive visual feedback.

Inspired by promising results in [6], the *Hierarchical Spatial Caching* method was introduced [36] to address the interactivity problem. The fundamental idea behind this technique is to dynamically insert spatial caches into the BlobTree as *cache nodes*. These nodes approximate the scalar field of their subtree using a set of regular discrete samples which are computed as needed. This reduces the cost of evaluating the subtree from $O(m)$ to amortized $O(1)$ (Fig. 11.19).

Unlike previous approaches [6, 14], the sample values at grid vertices are not precomputed, but rather evaluated as needed (Fig. 11.20). This lazy evaluation provides a significant benefit, as full evaluation of high-resolution grids is computationally intensive. In addition, if surface-tracking visualization algorithms are used, only cache samples near the surface are required. In this case most of the samples in a fully evaluated grid will never be used—particularly if they will be invalidated in the next frame as the user drags a primitive across the screen.

The resolution of spatial caches is key to visual fidelity—too low a resolution, and the subtree's scalar field will not be adequately reconstructed, while oversampling results in wasted computation. In practice, we err on the side of caution and use a fixed grid resolution of 128^3. An obvious improvement would be to utilize adaptively-sampled grids, as in [14]. Unfortunately, adaptive methods have high initial overhead, which is impractical when the cache is being discarded each frame, and to date also lack even basic C^0 continuity (see [33] for details).

A related issue is the positioning of cache nodes, which should be sparsely distributed throughout the tree, ideally above subtrees which define semantic "parts".

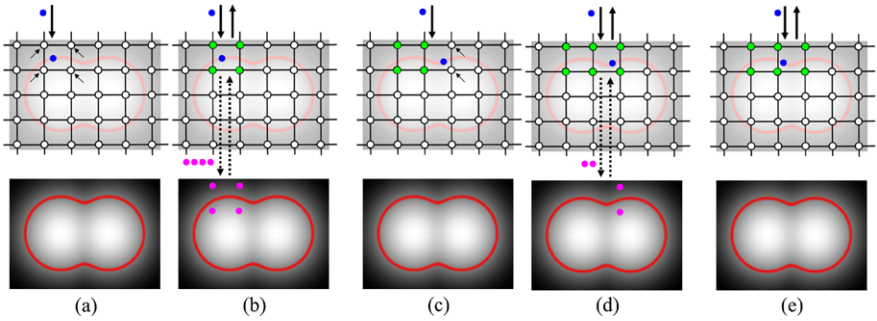

Fig. 11.20 In (**a**), the field values necessary to reconstruct the value at the incoming query point (in *blue*) are unavailable. The child field must be evaluated four times, once for each cache value (**b**). In (**c**), two cache values are missing and must be evaluated in the child field (**d**). Finally, in (**e**) all cache values are available. The incoming value query can be directly approximated in $O(1)$ time, no $O(m)$ evaluations of the child field are necessary

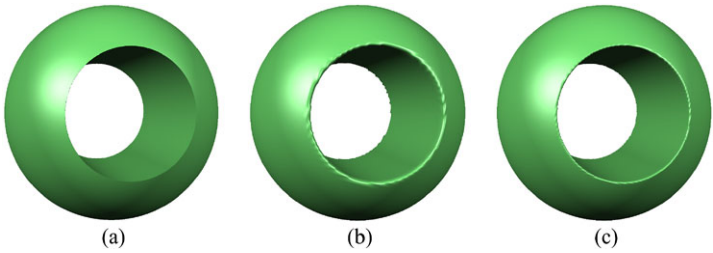

Fig. 11.21 Comparison of representation of a sharp edge without caching (**a**), with a cache resolution of 128^3 (**b**), and 256^3 (**c**). The Extended Marching Cubes polygonizer is used, producing a clear sharp edge in (**a**), but having no effect in (**b**) and (**c**) due to gradient smoothing at the crease

Since it is preferable that the designer not have to manually place cache nodes, ShapeShop uses simple heuristics to position cache nodes near the top of the tree. As with the sampling resolution, these ad-hoc solutions are effective in practice, but more principled approaches would be beneficial, and remain open problems.

In test cases simulating interactive modeling actions, hierarchical Spatial Caching results in an order-of-magnitude reduction in the computation necessary to triangulate a BlobTree model. This is a critical enhancement, making BlobTree modeling practical for use in interactive systems like ShapeShop. However, there are some drawbacks [33]. In particular, spatial approximation tends to smooth out sharp creases in the surface. This is a standard problem with surfacing implicit models, but recent polygonizers can use the field gradient to reconstruct sharp edges [21]. Unfortunately the interpolating filters to reconstruct smooth scalar fields from a sampled grid also smooth out the gradients, preventing sharp features from being recovered (Fig. 11.21). A solution to this problem will require the development of schemes which are sensitive to the properties of the scalar fields they are approximating.

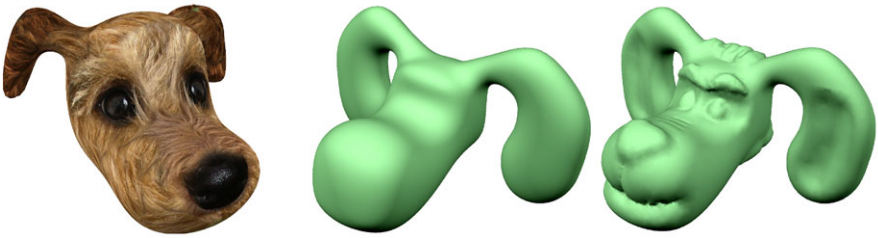

Fig. 11.22 Additions to the ShapeShop modeling system have included decal texturing (*left*) and mesh-based procedural surface editing layers (*right*)

11.4 The ShapeShop System

As with much of the research described in this book, one of our basic assumptions is that sketch-based interfaces will ultimately lead to more efficient and expressive 3D modeling tools. However, in our experience, working artists and designers have a hard time imagining how sketch-based tools could fit into their design pipelines. We have found that many artists do experiment with SBM research software that is publicly released, but these systems are regarded more as curiosities rather than real tools, as such systems (sensibly) tend to focus on demonstrating novelty rather than striving for production quality. Unfortunately, without user demand, the industrial 3D tool-makers with whom we have interacted remain skeptical of sketch-based modeling techniques. Hence, one of our goals with ShapeShop was to develop the system to the point where it could potentially be useful to real users.

ShapeShop was first made public at the SBIM Workshop in 2005, and included most of the techniques described here, as well as a few more which have since been removed. This was one of our early lessons—unlike systems with traditional interfaces, where one can always "just add another menu item", the designers of interfaces based on gestures and context-sensitivity must be much more self-critical, and willing to sacrifice infrequently-used tools if the system is to remain usable.

This initial version also lacked save/load capabilities, which severely limited the utility of the software. When demonstrating the system to artists, however, their biggest concern was texturing. Like many SBM systems, the output of ShapeShop is an unstructured triangle mesh, and hence manually assigning meaningful UV coordinates can be a time-consuming process. This lead to the development of decal texturing [38], shown in Fig. 11.22, which was released in ShapeShop V2 at SIGGRAPH 2006. This version also included saving and loading, making the system far more practical for real users.

Between the steady trickle of e-mail feedback, and posts discussing ShapeShop on web-based community forums, we have learned that working 3D designers are experimenting with ShapeShop in their professional workflows. Some of this exploration is purely artistic, such as the 3D sculptures displayed in Fig. 11.23, which were created by an artist who frequents the ShapeShop web forums. We have been contacted by bespoke jewelers, children's toy makers, elementary school teachers,

Fig. 11.23 3D sculptures created by first modeling in ShapeShop, and then importing the surface mesh into Modo [23] for texturing and rendering. Images ©Corien Klapwijk

and traditional-media artists who are using ShapeShop to experiment with 3D modeling. Much of this experimentation is short-lived, as the initial excitement tends to fade once the many limitations of the system become clear. Still, the comments we receive from users who have tried the software are almost uniformly positive, indicating a high level of interest in sketch-based modeling techniques.

There is one particular method in which 3D designers are integrating ShapeShop into their pipeline that warrants further explanation. As brush-based displacement painting systems like Z-Brush [29] and Modo [23] have become more capable, it is now common practice to build a basic model in traditional software, and then import it into these tools, where realistic levels of detail can be much more easily created. ShapeShop and other SBM software are quite effective in the initial blocking or massing stages, where they are more efficient than traditional control-point interfaces. We have attempted to encourage this workflow by adding mesh refinement tools to ShapeShop, as initial mesh quality is a necessity for these sculpting tools. However, it may be interesting to explore a more specific focus on this particular workflow, as it has significant implications for sketch-based modeling systems.

11.5 Discussion

In the previous sections, we have described the fundamental components of the ShapeShop sketch-based modeling system. From 2D drawing assistance to pen-based interaction to hierarchical, procedural shape representation, ShapeShop incorporates many aspects of our own research and that of others. By continually developing the software and releasing it “into the wild”, we have received extensive feedback from working 3D designers. This experience has given us much insight into the advantages and shortcomings of our work.

The use of a structured, procedural hierarchy to represent the sequence of sketched operations is perhaps the largest advantage of ShapeShop over its contemporaries. Not only does this permit greater complexity, it also allows designers to tweak and refine indefinitely. The response of 3D artists to this capability has been extremely positive. By utilizing implicit volumetric techniques, ShapeShop avoids the artificial distinction between "CAD-style" and "free-form" modeling that most tools make. The implicit approach also greatly enhances the ability to quickly explore a wide range of design variations (Fig. 11.24). Although iterative design is extremely common, few systems provide any specific support, and hence it is often very costly in 3D. Ultimately, we would like to reduce the burden on the designer when they wish to pick-and-choose from multiple variations.

Construction surfaces are another powerful feature of the procedural approach used in ShapeShop. Again, we have not designed in any specific support for construction surfaces, but are in the process of exploring how to do so. The use of a procedural hierarchy also introduces many new challenges. We have yet to find a straightforward way to even visualize the model tree in an intuitive and understandable way, let alone interact with it. In our informal observation of users, this is one of the most problematic areas of the system. Even computer graphics graduate students schooled in CSG techniques have trouble manipulating the model tree via an abstract tree-view widget.

Although ShapeShop simplifies many modeling tasks, the design space is ultimately constrained to shapes that can be practically constructed by blending sweeps and revolutions. Research is in progress to lift this limitation, such as the implicit push-and-pull deformations recently introduced [41]. The *Surface Tree* mesh-based procedural layered editing technique [34], has also been implemented in ShapeShop, providing powerful but non-implicit surface manipulation tools (Fig. 11.22).

Finally, a frequent comment from academia is that sketch-based modeling systems like ShapeShop must be proven through controlled evaluation, as has become standard practice for proposing novel interaction techniques in the field of human-computer interaction. However, those techniques can be reasonably tested in isolation, as the evaluation is largely based on human performance metrics. Such approaches are not applicable to evaluating the fitness of a complex SBM system, which in many cases integrates a wide range of novel interactions. Furthermore, we are not particularly interested in pure modeling speed, but rather a trade-off between efficiency and expressiveness, which is more difficult to measure. Direct comparisons to professional 3D modeling systems are also not particularly useful. Novice users must be trained for many hours to do anything productive in a complex modeling tool, while professional users have personal workflows so highly-optimized that any comparison to a new interface is hopelessly biased. And as there is so little in common between traditional and sketch-based interfaces, the most that can be learned from test subjects is personal preferences.

In some sense, we consider the public release of ShapeShop to be the ultimate test of usefulness. However, like the few attempts at evaluating SBM systems that have been performed [5, 19, 26], we have found that user response is essentially uniformly positive. Without variation, these data do not really tell us anything, except

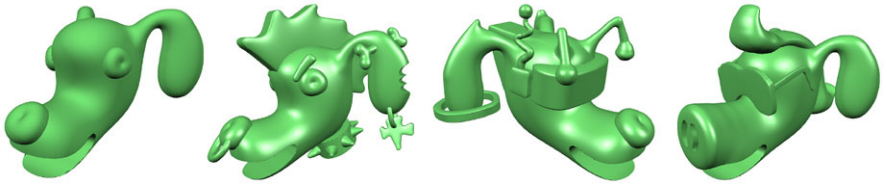

Fig. 11.24 Design iterations generated by adding detail to an initial base model (*left*). Volumetric implicit techniques free the designer from having to manage issues like model structure or discrete topology when exploring model variations. For example, the pig nose on the far right was simply drawn on top of the original dog nose

that the subjects have no basis for comparison and are probably only responding to the novelty of the system. We have also found that an initial positive response should not be taken to imply usefulness of the system in practice; once the novelty wears off, designers are apt to return to the tools they are more familiar with. Hence, we believe that the question of how to sensibly evaluate sketch-based modeling systems has yet to be answered, and is perhaps the most important open problem in the area.

References

1. Alexe, A., Gaildrat, V., Barthe, L.: Interactive modelling from sketches using spherical implicit functions. In: Proceedings of AFRIGRAPH 2004, pp. 25–34 (2004)
2. Apitz, G., Guimbretiére, F.: Crossy: a crossing-based drawing application. In: Proceedings of ACM UIST 2004, pp. 3–12 (2004)
3. Araújo, B., Jorge, J.: Blobmaker: Free-form modelling with variational implicit surfaces. In: Proceedings of 12th Encontro Português de Computação Gráfica (2003)
4. Autodesk Inc.: Autodesk Maya 2008 (2008). http://www.autodesk.com/maya
5. Bae, S.H., Balakrishnan, R., Singh, K.: ILoveSketch: as-natural-as-possible sketching system for creating 3d curve models. In: Proceedings of the ACM Symposium on User Interface Software and Technology (UIST), pp. 151–160 (2008)
6. Barthe, L., Mora, B., Dodgson, N., Sabin, M.: Interactive implicit modelling based on c^1 reconstruction of regular grids. International Journal of Shape Modeling **8**(2), 99–117 (2002)
7. Baudel, T.: A mark-based interaction paradigm for free-hand drawing. In: Proceedings of UIST '94, pp. 185–192 (1994)
8. Bloomenthal, J.: An implicit surface polygonizer. In: Graphics Gems IV, pp. 324–349. Academic Press, San Diego (1994)
9. Bloomenthal, J. (ed.): Introduction to Implicit Surfaces. Morgan Kaufmann, San Diego (1997). ISBN 1-55860-233-X
10. Carr, J.C., Beatson, R.K., Cherrie, J.B., Mitchell, T.J., Fright, W.R., McCallum, B.C., Evans, T.R.: Reconstruction and representation of 3d objects with radial basis functions. In: Proceedings of ACM SIGGRAPH 2001, pp. 67–76 (2001)
11. Cherlin, J.J., Samavati, F.F., Costa Sousa, M., Jorge, J.A.: Sketch-based modeling with few strokes. In: Proceedings of the Spring Conference on Computer Graphics (2005)
12. Crespin, B., Blanc, C., Schlick, C.: Implicit sweep objects. Computer Graphics Forum **15**(3), 165–174 (1996)
13. Fonseca, M.J., Ferreira, A., Jorge, J.A.: Towards 3d modeling using sketches and retrieval. In: Proceedings of the First Eurographics Workshop on Sketch-Based Interfaces and Modeling (2004)

14. Frisken, S., Perry, R., Rockwood, A., Jones, T.: Adaptively sampled distance fields: A general representation of shape for computer graphics. In: Proceedings of SIGGRAPH 2000, pp. 249–254 (2000)
15. Galbraith, C.: Modeling natural phenomena with implicit surfaces. PhD thesis, Department of Computer Science, University of Calgary (2005)
16. Igarashi, T., Hughes, J.F.: A suggestive interface for 3d drawing. In: Proceedings of ACM UIST 2001, pp. 173–181 (2001)
17. Igarashi, T., Matsuoka, S., Kawachiya, S., Tanaka, H.: Interactive beautification: a technique for rapid geometric design. In: Proceedings of ACM UIST '97, pp. 105–114 (1997)
18. Igarashi, T., Matsuoka, S., Tanaka, H.: Teddy: A sketching interface for 3d freeform design. In: Proceedings of ACM SIGGRAPH 99, pp. 409–416 (1999)
19. Kara, L.B., Shimada, K., Marmalefsky, S.D.: An evaluation of user experience with a sketch-based 3d modeling system. Computers & Graphics **31**(4), 580–597 (2007)
20. Karpenko, O., Hughes, J., Raskar, R.: Free-form sketching with variational implicit surfaces. Computer Graphics Forum **21**(3), 585–594 (2002)
21. Kobbelt, L.P., Botsch, M., Schwanecke, U., Seidel, H.P.: Feature-sensitive surface extraction from volume data. In: Proceedings of ACM SIGGRAPH 2001, pp. 57–66 (2001)
22. Lorensen, W.E., Cline, H.E.: Marching cubes: a high resolution 3d surface construction algorithm. Computer Graphics (Proceedings of SIGGRAPH 87) **21**, 163–169 (1987)
23. Luxology LLC.: Modo 302, July 2008. http://www.luxology.com
24. Nealen, A., Sorkine, O., Alexa, M., Cohen-Or, D.: A sketch-based interface for detail-preserving mesh editing. ACM Transactions on Graphics **24**(3), 1142–1147 (2005)
25. Nealen, A., Igarashi, T., Sorkine, O., Alexa, M.: Fibermesh: Designing freeform surfaces with 3d curves. ACM Transactions on Graphics **26**(3), 41–1419 (2007)
26. Oh, J.Y., Stuerzlinger, W., Danahy, J.: Sesame: Towards better 3d conceptual design systems. In: Proceedings of the 6th conference on Designing Interactive systems, pp. 80–89 (2006)
27. Olsen, L., Costa Sousa, M., Samavati, F.F., Jorge, J.: A taxonomy of modeling techniques using sketch-based interfaces. In: Eurographics 2008 STAR Reports (2008)
28. Owada, S., Nielsen, F., Nakazawa, K., Igarashi, T.: A sketching interface for modeling the internal structures of 3d shapes. In: Proceedings of the 4th International Symposium on Smart Graphics, pp. 49–57 (2003)
29. Pixologic Inc.: Zbrush 3.1 (2008). http://www.pixologic.com
30. Requicha, A.A.G.: Representations for rigid solids: theory, methods and systems. Computing Surveys **12**(4), 437–464 (1980)
31. Ricci, A.: A constructive geometry for computer graphics. Computer Graphics Journal **16**(2), 157–160 (1973)
32. Savchenko, V., Pasko, A., Okunev, O., Kunii, T.: Function representation of solids reconstructed from scattered surface points and contours. Computer Graphics Forum **14**(4) (1995)
33. Schmidt, R.: Interactive modeling with implicit surfaces. Master's thesis, Department of Computer Science, University of Calgary (2006)
34. Schmidt, R., Singh, K.: Sketch-based procedural surface modeling and compositing using surface trees. Computer Graphics Forum **27**(2), 321–330 (2008)
35. Schmidt, R., Wyvill, B.: Generalized sweep templates for implicit modeling. In: 3rd International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia (GRAPHITE 2005), pp. 187–196 (2005)
36. Schmidt, R., Wyvill, B., Galin, E.: Interactive implicit modeling with hierarchical spatial caching. In: Proceedings of International Conference on Shape Modeling and Applications (SMI 2005), pp. 104–113 (2005)
37. Schmidt, R., Wyvill, B., Costa Sousa, M., Jorge, J.A.: ShapeShop: Sketch-based solid modeling with blobtrees. In: Proceedings of the 2nd Eurographics Workshop on Sketch-Based Interfaces and Modeling, pp. 53–62 (2005)
38. Schmidt, R., Grimm, C., Wyvill, B.: Interactive decal compositing with discrete exponential maps. ACM Transactions on Graphics **25**(3), 605–613 (2006)
39. Schmidt, R., Isenberg, T., Jepp, P., Singh, K., Wyvill, B.: Sketching, scaffolding, and inking: a visual history for interactive 3d modeling. In: Proceedings of NPAR '07, pp. 23–32 (2007)

40. Schmidt, R., Singh, K., Balakrishnan, R.: Sketching and composing widgets for 3d manipulation. Computer Graphics Forum **27**(2), 301–310 (2008)
41. Sugihara, M., de Groot, E., Wyvill, B., Schmidt, R.: A sketch-based method to control deformation in a skeletal implicit surface modeler. In: Proceedings of SBIM 2008 (2008)
42. Tai, C.L., Zhang, H., Fong, J.C.K.: Prototype modeling from sketched silhouettes based on convolution surfaces. Computer Graphics Forum **23**(1), 71–83 (2004)
43. Turk, G., O'brien, J.F.: Modelling with implicit surfaces that interpolate. ACM Transactions on Graphics **21**(4), 855–873 (2002)
44. Wyvill, G., McPheeters, C., Wyvill, B.: Data structures for soft objects. Visual Computer **2**(4), 227–234 (1986)
45. Wyvill, B., Guy, A., Galin, E.: Extending the CSG Tree. Warping, blending and boolean operations in an implicit surface modeling system. Computer Graphics Forum **18**(2), 149–158 (1999)
46. Wyvill, B., Foster, K., Jepp, P., Schmidt, R., Costa Sousa, M., Jorge, J.A.: Sketch based construction and rendering of implicit models. In: Proceedings of the First Eurographics Workshop on Computational Aesthetics in Graphics, Visualization and Imaging 2005, pp. 67–74 (2005)
47. Yngve, G., Turk, G.: Robust creation of implicit surfaces from polygonal meshes. IEEE Transactions on Visualization and Computer Graphics **8**(4), 346–359 (2002)
48. Zeleznik, R.C., Herndon, K.P., Hughes, J.F.: SKETCH: an interface for sketching 3d scenes. In: Proceedings of ACM SIGGRAPH 96, pp. 163–170 (1996)