Eurographics Workshop on Sketch-Based Interfaces and Modeling (2005)
Takeo Igarashi, Joaquim A. Jorge (Editors)

Sketch-Based Mesh Augmentation

L. Olsen[1], F. F. Samavati[1], M. C. Sousa[1], J. A. Jorge[2]

[1]Department of Computer Science, University of Calgary, Canada
[2]Department of Information Systems and Computer Engineering, Technical University of Lisbon, Portugal

Abstract

This paper describes a sketch-based method for augmenting geometric meshes with variable-scale sharp features. Such mesh editing operations provide a powerful way to exploit an artist's natural sketching abilities for content creation using existing mesh libraries. We use adaptive subdivision to refine the mesh only in the region of the features and embed the augmentations into the input mesh, maintaining a unified representation. Moreover, we present new templates for adaptive subdivision that minimize the high-valence vertex count while maximizing the number of quadrilaterals in the new mesh, thereby improving its quality.

Categories and Subject Descriptors (according to ACM CCS): Geometric modeling, sketching, mesh editing, adaptive subdivision.

1. Introduction

The creation of large, complicated meshes is a task for which there are many tools available, primarily based on B-spline or NURBS patches. The ubiquitousness of these tools in modern modeling environments overshadows their steep learning curve. Indeed, the interfaces in such systems are far removed from drawing-based metaphors and are based on constructive geometry, menus and selection, which are more natural to computer scientists and engineers than to artists. This technical barrier to entry creates a high demand for experienced software modelers and excludes many talented artists from contributing to content creation.

Sketch-based modeling systems present an interesting alternative to traditional (parametric- or implicit-based) modelers. Ideally, sketch-based systems provide a more natural interface and enable the less technically-inclined artists to leverage their learned sketching and drawing skills for content creation. However, systems that create a mesh based purely on a user's two-dimensional *strokes* (lines input to a sketching system) must enforce constraints and make assumptions in order to create a 3D shape from the strokes. For instance, the SKETCH system of Zeleznik et al. [ZHH96] has predefined rules for creating geometric objects, such as interpreting three lines that meet at a point as a parallelepiped. The more recent Teddy system of Igarashi et al. [IMT99] assumes a spherical topology and "inflates" the user's strokes into bulbous shapes.

In this paper, we present a sketch-based system for creating variable-scale features on an existing mesh. By having an initial 3D shape to work from, our system does not have to place any constraints on the strokes or their interpretation. The user's strokes are no longer stuck in two dimensions, but can instead be projected into 3D using the mesh geometry.

When dealing with mesh augmentation instead of mesh creation, the interpretation and emphasis of user strokes changes. If user strokes are the only information available to the system, the noisiness of the strokes is a primary concern and dictates the quality of the output. For the purposes of augmenting a mesh, however, preserving the original quality of the mesh is as important as using the strokes to create features. To reduce distortion of the input mesh, we use adaptive subdivision only in the region of the user strokes and relax the position of vertices around augmentations.

The main contribution of this work is the combination of sketch-based methods and adaptive subdivision to create a robust mesh augmentation system. By refining the mesh only in the region of the user's sketches, we can localize the effects of the feature creation process; this allows the creation of small- to large-scale features while elsewhere preserving the quality of the input mesh. We also improve adaptive Catmull-Clark subdivision by using an incremental approach and introducing new patching templates that produce more quadrilateral faces and low valence vertices. These techniques combine to produce an augmented mesh with more regular connectivity and stable lighting calculations.

In the next two subsections, we review previous work in two categories related to our work: mesh augmentation and sketch-based modeling.

1.1. Mesh Augmentation

Our work is most closely related to the work of Biermann et al. [BMZB01]. In their paper, they present a method for creating sharp features on subdivision surfaces, using a set of strokes or *feature curves* as input. Their approach involves projecting the feature curves not onto the mesh per se, but into the mesh's parameter space. The process of aligning edges in the mesh with the curve ("snapping") is then performed in parameter space, and the modifications to the parameter space are finally reflected in the actual mesh by a resampling phase. The entire mesh is subdivided one or more times after the creation of the features to produce a high-quality mesh with sharp features.

There are a few shortcomings to their method however. Their snapping stage may introduce some distortion in the mesh. When a vertex is too far from the curve to be snapped, then the whole mesh must be subdivided; this creates a struggle between minimizing distortion and minimizing the number of faces in the mesh. Also, their method requires new subdivision rules to allow features to cross the diagonals of faces, which seems at odds with their goal of creating a unified representation for the mesh and the features.Finally, the feature creation requires an global increase in mesh complexity to achieve an accurate representation of the features. Ideally we could increase the complexity only in a local vicinity of the features.

Earlier methods for creating sharp features on existing meshes often resulted in features that were *extensions* of the original mesh representation. This means that the output of such a system is represented differently than the input. For instance, Khodakovsky and Schröder [KS99] keep the feature curves distinct from the mesh representation, reapplying the features at every level of subdivision.

Mesh augmentation can be thought of as the melding of two surfaces: the original mesh, and the sketched features. The concept of "pasting" surfaces together was first explored by Mann et al. [CMB97, MM01]. In that work, they consider only the pasting of parametric surfaces – which is by no means trivial – but parametric surfaces are generally easier to work with than arbitrary meshes.

Mann et al.'s work was extended by Biermann et al. [BMBZ02] to allow the pasting of polygonal meshes. This work also draws on their earlier work [BMZB01], in that the bulk of processing is done in parametric space. They present a very robust system that separates the source and destination meshes into a base mesh plus details, allowing for the two surfaces to be pasted together quite seamlessly. The main drawback is that both a source and destination mesh are required, so the creation of features that are not in an existing mesh is precluded.

The pasting idea was further extended by Yu et al. [YZX*04] to allow not only pasting, but also deformation of the source and destination meshes. Their work is based on Poisson gradient manipulation, though thankfully this fact is hidden from the user. Their system is capable of producing appealing meshes with clean deformations and blends, but there is still the problem of content creation: all elements in the final mesh are cobbled together from existing meshes.

In work that is loosely related to ours by its goal of producing small features on a base mesh, Peng et al. [PKZ04] forego actual mesh-level feature creation and instead use volumetric textures to add features at render-time. Their results are impressive, but the features are not embedded in the mesh representation. This eliminates the reusability of their detailed meshes in other applications.

1.2. Sketch-based modeling

In sketch-based modeling, there are other works that begin with an existing mesh to eliminate some of the typical sketching constraints. Lawrence and Funkhouser [LF03] allow the user to sketch a "velocity" on the surface of a mesh, with a subsequent physical simulation growing the mesh according to the sketched velocity. After several simulation iterations, they are able to make quite complex meshes that would be very difficult with other modeling approaches.

Funkhouser et al. [FKS*04] later presented a modeling-by-example system. Beginning from a base mesh, an assisted sketching system is used to select portions of the mesh for editing, as well as to cut out parts from other meshes to import into the base mesh. Because the user sketches on top of a mesh, they are able to provide "intelligent scissors" that follow the natural lines of the mesh. Like Yu et al., their system is designed to work with existing meshes, not to create arbitrary mesh details.

Zelinka and Garland [ZG04] also present a modeling-by-example system, except their "examples" are not other meshes but user-drawn curves that guide deformation along slices through the mesh. The deformations are created using the vertices in the input mesh, with no refinement through subdivision, which produces some artifacts in the deformed mesh.

1.3. Approach

Our approach is guided by the goal of producing a sketch-based system for creating features that are smoothly inte-

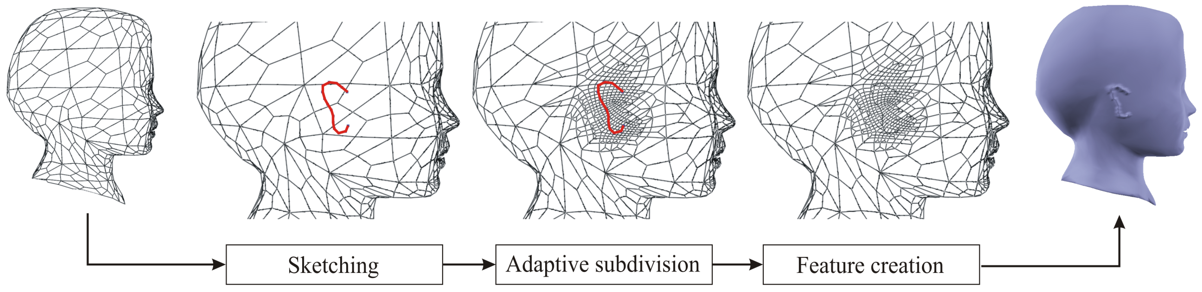


Figure 1: *Flow of data through our system: A base mesh is loaded; The sketching component allows the user to place feature curves on the surface of the mesh (red); The adaptive subdivision component increases the mesh resolution around the features; Finally, the feature creation displaces vertices along the features to create a sharp feature.*

grated with an input mesh. To reduce the amount of geometric distortion introduced into the mesh (any augmentation is essentially a distortion), we provide two different snapping strategies: snap-to-mesh, and snap-to-curve. To fully embed the features into the original mesh representation, it is important to avoid non-standard subdivision rules; to that end, when a feature curve crosses a face we split the face into two halves about the crossed diagonal. Finally, we would like to preserve the tessellation of the input mesh as much as possible; instead of refining the entire mesh to attain enough resolution to create features, we adaptively subdivide in the region of the features while leaving the rest of the mesh as it was before augmentation.

2. Methodology

Our system consists of three components (Figure 1). The sketching component, described in Section 2.1, handles the input and interpretation of the user's sketch input to create feature curves. The adaptive subdivision component, detailed in Section 2.2, uses the sketch input to increase the mesh resolution in the vicinity of the features. Finally, the feature creation component of Section 2.3 uses vertices in the high-resolution area of the mesh to approximate each features, and then displaces these vertices in the normal direction to create sharp features on the mesh in a unified representation.

The input to our system is assumed to be a quadrilateral (henceforth *quad*) mesh, M^0 (if the mesh contains non-quad faces, one application of Catmull-Clark subdivision will create a quad mesh). We can represent such a mesh with two lists: a list of vertices, V, and a list of faces, F. Consider a mesh with v vertices and f faces, and denote three-dimensional points as p and indices into V as w. Then:

$$M^0 = \{V, F\} ,$$

where

$$\begin{aligned} V &= \{p_0, p_1, \ldots, p_{v-1}\} , \\ F &= \left\{ \left\langle w_0^0, \ldots, w_4^0 \right\rangle, \ldots, \left\langle w_0^{f-1}, \ldots, w_4^{f-1} \right\rangle \right\} . \end{aligned}$$

We use Catmull-Clark subdivision, which is especially suited to quad meshes, for the adaptive subdivision component. This allows us to increase the resolution and the smoothness of the mesh in the vicinity of the features. Note that we could increase mesh resolution by simply splitting faces, but having both smoothness – the effect of the subdivision – and sharpness – the effect of displacement – allows us to model a variety of features and have them blend smoothly with the original mesh (see Figure 8, for instance).

2.1. Sketching

Our sketching interface employs a traditional mouse or tablet device for input. Strokes are converted from window coordinates to mesh/world coordinates by unprojecting each stroke sample into 3D. This reverse projection can be done with the viewing transformation matrix and the depth buffer of the rendering API (we use OpenGL). Each mouse or tablet event is unprojected onto the mesh, and linked together to form a feature curve. One drawback is that a feature curve could jump over a local minimum in the mesh if the sketching viewpoint is not chosen well; however, this behaviour fits with our sketching metaphor, so the user will intuitively choose a suitable viewpoint to sketch from.

For our adaptive subdivision component, it is important to note that the sketching component must determine which face $f \in F$ the feature falls on. This can determined via OpenGL's selection buffer interface [Bly99], whereby each face is rendered to a small viewport around the mouse pointer (this requires an additional rendering pass).

In order to enable a wide range of expressiveness, it is important to allow strokes to be linked together; otherwise, the user could not draw something as simple as a bracelet because the whole stroke cannot be made from a single viewpoint. We use a user-adjustable measure to decide whether we should connect a stroke point to an existing stroke or create a new stroke.

To formalize our feature curve representation, let p denote a three-dimensional point, g denote face indices, and n_j denote the number of samples in feature curve c_j. Then, the sketching component outputs a set C of feature curves

$$C = \{c_0, c_1, \ldots, c_n\} \ , \tag{1}$$

where

$$c_j = \left[\left\langle p_0^j, g_0^j \right\rangle, \ldots, \left\langle p_{n_j}^j, g_{n_j}^j \right\rangle \right] \ . \tag{2}$$

2.2. Adaptive Subdivision

Adaptive subdivision is a technique for increasing the resolution and smoothness of a mesh only in areas that satisfy some criteria, leaving other areas at their original resolution. Different applications and tasks require different criteria; for instance, an early use of adaptive subdivision was to increase mesh resolution only in high-curvature areas, because flat areas do not benefit from subdivision.

The first step of adaptive subdivision is to decide where to increase the mesh resolution; call this step *face tagging*. After the mesh resolution has been selectively increased, the next step is to blend the high-resolution and low-resolution regions of the mesh, which we refer to as *face patching*. We discuss each of these steps in detail below.

2.2.1. Face Tagging

For mesh augmentation, we want to subdivide only those faces that are visited by a feature curve. From the feature curves C output by the sketching component, we can determine all faces $F_{subdiv} \subseteq F$ that we would like to subdivide (each feature curve sample contains an index g indicating which face the sample is associated with; see Equation (2)):

$$F_{subdiv} = \left\{ f_j \mid \exists k, p \ \texttt{s.t.} \ \langle p, j \rangle \in c_k, \ c_k \in C \right\} .$$

When constructing F_{subdiv}, we frequently encounter gaps in the faces visited by a feature curve. These gaps are due to the sampling rate of the curve and are virtually unavoidable, especially as we increase mesh resolution through subdivision. Figure 2(a) illustrates this problem. From the curve samples (blue dots), we can determine that the gray faces belong in F_{subdiv}, but we would like to include *all* faces that fall under the curve.

To frame the problem in familiar terms, it is helpful to consider a mesh as a graph in which the nodes are faces and the edges are, well, edges. If we consider our definition of a feature curve c_j from Equation 2, then our goal is to find a *path* between faces f_m^j and f_{m+1}^j (for instance, the gray faces of Figure 2(a)).

Our approach to this problem is, however, not inspired by graph theory. We can instead exploit our geometric setting to find a path between the known face samples by casting a ray from sample m to sample $m+1$ and finding intersections with edges in the mesh. From the intersected edges, we can determine a sequence of faces that form a path between the samples.

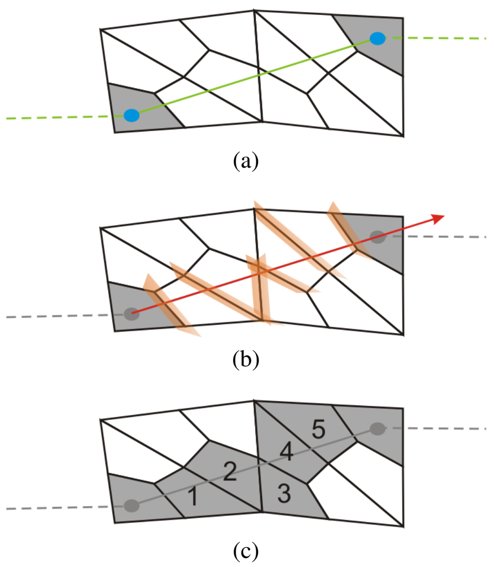


Figure 2: *Finding a path between faces on a feature: (a) We have intermittent face samples (gray faces) associated with each feature curve point (blue circles). (b) A ray (red arrow) is cast from the first to the second sample. The ray is then intersected with planes erected at each edge. Valid intersections are shown in orange. (c) Based on the ray-plane intersections, we can move from the first face to the second, visiting the intervening faces in the order shown.*

Figure 2 illustrates this approach. Though we consider a 2D example, one should note that in practice the computations are carried out fully in 3D; this is due to the fact that both our mesh and stroke representations (vertices, stroke samples) are in three dimensions. An equivalent 2D problem could be created by projecting the vertices and strokes to a plane, but the cost of this projection would not be recouped in solving the slightly simpler problem.

Consider sample m and $m+1$ of feature c_j, which are associated with faces f_m^j and f_{m+1}^j. In Figure 2(a), these curve samples and their associated faces are shown as blue circles and gray quads, respectively. To find a path between these faces, we construct a plane at each edge of the starting face such that the plane contains the edge and is perpendicular to the face. We then cast a ray r from the first sample point p_m^j to the second p_{m+1}^j:

$$r = p_{m+1}^j - p_m^j \ ,$$

and find the intersection of the ray with each edge plane. The valid ray-edge plane intersections are shown in orange in Figure 2(b).

To find a path, we use the valid intersections: from the left-

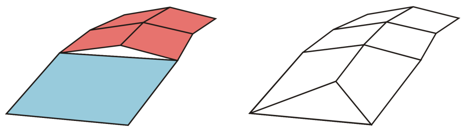


Figure 3: *Left: Cracks are created in the mesh when a subdivided face (red) and a non-subdivided face (blue) share an edge. Right: To fix these cracks, we patch the non-subdivided face.*

most intersection in Figure 2(b), we can move from face f_m^j to the face marked with a 1; from 1, we can use the next intersection to move to 2; and so on, until we reach face f_{m+1}^j. In cases where the ray is parallel to a plane being intersected, the path could flip-flop between two faces. We handle this by requiring that each face in the path is visited only once.

2.2.2. Face Patching

Regardless of our scheme for choosing which faces to subdivide, we must *patch* faces adjacent to those that we do subdivide. Denote all such faces as F_{patch}, where

$$F_{patch} = \{f \mid f \texttt{ adj to } g \in F_{subdiv},\ f \notin F_{subdiv}\}.$$

If no patching is done, cracks will appear in the mesh along edges between subdivided and non-subdivided faces; see Figure 3 for an illustration.

Because we assume our initial mesh M^0 has only quad faces, the faces $f_p \in F_{patch}$ will always be quadrilaterals. Thus f_p will have four neighbors, each of which may or may not be subdivided (but at least one must be); there are 15 permutations, but only 5 distinct cases to handle: one, two adjacent, two opposite, three, or four subdivided neighbors. (Note that to allow iterative augmentations, we must have a patching strategy for introduced triangle faces; this issue is addressed in [PS04].)

In the literature on adaptive subdivision [ZS00], they present templates for handling each of these cases; see the middle column of Figure 4. These patching templates are optimized for rendering, and therefore f_p is patched with triangles. While this approach is appropriate for many cases, it inevitably produces high-valence vertices. For instance, consider the top vertex of f_p in Figure 4(d)(middle): the valence of this vertex is $n = 7$, while the regular valance for quad meshes is $n = 4$. During subsequent subdivisions of the mesh, high valence vertices can result in visual artifacts.

In our system, we would like to maximize the number of quads while minimizing the number of high-valence vertices in our patched faces. To achieve these goals, we designed new patching templates. Our templates are shown in the right column of Figure 4. Note that we use the centroid of the face as a vertex in most cases; this is consistent with Catmull-Clark subdivision and in practice our patched faces produce a smooth transition between high and low resolution.

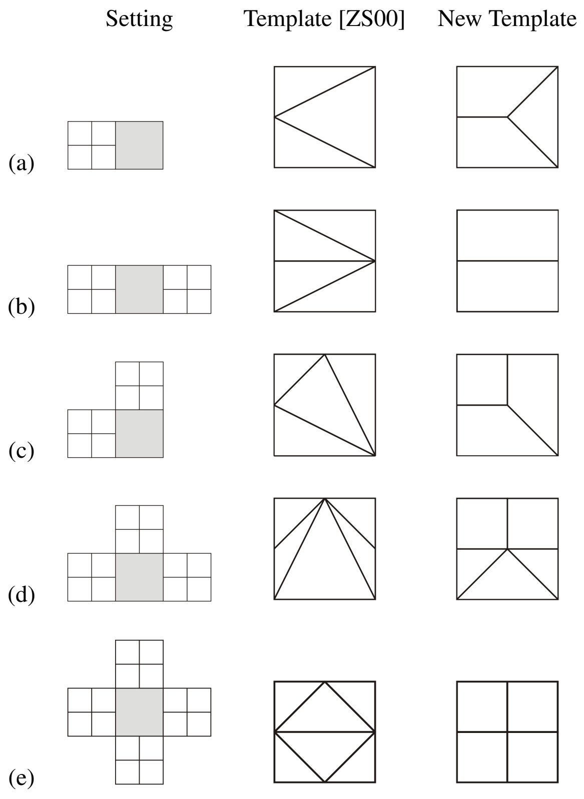


Figure 4: *Our patching templates versus the traditional templates. Left column: A face $f_p \in F_{patch}$ is shown in gray, with its subdivided neighbors $g \in F_{subdiv}$ shown in white. Middle: The traditional method for patching the face [ZS00]. Right: Our new method for patching the face.* ***(a)*** *One subdivided neighbor.* ***(b)*** *Two subdivided neighbors, on opposite sides.* ***(c)*** *Two subdivided neighbors, adjacent to one another.* ***(d)*** *Three subdivided neighbors.* ***(e)*** *Four subdivided neighbors (note that our strategy in this case is to just subdivide rather than patch).*

We can see that, over all cases, the traditional templates produce 0 quads and 22 triangles, while our templates produce 13 quads and only 4 triangles. Similarly, the traditional approach produces vertices with valence as high as $n = 7$, while our templates produces vertices of valence no more than $n = 5$.

Though our templates produce valence-3 vertices, which have only C^1 continuity in the limit surface, our templates reduce the total number of extraordinary cases in the patched faces. Further, this reduced limit continuity is preferable to the ripple effect that results from very high-valence vertices. There is no patching solution that eliminates extraordinary cases, so our templates merely strive to create a smaller number of less noticeable cases.

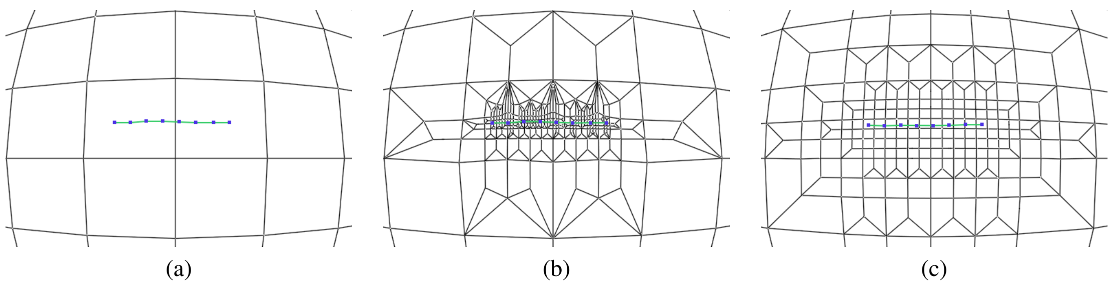


Figure 5: *Vanilla adaptive subdivision versus Incremental adaptive subdivision: (a) Original mesh with feature curve sketched on it. (b) Vanilla adaptive subdivision produces extremely high-valence vertices and thin triangles as faces are repeatedly patched. (c) Incremental adaptive subdivision produces a gradual transition from low to high resolution.*

2.2.3. Incremental Adaptive Subdivision

The traditional – or *vanilla* – adaptive subdivision process is an iterative one in which we decide which faces F_{subdiv} to subdivide based on some criteria, subdivide them, and then patch their neighbors F_{patch}. For only a single iteration, this approach works well. What happens, though, if we adaptively subdivide again? In particular, what happens if we again want to subdivide faces at the boundary between the low-resolution and high-resolution areas?

In those instances, we will find ourselves repeatedly patching descendants of our original patch faces. In general, patched faces create vertices with high valence; if we repeatedly patch them, we are repeatedly increasing vertex valence and creating long, thin faces. As an analogy, consider a pie cut into n slices as having a vertex of valence n at the center: as n increases, the slices (faces) connected to the vertex become thinner.

Figure 5 illustrates this problem. Our original mesh in (a) is to be subdivided based on faces falling under the green curve. The result of applying vanilla adaptive subdivision several times is shown in Figure 5(b); vertices at this level have valence as high as $n = 12$.

To address this problem, Pakdel and Samavati [PS04] describe an elegant solution called *incremental* adaptive subdivision. The idea is very simple: rather than only subdividing faces in F_{subdiv}, they extend their region of interest (on each iteration) by setting

$$F'_{subdiv} = F_{subdiv} \cup N(F_{subdiv}) \ ,$$

where $N(F_{subdiv})$ denotes all faces adjacent to the faces in F_{subdiv}. This requires us to recompute F_{patch} to reflect the larger F'_{subdiv}:

$$F_{patch} = \left\{ f \mid f \ \text{ adj to } g \in F'_{subdiv},\ f \notin F'_{subdiv} \right\} .$$

The effect of widening our region of interest is that we will never patch a face or its descendant more than once, assuming that our selection criterion is stationary. This leads to a smooth transition from high to low resolution, with the only expense being a slightly higher face count than vanilla adaptive subdivision.

Figure 5(c) shows the results of incremental adaptive subdivision applied to Figure 5(a). It is clear that the incremental approach produces much nicer results than vanilla. Also, the lack of thin triangles and high-valence vertices produces more stable calculations for lighting normals, which results in higher-quality renderings. The tradeoff is a higher number of faces: for instance, in Figure 5 the vanilla mesh has 561 faces, compared to 755 faces in the incremental mesh.

2.3. Feature Creation

Feature creation is the most important component in our mesh augmentation system. It is here that a visible feature is actually created by displacing vertices in the mesh.

We can split feature creation into two stages. First, we must select which vertices to displace and create an approximation of the features from these vertices; call this stage *snapping*, to be consistent with previous work. Second, we must displace the selected vertices to create a visually perceptible feature; call this stage *displacement*.

2.3.1. Snapping

The inputs to this component of our system are a refined mesh from adaptive subdivision, and a set C' of feature curves (the original set C of features must be updated to C', to reflect the new face structure that results from adaptive subdivision). The feature curves are a set of $\langle point,\ face \rangle$ samples. From these inputs, we want to determine a sequence of vertices Δ_j adjacent to one another that approximate each feature $c'_j \in C'$, i.e. a mapping $c'_j \mapsto \Delta_j \subseteq V$.

One might wish to further constrain the problem by requiring an edge to exist between adjacent vertices. We take the more relaxed approach of allowing vertices in Δ_j to cross the diagonal of a quad face. We differ from the approach of Biermann et al. [BMZB01] slightly, because we split faces that are crossed by an edge in Δ_j into two new faces. This

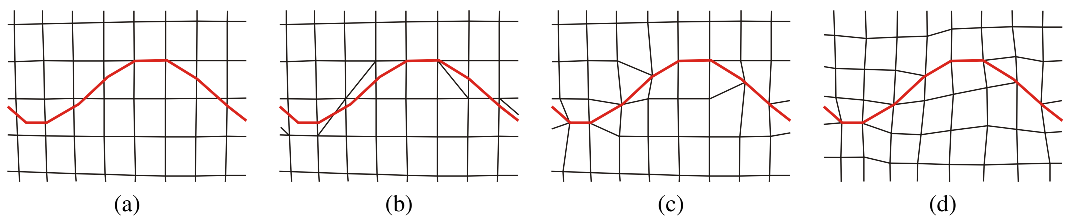


Figure 6: *Comparison of snapping strategies: (a) Mesh and curve before snapping; (b)* Snap-to-mesh *approximates the curve with a vertex path in the mesh; (c)* Snap-to-curve *uses the same vertex path, but vertices in the path are moved onto the curve; (d) Mesh distortion in the snap-to-curve strategy is reduced by a vertex relaxation process.*

allows us to use standard subdivision rules to refine the augmented mesh later.

The vertices in Δ_j are determined by finding the closest vertex to each feature curve sample. As in the face-tagging stage, the sampling rate of the feature may cause gaps in the vertex path; a gap is a sequence $\{f,g\} \subseteq \Delta_j$ such that f and g do not share any faces. To fill such a gap, we find the vertex neighbor f' of f that is closest to the feature, then repeat the process for f' and so on until g is reached.

There are two obvious snapping strategies. The first strategy, *snap-to-mesh*, approximates feature c_j by the vertices in Δ_j. This will preserve the mesh geometry – because vertices will not move at all – but cannot represent the feature curve exactly. This snapping strategy is well-suited to features that are composed of straight lines or that are meant to be guidelines. Figure 10 demonstrates how a snap-to-mesh strategy can create very clean features. For free-form curves, however, this strategy can cause aliasing and jagged edges. Figure 6(b) demonstrates the results of this snapping strategy for a free-form feature.

The second strategy, *snap-to-curve*, moves the vertices of Δ_j onto feature c_j. This strategy is used by Biermann et al. [BMZB01]. In order to keep mesh complexity at a reasonable level in their system it is necessary to displace the vertices quite far, which introduces distortion in the geometry beyond the distortion necessary to create the sharp feature. Figure 6(c) shows a mesh that has been snapped to a feature; the feature is represented exactly, but the faces surrounding the feature are clearly distorted.

To alleviate this distortion, we follow the snapping process with a simple vertex-relaxation stage. First, we compute the average edge length along each feature, d. This average is used to compute a desired position of vertices neighboring the feature: first-level neighbors should be approximately d units from the features, second-level neighbors should be $2d$, and so on. The final position of each neighboring vertex is a weighted average of its original and desired position. Figure 6(d) shows the relaxation of the mesh in (c). Note that the selection process for vertices in Δ_j guarantees that the relaxation process will not create concave faces.

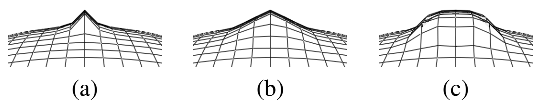


Figure 7: *The effect of the sharpness parameter, s: (a) $s = 10$; (b) $s = 1$; (c) $s = 0.1$.*

2.3.2. Displacement

After the snapping stage, we have a sequence of adjacent vertices Δ_j that approximate each feature c_j. To create a visually perceptible feature, we must displace these vertices somehow. We choose to displace the vertices in the direction of the surface normal.

The user can control the profile of each curve via several parameters, including width, height, and sharpness: the width of the feature is an integer value specifying the number of neighboring vertices $v_n \notin \Delta_j$ to displace; the height of the feature is a scalar that is applied to the vertex normals along the feature; and the sharpness is some value $s \in [0,\infty)$.

The shape of a feature profile is determined by the sharpness parameter, according to an exponential function:

$$h(x) = (1-|x|)^s ,$$

where $h(x)$ is the height of a feature-neighboring vertex at a distance x from the feature, and x is normalized to $[0,1]$. Figures 7 and 11 show the effect of sharpness on the feature profile.

3. Results

In our system, the features are applied to or “embedded” in the mesh only at the behest of the user, not while the user is sketching the features. This is more of a usability decision than a technical one. Timing results show that our system is usable at interactive rates for augmenting typical meshes, because adaptive subdivision – the most computationally expensive component – depends not on the complexity of the mesh, but on the complexity of the features.

While feature embedding requires some computational

Model	f_{in}	f_{out}	t_{sketch}	t_{aug}
Dinosaur	11328	12600	50s	0.66s
Hand	1464	3133	23s	0.09s
Etch sphere	384	1575	85s	0.22s
Wavy sphere	384	836	14s	0.05s
Fish	20466	38583	452s	42.13s
Dragon	21910	24183	247s	6.11s

Table 1: *Usage statistics for Figures 8-13, where f_{in} = input faces, f_{out} = output faces, t_{sketch} = sketching time, and t_{aug} = augmentation time.*

time, the actual interaction with the system provides real-time feedback because very little processing must be done while the user is sketching their feature curves. The sampling rate of the sketch is kept high by caching the mouse/tablet events as the user is sketching. Providing immediate feedback during the sketching phase is important for the user to be able to evaluate and make small adjustments while sketching.

Table 1 presents some usage statistics recorded while producing the results of Figures 8-13. The test machine contained a 2.0GHz Pentium M processor with 1GB of RAM. The reported timings include both the time spent sketching by the user, as well as the time spent applying the features by the system. The time required to augment a mesh depends on both the number and complexity of features, as well as the number of adaptive subdivision iterations. Note that the user, while trained to use the system, was not a trained artist.

Figure 8 demonstrates the ability of our system to create large-scale features that integrate naturally with the input mesh. The triceratops mesh in (a) has a few short features along the spine. After applying these features in (b), we have created a stegosaurus-like mesh with plates along the spine, and the augmentations blend with the original mesh seamlessly.

Figure 9 shows the creation of medium-scale freeform features on a hand model. The original model of (a) is augmented with some bulging veins in (b), by using features with small height and sharpness values. The vein features are smoothly recreated by snapping to the curve and then relaxing the surrounding vertices.

Figures 10 and 11 demonstrate the wide range of mesh augmentations achievable with our system. In Figure 10(a) we see a simple spherical base mesh. Using a negative height for the features produces an “etching” effect, as shown in (b); the clean lines result from the snap-to-mesh strategy. In (c), we can clearly see that adaptive subdivision impacts the mesh only around the features. Figure 11 shows both the ability of our system to cope with freeform features and the effect of varying the sharpness parameter. By varying s, we can create sharp features as in (a), gradual bumps as in (b), or smooth bulges as in (c) and (d).

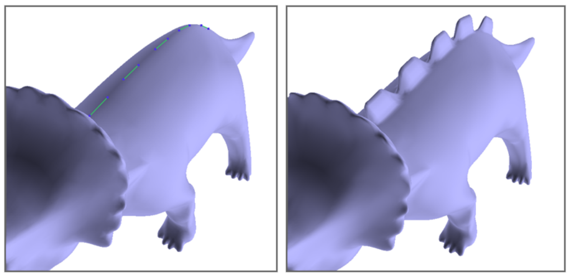


Figure 8: *A dinosaur mesh: (left) original smooth-shaded mesh, with sketched features along the spine; (right) the features blend nicely with the original mesh.*

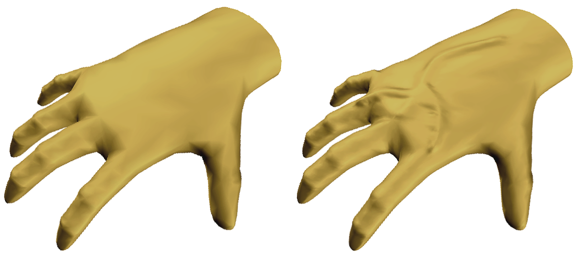


Figure 9: *A hand mesh augmented with medium-scale features: the original mesh (left) is augmented with veiny structures (right).*

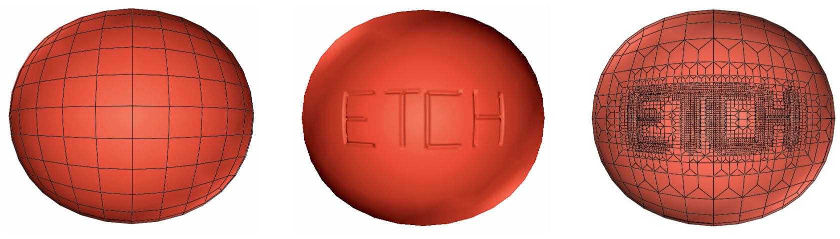
Figure 12 shows how our system can be used to create small-scale features on a fish mesh. The input mesh is shown in (a). By using small displacements, long strokes, and more levels of subdivision, we can create subtle surface details such as scales; see Figure 12(b).

Figure 13 uses a more detailed base mesh than the other figures. In (a), the base dragon mesh is shown; in (b), the dragon is shown along with the sketched features. The long strokes create nostrils, while the short strokes add teeth to the mesh; the result of applying these features is given in (c). The addition of these features turns a rather generic-looking dragon into a more detailed and menacing version.

4. Conclusion & Future Work

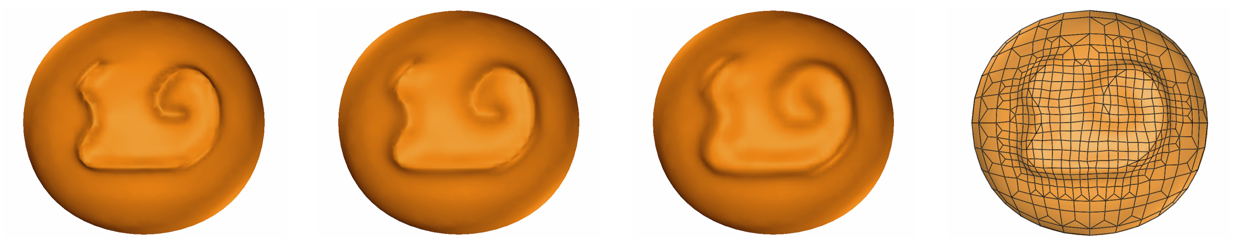
In this paper we have presented a robust method for augmenting an existing mesh with variable-scale sharp features. Our system presents several advantages over other similar systems: the augmented mesh retains the representation of the original mesh, allowing it to be seamlessly used in any other application; we do not require special subdivision rules to further enhance the augmented mesh; and finally, adaptive subdivision allows the mesh complexity to increase only in the vicinity of the features, meaning that global subdivision is not required to create high-quality features.

In the future, there are some improvements we would like



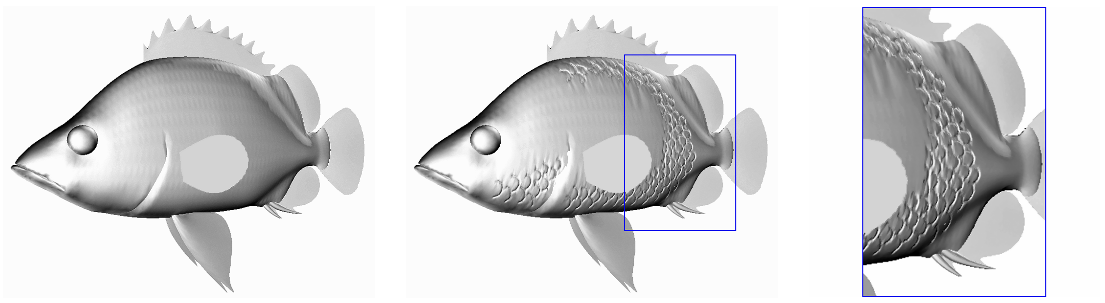
(a) (b) (c)

Figure 10: *A spherical base mesh is augmented with snap-to-mesh features: (a) Original mesh; (b) Using a negative displacement creates an "etched" effect; (c) A wire-frame rendering shows the localization of subdivision.*



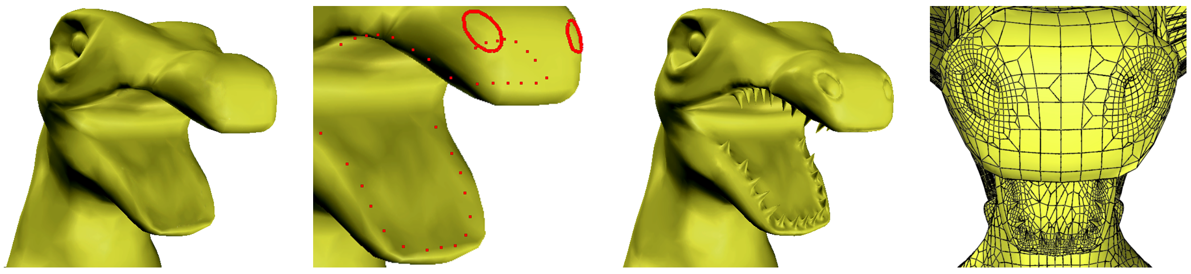
(a) (b) (c) (d)

Figure 11: *The effect of varying the feature sharpness: (a) $s = 100$; (b) $s = 1$; (c) and (d) $s = 0.1$. The base mesh is the same as in Figure 10(a).*



(a) (b) (c)

Figure 12: *A fish mesh augmented with small-scale features: (a) Original mesh; (b) Subtle surface details, such as scales, can be created with our system; (c) Magnified view of the selected area from (b).*



(a) (b) (c) (d)

Figure 13: *A simple dragon mesh is enriched with some features: (a) Original mesh; (b) A combination of stroke styles are used to create different effects; (c) The addition of nostrils and teeth make for a much more convincing and menacing dragon; (d) Wireframe rendering of the augmented mesh.*

to make to our system. We plan to extend our approach to handle triangular meshes, which are popular in modeling and are often easier to process than quadrilateral meshes. While the particular implementation presented here is most suitable to quadrilateral meshes due to our use of Catmull-Clark subdivision, the general method is applicable to any mesh structure or subdivision scheme. In particular, adaptive Loop subdivision could be used to produce great results for triangle meshes.

As Biermann et al. [BMZB01] recognized, a sketching interface provides a convenient way to allow mesh trimming operations. At the moment, we don't differentiate between open and closed feature curves, but a closed curve splits the mesh into two regions: inside and outside the curve. The mesh can be trimmed by discarding one of these regions.

Even with open curves, we may be able to partition the mesh into distinct halves on either side of the curve; think of drawing a line from one edge of a sheet of paper to the other. An exciting possibility is to interpret open curves as defining *crease lines* on the surface, and then allow the surface to be bent about that crease. This functionality would be extremely useful in CAD/CAM tasks such as automobile design.

Feature displacement in our system is always in the normal direction, which limits the expressiveness of the system. Rather than having parameterized profiles, we could extend the sketching metaphor to the feature profiles as well. Features could then have arbitrary profiles and be able to express complex details such as branching or asymmetric folding structures.

Finally, our sketching interface could be improved. Rather than having user-adjustable feature parameters such as height and sharpness, we could transition to a brush- or tool-based sketching metaphor, where each tool would represent some preset combination of feature parameters.

Acknowledgements

This research was sponsored in part by the Natural Sciences and Engineering Research Council of Canada and by the Portuguese Science Foundation (FCT) under grant BSAB-458-2004.

References

[Bly99] BLYTHE D.: Advanced graphics programming techniques using opengl. In *SIGGRAPH 1999 Course Notes* (1999).

[BMBZ02] BIERMANN H., MARTIN I., BERNARDINI F., ZORIN D.: Cut-and-paste editing of multiresolution surfaces. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002), ACM Press, pp. 312–321.

[BMZB01] BIERMANN H., MARTIN I., ZORIN D., BERNARDINI F.: Sharp features on multiresolution subdivision surfaces. *Graphics Models (Pacific Graphics 2001) 64*, 2 (2001), 61–77.

[CMB97] CHAN L. K. Y., MANN S., BARTELS R.: World space surface pasting. In *Proceedings of the conference on Graphics interface '97* (1997), pp. 146–154.

[FKS*04] FUNKHOUSER T., KAZHDAN M., SHILANE P., MIN P., KIEFER W., TAL A., RUSINKIEWICZ S., DOBKIN D.: Modeling by example. *ACM Trans. Graph. 23*, 3 (2004), 652–663.

[IMT99] IGARASHI T., MATSUOKA S., TANAKA H.: Teddy: A sketching interface for 3d freeform design. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques* (1999), pp. 409–416.

[KS99] KHODAKOVSKY A., SCHRÖDER P.: Fine level feature editing for subdivision surfaces. In *ACM Solid Modeling Symposium* (1999), pp. 203–211.

[LF03] LAWRENCE J., FUNKHOUSER T.: A painting interface for interactive surface deformation. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications* (2003), pp. 141–150.

[MM01] MA M., MANN S.: Multiresoluton editing of pasted surfaces. In *Mathematical Methods for Curves and Surfaces: Oslo 2000* (2001), Vanderbilt University, pp. 273–282.

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

[PS04] PAKDEL H., SAMAVATI F.: Incremental adaptive loop subdivision. In *International Conference on Computational Science and its Applications* (2004), pp. 237–246.

[YZX*04] YU Y., ZHOU K., XU D., SHI X., BAO H., GUO B., SHUM H.-Y.: Mesh editing with poisson-based gradient field manipulation. *ACM Trans. Graph. 23*, 3 (2004), 644–651.

[ZG04] ZELINKA S., GARLAND M.: Mesh modeling with curve analogies. In *Proceedings of Pacific Graphics 2004* (2004), pp. 94–98.

[ZHH96] ZELEZNIK R., HERNDON K., HUGHES J.: Sketch: An interface for sketching 3d scenes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (1996), pp. 163–170.

[ZS00] ZORIN D., SCHRÖDER P.: Subdivision for modeling and animation. In *SIGGRAPH 2000 Course Notes* (2000).