Surface Trees: Interactive Hierarchical Surface Modeling (papers_0295)

Ryan Schmidt and Karan Singh, University of Toronto

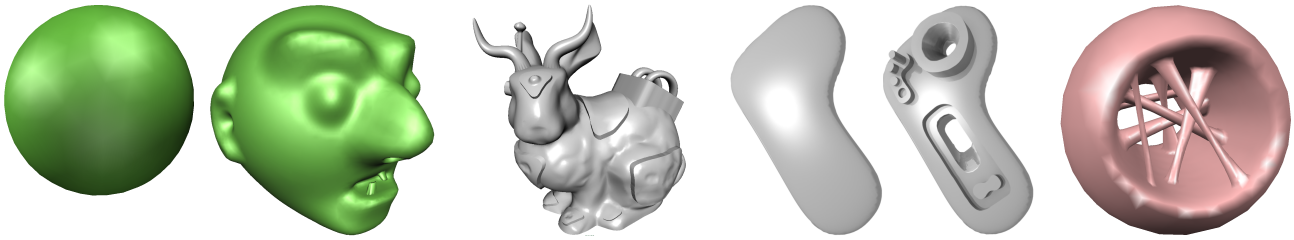

Figure 1: Surface trees are procedural 3D models composed of hierarchical layered surface deformations. The examples shown here were created in our interactive surface tree editor, each taking less than 30 minutes. Surface trees support smooth deformation, sharp extrusion, and fast hole and handle topology-changing tools. The surface tree is a full construction history - any previous operation can be interactively manipulated, overlapping deformations are automatically recomputed.

Abstract

A method is described for representing and manipulating a hierarchy of surface editing operations, in the context of an interactive shape modeling tool. Surface deformations are cast as dynamic geometric textures, applied to locally-parameterized regions of the surface which can be interactively manipulated, and also layered. This use of dynamically layered deformation is characterized as "surface compositing", and leads to the definition of the *surface tree* - a hierarchical, procedural representation of a 3D surface. Like CSG trees, surface trees allow any deformation to be manipulated at any time. Editing operations higher in the model tree are automatically recomputed, with relative position of layered elements maintained by "anchoring" them in the parameter space of lower layers. In addition to surface deformation, dynamic holes and handles can be created between (possibly non-manifold) surfaces. These techniques are demonstrated in an interactive "drag-and-drop" mesh editing system. To efficiently implement this system, a novel procedural mesh data structure is described.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction Techniques

Keywords: surface modeling, procedural modeling, decal parameterization

1 Introduction

Research in geometric modeling continues to battle with the challenging question: "How quickly and effectively can a designer transform a mental concept into a digital object, that is easy to refine and reuse?". While a large body of existing work addresses the efficient construction geometric models [Igarashi et al. 1999; Schmidt et al. 2005; Zwicker et al. 2002], less attention has been given to model refinement and reuse. Concepts of construction history, layering and parametric or procedural editing make refinement and reuse simple. While these paradigms are commonplace for editing naturally parameterized signals such as animation curves [Autodesk Inc. 2007] or images[Adobe Systems Inc. 2007], they have had limited success with general 3D surface representations. We address this problem by developing a framework for procedural surface modeling based on locally layered 2D parameterizations, which are used to generate 3D surface geometry.

The concept of three-dimensional geometry modification generated based on 2D parameterizations can be traced back to displacement mapping [Cook 1984], which is widely recognized as an efficient method of increasing geometric surface detail. Recent extensions such as geometric texture [Elber 2005] and shell mapping [Porumbescu et al. 2005] allow surfaces to be tiled with arbitrary geometric detail, greatly enhancing visual complexity.

Similar to traditional texture mapping, geometric texturing has been cast as a stage in the rendering pipeline. A geometric texture is a surface attribute, with the necessary geometry being generated on-the-fly during rendering [Elber 2005]. However, geometric texture can also be considered a modeling operation. Recent developments in surface parameterization have produced the discrete exponential map [Schmidt et al. 2006], an efficient algorithm for parameterizing small regions of the surface. These *decal parameterizations* can be used to apply geometric texture locally, resulting in a "drag-and-drop"-style surface deformation interface.

One issue that immediately occurs in this interface is that decal parameterizations may overlap. Since geometric textures contain arbitrary geometry, any sort of automatic merging is problematic. Another solution is to impose a layer ordering on the decals. Hence, decals with a higher layer order are applied on top of previous geometric texture. This layer ordering introduces a significant conceptual difference - geometric texture is no longer simply a means for adding surface detail. Instead, it has become a more fundamental modeling operation.

Traditionally, surface modeling interfaces have been destructive in nature. Overlapping deformations or other manipulations are ap-

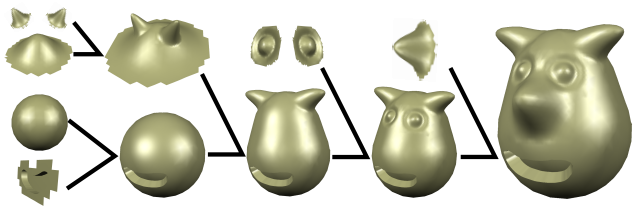

Figure 2: A *surface tree* is a procedural definition of a complex surface, created by compositing a series of editing operations. Any of the individual operations in the tree can be modified at any time.

plied sequentially, and previous changes to the surface cannot be directly modified once another change is made. However, If the surface underlying a decal-based geometric texture is modified, the decal can simply be re-computed on the new surface, and the geometric texture re-applied. In this way, layered geometric texture decals can be incrementally updated. Building on this idea, complex geometric models can be constructed out of layered geometric textures, each of which can be modified at any time. The resulting model is essentially a procedural surface model.

The contribution of this work is a procedural approach to surface modeling, described in the context of an interactive mesh editing tool. Since there is no inherent reason that geometric textures be static, surface editing tools will be re-cast as dynamic geometric texture elements which can be interactively modified (Section 4). Instead of a static 3D mesh, the current model will be represented by a *surface tree* - a hierarchy of layered, local geometric textures applied to an initial surface (Section 3). This novel procedural surface model supports interactive manipulation of any editing operation in the entire modeling history. To preserve hierarchical semantics, surface tree nodes are defined relative to surface parameterizations, and positioned relative to previous tree nodes. Hence, when underlying surface nodes are modified, more recent operations are updated automatically. Supporting efficient non-linear editing also requires replacing the traditional mesh data structure with a procedural variant (Section 5). The non-destructive layering of surface manipulations is reminiscent of layer-based image compositing interfaces, such as Adobe Illustrator [2007], leading us to characterize our approach as *surface compositing*.

2 Background

The notion of increasing the geometric detail of a parameterized surface by displacing it based on some parameter-space signal, or *displacement mapping*, was first introduced by Cook [1984]. Extensive literature on rendering displacement maps exists, but is outside the scope of our work.

Displacement mapping has recently been generalized to allow for the direct insertion of new geometry [Peng et al. 2004; Elber 2005; Porumbescu et al. 2005; Zhou et al. 2006]. The goal of these methods is to tile complex 3D geometric elements on the target surface. The standard approach is to define the desired element in a canonical 3D volume, and map this canonical volume to local patches of the surface, where the deformed 3D volume is usually defined by normal displacement. These techniques are generally applied automatically, in the rendering pipeline. Use as a modeling primitive has been limited to simple displacement-painting tools common in software such as Zbrush [2007].

Volumetric spatial deformation techniques [Barr 1984; Sederberg and Parry 1986; Singh and Fiume 1998; von Funck et al. 2006] are frequently applied in shape deformation. At a conceptual level, these methods "warp" some volume containing the 3D model, and hence exist independently both of the surface being modified, and of other warps. This simplifies their use in procedural frameworks, but often limits the control available to the artist. In response, *surface deformation* techniques have been developed which are defined relative to the original surface [Welch and Witkin 1992; Sorkine et al. 2004; Yu et al. 2004; Botsch et al. 2006]. However, by introducing dependence on the un-deformed surface, the ability to procedurally compose deformations is lost (Figure 3). It is this specific issue which our work aims to address.

Multi-resolution techniques can also be used to apply surface deformation [Zorin et al. 1997], and are (in some sense) procedural surfaces. Intuitive interaction operations such as cut-and-paste [Biermann et al. 2002] have been demonstrated on multi-resolution surfaces, as well as level sets [Museth et al. 2002] and point sets [Zwicker et al. 2002]. However, regardless of surface representation, these techniques are still destructive - non-sequential manipulation of overlapping pasted elements is not supported.

While the procedural modeling paradigm has been applied heavily in automatic generative modeling [Ebert et al. 2002; Prusinkiewicz and Lindenmayer 1991], there have been relatively few attempts to leverage it in surface editing interfaces. Real-time CSG [Hable and Rossignac 2005] has been demonstrated, and the ShapeShop system [Schmidt et al. 2005] supports non-linear editing of hierachical implicit models, but these are volumetric shape representations. Procedural mesh editing is described by [Lewis and Jones 2004], but again the underlying algorithms are volumetric in nature. None of these frameworks support procedural surface deformation. Commercial modeling systems such as Autodesk Maya [2007] do support a limited form of procedural surface modeling in the form of construction history, but this interface is limited to certain NURBS surface-construction tools such as sweeps and surfaces of revolution.

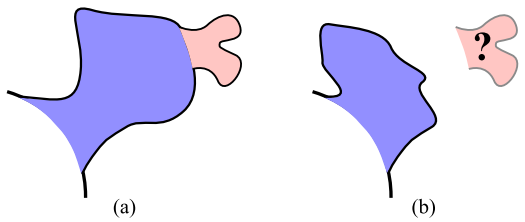

Figure 3: An initial surface is first modified by surface deformation A, followed by deformation B. While B is specified relative to A, the relative context (vectors, control points, and so on) are exist in global 3D position. If A is modified, this contextual information becomes meaningless, and it is unclear how B should be re-applied to A. This is the *surface transport* problem.

3 Procedural Surface Editing

The main challenge in creating a procedural definition of a surface suitable for interactive editing is illustrated in Figure 3. A local deformation A is applied to some base surface, followed by deformation B which affects the region of the surface modified by A. As is standard in 3D modeling interfaces, each of these deformations is specified by contextual information (control points, handles, etc) which have meaning relative to the underlying surface, but are stored in global (3D) position. Assuming that this information is stored for both A and B, the artist may then wish to change A without destroying B. However, since B is defined relative to A, modifying A implicitly modifies B, and the contextual information which defines B must somehow be updated. Essentially, the definitions of A and B are are not independent, which is problematic as it leaves the notion of procedural surface manipulation somewhat ill-posed.

To support interactive editing of A while preserving B, two issues must be addressed. First, we devise a method for representing overlapping surface deformations, and second, a mechanism for updating the set of deformations when one of them changes.

3.1 Surface Trees

In traditional modeling tools, a mesh is represented as tuple $\{\mathcal{V},\mathcal{E},\mathcal{F}\}$ of sets of vertices, edges, and faces. Sequential editing operations destructively update this set. As we have noted, it is generally not possible to manipulate previous edits because the "semantics" of each operation are dependent on the previous state of the mesh. If the previous state is modified in any way, the semantics of later edits are lost.

To mitigate this problem, we eliminate the dependency between an editing operation and the surface it is applied to. Instead, the semantics of an edit are defined in a canonical space, and dynamically transferred to the current surface based on a local uv-parameterization. Although it is somewhat of a simplification, each edit can be thought of as an interactive geometric texture, which can be dynamically applied to any parameterized surface.

By defining surface edits relative to a canonical space, rather than to the surface they are currently applied to, the dependency between operations is severed. Each surface edit is now an independent entity, which can be transferred to any other surface simply by reparameterizing that new surface. Recent work in surface parameterization [Schmidt et al. 2006] described a fast and stable technique for locally parameterizing point-sampled surfaces. These local surface parameterizations, called *decals*, were applied to interactive procedural surface texture compositing. The general approach is to segment a geodesic disc which tightly contains the necessary "support region" of the surface, and then parameterize it, creating a local uv-domain on the surface. In [Schmidt et al. 2006], the Discrete Exponential Map algorithm efficiently computed both these steps simultaneously. We use that algorithm here, but note that any alternate schemes for geodesic disc segmentation and surface parameterization could be used instead.

We now have a series of surface edits, each defined with respect to their local decal parameterization. However, decals can overlap - this is problematic, as edits may be arbitrarily complex and hence difficult to automatically combine. Instead we impose a layer ordering on the edits and their respective decals. Edits higher in the layer ordering are applied on top of lower decals (Figure 4).

Given this layer ordering, a procedural mesh model can be described as a sequential list of editing nodes $\mathcal{N}$ applied to some base surface. The inputs to $\mathcal{N}$ are a surface $\mathcal{S}_{in}$ and an editing operator $\mathcal{E}$. Each node produces an output surface $\mathcal{S}_{out} = \mathcal{N}(\mathcal{S}_{in},\mathcal{E})$. Clearly this definition is recursive, allowing sequential operations to be chained together. However, since $\mathcal{E}$ could be a geometric texture defined by another series of procedural edits, trees of nodes can also be constructed. We refer to this tree of editing nodes as a *surface tree*. An example is shown in Figure 2. The hierarchical definition of a 3D model is strongly reminiscent of procedural implicit volume modeling frameworks [Wyvill et al. 1999].

3.2 Surface Transport

The surface tree defines a procedural 3D model by assembling a complex surface from a series of simpler editing operations. However, the surface transport problem described in Figure 3 still exists. Luckily, the decal framework provides a mechanism for transferring editing operations between surfaces. Remember, each edit $\mathcal{E}$ is applied using a decal parameterization, and hence the support region of $\mathcal{E}$ is entirely defined by the decal seed point and geodesic radius. To transfer the decal parameterization between surfaces, the seed point must simply be moved to the new surface. The tangent frame is then aligned with the surface normal at the new seed point, and the decal can be regenerated.

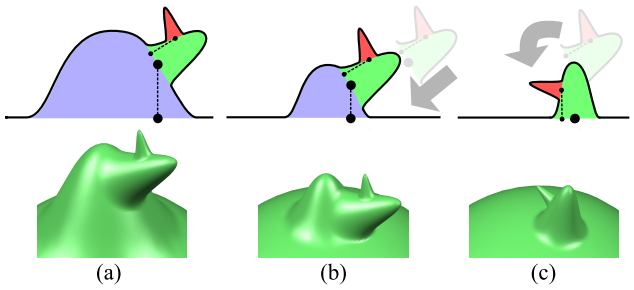

Figure 4: Three surface deformations are layered one on top of another (a), with seed point mappings shown as dotted lines. If the bottom deformation is modified (b) or deleted (c), the seed point lying on top of it is automatically transferred to the new surface.

In [Schmidt et al. 2006], decal seed points were transferred to a changing implicit surface using gradient walks. Similarly, we can find the nearest point on the new surface. However, this approach does not maintain relative position between edits, which make controllable positioning of multiple edits very difficult. To maintain relative position, the decal seed point is "anchored" as a 2D point in the parameter space of a lower node, rather than an absolute 3D point. To produce the 3D seed point required in the ExpMap algorithm, the 2D seed point is mapped onto the surface using the anchor node parameterization. Likewise, if the anchor node is deleted, the uv seed point is automatically mapped back onto the input surface for that node, and possibly attached to another node in the surface tree. (Figure 4). Using this approach, nodes lower in the surface tree can be coherently manipulated (Figure 5).

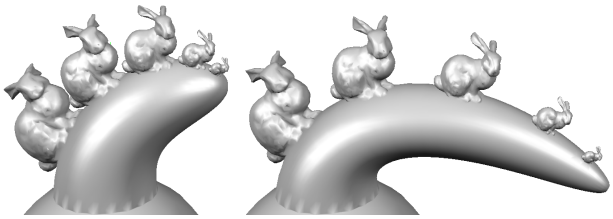

Figure 5: Several geometric textures (Stanford bunnies) are layered on top of a curve-following deformation. The bunnies are anchored in the parameter space of this deformation, so they maintain relative position as it is stretched.

4 Procedural Editing Operations

To demonstrate the benefits of hierarchical surface tree editing, we have developed a prototype interactive modeling system. The interface is sketch-based in nature, the user specifies the shape of new geometry by drawing curves directly on the surface. These concepts have been described in existing sketch-based modeling works [Igarashi et al. 1999; Schmidt et al. 2005]. The interactive tools used in the system are generally very simple, based on standard geometric techniques such as displacement, extrusion, and parametric curves [Foley et al. 1990]. The novelty is not in the implementation of these algorithms, but in how they are applied to the surface.

Existing geometric texturing systems have either avoided modifying the original surface geometry [Porumbescu et al. 2005], or replaced it entirely by seamless tiling of the geometric texture element [Elber 2005; Zhou et al. 2006]. In our interactive system only a single geometric texture is being placed. To preserve manifold properties, it's geometry must be seamlessly stitched into the target surface.

We use the local decal parameterization to robustly and efficiently stitch geometric texture elements into the existing mesh. Geometric textures are attached via well-defined boundary loops, which can be projected into the decal parameter space, as can the relevant portion of the surface mesh. Once in this 2D space, the efficient constrainted Delaunay triangulation code TRIANGLE [Shewchuk 1996] is used to insert the boundary loops and discard interior triangles. This modified 2D mesh is projected back to the surface, and the geometric texture element is inserted (Figure 9).

Note that this projection approach is unnecessary in some operations, such as for simple displacement mapping. However, we still find it useful in that case, as it permits the displacement map to be accurately meshed as a pre-process.

4.1 Generalized Surface Displacement

The standard formulations of displacement mapping and geometric texture rely on the notion of normal offsets. A more general approach is to consider a uv-parameterized domain $\mathcal{D}_\mathcal{S}$ lying on some surface $\mathcal{S}$, where $uv \in [0,1]^2$. Assume another uv-parameterized domain $\mathcal{D}_\mathcal{O} \in [0,1]^2$ exists on some other surface. These two domains have a *mutual parameterization*, meaning that there exists a trivial bijective mapping between the parameterized regions of the two surfaces (via the parameterizations).

A mutual parameterization also defines a trivial map from the canonical 3D volume $uvw \in [0,1]^3$ to the 3D space existing between the two surface domains. Given some coordinate (u,v,w), two unique points $\mathbf{p} \in \mathcal{D}_\mathcal{S}$ and $\mathbf{q} \in \mathcal{D}_\mathcal{O}$ are defined by (u,v), and w specifies a 3D point along the linear path $\mathbf{p} + w \cdot (\mathbf{q} - \mathbf{p})$. In this way, the mutual parameterization between two surfaces can be used to apply geometric texture. Traditional normal displacement is simply a particular method of defining $\mathcal{D}_\mathcal{O}$ (Figure 6).

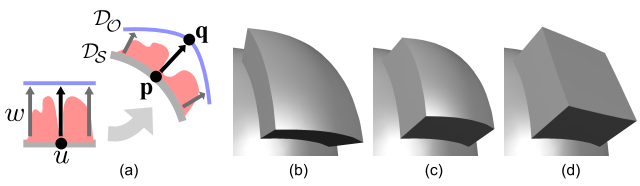

Figure 6: Geometric textures are applied by defining a volume deformation from a canonical unit cube to 3D world space (a). Linear deformation paths can be defined by surface normals (b), a constant vector (c), or another parameterized surface, such as a plane (d).

There is no requirement that the paths in world space be linear. In Figure 7, a control curve defines a bundle of Bezier splines between $\mathcal{D}_\mathcal{S}$ and $\mathcal{D}_\mathcal{O}$. To simplify the mapping from the canonical geometric texture domain, each curve is parameterized by $w \in [0,1]$. The result is essentially a dynamic volume deformation, which changes to conform to $\mathcal{D}_\mathcal{S}$ as the underlying decal moves across the surface.

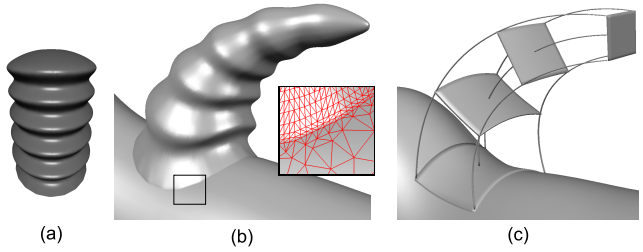

Figure 7: A geometric texture (a) applied to a surface using a non-linear displacement volume (c) defined by a bundle of Bezier curves. The new geometry is dynamically stitched into the existing mesh (b, inset).

4.2 Topology Change

One limitation of geometric texture is that it is strictly additive - while it can be used to append surface elements with non-zero genus (Figure 8a), it cannot cut a hole in a sphere. However, when combined with local parameterization, the geometric texture framework can be adapted to support insertion of topological *holes* and *handles*. To do so, $\mathcal{D}_\mathcal{O}$ and $\mathcal{D}_\mathcal{S}$ are both specified using separate decals. Profile curves lying in these decals define a cylindrical "tube" in canonical space, which is mapped to 3D space based on the mutual parameterization of $\mathcal{D}_\mathcal{O}$ and $\mathcal{D}_\mathcal{S}$, and stitched in to the existing surface.

Like generalized displacement operations, these topological holes and handles are simply additional nodes in the procedural surface tree. Both ends of the new geometry can be interactively manipulated by dragging the decals across the underlying surface. To support arbitrary profile curves, we (linearly) interpolate between the distance fields of the two curves, and mesh this implicit surface in canonical space. This produces holes and handles with sharp edges. Deformations can also be applied to the canonical-space mesh to create smooth transition regions, or other transition shapes. Note that since these topological operations depend only on local parameterizations, they can be applied to non-manifold surfaces (Figure 8c).

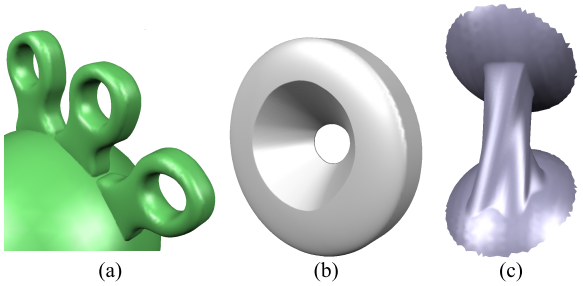

Figure 8: Geometric texture elements can contain topological holes (a), however they cannot create holes in the existing surface. Topological holes can be generated by cutting the manifold using multiple decals and connecting the boundary loops (b). Handles between non-manifold surfaces can also be created (c).

5 Procedural Mesh Data Structure

Conceptually, the procedural mesh model proposed in Section 3 is straightforward to implement. Each mesh edit node takes a base

mesh as input, modifies it, and outputs a new mesh. Note that it is not possible to pass a single vertex through multiple deformations, as is the case with volume deformations. Hence, each intermediary surface must be fully computed before the next can be updated. However, it is infeasible to generate a new mesh at each node. Memory constraints prevent the storage of a large number of high-resolution meshes, and even at lower resolution, if an operation deep in the model stack is changed then the overhead of generating all the intermediate meshes becomes overwhelming. This global approach fails to provide the real-time feedback that 3D designers have come to expect.

Our solution is to leverage the local support of each mesh edit. The decal parameterization limits the region of the mesh which can possible be affected by an edit. Hence, we define a *Mask* operator which takes as arguments an input mesh $\mathcal{S}$ and a decal parameterization $\mathcal{P}$. *Mask*$(\mathcal{S},\mathcal{P})$ produces a light-weight "surface" which provides the same interface as $\mathcal{S}$ by forwarding requests directly to $\mathcal{S}$. However, when iterating over the triangles of *Mask*$(\mathcal{S},\mathcal{P})$, those which lie in the masked region are transparently skipped. The resulting surface contains a hole whose inner boundary coincides with the outer boundary of $\mathcal{P}$. the interior triangles are not discarded, but simply hidden by *Mask*. Since none of the data in $\mathcal{S}$ is copied or modified, it can be instantly recovered by removing the mask. Further, since the decal has local support, so does *Mask*, making it efficient on large meshes.

To re-insert the portion of $\mathcal{S}$ modified by an editing operation, the *Weld* operator is applied. This operator takes as input the surface output by a mask operation, $\mathcal{S}_M$, and the modified decal surface $\mathcal{S}_P$. Like *Mask*, the output of *Weld*$(\mathcal{S}_M,\mathcal{S}_P)$ masquerades as a surface, passing requests on to either $\mathcal{S}_M$ or $\mathcal{S}_P$ as necessary. Iterating over the triangles of *Weld*$(\mathcal{S}_M,\mathcal{S}_P)$ first outputs the triangles triangles of $\mathcal{S}_M$, followed by the those of $\mathcal{S}_P$. Again, the *Weld* operator does not copy or modify either of it's input surfaces.

Additional care must be taken in the *Weld* operator to avoid introducing "cracks" along the common boundary vertices which exist in both $\mathcal{S}_M$ and $\mathcal{S}_P$. *Weld* transparently re-writes incoming and outgoing boundary vertex indices, presenting the outward appearance of a manifold mesh. To avoid any complications in this boundary rewriting, all surface editing operators are required to preserve the outer boundary loop of the decal mesh.

Given a surface and a decal, a procedural mesh deformation can then be implemented as *Weld*$(\textit{Mask}(\mathcal{S},\mathcal{P}),\mathcal{S}_P)$. Since the output of *Weld* presents the same interface as $\mathcal{S}$, these operators can be recursively applied, creating a procedural mesh data structure. Note that *Mask* and *Weld* are easily implemented on point set surfaces. In fact, the point set implementation of *Weld* is less complex, since the explicit boundary re-writing is unnecessary.

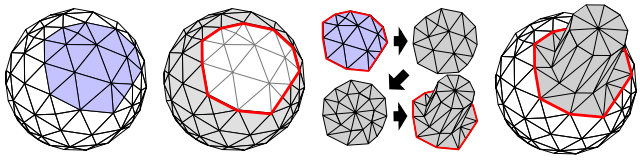

Figure 9: To apply a surface edit, a local uv-parameterization is used to segment a set of triangles (a). The *Mask* operator produces a new mesh which hides these triangles (b). The masked region is then copied and projected down into the uv space, where the edited region is re-meshed and then projected back into 3D (c). Finally, the *Weld* operator is applied to synthesize a manifold mesh (d).

Although *Mask* and *Weld* operate locally, edits may have large support regions, and the overhead of repeated applications will limit interactivity as the node tree grows. To mitigate this, we have found it useful to dynamically insert *cache nodes* into the mesh tree. These cache nodes simply copy the abstract mesh produced by a procedural operation into a single manifold mesh, which is significantly more efficient to iterate over. Cache nodes are dynamically inserted directly below any edit node that the user is interacting with, and discarded when the interaction is complete. In addition, if the user is manipulating a node deep in the tree, interactivity can be maintained by limiting the number of parent nodes which are recomputed during dynamic interaction, and reducing the resolution of procedural editing operations.

Figure 10: A series of variations on an initial surface tree model (top left) derived by manipulating the parameters of different nodes in the tree.

6 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 they have made in the past. The first step in this direction was to develop a surface representation which could support this style of interaction. Hence, we have described the surface tree, a novel approach to representing complex hierarchies of surface editing operations. By combing traditional geometric modeling techniques with dynamic surface parameterization, the surface tree is capable of representing a wide range of 3D models (Figure 1). To demonstrate the capabilities of surface trees, we developed a prototype modeling environment which allows the designer to efficiently composite a series of surface manipulation. Each discrete edit is represented as an independent node in our procedural mesh data structure, which enables dynamic visualization of the surface tree as it is modified. This easily allows a designer to explore different design variations (Figure 10).

As with any prototype interactive tool, our system has a variety of

limitations. One key restriction is that the discrete exponential map parameterization is known to produce significant foldovers when parameterizing large regions of varying curvature [Schmidt et al. 2006]. While more robust and efficient techniques can likely be developed, parameterizations over large domains necessarily contain distortion. This fundamentally limits the capabilities of a geometric texture approach. We are exploring the use of Laplacian surface reconstruction [Sorkine et al. 2004] as a means for expanding the range of deformations that can be applied. Note that the surface tree concept is not dependent on geometric texture, and is easily adaptable to other surface manipulation techniques.

Although we have not implemented it, our implementation framework can trivially be applied to point set surfaces [Zwicker et al. 2002]. The discrete exponential map computes geodesic discs and uv-parameterizations directly on point sets, and our procedural mesh data structure (Section 5) is easily implemented for point sets. If all editing operations are similarly procedural in nature, a surface tree created in our mesh-based interface could even be "played back" on a point set surface, or vice-versa. Similarly, the resolution of procedural operations can be dynamically varied, to automatically construct output models at various levels of detail while faithfully capturing salient model features.

Another aspect of surface trees yet to be explored is applications in computer animation. Procedural models are trivial to animate, and the ability to dynamically manipulate layered surface geometry may be particularly beneficial in this domain.

References

ADOBE SYSTEMS INC., 2007. Adobe Illustrator. www.adobe.com/illustrator.

AUTODESK INC., 2007. Maya. www.autodesk.com/maya.

BARR, A. H. 1984. Global and local deformations of solid primitives. In *Proceedings of SIGGRAPH 84*, 21–30.

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

BOTSCH, M., PAULY, M., GROSS, M., AND KOBBELT, L. 2006. Primo: Coupled prisms for intuitive surface modeling. In *Eurographics Symposium on Geometry Processing*, 11–20.

COOK, R. L. 1984. Shade trees. In *Proceedings of SIGGRAPH 84*, 223–231.

EBERT, D., MUSGRAVE, K., PEACHEY, D., PERLIN, K., AND WORLEY, S. 2002. *Texturing and Modeling: A Procedural Approach*, 3rd ed. Morgan Kaufmann.

ELBER, G. 2005. Geometric texture modeling. *IEEE Comput. Graph. Appl. 25*, 4, 66–76.

FOLEY, J. D., VAN DAM, A., FEINER, S. K., AND HUGHES, J. F. 1990. *Computer graphics: principles and practice (2nd ed.)*. Addison-Wesley.

HABLE, J., AND ROSSIGNAC, J. 2005. Blister: Gpu-based rendering of boolean combinations of free-form triangulated shapes. *ACM Trans. Graph. 24*, 3, 1024–1031.

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: A sketching interface for 3d freeform design. In *Proceedings of SIGGRAPH 99*, 409–416.

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

MUSETH, K., BREEN, D. E., WHITAKER, R. T., AND BARR, A. H. 2002. Level set surface editing operators. In *Proceedings of SIGGRAPH '02*, 330–338.

PENG, J., KRISTJANSSON, D., AND ZORIN, D. 2004. Interactive modeling of topologically complex geometric detail. *ACM Trans. on Graph. 23*, 3, 635–643.

PIXOLOGIC INC., 2007. ZBrush. www.pixologic.com.

PORUMBESCU, S. D., BUDGE, B., FENG, L., AND JOY, K. I. 2005. Shell maps. *ACM Trans. Graph. 24*, 3, 626–633.

PRUSINKIEWICZ, P., AND LINDENMAYER, A. 1991. *The Algorithmic Beauty of Plants*. Springer.

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

SCHMIDT, R., GRIMM, C., AND WYVILL, B. 2006. Interactive decal compositing with discrete exponential maps. *ACM Transactions on Graphics 25*, 3, 605–613.

SEDERBERG, T. W., AND PARRY, S. R. 1986. Free-form deformation of solid geometric models. In *Computer Graphics (Proceedings of SIGGRAPH 86)*, vol. 20, 151–160.

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

SINGH, K., AND FIUME, E. L. 1998. Wires: A geometric deformation technique. In *Proceedings of SIGGRAPH 98*, 405–414.

SORKINE, O., COHEN-OR, D., LIPMAN, Y., ALEXA, M., RÖSSL, C., AND SEIDEL, H.-P. 2004. Laplacian surface editing. In *Eurographics / ACM SIGGRAPH Symposium on Geometry Processing*, 175–184.

VON FUNCK, W., THEISEL, H., AND SEIDEL, H.-P. 2006. Vector field based shape deformations. *ACM Trans. Graph. 25*, 3, 1118–1125.

WELCH, W., AND WITKIN, A. 1992. Variational surface modeling. In *Computer Graphics (Proceedings of SIGGRAPH 92)*, vol. 26, 157–166.

WYVILL, B., GUY, A., AND GALIN, E. 1999. Extending the csg tree. warping, blending and boolean operations in an implicit surface modeling system. *Comp. Graph. Forum 18*, 2, 149–158.

YU, Y., ZHOU, K., XU, D., SHI, X., BAO, H., GUO, B., AND SHUM, H.-Y. 2004. Mesh editing with poisson-based gradient field manipulation. *ACM Trans. Graph. 23*, 3, 644–651.

ZHOU, K., HUANG, X., WANG, X., TONG, Y., DESBRUN, M., GUO, B., AND SHUM, H.-Y. 2006. Mesh quilting for geometric texture synthesis. *ACM Trans. Graph. 25*, 3, 690–697.

ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. 1997. Interactive multiresolution mesh editing. In *Proceedings of SIGGRAPH 97*, 259–268.

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