EUROGRAPHICS 2008 / G. Drettakis and R. Scopigno
(Guest Editors)

Volume 27 (2008), Number 3

Sketch-Based Procedural Surface Modeling and Compositing Using Surface Trees

Ryan Schmidt† and Karan Singh†

Dynamic Graphics Project
University of Toronto, Canada

Abstract

We present a system for creating and manipulating layered procedural surface editing operations, which is motivated by the limited support for iterative design in free-form modeling. A combination of sketch-based and traditional modeling tools are used to design soft displacements, sharp creases, extrusions along 3D paths, and topological holes and handles. Using local parameterizations, these edits are combined in a dynamic hierarchy, enabling procedural operations like linked copy-and-paste and drag-and-drop layer-based editing. Such dynamic, layered "surface compositing" is formalized as a Surface Tree*, an analog of CSG trees which generalizes previous hierarchical surface modeling techniques. By "anchoring" tree nodes in the parameter space of lower layers, our surface tree implementation can better preserve the semantics of an edit as the underlying surface changes. Details of our implementation are described, including an efficient procedural mesh data structure.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling

1. Introduction

Creating 3D models is a notoriously difficult task, in part because 3D modeling interfaces are so complex. One of the goals underlying much of the research in shape modeling is to make creating models more efficient. Although 3D design is largely a process of trial-and-error, most tools operate under the assumption that the designer will carry out editing operations sequentially. Operation n can be tweaked indefinitely, but becomes immutable once operation $n+1$ is initialized. "Undo" allows operation n to be modified, but only by discarding all following operations. This workflow results in much repeated work during design iterations.

Procedural modeling interfaces such as the ShapeShop system [SWSJ05] support a more efficient workflow by allowing the designer to "go back in time" and directly modify any offending editing operation. However, we have found that 3D artists use ShapeShop only to create coarse models, and prefer to add detail by exporting static meshes to traditional surface deformation tools. As a result, all the benefits of the procedural representation are lost. The system we describe here is motivated by the desire to support the direct surface editing tools that artists demand, in a procedural interface. Recent works such as Fibermesh [NISA07] and layered subdivision [WM07] also address procedural surface modeling, but neither supports the intuitive drag-and-drop layered surface editing that our system demonstrates.

Our main contribution is a system which fuses sketch-based interaction with a 3D analog of the intuitive *layer-based* metaphors found in 2D graphic design tools such as Adobe Illlustrator [Ado07]. With our interface, designers use sketches to layer large-scale edits, fine details, and even topological change onto an initial base surface. These edits are completely procedural - at any time the designer can manipulate parameters, copy-and-paste, or even drag edits along the surface (Figure 1). Layered updates attempt to preserve the intended "semantics" of edits - if operation n is modified, operation $n+1$ is "played back" relative to the new output of operation n.

† {rms | karan}@dgp.toronto.edu

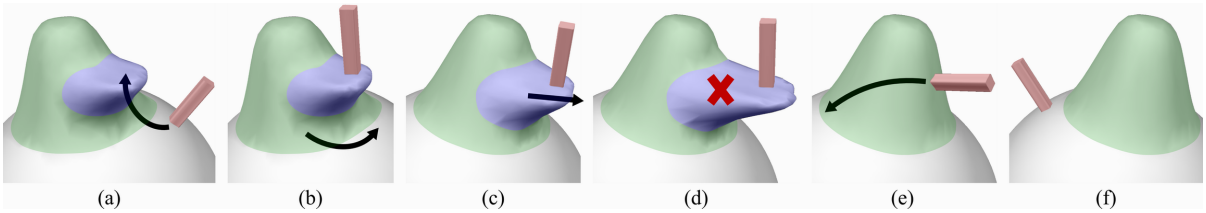

Figure 1: *In (a), an extrusion is dragged-and-dropped onto a layered displacement. The green displacement is dragged across the base surface (b), taking the overlapping elements with it. Then the blue edit is stretched (c) and deleted (d). The extrusion is automatically mapped back onto the green displacement (e), and then interactively dragged back onto the base surface (f).*

Interactive layered surface modeling has been studied primarily in the context of H-Splines [FB88] and Surface Pasting [BBF94], which are based on hierarchies of spline overlays. In Section 3 we generalize this concept to arbitrary manifolds, resulting in the *Surface Tree*. H-Splines are a specific instance of the Surface Tree in which all nodes share a global planar parameterization, which tends to result in surface distortion when layers are significantly deformed (Figure 2c). In Section 4 we describe an alternative approach to Surface Tree implementation suitable for meshes and point-set surfaces which does not assume the existance of a global parameterization, instead relying on a local parameterization at each node, resulting in significantly reduced distortion (Figure 2e). We also describe an efficient procedural mesh data structure (Section 5), details on our interactive techniques and sketch-based tools (Section 6), and a discussion of the major limitations of our approach (Section 7).

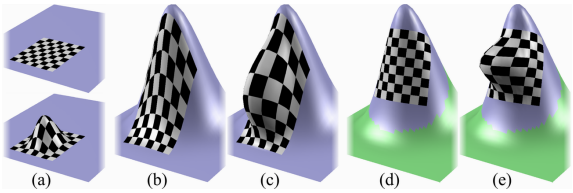

Figure 2: *In (a), a displacement "bump" is layered onto a planar NURBS patch. As in H-Splines, the intrinsic NURBS parameterization stretches as the planar patch is deformed (b), resulting in a deformed displacement (c). Dynamically computing a new local parameterization (d) reduces distortion in the layered edit (e).*

2. Background

Sketch-based tools simplify the 3D modeling process by allowing designers to leverage existing artistic ability. Early systems such as SKETCH [ZHH96] specified simple shapes by gestural commands. To support free-form modeling, the Teddy system [IMT99] directly inflated sketched 2D curves into 3D, an approach adapted in many later works [KH06, Ske07]. Recent profile-based techniques [LGS06] have expanded the range of sketchable shapes. Volumetric sculpting tools [GH91, AWC04, vFTS06] also leverage artistic talent in free-form modeling, and are capable of producing highly detailed models. However, much like physical sculpture, trial-and-error iterative design is quite time-consuming.

Procedural modeling techniques [Ebe02] such as hierarchical Constructive Solid Geometry, or CSG [RV83], support infinite design refinement, with sketching tools in development [JSC03]. Hierarchical implicit modeling [WGG99, ABGA04] extends CSG with blending and warping operators, and is the basis of the ShapeShop system [SWSJ05]. Volumetric spatial deformation [Bar84, SP86, SF98, LJ04] is widely used in commercial systems [Aut07] and is compatible with CSG-style procedural hierarchies.

A key limitation of volumetric techniques is that they do not easily support the intuitive notion of directly manipulating a 3D surface. In contrast, *surface deformation* techniques allow the designer to explicitly "push-and-pull" the 3D surface. Variational approaches [WW94, SCOL*04] have become quite popular (see [BS08] for a recent survey). However, introducing a dependency on the surface makes procedural composition non-trivial. The recent FiberMesh system [NISA07] fuses variational techniques with a global control curve network which can be arbitrarily refined, but layering and relative definition are not supported.

Most procedural surface deformation techniques are based on the notion of displacement [Coo84]. Originally developed to add surface detail during rendering, displacement mapping has evolved into powerful geometric texturing techniques [PKZ04, Elb05]. Applying geometric texture to locally-parameterized regions is essentially a mesh pasting technique [BMBZ02], and a recent extension [BMPB07] takes exactly this approach. Like our work, these approaches rely on local parameterizations, but none include an underlying dynamic procedural hierarchy.

Forsey and Bartel's seminal Hierarchical Splines [FB88] offered the first truly procedural approach to interactive surface editing. B-Spline *detail overlays* were layered onto a base patch and encoded using relative offset-vectors. These detail surfaces naturally deformed as underlying layers were interactively modified, but the shape of overlays was restricted by the base knot structure. Surface Pasting [BBF94]

addresses this limitation, by allowing arbitrary NURBS patches to be hierarchically pasted to a base spline patch.

Surface pasting was initially developed to quickly approximate layered spline displacement [BF93] by trimming holes in the base spline and simply positioning the "paste" surface in the trimmed hole. Although the result is only "ε-continuous", it is highly interactive, and has been applied in the commercial modeling package Houdini [Sid07]. While the initial system was limited to interaction in parameter space, an interface for directly positioning pasted surfaces has been developed [CMB97]. However, this "world-space" interaction is ultimately mapped into the parametric domain, exposing a critical limitation of surface pasting - it's reliance on the base patch parameterization. As underlying patches are manipulated, the shape of the pasted surface follows the (frequently un-intuitive) distortions that occur in parameter space [TM98]. The assumption of an underlying global planar parameterization also makes topology change problematic [SM03]. One of the key contributions of our work is to develop a framework which resolves these limitations.

Multiresolution surfaces [ZSS97, LDW97, GKSS02] adapt the H-Spline concept to mesh representations. However, the detail vector hierarchies used in these approaches are derived automatically, and thus lack the user-constructed semantics found in Surface Pasting and our Surface Trees. A notable exception is [WM07], in which a dependency graph of sketched subdivision curves and surfaces are inflated into 3D volumes. A layered subdivision "skin" blends these volumes together. While this approach is promising, procedural editing operations are constrained to mesh faces in the current subdivision hierarchy, preventing the intuitive drag-and-drop surface compositing provided by our system.

3. Defining Surface Trees

Hierarchical volumetric modeling techniques [RV83, WGG99] represent complex volumes using trees of *composition nodes*, with *primitive* shapes at the leaves. The power of these data structures is that any composition node or primitive can be trivially replaced with another. In this section we define an analogous structure for representing direct surface manipulations - a *Surface Tree*.

At a conceptual level, a surface editing operation replaces a bounded region $\mathcal{U}$ on a surface $\mathcal{S}$ with an open surface $\mathcal{V}$, where the boundary $\partial\mathcal{V}$ coincides with $\partial\mathcal{U}$. Each node $\mathcal{N}$ in our Surface Tree simply carries out this replacement:

$$\mathcal{N}(\mathcal{S},\mathcal{U},\mathcal{V}) = (\mathcal{S}\setminus\mathcal{U}) \quad \cup \quad \mathcal{V} \tag{1}$$

$\mathcal{N}$ can be thought of as a surface *compositing* operation, and hence a complex surface can be recursively defined by applying nodes to a *base surface* $\mathcal{B}$:

$$\mathcal{S}_{i+1} = \mathcal{N}_i(\mathcal{S}_i,\mathcal{U}_i,\mathcal{V}_i)$$
$$\mathcal{S}_1 = \mathcal{N}_0(\mathcal{B},\mathcal{U}_0,\mathcal{V}_0)$$

Intuitively, the final output surface is defined by incrementally *layering* a series of surface patches $\mathcal{V}_i$ onto $\mathcal{B}$. Although the recursion above is sequential in nature, any $\mathcal{V}_i$ can be defined by another series of compositions. Hence, a Surface Tree is a structured binary tree of composition nodes $\mathcal{N}$, with a *primary branch* that contains $\mathcal{B}$ as the initial leaf node, and a series of nested *secondary branches* which feed into the $\mathcal{V}_i$'s of the primary branch (Figure 3).

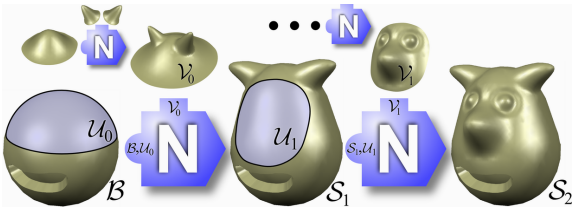

Figure 3: *A surface editing operation locally replaces some region $\mathcal{U}$ of a surface. In a Surface Tree, editing operations can be applied hierarchically, although* secondary *branches must always output open surfaces to be properly merged.*

Note that the arguments to $\mathcal{N}$ are not independent. $\mathcal{S}$ can be any surface, but it is necessary that $\mathcal{U} \subseteq \mathcal{S}$, and $\mathcal{V}$ must always be an open surface patch. This implies that only $\mathcal{B}$ can be a closed surface - all $\mathcal{V}_i$'s must have an open boundary (and hence so must all secondary branches).

In the equations above, we assumed that $\partial\mathcal{V} = \partial\mathcal{U}$. The easiest way to ensure this is to define $\mathcal{V} = \mathsf{E}(\mathcal{U})$, where E is a boundary-preserving editing operation. Although arbitrarily complex edits are possible, assume for now that E is a displacement map. The above definition allows $\mathcal{V}$ to be procedurally re-computed if $\mathcal{U}$ changes. However, if the designer changes $\mathcal{U}_0$ in Figure 3, $\mathcal{S}_1$ will change, and $\mathcal{U}_1$ will need to be procedurally re-computed (Figure 4).

To ensure maximum flexibility, It is desirable that the recomputation of $\mathcal{U}$ be as independent of $\mathcal{S}$ as possible. One way to accomplish this is to use the tools of Riemannian geometry [DoC94]. We restrict $\mathcal{S}$ to 2-manifolds embedded in $\mathbb{R}^3$, guaranteeing that any point on $\mathcal{S}$ has a local neighbourhood with disc-like topology. $\mathcal{S}$ is then covered with a finite *atlas* of topological discs, referred to as *coordinate patches*. A mapping $\mathcal{P}$ known as a *planar parameterization* exists between each 3D patch and $\mathbb{R}^2$. Given an atlas on $\mathcal{S}$, we can now *encode* $\mathcal{U}$ as a mapping from some 2D region $\mathbf{u}$ of the atlas parameter-space to the 3D surface. To simplify the following exposition, assume that $\mathcal{U}$ is contained within a single coordinate patch with parameterization $\mathcal{P}_\mathcal{U}$. Then we can write $\mathcal{U}(\mathbf{u}) = \mathcal{P}_\mathcal{U}(\mathbf{u})$, and rewrite Equation 1 as

$$\mathcal{N}(\mathcal{S},\mathbf{u},\mathsf{E}) \;=\; (\mathcal{S}\setminus\mathcal{U}(\mathbf{u})) \;\cup\; \mathsf{E}(\mathcal{U}(\mathbf{u})) \tag{2}$$

With this formulation, Surface Tree nodes can be procedurally re-computed because the editing region $\mathcal{U}$ is encoded

independent of the current 3D embedding of the input surface $\mathcal{S}$. Instead, it depends on the parameterization. Unfortunately, to build a practical system using arbitrary mesh or point-set representations, we must avoid assuming the existence of a global, consistent manifold parameterization as we have no way of maintaining such a manifold in real-time. This is problematic because we now lack a global embedding space in which to fix the location and shape of each layer. In the next section, we describe how we deal with this issue and implement Surface Trees in our interactive system.

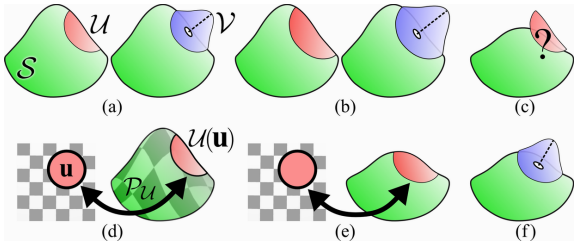

Figure 4: *In (a), a displacement is applied to $\mathcal{U} \subseteq \mathcal{S}$ to create an edited region $\mathcal{V}$. If $\mathcal{U}$ is changed, $\mathcal{V}$ can be re-computed (b), but how can $\mathcal{U}$ be re-computed if $\mathcal{S}$ changes (c)? One solution is to write $\mathcal{U}$ as a function of a 2D parameterization of $\mathcal{S}$ (d). Then if $\mathcal{S}$ is deformed, $\mathcal{U}$ can be re-computed from the parameterization (e), allowing $\mathcal{V}$ to be applied to the new surface (f).*

4. Implementing a Surface Tree

In Section 3 we described a mathematical formulation of a Surface Tree. In this section we provide details on our implementation, focusing on how the tree is constructed and updated. To begin, we assume that surfaces $\mathcal{S}$ are triangle meshes, although our approach is also applicable to point sets and arbitrary polygonal meshes.

Following Equation 1, a Surface Tree node $\mathcal{N}$ is a procedural operator which applies an editing operation to an input surface. In our system, the node operator is defined as

$$\mathcal{S}_{out} = \mathcal{N}(\mathcal{S}_{in}, \mathbf{u}, r, \mathsf{E}) \tag{3}$$

where $\mathbf{u}$ is a single point embedded in some parameterization, r is the radius of a geodesic disc which contains $\mathcal{U}$ (explained below), and E is a mesh editing operation.

Our basic approach is as follows. The primary branch of our Surface Tree begins with a base mesh $\mathcal{B}$ and incrementally applies procedural editing operations:

$$\mathcal{S}_{i+1} = \mathcal{N}_i(\mathcal{S}_i, \mathbf{u}_i, r_i, \mathsf{E}_i)$$
$$\mathcal{S}_1 = \mathcal{N}_0(\mathcal{B}, \mathbf{u}_0, r_0, \mathsf{E}_0)$$

Editing regions $\mathcal{U}$ are defined as parameterized triangle patches which approximate geodesic discs of radius r centered at *seed points* $\mathbf{p}$. These per-node parameterizations are re-used to approximate a global manifold - the *anchor point* $\mathbf{u}_i$ of each node is embedded in the parameterization computed for some upstream node $\mathcal{N}_{j<i}$. Hence, when evaluating Equation 3, we project $\mathbf{u}_i$ to the seed point $\mathbf{p}_i$ on the current surface $\mathcal{S}_i$ to find $\mathcal{U}_i$ (see Figure 5).

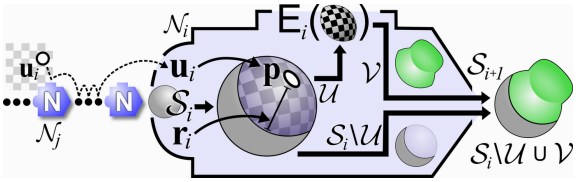

Figure 5: *To evaluate node $\mathcal{N}_i$, we first map the anchor point $\mathbf{u}_i$ forward from the upstream node $\mathcal{N}_{j<i}$ that it is embedded in to the seed point $\mathbf{p}$ on the input surface $\mathcal{S}_i$. Next we compute the editing region $\mathcal{U}$ by segmenting from $\mathcal{S}_i$ an approximate geodesic disc with radius r_i around $\mathbf{p}$, parameterize it, and generate the edited region $\mathcal{V}$. Finally $\mathcal{V}$ is combined with $\mathcal{S}_i \setminus \mathcal{U}$ to produce the output surface $\mathcal{S}_{i+1}$.*

Our procedural editing operations (Section 6) share a common foundation, in that they are defined as boundary-preserving modifications to one or more parameterized geodesic discs. Given a *seed point* $\mathbf{p}$ and geodesic radius r, we use Dijkstra's algorithm [Dij59] to segment an (approximate) geodesic disc from the mesh and parameterize it, creating the editing region $\mathcal{U}$ (Figure 6). While many parameterization algorithms are known [SPR06], we use the Discrete Exponential Map [SGW06] because it produces consistent distortion and is computed in-line with Dijkstra's algorithm.

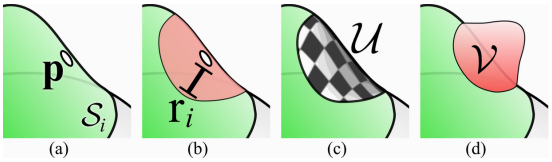

Figure 6: *Given a seed point $\mathbf{p}$ on surface $\mathcal{S}_i$ (a), we segment a geodesic disc with radius r_i (b), parameterize it (c), and apply a deformation (d).*

4.1. Node Anchoring and Updating

As noted, we lack a global atlas which can be used to fix the position of each editing layer. We do, however, have a parameterized support region computed for each editing operation. Hence, we can embed the seed point $\mathbf{p}$ of an edit as an *anchor point* $\mathbf{u}_i$ in the existing parameterization of some upstream node $\mathcal{N}_j$. This creates a dependency between $\mathcal{N}_i$ and $\mathcal{N}_j$ (Figure 7), so our Surface Tree is not strictly a tree, but rather a highly structured dependency graph [Hae88].

This anchor point approach inherently avoids the ambiguous case where an editing region $\mathcal{U}$ overlaps the boundaries

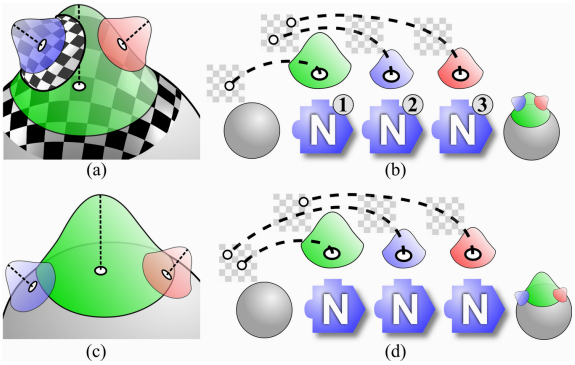

Figure 7: *Three bumps are layered onto a sphere. In (a) the parameterized edit regions for Nodes 1 and 2 are shown. Node 3 is anchored to Node 1 because it's seed point does not lie within Node 2's support (b). The potential ambiguity due to boundary overlaps in (c) is avoided because only the seed points are embedded in the parameterizations (d).*

of several underlying layers, as in Figure 7c. Since the anchor is a single point, it can be embedded in the parameter space of a single input node. This embedded anchor can always be mapped forward through the tree to unambiguously fix the position of the seed point using the algorithm listed in Figure 8. Note that as a parameterized point $\mathbf{u}$ has both 2D and 3D coordinates, here usage is determined by context.

$\mathbf{p} \leftarrow$ ProjectToSurface($\mathbf{u}_i$)
$k = j \ : \ \mathbf{u}_i \subset \mathcal{N}_j$
while k != i
$\mathbf{u} = \mathsf{E}_k(\mathbf{u})$
$k = k+1$
$\mathbf{p} = \mathbf{u}$

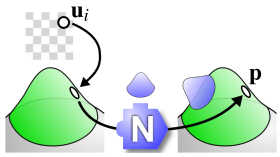

Figure 8: *The algorithm on the left maps the seed point* $\mathbf{u}_i$ *for the rightmost red node in Figure 7b from it's embedding in the green node to the output surface of the blue node.*

With these encoding and anchoring schemes in place, we can give an algorithmic description of how Equation 3 is implemented (Algorithm 1). The steps in this algorithm are equivalent to the visualization in Figure 5. Note that this algorithm only explicitly handles the primary branch. In our implementation, secondary branches generate arbitrary open meshes which are layered onto the primary branch using a geometric texturing editing operation, effectively "pasting" the output of the secondary branch onto the surface.

4.2. Surface Tree Manipulation

One advantage of our surface tree construction is that layers can be dynamically manipulated simply by interacting with

$\mathcal{S}_{i+1} \leftarrow \mathcal{N}(\mathcal{S}_i, \mathbf{u}_i, r_i, \mathsf{E}_i)$
$\mathbf{p} = ProjectToSurface(\mathbf{u}_i)$
$\mathcal{U} = GeodesicDisc(\mathbf{p}, r_i)$
$\mathcal{P}_\mathcal{U} = ExpMapParameterization(\mathcal{U})$
$\mathcal{S}_{i+1} = (\mathcal{S}_i \setminus \mathcal{U}) \quad \cup \quad \mathsf{E}_i(\mathcal{U}, \mathcal{P}_\mathcal{U})$

Algorithm 1: Computing the output of a Surface Tree node.

the anchor points using a surface-constrained handle. This 3D widget, which also incorporates components for rotating and scaling [SGW06], allows edits to be quickly "dragged-and-dropped", re-ordered, and deleted.

Interactive actions often require dynamic re-anchoring of nodes, for example if the user drags a seed point outside the support region of it's anchor node. Hence, each time node $\mathcal{N}$ is explicitly moved by the designer, we automatically re-anchor it's seed point in the "nearest" input node using Algorithm 2. We implement the inverse mapping E^{-1} in this algorithm by finding the triangle containing $\mathbf{p}$ and interpolating $\mathbf{u}$ from the triangle uv-coordinates.

$\mathbf{u} \leftarrow$ FindAnchorPoint($\mathbf{p} \in \mathcal{N}_i$)
$k = i - 1$
$\mathbf{u} = \mathsf{E}_k^{-1}(\mathbf{p})$
while $\mathbf{u} \notin \mathcal{U}_k$
$\mathbf{u} = \mathsf{E}_k^{-1}(\mathbf{u})$
$k = k - 1$

Algorithm 2: Finding the anchor point for a surface point.

Algorithm 2 is also called during tree manipulation. Before removing node $\mathcal{N}_i$, Algorithm 2 is used to transfer any seed points anchored in $\mathcal{N}_i$ to some input node $\mathcal{N}_{j<i}$, after which $\mathcal{N}_i$ can be safely removed by connecting it's input $\mathcal{N}_{i-1}$ to it's output $\mathcal{N}_{i+1}$. Similarly, to insert $\mathcal{N}_k$ between $\mathcal{N}_i$ and $\mathcal{N}_{i+1}$, any seed points anchored to $\mathcal{N}_i$ lying within the new support of $\mathcal{N}_k$ are "pushed forward" to $\mathcal{N}_k$.

Although it is not possible to perfectly anticipate the user's intent in all cases, we find that our approach gives the expected behavior most of the time. Essentially, if node A is anchored to node B, then when B is dragged across the surface, A will "stick" to it. This emulates the layer "grouping" found in other layer-based tools such as Illustrator [Ado07]. Automatic anchoring will only link A to B if the user explicitly drags A on to B - anchoring does not change if B is dragged underneath A. Note that the "layer order" is implicitly defined by the dependency structure of the Surface Tree, the user must explicitly re-order layers to, for example, move a small edit at layer n on top of a larger edit at layer $n+1$. Surface Pasting tried to avoid this situation by automatically re-ordering layers [BBF94], but there are many cases where this behavior can produce un-expected results, and it diverges from 2D layer-based interfaces [Ado07] where layer ordering is explicitly under user control.

5. A Procedural Mesh Data Structure

Our system operates on triangle meshes. To simplify notation, we will consider a mesh to be a simple list of triangles. Each node in our Surface Tree takes an input mesh $\mathcal{S}_{in}$, modifies it, and outputs a new mesh $\mathcal{S}_{out}$. As the edit region $\mathcal{U}$ is defined with respect to $\mathcal{S}_{in}$, each node is dependent on it's input mesh, making a full tree update $O(N)$ in the number of nodes. This can be reduced by storing all intermediate meshes - then if the designer edits node i, only nodes $j \geq i$ need be evaluated. Unfortunately, the overhead of copying and storing the mesh at each node is overwhelming.

Since our edits are locally supported, it is possible to construct a more efficient data structure for representing intermediate meshes. First we define two abstract mesh editing operations - *Mask* and *Weld*. *Mask* simply removes the triangle subset $\mathcal{U} \subset \mathcal{S}$ from $\mathcal{S}$ (Figure 9b), and, recalling our previous notation $\mathcal{V} = \mathsf{E}(\mathcal{U})$, *Weld* inserts $\mathcal{V}$ into the hole created by *Mask* (Figure 9d):

$$Mask(\mathcal{S},\mathcal{U}) = \mathcal{S} \setminus \mathcal{U} \qquad Weld(\mathcal{S},\mathcal{V}) = \mathcal{S} \cup \mathcal{V}$$

As previously noted, a procedural edit E must preserve the boundary of $\mathcal{U}$, to avoid introducing "cracks" between $\mathcal{S}$ and $\mathcal{V}$. Hence, assuming that $\mathcal{U}$ and $\mathcal{V}$ have been computed, these operators can be chained together:

$$\mathcal{S}_{out} = Weld(\, Mask(\mathcal{S}_{in},\mathcal{U}) \,,\, \mathcal{V} \,)$$

In our system, the geometry of $\mathcal{S}$ is accessed through iterators, so *Mask* and *Weld* can be implemented as iterators which either skip certain triangles (*Mask*) or iterate over multiple meshes (*Weld*). With this approach, a node never modifies $\mathcal{S}_{in}$, but rather generates an iterator which masquerades as a manifold mesh by transparently hiding the triangles in $\mathcal{U}$ and inserting $\mathcal{V}$. Clearly, *Mask* and *Weld* can be applied recursively, creating a procedural mesh data structure.

In practice, our mesh is not simply a list of triangles, but an efficient vertex/edge/face manifold representation. This complicates the implementation of *Weld*, as it must transparently re-write the incoming and outgoing indices of boundary vertices and edges, to preserve the outward appearance of a manifold mesh. Note that *Mask* and *Weld* can be applied to point set surfaces, where the implementation is simplified because the explicit boundary re-writing is unnecessary.

Since *Mask* and *Weld* do not copy or modify $\mathcal{S}_{in}$, they are highly efficient even when applied to large meshes. They do, however, add overhead to mesh iterations, which can limit interactivity as the surface tree grows. Hence, we have found it useful to occasionally cache a full copy of $\mathcal{S}_{in}$ at a node. This cache simply copies the geometry produced by the incoming *Mask*/*Weld* iterators into a single manifold mesh, which is much more efficient to iterate over. In particular, if the user selects $\mathcal{N}_i$ for editing, we cache the output of $\mathcal{N}_{i-1}$ to ensure that interactive feedback is as fast as possible.

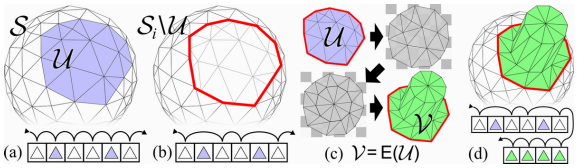

Figure 9: *The* Mask *operator creates a hole in $\mathcal{S}$ by hiding triangles in the editing region $\mathcal{U}$ (a) during mesh iterations (b). Edits* E *generate $\mathcal{V}$ by copying and modifying $\mathcal{U}$ (c), which often involves mapping to uv-space for re-meshing. Finally,* Weld *synthesizes a manifold mesh by transparently combining* Mask$(\mathcal{S},\mathcal{U})$ *and $\mathcal{V}$ during iteration (d).*

6. Surface Tree Creation and Interaction

Our Surface Tree editor is implemented as an extension to the ShapeShop modeling system [SWSJ05]. Similar to other sketch-based modeling tools, ShapeShop's workflow involves a mixture of sketch-based and traditional interfaces. A suggestion-list interface [IH01] allows the user to create and modify a variety of edits based on sketched curves. Other parameters are controlled using 2D and 3D widgets.

Visualizing and interacting with the existing Surface Tree is a difficult task, one which we have only begun to explore. To select an existing node for manipulation, the user clicks on it's support region. If multiple nodes lie under the cursor, repeated clicking cycles through them. The selected node is hilighted, and all overlapping layers are rendered transparently (Figure 11b). As the user manipulates the selection, overlapping nodes are dynamically re-evaluated and rendered. We limit re-evaluation of overlapping layers to maintain interactivity, the full tree update is deferred until the user finishes with the manipulation (Figure 11d).

We now briefly describe the editing tools available in our system. As previously noted, these tools operate on parameterized approximate geodesic discs segmented from the mesh, which correspond to the *decal* parameterizations of [SGW06]. Generally, edits take one or more user-sketched strokes as additional parameters. Since our strokes are represented by spline curves, and displacement offsets by func-

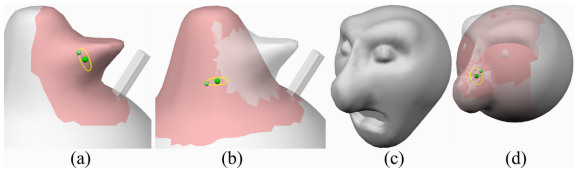

Figure 11: *The support region of an edit node is highlighted when it is selected (a), and all overlapping layers are rendered as transparent (b). Computation of layers in (c) can be deferred as the user manipulates an underlying layer (d) to maintain a minimum interactive frame-rate.*

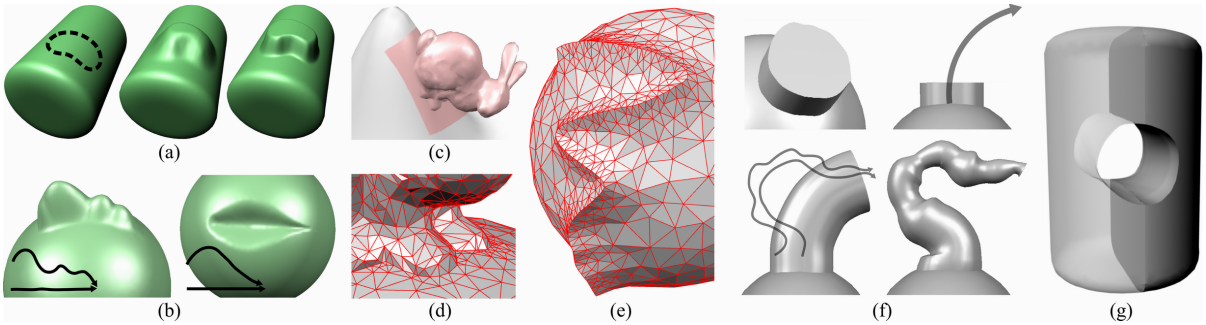

Figure 10: *Our system supports a variety of editing operations. Drawing a closed loop on the surface specifies a region for displacement with varying falloff strength (a). Surface curves also can specify displacement (b), with an optional height curve (inset strokes have been shifted). Arbitrary open meshes can be dynamically stitched into the mesh (c), creating a watertight surface (d). Edges of sketched creases are also inserted into the mesh, to preserve sharpness (e). Displacements can be pulled along 3D curves, and optional profile curves produce arbitrary generalized cylinders (f). Multiple surface patches can also be connected to create dynamic holes and handles (g).*

tional scalar fields, our edits can be computed at arbitrary resolution, allowing high-quality surfaces to be generated.

6.1. Sketched Displacement

Displacement mapping [Coo84] is one of the simplest types of procedural surface manipulation, and displacement-painting tools [LF03] have become popular in commercial systems [Pix07]. Although here we take a sketch-based approach, our techniques would be a useful addition to these systems, as displacements painted into procedural decals can be easily moved or later modified.

In our system, arbitrary surface regions can be displaced by sketching a closed loop around the desired area (Figure 10a). The displacement offset is based on a smoothed approximation to the 2D distance field of the region contour [PKZ04, SWSJ05], allowing for a continuous range of transition smoothness. Sharp transition edges are created by inserting the sketched polyline directly into the mesh using constrained Delaunay triangulation [She96].

Sketched open surface curves can also be used to displace the surface, with an optional second curve being used to modulate the displacement height (Figure 10b). The width can also be tied to height, producing a variable-width displacement reminiscent of the area-proportional inflation used in the Teddy system [IMT99]. The scalar displacement maps are created by accumulating radial 2D fields at regular intervals along the sketched curve. The combined field is defined as $\max_i(h_i * max(1-(d_i/r_i)^2)^3, 0))$, where d_i is the distance to the origin of the ith field, r_i is it's radius, and h_i the height. To create sharp creases, we remove the square on the $(d/r)^2$ term, producing a sharp, "inverted" falloff region. To accurately reproduce the crease, we directly insert the sketched curve into the mesh, again using Delaunay triangulation in parameter space (Figure 10e).

Displacements can also be extended along curved paths (Figure 10f), supporting the construction of larger-scale edits. The path can be dynamically edited by re-sketching, and an optional second curve can be used to define a profile function, turning the displacement into a flexible generalized cylinder. Note that the profile width must be blended with the orignal width near the base of the edit to ensure a smooth transition. Scalar parameters control this transition region, and explicit tapering control is also available.

6.2. Holes and Handles

In addition to arbitrary sketched displacements, our implementation supports topology-changing operations such as construction of holes and handles. This requires two decals, each of which create an opening in the manifold. These 2D holes are connected together with a generalized cylinder, resulting in a 3D topological hole or handle (Figure 12a-e). This manifold-stitching approach can be applied to arbitrary

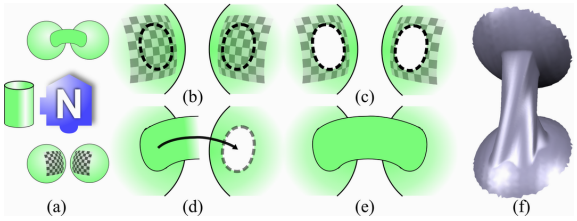

Figure 12: *A Handle node inserts an open "tube" into the surface (a) based on two parameterized patches (b), by opening two holes in the manifold (c) and mapping the tube geometry along a path between them (d,e). This approach does not rely on any notion of interior vs. exterior, allowing arbitrary non-closed surfaces to be procedurally joined (f).*

non-closed surfaces (Figure 12f), as it does not require a consistent inside/outside spatial partitioning.

To create a handle, the user first sketches an area displacement, and then draws a line from it to another point on the surface. Selecting the resulting suggestion icon creates a hole or handle with the same contour at both ends. Alternately, the user can draw a second contour, in which case we interpolate between the two along the handle path.

6.3. Linked Copy and Paste

Similar to [BMPB07], our local parameterizations can be used to paste arbitrary open meshes as geometric texture (Figure 10c). Tree nodes can also be copied, including selective copying of edits lying "underneath" higher layers. A more significant advantage is that copied nodes can be *linked*, such that when parameters of one node are modified, linked copies are automatically updated (Figure 13).

In some cases, it can be useful to link some parameters but not others (Figure 13d). It is even possible to link parameters between arbitrary nodes - one can imagine a whole set of edits whose parameters are driven by a few simple controls. A sensible interface for constructing linked parameter networks is one direction for future research.

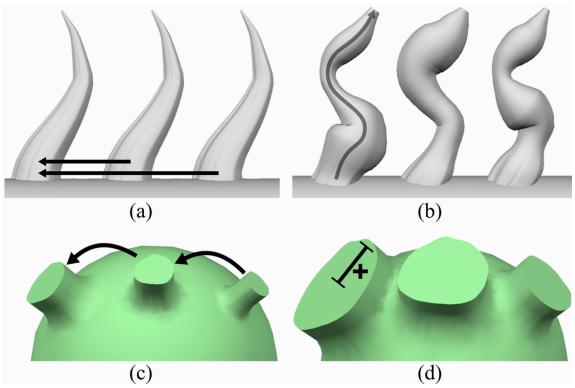

Figure 13: *The copies in (a) are linked to the shape of the parent (leftmost) edit. In (b) only the path curve is linked, allowing for different profile curves. The radius of each bump in (c) is defined as* 0.75 *times the radius of the bump to it's left, allowing the leftmost edit to control all three (b).*

7. Limitations

While our interactive system demonstrates the potential of Surface Tree modeling, there is extensive room for improvement. One major limitation is the use of the discrete exponential map (DEM) to parameterize edit regions, as it frequently breaks down when crossing regions of widely varying curvature [SGW06]. However, more robust algorithms [SPR06] tend to "wobble" as the underlying mesh changes, whereas parametric distortion in the DEM remains very consistent, resulting in an interactive response that feels more rigid. Improved parameterization algorithms would immediately increase the range of our system.

Another limitation is that the response time of the system varies depending on which edit is being manipulated, as the computational cost of a tree update depends on the layer depth of the modified node. The cost of updating a node is highly variable - major factors include the resolution of the input mesh, the size of the node support region, and the level of refinement necessary to faithfully represent the edit. Generally, the 3-5 edits at the top of the tree are highly interactive at moderate resolutions. Beyond that depth, some action must be taken to guarantee real-time feedback. Currently we use the partial-update scheme described in Section 6, which can limit the designer's ability to see overlapping layers deforming. An alternative is to compute edits at lower resolution during interaction, but the loss of fidelity is undesirable. We note that the figures in this paper are all taken at interactive mesh resolutions. "Production" surfaces can be generated by adapting the resolution at each node to ensure visual fidelity, but this generally precludes real-time feedback.

The cost of tree updates is exacerbated by the dependency structure of our tree. If node A is an input to node B, it will *always* be applied before B. Hence, each "layer" contains only a single edit, and each must be sequentially re-evaluated to avoid editing region conflicts. However, the editing regions of A and B are often disjoint. In this case, as B is not actually dependent on A, they could be computed in parallel. Implementing this optimization requires an automatic surface partitioning, but will significantly enhance the user experience.

8. Results and Discussion

The primary goal of this work is to increase the power of surface modeling tools available to designers, by allowing them to "go back in time" and non-destructively modify any modeling decisions made in the past. The model in Figure 14 demonstrates our progress towards this goal. The base shape

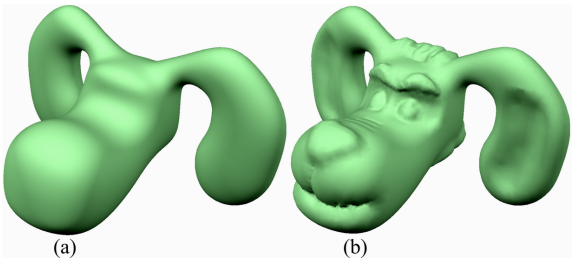

Figure 14: *A Surface Tree can be layered on top of a simple BlobTree model (a) to increase surface detail (b). The level of fine detail in (b) is quite difficult to create using ShapeShop's implicit surfaces.*

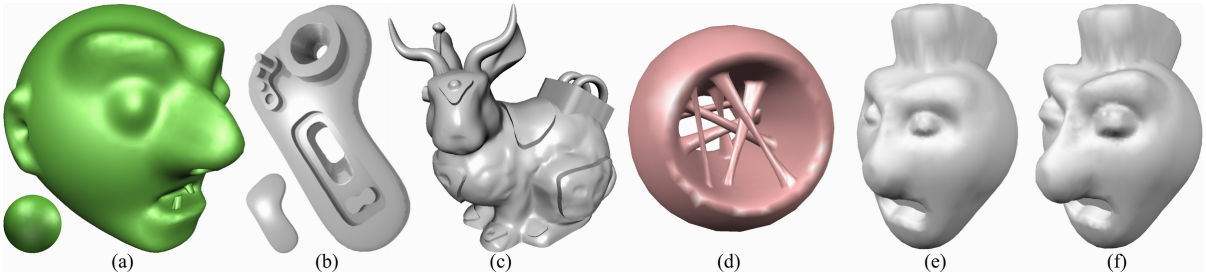

Figure 15: *Using our Surface Tree modeling system, procedural details can be layered on simple shapes to create organic (a) and mechanical (b) models. We can also edit existing surfaces (c), and quickly construct shapes with high genus (d). Since the models are fully procedural, the designer can manipulate underlying features at any time (e,f).*

is an implicit volume sketched in ShapeShop, onto which we have layered significantly more detail using our Surface Tree interface. This approach results in a 3D model which is completely procedural - each surface edit, as well as the elements of the base surface, can be interactively modified.

Although our current system has many limitations, we have found that this approach of compositing layered sketched edits onto a sketched base surface is quite effective for quickly producing 3D models. By combining traditional geometric modeling techniques with dynamic surface parameterization, our system is capable of creating and manipulating a wide range of 3D surfaces (Figure 15). In particular, layered displacements allow for the creation of levels of surface detail that have not been demonstrated in previous sketch-based systems, and sketched holes and handles enable procedural topology change without resorting to a volumetric approach. The underlying procedural hierarchy allows designers to efficiently explore design variations without having to discard existing work. Linked copy-and-paste simplifies many repetitive modeling tasks and we have begun to explore other ways of exploiting procedural techniques in the interface. One interesting direction is to attempt to incorporate CAD-style parametric modeling techniques into the surface editing domain.

One untapped benefit of our system is that our implementation techniques can also be applied to point-set surfaces [ZPKG02]. The key components - Dijkstra's algorithm, the Discrete ExpMap parameterization [SGW06], and our procedural mesh data structure - can all be applied directly to point sets. A Surface Tree created using our mesh-based interface could even be "played back" on a point-set surface. A point-set implementation is under development.

Another aspect of Surface Trees yet to be explored is computer animation. Procedural models are trivial to animate, and linked node parameters could be an efficient way to animate many effects. The ability to dynamicaly manipulate layered surface geometry may be particularly beneficial in this domain.

Acknowledgements

The authors wish to thank the anonymous reviewers, Cindy Grimm, Christian Lessig, and the rest of the DGP for their invaluable discussions and feedback. This work was funded by MITACS and NSERC.

References

[ABGA04] ALLÈGRE R., BARBIER A., GALIN E., AKKOUCHE S.: A hybrid shape representation for free-form modelling. In *Proc. Shape Modeling International* (2004).

[Ado07] ADOBE SYSTEMS INC.: Adobe Illustrator, 2007. www.adobe.com/illustrator.

[Aut07] AUTODESK INC.: Maya 8.5, September 2007. http://www.autodesk.com/maya.

[AWC04] ANGELIDIS A., WYVILL G., CANI M.-P.: Sweepers: Swept user-defined tools for modeling by deformation. In *Proc. Shape Modeling International* (2004).

[Bar84] BARR A. H.: Global and local deformations of solid primitives. In *Proc. SIGGRAPH 84* (1984), pp. 21–30.

[BBF94] BARGHIEL C., BARTELS R., FORSEY D.: Pasting spline surfaces. In *Mathematical Methods for Curves and Surfaces* (1994), pp. 31–40.

[BF93] BARTELS R. H., FORSEY D. R.: *Spline Overlay Surfaces*. Tech. Rep. TR-93-48, Univ. British Columbia, 1993.

[BMBZ02] BIERMANN H., MARTIN I., BERNARDINI F., ZORIN D.: Cut-and-paste editing of multiresolution surfaces. *ACM Trans. Graph. 21*, 3 (2002), 312–321.

[BMPB07] BRODERSEN A., MUSETH K., PORUMBESCU S., BUDGE B.: Geometric texturing using level sets. *IEEE Trans. Vis. Comp. Graph.* (2007). to appear.

[BS08] BOTSCH M., SORKINE O.: On linear variational surface deformation methods. *IEEE Trans. Vis. Comp. Graph. 14*, 1 (2008), 213–230.

[CMB97] CHAN K., MANN S., BARTELS R.: World space surface pasting. In *Proc. Graph. Interface '97* (1997), pp. 146–154.

[Coo84] COOK R. L.: Shade trees. In *Proc. SIGGRAPH 84* (1984), pp. 223–231.

[Dij59] DIJKSTRA E.: A note on two problems in connexion with graphs. *Numerische Mathematik 1* (1959), 269–271.

[DoC94] DOCARMO M.: *Riemannian Geometry*. Birkhauser, 1994.

[Ebe02] EBERT D. S. (Ed.): *Texturing and Modeling: A Procedural Approach*. Morgan Kaufmann, 2002. ISBN 1558608486.

[Elb05] ELBER G.: Geometric texture modeling. *IEEE Comput. Graph. Appl. 25*, 4 (2005), 66–76.

[FB88] FORSEY D., BARTELS R.: Hierarchical b-spline refinement. In *Proc. SIGGRAPH 88)* (1988), pp. 205–212.

[GH91] GALYEAN T., HUGHES J.: Sculpting: An interactive volumetric modeling technique. In *Proc. SIGGRAPH 91)* (1991), pp. 267–274.

[GKSS02] GUSKOV I., KHODAKOVSKY A., SCHRODER P., SWELDENS W.: Hybrid meshes: multiresolution using regular and irregular refinement. In *Proc. Symp. Comp. Geom.* (2002), pp. 264–272.

[Hae88] HAEBERLI P.: Conman: a visual programming language for interactive graphics. *Proc. SIGGRAPH '88* (1988), 103–111.

[IH01] IGARASHI T., HUGHES J. F.: A suggestive interface for 3d drawing. In *Proc. UIST '01* (2001), pp. 173–181.

[IMT99] IGARASHI T., MATSUOKA S., TANAKA H.: Teddy: a sketching interface for 3d freeform design. In *Proc. SIGGRAPH '99* (1999), pp. 409–416.

[JSC03] JORGE J., SILVA N., CARDOSO T.: Gides++. In *Proc. 12th Encontro Português de Computação Gráfica* (2003).

[KH06] KARPENKO O., HUGHES J.: Smoothsketch: 3d free-form shapes from complex sketches. *ACM Trans. Graph. 25*, 3 (2006), 589–598.

[LDW97] LOUNSBERY M., DEROSE T. D., WARREN J.: Multiresolution Analysis for Surfaces of Arbitrary Topological Type. *ACM Trans. Graph. 16*, 1 (1997), 34–73.

[LF03] LAWRENCE J., FUNKHOUSER T.: A painting interface for interactive surface deformations. In *Proc. Pacific Graphics 2003* (2003), pp. 141–150.

[LGS06] LEVET F., GRANIER X., SCHLICK C.: 3d sketching with profile curves. In *Intl. Symp. Smart Graphics* (2006).

[LJ04] LEWIS T., JONES M. W.: A system for the non-linear modelling of deformable procedural shapes. *Journal of WSCG 12*, 2 (2004), 253–260.

[NISA07] NEALEN A., IGARASHI T., SORKINE O., ALEXA M.: Fibermesh: designing freeform surfaces with 3d curves. *ACM Trans. Graph. 26*, 3 (2007).

[Pix07] PIXOLOGIC, INC.: ZBrush 3.1, September 2007. http://www.pixologic.com/zbrush/.

[PKZ04] PENG J., KRISTJANSSON D., ZORIN D.: Interactive modeling of topologically complex geometric detail. *ACM Trans. Graph. 23*, 3 (2004), 635–643.

[RV83] REQUICHA A. A. G., VOELCKER H. B.: Solid modeling: Current status and research directions. *IEEE Comp. Graph. & Appl. 3* (1983), 25–37.

[SCOL*04] SORKINE O., COHEN-OR D., LIPMAN Y., ALEXA M., RÖSSL C., SEIDEL H.-P.: Laplacian surface editing. In *Proc. Symp. Geom. Proc.* (2004), pp. 175–184.

[SF98] SINGH K., FIUME E. L.: Wires: A geometric deformation technique. In *Proc. SIGGRAPH 98* (1998), pp. 405–414.

[SGW06] SCHMIDT R., GRIMM C., WYVILL B.: Interactive decal compositing with discrete exponential maps. *ACM Transactions on Graphics 25*, 3 (2006), 605–613.

[She96] SHEWCHUK J. R.: Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, Lin M. C., Manocha D., (Eds.), vol. 1148 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996, pp. 203–222.

[Sid07] SIDE EFFECTS SOFTWARE INC.: Houdini 9, September 2007. http://www.sidefx.com.

[Ske07] Sketch-based interfaces: techniques and applications, 2007. ACM SIGGRAPH 2007 courses.

[SM03] SIU S., MANN S.: Computer aided ferret design. In *Proc. Geometric Modeling and Graphics* (2003), pp. 195–200.

[SP86] SEDERBERG T. W., PARRY S. R.: Free-form deformation of solid geometric models. In *Proc. SIGGRAPH 86* (1986), vol. 20, pp. 151–160.

[SPR06] SHEFFER A., PRAUN E., ROSE K.: *Mesh Parameterization Methods and their Applications*. Now Publishers, 2006.

[SWSJ05] SCHMIDT R., WYVILL B., SOUSA M. C., JORGE J. A.: Shapeshop: Sketch-based solid modeling with blobtrees. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling* (2005), pp. 53–62.

[TM98] TSANG C., MANN S.: *Animated Surface Pasting*. Tech. Rep. CS-98-19, University of Waterloo, 1998.

[vFTS06] VON FUNCK W., THEISEL H., SEIDEL H.-P.: Vector field based shape deformations. *ACM Trans. Graph. 25*, 3 (2006), 1118–1125.

[WGG99] WYVILL B., GUY A., GALIN E.: Extending the CSG Tree. warping, blending and boolean operations in an implicit surface modeling system. *Comp. Graph. Forum 18*, 2 (1999), 149–158.

[WM07] WANG H., MARKOSIAN L.: Free-form sketch. In *Proc. Sketch Based Interaction and Modeling* (2007).

[WW94] WELCH W., WITKIN A.: Free-form shape design using triangulated surfaces. In *Proc. SIGGRAPH 94* (1994), pp. 247–256.

[ZHH96] ZELEZNIK R. C., HERNDON K. P., HUGHES J. F.: Sketch: An interface for sketching 3d scenes. In *Proc. SIGGRAPH 96* (1996), pp. 163–170.

[ZPKG02] ZWICKER M., PAULY M., KNOLL O., GROSS M.: Pointshop 3d: An interactive system for point-based surface editing. *ACM Trans. Graph. 21*, 3 (2002), 322–329.

[ZSS97] ZORIN D., SCHRÖDER P., SWELDENS W.: Interactive multiresolution mesh editing. In *Proc. SIGGRAPH 97* (1997), pp. 259–268.