Image-Assisted Modeling from Sketches

Luke Olsen*
University of Calgary

Faramarz F. Samavati
University of Calgary

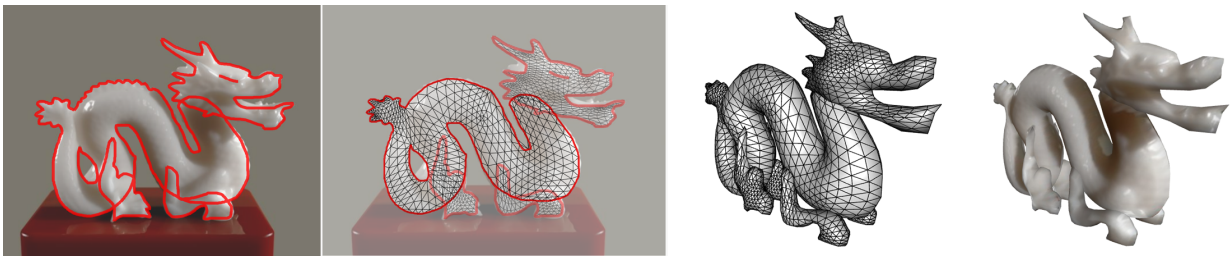


Figure 1: A porcelain dragon is modeled from a single image, with part boundaries sketched by the user. A feature-sensitive meshing algorithm, combined with automatic texturing, produces a visually pleasing 3d representation of the object.

ABSTRACT

In this paper, we propose a method for creating freeform surfaces from sketch-annotated images. Beginning from an image, the user sketches object boundaries, features, and holes. Sketching is made easier by a magnetic pen that follows strong edges in the image. To create a surface from the sketch, a planar mesh is constructed such that its geometry aligns with the boundary and interior features. We then inflate to 3D using a discrete distance transform filtered through a cross-sectional mapping function. Finally, the input image is applied as a texture to the surface. The benefits of our framework are demonstrated with examples in modeling both freeform and manufactured objects.

Index Terms: I.3 Computer Graphics [I.3.5]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems [I.3.6]: Methodology and Techniques—Interaction Techniques [I.3.7]: 3D Graphics and Realism—Texture

1 INTRODUCTION

It is apparent to anyone visiting the cinema or playing a modern video game that the range of possible 3D models is limited only by one's imagination. The question is, how much effort does it take to create a high-quality model? 3D modeling is a pursuit that requires extensive training and experience. Traditional interfaces offer tedious mouse-based click-and-drag interaction, but great advances in usability have been made in recent years as the research focus has shifted toward improved interaction.

Sketch-based interfaces for modeling (SBIM) are a promising trend in user interaction, offering simple and fast ways to perform certain modeling tasks. Freehand input has been used to replace traditional click-and-drag control point manipulation in tasks such as object creation, composition, deformation, and augmentation. From a broader perspective, the problem of interpreting freehand sketch input as a 3D object involves several disciplines, including computer vision, human-computer interaction, and cognitive science (the study of perception).

A typical sketch-based system for model creation presents the user with a blank "canvas" within which they draw some representation of the object. For novice artists, drawing an accurate shape without assistance is an arduous task – even drawing a square in the correct proportions is difficult, and trained artists tend to draw with incorrect perspective effects [28]. Tracing an image, however, can be done by anyone.

In this work, we explore an image-centric interface to make sketch-based modeling more accessible to inexperienced users. A planar-mesh inflation approach is used for shape reconstruction, in which the sketched silhouette is embedded in the 2D image plane. A single image of an object is often enough to create a model in our system; for more complex objects, the object can be modeled parts-wise from multiple images and composited together. To make the output models suitable for further editing, we propose a surface reconstruction method that creates semi-regular meshes with feature-aligned geometry. The input image is also used as texture information for the output model, increasing visual quality without a corresponding increase in geometric complexity.

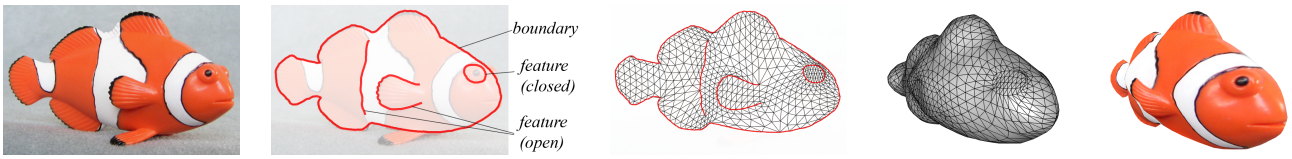
Our main contributions are: an image-assisted sketching interface, in which a "magnetic" pen can be used to align input strokes with image features; a novel planar meshing approach that supports interior features and holes; an inflation method based on the discrete distance transform, allowing for custom cross-sections; and finally, automatic texture-mapping of the created surfaces with the input image.

1.1 Related Work

Our work is most related to inflation- and "skeleton"-based freeform sketching systems [10, 30, 3, 19], in which the common element is the interpretation of a sketched line as an object boundary contour. A smooth 3D surface passing through the boundary can be constructed from a 2D skeleton by relating the surface-to-skeleton distance to the boundary-to-skeleton distance [23]. An alternative approach [19] uses functional optimization to define a smooth surface passing through the sketched boundary.

Mesh-based inflation systems typically use a planar (2D) meshing algorithm to construct the mesh topology and connectivity. The original Teddy [10] computes the Delaunay triangulation (DT) of dense boundary points; the lack of points inside the boundary often results in poorly-shaped triangles, however. Conformal triangulations [27] add points to the DT to satisfy constraints on triangle aspect ratio and angles. Nealen et al. [19, 20] trim a regular triangle grid to the sketch area, ensuring well-shaped triangles with mostly regular valence.

*e-mail: olsenl@cpsc.ucalgary.ca



(a) Input image (b) Sketch (c) Planar mesh (d) Inflated surface (e) Textured

Figure 2: System overview: (a) the user provides an image, then (b) sketches contours, features, and holes of the object; (c) the system generates a feature-sensitive semi-regular planar mesh and (d) inflates in 3D; (e) the image is used to texture-map the object.

Our work is also related to image-based modeling. Automatic methods exist for constructing a 3D representation of an object from multiple images, such as the "turntable" approach [6]. When the images come from less controlled sources, some user interaction is required to indicate objects of interest [35, 39]. Modeling from a single image is not possible in general due to incomplete information, though properties such as symmetry can be exploited [38, 12].

In the context of image-based modeling and SBIM, perhaps the first example is Williams' 3D Paint system [36], in which sketched Coons patch boundaries and a user-edited displacement map are combined with a painted or acquired texture to create a 3D model. Liu & Zhang [16] use edge detection to automatically extract a boundary contour from an image, followed by inflation to a 3D surface; however, the mesh quality is poor and limited to genus-0 topology, and automatic segmentation is not very reliable. Kara et al. [13] allow the user to trace over a conceptual design image, though the image plays no active role in the system. Yang et al. [37] use traced images to texture-map parametric models created from sketched parts. Most recently, Gingold et al. [8] describe an SBIM system in which complex models are created by sketching primitive shapes and then *annotating* the drawing to indicate inter-relationships such as symmetry and connectivity. The system allows for a user-specified image as a passive drawing guide, but the image does not inform any internal system component.

2 Modeling from a Single Image

In this section, we describe the components of our system for modeling objects from a single image and a sketch (Fig. 2). We first describe the input, an automatically-refined user sketch (Sec. 2.1). Each stroke of the sketch is then classified as an object boundary or feature (Sec. 2.2.1), and that information is used to construct a planar mesh of the sketch area (Sec. 2.2.2). Finally, the planar mesh is inflated or extruded into a 3D surface, and the input image is applied as a texture map (Sec. 2.3-2.4).

Each constructed object is symmetric about the drawing canvas, a result of the planar meshing approach. Thus, the method works best when the image depicts a symmetric view of the object – for example, the dragon and the fish of Figs. 1-2 are side views. A large class of objects can be modeled from piecewise-symmetric parts, as shown in our results (Sec. 3). However, more complex shapes may require other modeling tools such as free-form deformation.

2.1 Image-Assisted Sketching

Sketching an object with complex boundary and internal features is a difficult and time-consuming task, especially for untrained artists. That is why most modeling software allows the user to display an image in the editing window – to assist with the proportions and position of various elements. This approach is very passive, as the image is not exploited by the software. A single image is full of useful information that can be used to refine or assist the input sketch.

We provide two drawing tools to the user: a regular freehand pen, and a "magnetic" pen that tries to follow nearby edges in the image (Fig. 3). Tsang et al. [32] describe a similar "image-guided" sketching system, based on active contours [17], for snapping input to scanned sketch images; that is, the important edges are clear and distinguishable. In our scenario we are more concerned with photographs.

The magnetic pen in our system is a variation of the *intelligent scissors* image segmentation tool introduced by Mortensen & Barrett [18]. In their method, an image is modeled as a graph with edges between neighboring pixels. The edge weights are derived from the image gradient, such that moving along strong edges has lower cost than moving across homogeneous regions. A pathfinding algorithm is then used to find the lowest-cost path between "seed" points placed by the user. In this way, an object can be segmented from the background with as few as two seed points. The seed points create localized and more predictable snapping behavior than the globally-optimal active contour model.

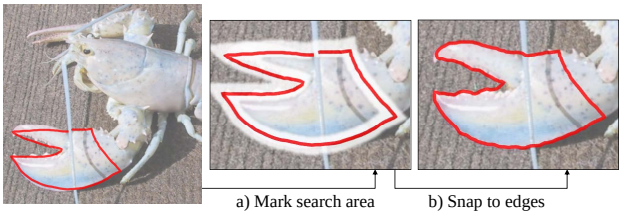


Figure 3: The magnetic pen: the input stroke defines a search region (a) within which the stroke is snapped to strong edges (b).

Our usage scenario is different, in that we want to support freeform and natural sketching. We only want to refine or *snap* the stroke when the input stroke appears to be following an edge. Thus we restrict the search space to a fixed distance from the stroke. For seed points, the polyline approximation method described in Sec. 2.2.2 is used to automatically extract them from the input strokes. As well, we really only want to snap to strong edges; if the user draws a stroke cutting through a low-gradient region, we preserve their original stroke/intent rather than choose a path that is less direct but slightly more optimal.

The input strokes $\{S_1, \ldots, S_k\}$, obtained from a mouse or tablet device, are represented as a sequence of point samples $S_i = \{p_{i0}, \ldots, p_{in}\}$. Each point $p_{ij} \in \Re^3$ is projected from window coordinates onto the drawing (x-y) plane (Fig. 4a). (Because our interface supports panning and zooming of the sketch canvas, window coordinates are not used directly.) The input is uniformly resampled by discarding extraneous points and interpolating between widely-spaced samples. Since we wish to support small-scale features and sharp corners, we do not filter or smooth the input. Instead, oversketching [7] is supported to allow the user to correct mistakes.

Though strokes are stored in projected world coordinates, some operations in our pipeline occur in raster space – that is, a discrete pixel sampling of the drawing canvas (Fig. 4b). World-space strokes are useful for determining the speed and direction of drawing, identifying feature points, resampling, and rendering. We use raster-space processing to classify strokes (Sec. 2.2.1), which makes the method agnostic to the drawing order and leads to more natural sketching.

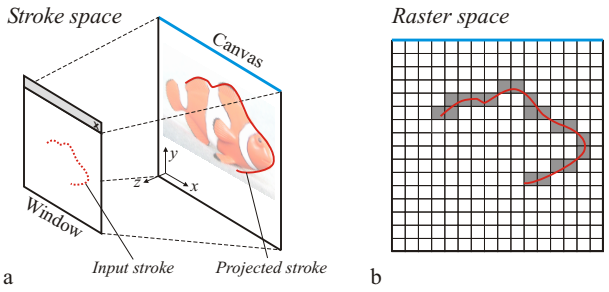


Figure 4: (a) Stroke space: input strokes (in window coordinates) are projected onto the sketch canvas (in world coordinates); (b) Raster space: the sketch canvas is drawn to a discrete pixel grid for stroke classification (Sec. 2.2.1).

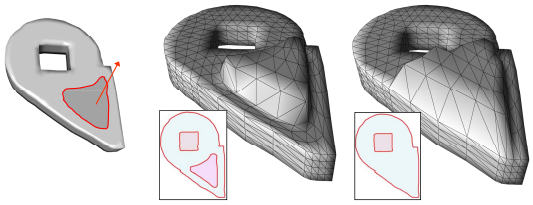


Figure 5: *Left:* Feature-based meshing enables intuitive editing. *Right:* If the geometry does not align with the desired deformation, the results are poor.

2.2 Planar Meshing

Many mesh editing operations (e.g. [24, 21]) require aligned edges and vertices to represent the details (Fig. 5). If the required geometry does not exist, it has to be created by localized re-meshing or vertex relocation. These operations can often create artifacts or poor-quality triangles, especially over successive applications. We propose a feature-sensitive planar meshing in which all strokes in the original sketch are taken into consideration during the meshing process. The required geometry then exists for further editing.

The current state-of-the-art in inflation-based SBIM is FiberMesh [19, 20], a descendant of Teddy. To create a planar mesh, the area enclosed by a boundary stroke is filled with a regular triangle mesh by growing outward from a single seed triangle. To increase the regularity of the triangles and vertex valences, a boundary energy condition is imposed. This approach has some limitations: first, the triangle size determines the scale of sketch features that can be captured, necessitating a fairing approach to widen narrow regions; second, the triangle size is uniform over the whole area; most importantly, adding interior features requires local remeshing, which degrades the mesh quality over subsequent applications.

We also strive for vertex and triangle regularity in the planar mesh, but approach the problem from a different direction. We note that most subdivision schemes produce regular-valence vertices and well-shaped triangles. For example, in Loop's scheme all inserted interior *edge* vertices are regular (6 neighbors for triangular meshes); thus, the only irregular vertices are those that exist at the base level or lie on a boundary. To exploit this behavior in the planar meshing stage, we first create a coarse tessellation and then refine it via subdivision. This involves three main steps: stroke classification, coarse tessellation, and refinement.

2.2.1 Stroke Classification

Whereas most preceding SBIM systems operate on single-stroke input, our system accepts an arbitrary number of strokes and places no constraints on directionality or order. Therefore, before we can construct a planar mesh, we need to determine some high-level structure of the input sketch. As shown in Fig. 2b, a stroke can define either an object *boundary* or a *feature*. Feature strokes can be either *open* or *closed*, and are associated with their containing boundary stroke. The problem then becomes one of determining a containment hierarchy.

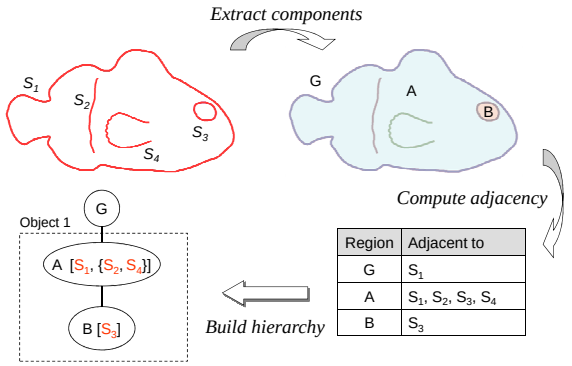


Figure 6: Stroke classification: first, the connected components, or "blobs", are extracted; second, the blobs are processed to extract adjacency information; finally, the adjacency information is used to construct a region hierarchy. Each child of the background blob is a unique object in the sketch.

An important decision is whether to construct the hierarchy in stroke- or raster-space. We choose the latter for two reasons. First, rasterizing the strokes removes any effect of drawing direction, order, and continuity; if strokes are connected in raster space, then they're adjacent in stroke space. This allows for more natural sketching, because the algorithm is concerned with *what* was drawn rather than *how*. Second, the raster algorithms we use are computationally efficient and depend only on the number of pixels, not the number of stroke samples. This allows the hierarchy to be updated transparently in the background as the user draws.

To find the stroke containment hierarchy, we employ the connected component (CC) labeling algorithm [31] for identifying contiguous 'blobs' in an image. The input to the algorithm is an image I_{sk} containing the rasterization of all user strokes, each colored uniquely. The output is a set of blobs $\{B_1, \ldots, B_n\}$, where each blob is a connected set of pixels of a single color. This allows us to distinguish between blobs formed by stroke and background pixels. We currently assume that each stroke is a single unbroken line; this could be overcome by using a stroke-blending method [26] or by tracing the contours of the background blobs.

In the simplest case, a single boundary stroke will have three blobs: the background, the *region* bounded by the stroke, and the stroke itself. In the example shown in Fig. 6, there are 7 blobs: the background G, regions A and B, and the four strokes.

After identifying the blobs, the labeled image (in which all pixels belonging to blob B_i have unique color C_i) is post-processed to extract adjacency information. For stroke blobs, we're interested in the adjacent background blobs. If a stroke is adjacent to only 1 blob (e.g. S_4 adjacent to region A in Fig. 6), then it is classified as an open feature stroke. Strokes adjacent to 2 blobs can be either boundary or closed feature strokes, contingent on whether one is the background blob. For example, S_2, adjacent to A and B, is a feature stroke, while S_1 is a boundary stroke. For each background blob (that is, a region enclosed by a stroke), we need to find adjacent background blobs – these will be separated by a stroke. Thus, for each pixel we check pixels $\pm w$ rows and columns away, where w is the width of a rasterized stroke. Let the largest background blob be the root blob G – it will be the size of I_{sk}, so long as no strokes touch the boundary. Any background blob adjacent to it defines a

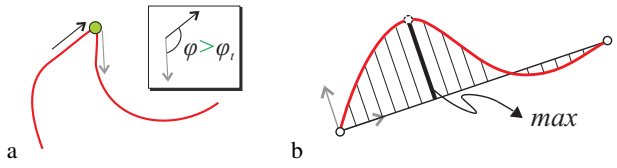


Figure 7: Polyline approximation of a stroke: (a) corner points are placed when the direction change is high; (b) stroke segments are split when the maximum deviation is too high.

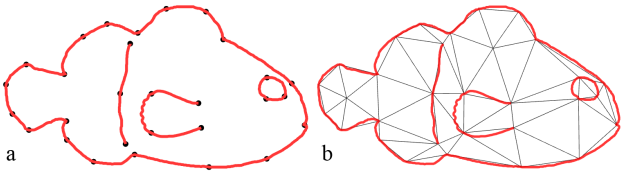


Figure 8: Planar meshing: (a) the input strokes are approximated with polylines (black points); (b) the CDT of those points yields a coarse mesh.

unique object (e.g. region A); all other blobs are children of these objects.

From this adjacency information, we construct a region hierarchy. A sketch region $R = [S_b, F, C]$ is defined by a set of strokes, where S_b is a closed stroke defining the boundary and $F = \{S_{f1}, \ldots S_{fm}\}$ is a set of feature strokes contained in the region. The final set $C = \{c_1, \ldots, c_k\}$ is the set of other sketch regions fully contained within R. Figure 6 illustrates the region hierarchy of a simple sketch.

2.2.2 Coarse Tessellation

Each object is composed of a boundary stroke and zero or more features. To create a coarse mesh, we first approximate each stroke with a sparsely-sampled polyline and then compute the Delaunay triangulation of those points. There are two main qualities desired in the polyline approximation $P_i \subset S_i$: feature sensitivity (i.e. low approximation error), and compactness. It is important to minimize the number of points because P_i is not a final approximation of S_i – the base mesh will later be refined with subdivision. Limiting the approximation error is also important, as high error is difficult to overcome in the subdivision stage. The benefit of a feature-sensitive approach over uniform spacing is that sharp features can be captured, and less geometry is devoted to on low-detail areas.

We use a two-stage approach to construct P_i. First, sharp corners are found by looking the local change of direction ϕ at each stroke sample (Fig. 7a). A directional change exceeding a threshold angle ϕ_t indicates that the sample is at or near a sharp corner. There will typically be a few consecutive samples meeting this criteria at each corner; these are clustered to a single point, and all such points are retained as vertices in P_i. In the second stage, we employ an iterative splitting scheme [4] to split the stroke segments between corner points. The point of maximal deviation from a straight-line approximation is chosen as the location of a new polyline vertex, splitting the segment into two parts (Fig. 7b). Each sub-segment can then be split in the same fashion, until either the maximal point is within δ_{DP} of the straight line, or the segment is too short to split.

This construction has two parameters for controlling the approximation quality, ϕ_t and δ_{DP}. We have found values that work well enough in most cases ($\phi_t = \frac{\pi}{4}$, $\delta_{DP} = 5\%$ of the canvas size), although having them adjusted automatically according to the sketch would be ideal.

After each stroke has been approximated with a polyline, we compute a coarse mesh M_{p0} as a constrained DT (CDT) of the vertices, with the stroke segments acting as constraints (Fig. 8). That is, if two vertices are connected by a stroke segment, then the DT is constrained to contain an edge between those vertices; this is consistent so long as no two strokes intersect each other. Terminal triangles (those having three boundary vertices) are problematic during surface construction, since they result in non-manifold connectivity. To remove terminal triangles, we use a merge-and-split approach similar to Teddy, in which each terminal triangle is merged with adjacent faces, and the resulting (non-triangle) face is then split by inserting a new vertex at the center of the face.

2.2.3 Mesh Refinement

The coarse mesh M_{p0} only approximates the input sketch, but the final planar mesh should match the user's sketch as closely as possible. To better match the input strokes, we employ a subdivision-with-snapping technique (Fig. 9). Stroke segments are associated with edges based on the polyline approximation. As the mesh is subdivided, new vertices are created along edges and then "snapped" onto the stroke segment. After several levels, the edges in the planar mesh follow the original sketch very closely.

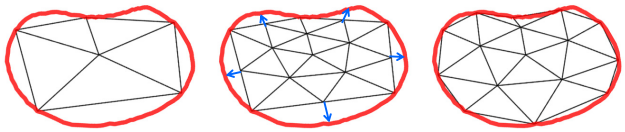


Figure 9: Mesh refinement (*left to right*): the coarse base mesh; after subdivision; after snapping.

We use smooth subdivision rather than linear, as it results in better vertex distribution in the sketch interior. In the presence of interior feature strokes, however, the use of smooth subdivision can lead to unwanted artifacts as triangles are compressed or stretched. To mitigate these issues, when interior vertices are snapped onto a stroke a portion of their displacement (we used 30%) is passed on to adjacent vertices.

2.2.4 Extensions

To offer more control and variety in the planar mesh geometry, we support a couple of variations to the method described above. First, quadrilateral meshing is often preferable to triangulation (e.g. for manufactured object design). To create a planar quad mesh, we perform the coarse CDT triangulation and then employ a greedy triangle-merging approach [11] (Fig. 10). Although some triangles may remain in the base mesh, Catmull-Clark subdivision [2] – which produces only quads after one iteration – is used instead of Loop in the refinement stage.

A second improvement is to add some points to the base mesh; as Fig. 11a shows, when the sketch has a large interior with no features, the lack of Delaunay points can result in large region-spanning triangles. We use a random dart-throwing approach [33], with the restriction that any inserted dart p_d must be at least r_d from the boundary and all other darts. In our experiments, setting

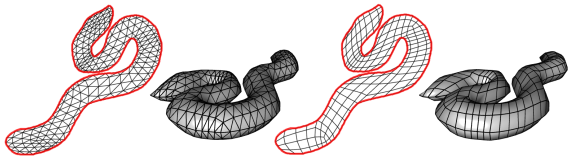


Figure 10: Our system can produce either triangle or quad meshes.

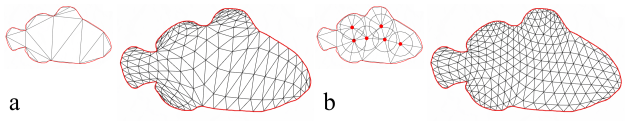


Figure 11: Planar mesh (a) without and (b) with dart-throwing to generate interior points.

$r_d = \frac{1}{2} max(T(I))$ works well, where $T(I)$ is the discrete distance transform discussed in Sec. 2.3. The result is a more regular distribution of triangles in the sketch interior (Fig. 11b).

2.3 Surface Reconstruction

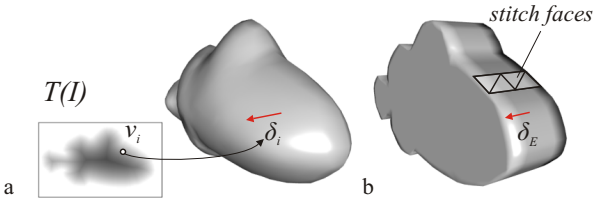


Figure 12: (a) Inflation uses a distance transform $T(I)$ to displace vertices in M_p; (b) extrusion uses uniform displacement and requires additional "stitch" faces.

The planar mesh we have constructed could be used in any previous inflation method, such as the non-linear optimization approach of FiberMesh [19]. Optimization is a "black-box" approach that offers no recourse for the user if the result is unsatisfactory or if a non-smooth result is desired. In this section, we explore a constructive inflation method that allows for customization and control of the surface cross-section. We provide two reconstruction variants: *inflation* for smooth objects, and *extrusion* for rectilinear shapes (Fig. 12).

In either case, our default interpretation is to construct a mesh M symmetric about the drawing (x-y) plane. In the absence of any depth (z) information, this is a justifiable assumption. Let M_{front} be the portion of M visible from positive z, and M_{back} be the $-z$ portion; then $M = M_{front} \cup M_{back} \cup M_{stitch}$ where M_{stitch} is a set of faces that may be necessary to bridge the front and back. The final mesh is also semi-regular, suitable for multiresolution editing [40, 22] and compression [14] applications.

To inflate a planar mesh M_p, interior vertices are "pushed away" from the drawing plane by an amount determined by the distance d to the boundary, while the boundary remains fixed. This requires an efficient way to compute d. A chordal axis, approximated from a Delaunay triangulation, can be used to estimate the distance. However, because we are using a very coarse DT, the chordal axis is not very accurate. Furthermore, it is not clear how to define the chordal axis in the presence of feature strokes and holes. Therefore, we need a more robust way to estimate d.

We use a discrete image-based distance transform $T(I)$ [5]. Given a binary image I, $T(I)$ is an image in which the value of a pixel encodes the distance to the nearest black pixel. That is, if $I(i,j)$ is black $T(I(i,j)) = 0$; pixels adjacent to black pixels have value 1, and so forth. Figure 13a shows an example of $T(I)$ (inverted for clarity). The purpose of a distance transform is to estimate the distance from an interior vertex to the boundary. Thus, only boundary strokes should be drawn into I – including hole boundaries, but not tunnels (see Fig. 15). In a sense, this creates a displacement map whose usage is similar to Williams' system [36], but derived automatically from the input strokes rather than acquired from a range scanner or painted by a user.

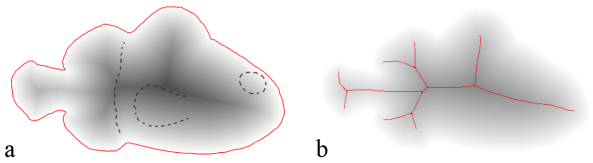


Figure 13: (a) The distance transform $T(I)$ defined by a boundary stroke; (b) The set of local-max pixels defines the object's skeleton.

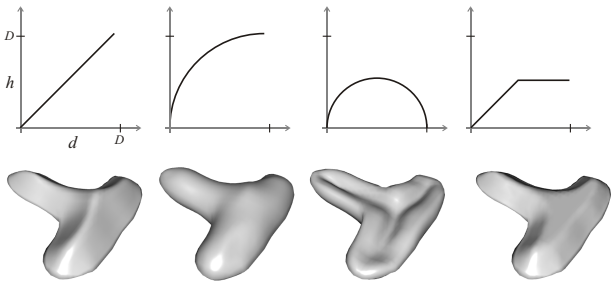


Figure 14: The distance d in $T(I)$ can be mapped to an inflation offset h by an arbitrary function, giving control over the cross-section.

For any vertex $v_i \in M_p$, we can map it to pixel-space, look up the distance d to the boundary in $T(I)$, and map that distance to displacement δ_i. We generally do not want to use d directly, however, because it is linear. To inflate a smooth object, a circular cross-section can be attained by the mapping $\delta = \sqrt{D^2 - (D-d)^2}$, where $D = \max_d T(I)$. In practice, we use a local maximum, which corresponds to the radius on the skeletal axis (Fig. 13b). A more robust skeleton-extraction method, such as [15], could be used to remove the spurious branches (for example, in the fish's tail).

Other mapping functions can be used to achieve different smooth or non-smooth cross-sections; Fig. 14 shows some examples. This approach is similar to the cross-sectional blending surfaces proposed by Cherlin et al. [3], but extended to arbitrary topology surfaces (equivalently, shapes with branching or looping skeletons).

Let $\Delta = \{\delta_1, \delta_2, \ldots\}$ be the set of these per-vertex displacements. In the inflated mesh M, then, we have $M_{front} = M_p + \Delta$ and $M_{back} = M_p - \Delta$. No stitch faces are required ($M_{stitch} = \emptyset$), but boundary vertices in the front and back should be merged to create a sealed mesh. Because M_{front} and M_{back} are both semi-regular, the union M is as well.

Inflation by displacing vertices purely in the z direction results in triangles near the planar mesh boundary being stretched in 3D. To have nice triangles in 3D requires either irregularly-shaped triangles in the plane (i.e. the near-orthogonal projection of a well-shaped triangle), post-inflation vertex movement [9], or remeshing [25]. Currently this is not implemented in our system.

To have more expressive range, our system also supports *extrusions*, which are more suitable to mechanical or engineered surfaces. To extrude the planar mesh, we displace the vertices uniformly by δ_E to create the front mesh, and duplicate them in the negative z direction for the back mesh. The front and back then need to be stitched together with new faces (Fig. 12b). This stitching must be done with some care if we want to preserve semi-regularity. At the base level, each boundary edge in M_{front} should be connected by two triangles (or a single quad) to the corresponding edge in M_{back}. Therefore, if M_p were subdivided k times, the gap should be stitched with $2k$ triangles (k quads).

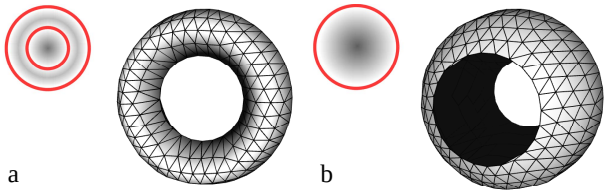


Figure 15: (a) Holes are created by trimming faces from M_p and including the hole's boundary in $T(I)$. (b) Tunnels are created by trimming faces from M, with no change to $T(I)$.

2.3.1 Holes & Tunnels

To create a torus, one would have to sketch two concentric circles, one corresponding to the surface and the other to the hole. However, the same sketch might correspond to a rim-and-tire object, with no hole but simply a boundary between the regions. Because of this inherent ambiguity in interpretation, we leave it to the user to identify holes. By default all regions are assumed to be filled, but any closed region that is a child of another region can be designated as either a *hole* or a *tunnel*. The difference between these two is illustrated in Fig. 15.

A hole is created *before* reconstruction by removing faces overlapping the hole from M_p. Vertices around the hole should have zero displacement, so its boundary stroke is included in the distance transform. A tunnel is created *after* reconstruction by removing faces in the hole region from M. Vertices on the hole's boundary should be displaced as usual, so the boundary stroke is not included in the distance transform.

2.4 Texturing

In most modeling programs, the modeling and texturing phases are disjoint – first the model is created, and then the textures are painstakingly applied. Predefined mappings such as cube- or sphere-maps can make the process easier, but are suitable only for certain objects. In our system modeling and texturing are intertwined, as the nature of our symmetric mesh construction provides a natural way to interpret the image as a texture. We use a simple projective mapping: for a texture with width w, height h, and origin (x_o, y_o) (in world coordinates), the texture coordinate (s,t) of a vertex $v = (x,y,z)$ is $s = (x - x_o)/w$ and $t = (y - y_o)/h$.

This type of mapping exhibits distortion on geometry that is aligned with the projection axis (z). Where the surface is parallel to the drawing plane the projection is roughly uniform; near the boundary, however, a large portion of the surface projects to a relatively small portion of the texture. This is especially problematic with extruded surfaces, because all stitch faces are parallel to the view direction and thus project to the same texture coordinates.

Note that from the input sketch we have a set of image-texture correspondences. To reduce texture distortion along the seam, in the future these could be used as input to a more robust texturing approach, such as Tzur and Tal's method based on local camera parameter recovery [34].

3 Results

We have used the techniques described in the paper to create a variety of models from photographs. The authors are not professional modelers or artists, so sketching realistic objects without an image-assisted system is beyond our capabilities. Using our framework, we are able to create high-quality models quickly and easily.

As Fig. 16 shows in a detailed view, our system produces meshes whose geometry closely follows the sketched feature strokes. This is beneficial for making complex models, as the mesh has the geometry in the right place for performing further deformations and refinements.

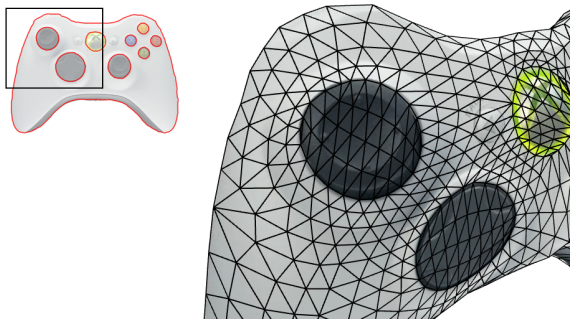


Figure 16: Mesh geometry respects the sketched features.

Figure 17 shows several results demonstrating the capabilities of our system. Most are multi-part objects that were reconstructed piecewise and then assembled via traditional methods. Together they demonstrate all of the benefits of our framework: the lock and key have complex holes and tunnels; the gun and plane show both triangle and quad meshing. The automatic texturing gives each model a strong visual impact, and the underlying geometry is simple and clean. Most importantly, these models were created easily by untrained modelers in under ten minutes.

The creation of each part via the methods described in this paper is very fast, taking only a few seconds. The actual sketching phase can be moderately time-consuming (a minute or two in these examples), depending on how many features, and how carefully, the user chooses to sketch. The planar mesh construction happens interactively as the user sketches, as does the distance transform computation. Thus, when the user is satisfied with their sketch and clicks *Inflate* or *Extrude*, the 3D model can be constructed instantly. The bulk of the creation time for the results shown here is spent placing the parts: primarily, translating the part in the depth. In the future, we'd like to offer quicker sketch-based composition tools, such as sketching part connection points.

4 Conclusion & Future Work

In this paper, we have described a sketch-based system for modeling single-image objects. Our planar mesh construction supports sketches with not just boundary strokes, but also interior features and holes. This allows for more topological variety and feature-aligned mesh geometry. To create a 3D surface, we proposed a discrete distance transform method that allows for a variety of cross-sections. The reconstructed objects are planar-symmetric; to attain more complex shapes, the user can sketch multiple parts either from a single image or multiple images, and then assemble the parts with affine transformations.

We have also advocated an image-oriented approach, in which an image acts as an active drawing guide. The clear image–sketch correspondence allows for "free" texture mapping, increasing the visual quality of the output and automating a tedious task. This allows skilled and unskilled users alike to create visually appealing and proportionally correct models by simply tracing an image. The benefits of this construction were illustrated with several examples.

Future work. While some informal feedback has been garnered from modelers and artists, a formal user study is necessary to evaluate the usability of our framework. Undertaking such a study is the next stage of our research.

Our current system currently does not provide tools for modifying an existing surface, such as extruding one surface from another, or blending surfaces together. In the future, we would like to of-

Figure 17: Several models created with our system; for each result, the input sketch(es), textured model, and underlying geometry are shown. The modeling times ranged from 5-10 minutes for each. [Image Credits: Lobster: *glf.dfo-mpo.gc.ca*. Lock: *tribalartifacts.com*. Key: *charmfactory.com*. Xbox controller: *microsoft.com*. Airplane: *www.cruzware.com*. Seal: *fotolia.com*. Car: *sandsmuseum.com*. Gun: *gamespot.com*.]

fer better sketch-based tools for creating and assembling multi-part objects. Methods such as sketching the connection points on each part [29] and annotating inter-part relations [8] can serve as inspiration here.

The texturing component should also be extended to improve the quality along the object's seam. This could include a multi-image integration technique [1], or perhaps a non-linear mapping. We would also like to allow for the front and back surfaces to have separate textures.

The raster-based region extraction used for stroke classification has some limitations in the current implementation. In particular, it fails if two strokes intersect. This prevents the user from drawing, for instance, an object with a feature that spans the whole object. Fortunately, this limitation can likely be overcome with minor modifications to our current system.

Some aspects of the system could use further automation. The final mesh quality depends on the polyline precision and the number of subdivisions levels; the results presented here were attained with some tweaking of the parameters. There is an interrelationship between the polylines as well; for example, the best place to connect the endpoint of a feature line is to the nearest boundary point, but there may not be a point there in the boundary's polyline. These aspects should be explored more fully to further improve the mesh quality.

Acknowledgements

This research was sponsored in part by the National Science and Engineering Research Council (NSERC) of Canada and the Informatics Circle of Research Excellence (iCore) of Alberta.

References

[1] J. Arnabat, S. Casanovas, and G. Medioni. 3d modeling from turntable sequences using dense stereo carving and multi-view consistency. In *Proc. of Int. Conf. on Pattern Recognition (ICPR04)*, pages 36–39, 2004.

[2] E. Catmull and J. Clark. Recursively generated b-spline surfaces on arbitrary topological surfaces. *Computer-Aided Design*, 10(6):350–355, 1978.

[3] J. J. Cherlin, F. Samavati, M. C. Sousa, and J. A. Jorge. Sketch-based modeling with few strokes. In *Proc. of Spring Conference on Computer Graphics (SCCG '05)*, pages 137–145, 2005.

[4] D. Douglas and T. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2), 1973.

[5] R. Fabbri, L. D. F. Costa, J. C. Torelli, and O. M. Bruno. 2d Euclidean distance transform algorithms: A comparative survey. *ACM Computing Surveys*, 40(1):1–44, 2008.

[6] A. W. Fitzgibbon, G. Cross, and A. Zisserman. Automatic 3d model construction for turn-table sequences. In *Proc. of the European Workshop on 3D Structure from Multiple Images of Large-Scale Environments (SMILE'98)*, pages 155–170, 1998.

[7] T. Fleisch, F. Rechel, P. Santos, and A. Stork. Constraint stroke-based oversketching for 3d curves. In *Proc. of Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM '04)*, 2004.

[8] Y. Gingold, T. Igarashi, and D. Zorin. Structured annotations for 2d-to-3d modeling. In *Proc. of SIGGRAPH Asia 2009*, pages 1–9, 2009.

[9] T. Igarashi and J. Hughes. Smooth meshes for sketch-based freeform modeling. In *ACM Symposium on Interactive 3D Graphics*, pages 139–142, 2003.

[10] T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy: A sketching interface for 3d freeform design. In *Proc. of SIGGRAPH'99*, 1999.

[11] T. Itoh, K. Shimada, K. Inoue, A. Yamada, and T. Furuhata. Automated conversion of 2d triangular mesh into quadrilateral mesh with directionality control. In *Proc. of 7th International Meshing Roundtable*, 1998.

[12] N. Jiang, P. Tan, and L.-F. Cheong. Symmetric architecture modeling with a single image. In *Proc. of SIGGRAPH Asia 2009*, 2009.

[13] L. Kara, C. D'Eramo, and K. Shimada. Pen-based styling design of 3d geometry using concept sketches and template models. In *Proc. of ACM Solid and Physical Modeling Conference (SPM '06)*, 2006.

[14] A. Khodakovsky, P. Schröder, and W. Sweldens. Progressive geometry compression. In *Proc. of SIGGRAPH '00*, pages 271–278, 2000.

[15] F. Levet and X. Granier. Improved skeleton extraction and surface generation for sketch-based modeling. In *Graphics Interface 2007*, 2007.

[16] S. Liu and Z. Huang. Interactive 3d modeling using only one image. In *Proc. of ACM Sym. on VR Software and Technology*, pages 49–54, 2000.

[17] D. T. M. Kass, A. Witkin. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.

[18] E. N. Mortensen and W. A. Barrett. Intelligent scissors for image composition. In *Proc. of SIGGRAPH '95*, pages 191–198, 1995.

[19] A. Nealen, T. Igarashi, O. Sorkine, and M. Alexa. Fibermesh: Designing freeform surfaces with 3d curves. In *ACM Transactions on Graphics (Proc. of SIGGRAPH '07)*. ACM Press, 2007.

[20] A. Nealen, J. Pett, M. Alexa, and T. Igarashi. Gridmesh: Fast and high quality 2d mesh generation for interactive 3d shape modeling. In *Proc. of IEEE Int'l Conf. on Shape Modeling and Applications (SMI 2009)*, pages 155–162, 2009.

[21] A. Nealen, O. Sorkine, M. Alexa, and D. Cohen-Or. A sketch-based interface for detail-preserving mesh editing. *ACM Trans. Graph.*, 24(3):1142–1147, 2005.

[22] L. Olsen, F. Samavati, and R. Bartels. Multiresolution for curves and surfaces based on constraining wavelets. *Computers & Graphics*, 31:449–462, 2007.

[23] L. Olsen, F. Samavati, M. Sousa, and J. Jorge. A taxonomy of modeling techniques using sketch-based interfaces. In *Eurographics 2008 State-of-the-Art Report (EG'08 STAR)*, 2008.

[24] L. Olsen, F. Samavati, M. C. Sousa, and J. Jorge. Sketch-based mesh augmentation. In *Proc. of the 2nd Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM)*, 2005.

[25] C. G. M. A. P. Alliez, G. Ucelli. *Recent Advances in Remeshing of Surfaces*, pages 53–82. Mathematics and Visualization. Springer Berlin Heidelberg, 2008.

[26] R. Pusch, F. Samavati, A. Nasri, and B. Wyvill. Improving the sketch-based interface: Forming curves from many small strokes. *The Visual Computer*, 23(9-11):955–962, 2007.

[27] J. Ruppert. A delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms*, 18(3):548–585, 1994.

[28] R. Schmidt, A. Khan, G. Kurtenbach, and K. Singh. On expert performance in 3d curve-drawing tasks. In *Proc. of Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM '09)*, 2009.

[29] R. Schmidt and K. Singh. Sketch-based procedural surface modeling and compositing using Surface Trees. In *Proc. of Eurographics 2008*, 2008.

[30] R. Schmidt, B. Wyvill, M. C. Sousa, and J. A. Jorge. Shapeshop: sketch-based solid modeling with blobtrees. In *Proc. of EG Workshop on Sketch-Based Interfaces and Modeling (SBIM '05)*, 2005.

[31] L. Shapiro and G. Stockman. *Computer Vision*. Prentice Hall, 2001.

[32] S. Tsang, R. Balakrishnan, K. Singh, and A. Ranjan. A suggestive interface for image guided 3d sketching. In *Proc. of the SIGCHI conference on Human factors in computing systems (CHI '04)*, pages 591–598. ACM, 2004.

[33] G. Turk. Re-tiling polygonal surfaces. *SIGGRAPH Computer Graphics*, 26(2):55–64, 1992.

[34] Y. Tzur and A. Tal. Flexistickers: photogrammetric texture mapping using casual images. *ACM Trans. Graph.*, 28(3):1–10, 2009.

[35] A. van den Hengel, A. Dick, T. Thormählen, B. Ward, and P. H. S. Torr. Videotrace: rapid interactive scene modelling from video. In *Proc. of SIGGRAPH '07*, page 86, 2007.

[36] L. Williams. 3d paint. In *Proc. of 1990 Symposium on Interactive 3D graphics (SI3D '90)*, pages 225–233, 1990.

[37] C. Yang, D. Sharon, and M. van de Panne. Sketch-based modeling of parameterized objects. In *Proc. of Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM '05)*, 2005.

[38] L. Zhang, G. Dugas-Phocion, J.-S. Samson, and S. M. Seitz. Single view modeling of free-form scenes. In *Proc. of CVPR'02*, 2002.

[39] R. Ziegler, W. Matusik, H. Pfister, and L. McMillan. 3d reconstruction using labeled image regions. In *Proc. of the Eurographics/ACM SIGGRAPH Symposium on Geometry processing (SGP '03)*, pages 248–259, 2003.

[40] D. Zorin, P. Schröder, and W. Sweldens. Interactive multiresolution mesh editing. In *Proc. of SIGGRAPH '97*, pages 259–268, 1997.