NaturaSketch: Modeling from Images and Natural Sketches

Luke Olsen[1], Faramarz F. Samavati[1], and Joaquim A. Jorge[2]

[1]University of Calgary, Alberta, Canada
[2]Instituto Superior Técnico, Lisbon, Portugal

***Abstract*—Sketching on paper is a quick and easy way to communicate ideas. However, many sketch-based systems require people to draw in contrived ways instead of sketching freely as they would on paper. Our NaturaSketch system affords more natural interfaces by allowing them to use multiple strokes that overlap, cross and connect. Other contributions include a meshing algorithm to support multiple strokes of different classifications, which enables users to design complex 3D shapes from sketches drawn on images. To provide a familiar workflow for object design, modeling and editing operations can also be specified through a set of sketch annotations. User tests indicate that this approach empowers designers to produce a variety of models quickly and easily.**

I. Introduction

Sketching on paper is a quick and easy way to communicate ideas. In computer modeling, sketches might be used for both rough conceptualizations and detailed design drawings. Many experienced artists have workflows based around 2D software such as Photoshop, which allows them to create detailed design sketches from one or two viewpoints.

Sketch-based interfaces for modeling (SBIM) have been proposed to leverage this sketching ability in the digital domain. However, understanding and interpreting sketches depends on deeply-ingrained visual rules and vast shape memories [Hof00], and translating these processes to the computer is a challenging task. As such, in many sketch-based systems users must draw in a contrived manner instead of sketching freely as they would on paper. We believe that how a sketch is drawn (eg. number and order of pen strokes) should make little difference to what the systems perceives it to be.

Most sketch-based mesh creation tools offer single-stroke input for creating an initial mesh, perhaps followed by a sketch-rotate-sketch workflow for finding the correct viewpoint and adding details with 3D editing techniques [IMT99], [NISA07]. While experienced modelers can create detailed objects in this way, it lacks the exploratory and evolutionary aspects of sketching on paper. If we instead wait for the user to initiate the construction, users have the opportunity to sketch with many strokes, erase, reposition the canvas, and sketch over existing lines as they would on paper, leading to a more natural sketching interface. Such an interface fits well with the workflow used by experienced 2D artists.

The NaturaSketch system supports more natural sketching by allowing input to contain multiple strokes that overlap, cross, and connect. Combined with a meshing algorithm that supports multiple strokes of different classification, users can design complex 3D meshes easily from sketches and images. Figure 1 shows the process used by NaturaSketch to convert a sketch to a 3D model. The power and utility of the system – of which some aspects have been published previously [OS10a], [OS10b] – is demonstrated with several results and an application to 3D image conversion.

II. User Interface

The NaturaSketch interface is primarily a canvas onto which the user draws, along with a few buttons and menu items. The user can also provide an image of the object they are modeling, which will help them to draw in accurate proportion. The sketch is drawn on top of the image with freehand pen marks (shown in black ink) and some annotations indicating operations (red ink).

A sketch is simply a set of strokes acquired from a mouse or tablet. Positional information alone is sufficient, but additional data such as pressure and timestamps (for velocity) are retained when available and used to adjust parameters for stroke extraction (see Section III). Each stroke point is projected from window coordinates onto a 3D drawing canvas, and these projected coordinates are retained. This allows for a robust interface in which the canvas can be panned and zoomed, since the canvas can be any bounded plane in 3D.

Once their sketch is completed, 3D model construction is initiated by the user via a button. Changes to the 3D model can be made easily by returning to the sketch, modifying or adding to it, and re-constructing.

For creating their sketch, we provide three tools to the user. The regular Pen tool allows the user to create their sketch with any number of strokes (Figure 2a). The strokes can overlap, and their relative order is not important. As shown in the example, the user is free to sketch complex objects with multiple parts, regions, and interior features. The algorithm for processing these sketches is described in Sections III-IV.

The Magnetic pen (Figure 2b) is a tool that enables easy tracing of the boundary and feature lines in an image, and is based on *intelligent scissors* image segmentation [MB95]. The image is modeled as a graph with edges between neighboring pixels, and edge weights are derived such that moving along strong edges has lower cost than moving across homogeneous regions. To segment an object from an image, seed points are placed on the object boundary and a pathfinding algorithm

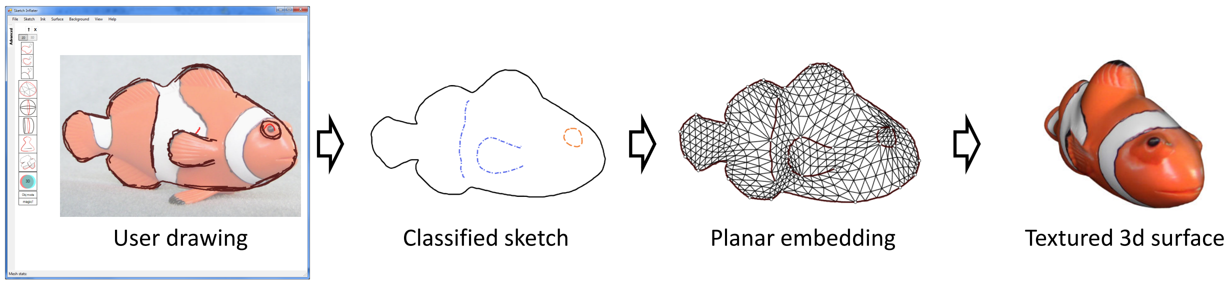


Fig. 1. The NaturaSketch system. First the user freely sketches over an image. Then the system automatically extracts and classifies the sketches lines, creates a planar embedding, and inflates to a textured surface.

Related Work

There are many varied approaches to understanding and interpreting sketched input [OSSJ09]. Our work is most related to those that consider sketches with many strokes, such as Rajan & Hammond's image-based method for 2D sketch recognition [RH08] and Pusch et al.'s space-partitioning approach to blending multiple strokes [PSNW07]. We are also inspired by works that use drawing characteristics such as pen speed to assist in processing, such as Dematapitiya et al. [DKKS05] and Sezgin & Davis [SD04].

Our work is closely related to mesh inflation systems such as Teddy [IMT99] and FiberMesh [NISA07], which allow users to easily create 3D models by sketching freeform lines. The latter system constructs a surface by treating the input strokes as constraints on the surface position and solving a non-linear system, with the mesh structure coming from a regular triangulation of the 2D sketch region. In contrast, our system uses a coarse triangulation followed by subdivision to produce a subdivision surface, and constructs the surface geometry using functional z-offset displacement. We use a distance transform to compute the offset distance, similar to the Matisse system [BPCB08].

This work is also inspired by image-based modeling systems, such as William's 3D Paint [Wil90], where a user-edited displacement map is combined with a painted or acquired texture to create a 3d model, and the more recent Repoussé system [JC08], which uses mesh inflation to add depth to photographs using smooth and sharp stroke constraints on a dense triangle mesh.

Gingold et al. [GIZ09] introduced the concept of structured annotations, in which the user is able to augment a sketch with markings that indicate structural relationships such as symmetry and alignment between the legs of an animal. The idea of adding non-geometric elements to a sketch inspired our own annotations described in this article.

References

[BPCB08] Bernhardt A., Pihuit A., Cani M. P., Barthe L.: Matisse: Painting 2d regions for modeling free-form shapes. In *Proc. of Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM '08)* (2008).

[DKKS05] Dematapitiya S., Kawazoe M., Khand Q. U., Saga S.: Object snapping method using multi-resolution fuzzy grid snapping technique. In *Proc. of Eurographics Symposium on Sketch-Based Interfaces and Modeling (SBIM '05)* (2005).

[GIZ09] Gingold Y., Igarashi T., Zorin D.: Structured annotations for 2d-to-3d modeling. In *Proc. of SIGGRAPH Asia 2009* (2009), pp. 1–9.

[JC08] Joshi P., Carr N.: Repoussé: Automatic inflation of 2d artwork. In *Proc. of Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM '08)* (2008).

[OSSJ09] Olsen L., Samavati F., Sousa M., Jorge J.: Sketch-based modeling: A survey. *Computers & Graphics 33* (2009), 85–103.

[PSNW07] Pusch R., Samavati F., Nasri A., Wyvill B.: Improving the sketch-based interface: Forming curves from many small strokes. *The Visual Computer 23*, 9-11 (2007), 955–962.

[SD04] Sezgin T., Davis R.: Scale-space based feature point detection for digital ink. In *Making Pen-Based Interaction Intelligent and Natural* (October 21-24 2004), AAAI Fall Symposium, pp. 145–151.

[Wil90] Williams L.: 3d paint. In *Proc. of 1990 Symposium on Interactive 3D graphics (SI3D '90)* (1990), pp. 225–233.

is used to find the optimal path between them. To maintain a sketching metaphor in our system, the seed points are automatically extracted from the magnetic pen stroke. In particular, the corner regions of the stroke are used as seeds. To prevent snapping to distant edges, the search space is restricted to a fixed distance from the original stroke. Together, the seed points and search region are used as criteria to find the optimal path along the image edges.

Finally, the Annotation tool is used to indicate modeling operations on the sketched object, such as extrusion and hole-cutting. This enables the user to create more varied shapes with arbitrary topologies. Annotations are created with freehand strokes simultaneously with the sketch itself. Typically, the user first sketches the object and features, and then annotates the sketch. From the application point of view, the order of drawing does not matter – annotations strokes can be made before the sketch itself, if the user desires. However, unlike the main sketch, it is assumed that the annotations are drawn with continuous strokes. This is because the annotations are iconic symbols denoting operations, akin to gestures. The full set of supported annotations is shown in Figure 2c, and their implementation is described in the next section.

(a)

(b)

(c)

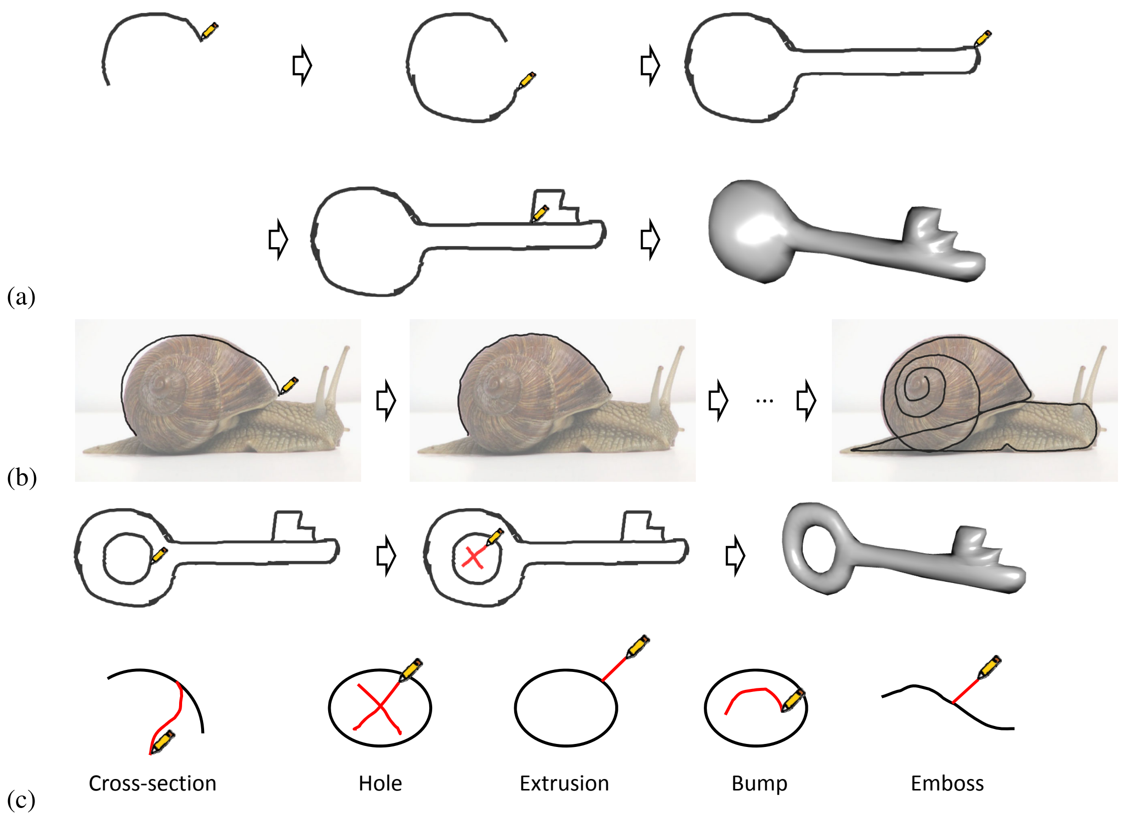


Fig. 2. User interface: (a) the main input mode is the Pen, used for defining object geometry and feature lines; (b) the Magnetic pen automatically aligns the input with image edges; (c) the Annotation pen is used for indicating operations such as hole-cutting.

III. Sketch Analysis: Stroke Extraction & Classification

There are two main components in our system. The Sketch Analysis component (Section III) is responsible for converting the unstructured user sketch into a useful form; this involves first extracting salient strokes from the sketch, and then classifying the extracted strokes according to their role in object definition. The Surface Construction component (Section IV) uses the set of classified strokes to create a smooth surface; a planar embedding of the strokes is created in 2D, followed by inflation to 3D using functional displacement.

To support complex sketches for mesh creation, we must consider how the input strokes combine to form perceptual strokes, and how these perceptual strokes combine to define an object. These questions lead us to a 2-stage sketch analysis approach (see Figure 3). First, the input sketch is rasterized and traced to extract contiguous stroke segments. Second, these segments are classified according to their relative positions and containment within the regions defined by them.

Our proposed method for stroke extraction is similar to previous work based on thinning and contour tracing [RH08]. Our contributions are identifying branch points, restoring sharp features, and using on-line sketching information to improve accuracy. The approach is focused on line drawings without shading cues, a simplification that makes the problem of stroke classification tractable while still allowing for sketches with many lines and complex shapes.

To perform contour tracing, the strokes should be drawn to a raster image. After rasterization, a binary image with white (paper) background pixels and black (ink) foreground pixels is generated by applying a Gaussian blur to close small gaps between strokes, and then thresholding. Finally, morphological thinning is applied to produce lines that are only 1-pixel wide.

To trace the contours of a binary image I, we use a label image L of the same size with all pixels initially unlabeled ($L_{i,j} = 0$). The goal is to assign a label to all foreground pixels of I such that connected pixels (lines) have the same label. Starting from an unlabeled foreground pixel, tracing proceeds by advancing to the first unlabeled clockwise foreground neighbor. Tracing terminates when the start pixel is reached, or a pixel with multiple unlabeled neighbors is encountered. The latter case corresponds to a point where several strokes meet, which we call a *branch point*.

In our implementation, the active contour is terminated when a branch point pixel is reached. This ensures that after extraction, no stroke crosses another. Tracing then resumes at the next unmarked foreground pixel until the entire image has been processed. After tracing, each traced line is converted back to a stroke by a simple 2D mapping.

The stroke thinning step can cause artifacts around sharp features, where the width variance in rasterized lines results in an unwanted branch point with a short line attached. In most cases (when w is small) the effect is not prominent; however, for very sharp features, or very messy sketches, the erosion results in an unwanted branch point.

Sharp features are restored by identifying branch points with three connected strokes, where the angle between the two

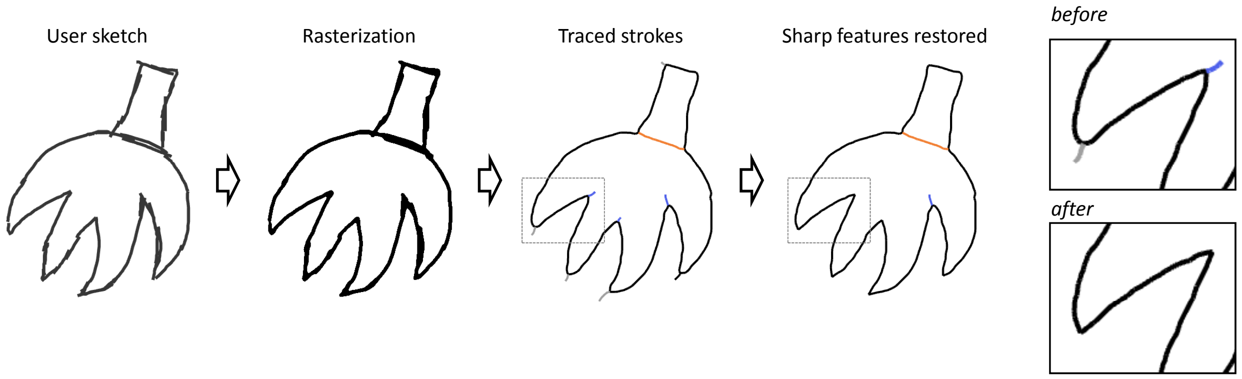


Fig. 3. Sketch analysis overview: the input sketch is rasterized and traced to extract stroke segments. Sharp features are restored in a post-processing step.

longer strokes is acute. When these conditions are satisfied, the short is discarded and the longer strokes are merged to create a continuous stroke with no branch point (see the insets of Figure 3).

After extracting strokes from the sketch, we should consider how they combine to define an object. Where is the object boundary? What do non-boundary lines tell us about the object? For example, consider a simple sketch of a face: the two eyes are separate objects if seen alone, but in the context of other lines – contained within the head, near the mouth – we can perceive them as features of a larger object. Therefore, a stroke classification depends on where the strokes are located as well as the regions they define. To support objects with features and general topology, the relative containment of strokes is also important. Domain-specific knowledge – such as the arrangement of human facial features used in Sharon & van de Panne's constellation models [SvdP06] – can aid in the classification task, but our system is targeted at general modeling tasks.

We employ an image-based approach to find the containment hierarchy, based on a connected components (CC) labeling. The CC algorithm is suitable for the classification task because of its awareness of not only the strokes, but the space (or regions) between them. To identify distinct regions in a sketch, the CC labeling algorithm is used on the rasterized stroke image where each pixel is either foreground (stroke) or background (non-stroke). (For classification, we are interested in the regions defined *between* the strokes, but not the stroke regions themselves since the strokes are already known. In the following discussion, *region* refers to a non-stroke region.) The labeling of a simple sketch is shown in Figure 4, where the strokes define four non-stroke regions: A, B, C, and the background region BG.

To identify objects in a sketch – and then classify the extracted strokes according to how they define those objects – we can examine the relationship between strokes and regions. Because strokes are terminated at branch points, each stroke must be adjacent to two regions – it could not be adjacent to more regions, as any point where three or more regions meet would be a branch point. Thus it is sufficient to check the regions adjacent to a stroke at any point along it, such as the midpoint.

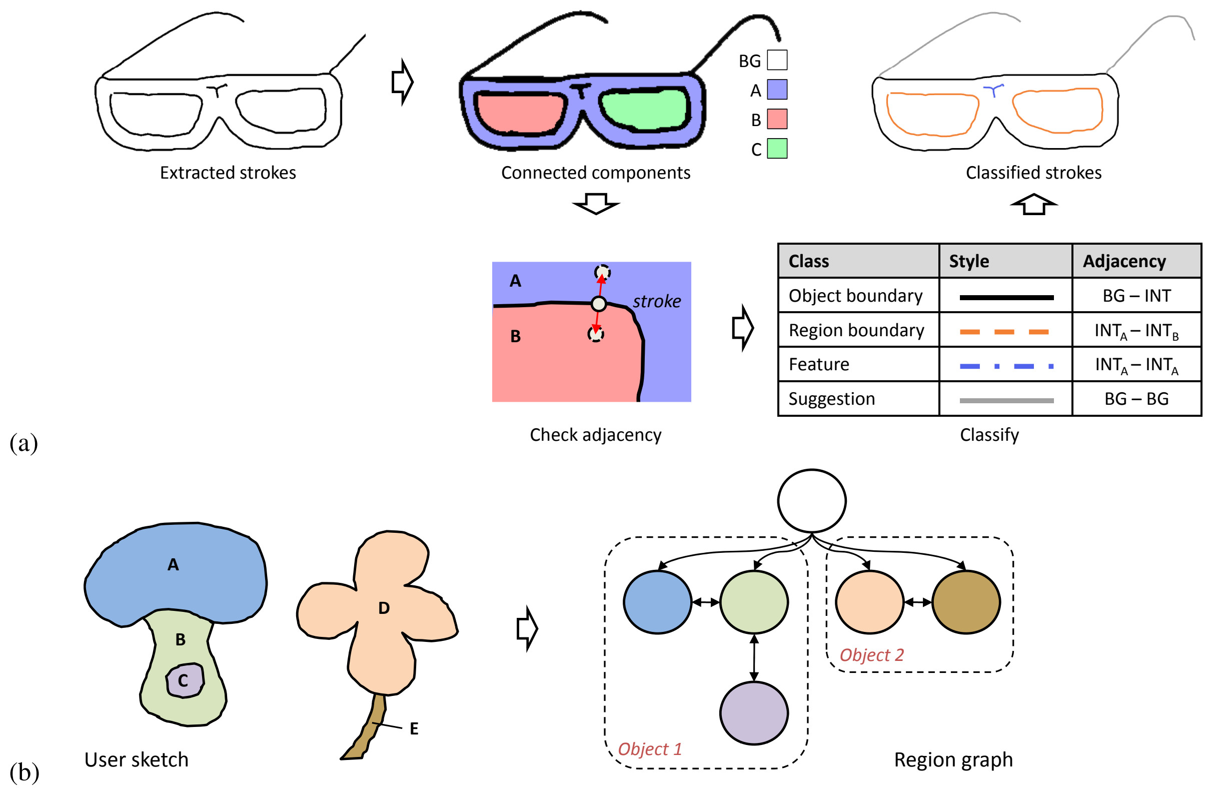
In a labeled image, there are two types of region: background (BG), and interior (INT; labeled A, B, $\ldots$ in the figures). Since each stroke is adjacent to two regions, the possible adjacency arrangements are BG-BG, BG-INT, and INT-INT. In the latter case, the interior region can be the same or different, leaving a total of four possible arrangements, or classes. Based on a subjective analysis of where these strokes appear in a sketch, we have named these classes as *object boundary* (BG-INT), *region boundary* (INT$_A$-INT$_B$), *feature* (INT$_A$-INT$_A$), and *suggestion* (BG-BG).

This classification is suitable for mesh creation, as the object boundary indicates the region to be filled with the surface, while region boundaries and features can be used to guide the meshing process. Region boundaries, when closed and not adjacent to any object boundary, define potential holes in the object. Suggestion strokes may or may not be useful to a particular application; for example, they could be mapped to generalized cylinders in 3D, or aligned pairs could be used to construct surfaces of revolution.

The stroke-region adjacency information can be thought of as a graph whose nodes correspond to regions, and edges correspond to strokes separating the regions. Each stroke class corresponds to a particular kind of edge in the graph. Suggestions and feature lines are self-edges of a BG or INT region, respectively, while object and region boundaries are edges between different regions.

The region graph offers insight into the object structure, such as the presence of multiple objects, containment of regions, and the location of holes. Consider Figure 4b. Region C is only adjacent to B, and so C must be contained within B. In general, it can be observed that if region X is adjacent only to non-BG region Y, then X is contained by Y. Further observation shows that each unique object corresponds to a connected component (in the graph-theoretic sense) of the region graph after removal of the BG node. In the example graph, there are two objects: the first is defined by regions A, B, and C, while the second is defined by regions D and E.

Extending the construction to allow for holes makes it useful for a wider variety of modeling tasks. A hole is of course an empty space surrounded by non-empty space. Thus, in a sketch, a hole region must be surrounded by, or contained in, another region. In terms of the region graph, this means that



Class	Style	Adjacency
Object boundary	━━━	BG – INT
Region boundary	╌╌╌	INT_A – INT_B
Feature	─ · ─ ·	INT_A – INT_A
Suggestion	━━━	BG – BG

Fig. 4. (a) Classification is based on which regions are adjacent to each stroke; the four possible classes are object and region boundaries, features, and suggestions; (b) The adjacency information defines a region graph, which can be used to identify unique objects and potential holes in an object.

a hole region must be two or more levels deep in the graph. Because of inherent ambiguity in interpretation (is it a hole or simply a region?), the user must identify holes by drawing a 'cross' annotation over the hole region.

A. Adapting to Drawing Styles

Artists often sketch iteratively, at first making light, hasty strokes to define rough boundaries and guidelines, then tracing these rough lines with harder, more deliberate strokes. The viewer naturally perceives the harder (and thus darker and thicker) strokes as being more important, and uses these lines to understand the sketch. Using this observation of artistic drawing tendencies, we consider an approach for using stroke speed and pressure to adapt to a user's drawing style. Strokes with very low pressure or high speed can be de-emphasized or ignored, while strokes with high pressure or low speed can be trusted as truly capturing the artist's intent.

The extraction's success depends largely on the strokes' rasterization width w. If drawn too thin, then perceptual connections may not be made, but drawing too thick can conversely result in unwanted connections or region closing. In general, a thin stroke rasterization is preferable to a thick rasterization, because fewer artifacts are introduced by thinning and the original strokes are captured more accurately. However, a thin rasterization will not always lead to a perceptually-correct extraction, because perceived connections between overlapping strokes will not be created. Consider the two apples shown in Figure 5: if these sketches were traced directly, the "messy" sketch would result in many strokes along the apple boundary even though they combine in the eye to form a single contour. The question is, how can a system adapt to different drawing styles?

Figure 5 shows two apples sketched in different styles. The left apple is sketched in a deliberate style, and the right with a hasty style. Deliberate strokes are characterized by high pressure and low pen velocity, while a hasty stroke has high velocity and low pressure. In the former case, a smaller rasterization width w can be used so that the original intent is preserved. In the latter case, a larger w should be used to create the perceptual connections and overlaps in the strokes.

For each apple in Figure 5, two types of rasterization have been used: thin lines and thick lines. We can see that for the deliberate sketch, the thin rasterization results in the original details being preserved and ultimately in a successful extraction and classification. Conversely, a thick rasterization of the deliberate sketch results in details being obscured and an incorrect classification. For the hasty sketch, the opposite is true: thin rasterization results in a poor classification because the small gaps between strokes are not closed, while a thick rasterization results in a correct classification.

Based on such observations, our system uses an automatic tuning phase in which each stroke's average pressure and speed controls the rasterization width w. To do this, the stroke pressure and velocity are quantized to three levels: low, medium, and high. This defines nine possible pressure-velocity combinations, which are mapped to w. This width adjustment does not change what the user sees, but rather happens internally for the purposes of stroke extraction. Using this

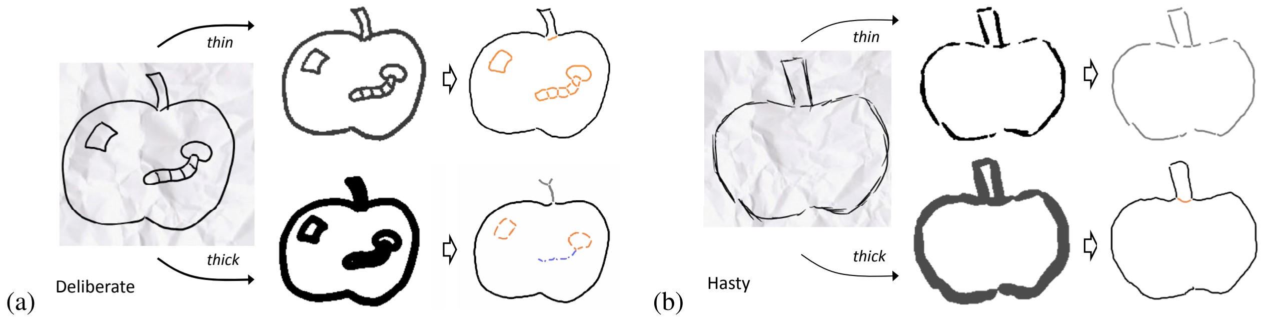


Fig. 5. Adapting to drawing styles: (a) for deliberate sketches, thin rasterization (top) is best; (b) for hasty sketches, thicker lines (bottom) that overlap and blend together lead to better results. Reversing the settings can cause unwanted blending (eg. the stem in (a-bottom)) or misclassification (b-top).

automatic tuning, the system is able to respond to the user's drawing style and achieve better classification performance.

IV. Surface Construction

The guiding principle in surface construction is that the sketched lines are important, and every sketched line should be represented in the created mesh. In particular, the goal is feature-sensitive meshing, where edges of the mesh are enforced to follow the sketched lines. A secondary goal is to create a mesh that has subdivision connectivity, since subdivision results in a nice mesh structure with well-shaped faces and mostly regular-valence vertices. Meshes with subdivision connectivity can also be used in multiresolution applications for multi-scale editing or level-of-detail rendering.

In our meshing approach, we first create a planar embedding of the sketched object, based on the classified strokes. The planar mesh is then inflated to a 3D surface using a functional vertex displacement. This approach bridges the gap between mesh quality and topological variety that exists in previous work. On one side of the gap are approaches such as rotational blending surfaces [CSSJ05], which are limited to non-branching topologies but produce nicely-parameterized NURBS surfaces. On the other side, works such as FiberMesh [NISA07] support arbitrary topologies but suffer in terms of mesh quality, especially after adding surficial details.

A. Planar Embedding

Each sketched object is composed of boundary strokes, region boundaries, features and suggestive strokes. A planar embedding should have triangles with edges that follow the strokes, but also respect the boundaries and holes so that only the object interior is covered with triangles. To create a planar embedding of these strokes, we first approximate each stroke with a sparsely-sampled polyline and triangulate those points, then perform a snapped subdivision (Figure 6a).

The polyline approximation is constructed in two steps. First, sharp corners are found by looking the local change of direction ϕ at each stroke sample (Figure 6b). A directional change exceeding a threshold angle ϕ_t indicates that the sample is at or near a sharp corner. To reduce the effect of small-scale jitter that can occur in the input points, the incoming and outgoing directions at a point p_i are considered over a small window of k points.

These corner points are typically not enough to approximate a sketch – for example, a circle would have no corner points. Therefore, we also use an iterative splitting scheme in which the strokes are split between corner points at the point of maximal deviation from a straight-line approximation. Each sub-segment can then be split in the same fashion, until either the maximal point is within a threshold distance of the straight line, or the segment is too short to split. The benefit of this approach over something like least-squares is that it uses just enough points to represent the data, rather than selecting the best k points.

After each stroke has been approximated, a coarse mesh is created from a Constrained Delaunay Triangulation (CDT) of the polyline vertices, with the stroke segments acting as constraints. That is, if two vertices are connected by a stroke segment, then the triangulation must contain an edge between those vertices. This is consistent as long as no two stroke segments intersect each other, a condition that is ensured by terminating strokes at branch points during the tracing stage.

Since the polylines contain a small number of points, this coarse mesh only approximates the input strokes. To provide a better match, we use a snapped-subdivision approach in which newly-created vertices are moved onto adjacent stroke segments as the mesh is subdivided (Figure 6c).

To do this, stroke segments are associated with edges based on the polyline approximations. As the mesh is subdivided, each edge is split into two edges by inserting a new vertex v_e. In the case where the edge has an associated stroke segment, snapping occurs by by moving v_e to the midpoint of the stroke. So that the snapping process can be subsequently applied, the stroke segment should also be split at the midpoint, and each half-segment associated with the newly-created edges. After a few iterations of subdividing and snapping, the edges in the planar mesh follow the original sketch very closely.

B. Inflation

The planar mesh constructed thus far could be used in any previous inflation method, such as the non-linear optimization approach of FiberMesh [NISA07]. In NaturaSketch, however, we explored a constructive inflation method that allows for customization and control of the surface's cross-section.

To inflate a smooth 3D surface (Figure 7), each vertex in the planar mesh should be offset in the depth by an amount proportional to its distance to the boundary. We use a discrete

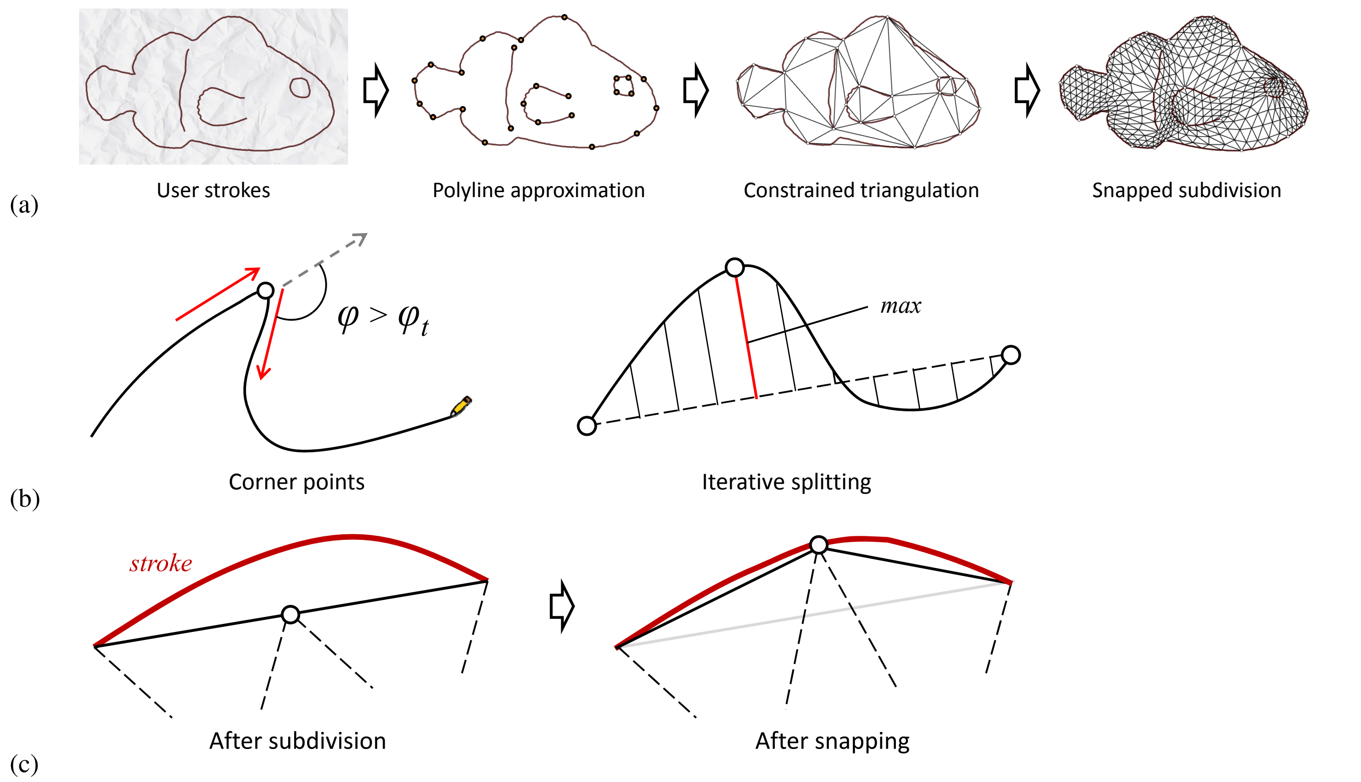


Fig. 6. (a) A planar embedding of the sketch is created by approximating each stroke with a small number of points, constructing a constrained triangulation of those points, and then using snapped subdivision; (b) corner conditions and iterative splitting are used to create a polyline from a stroke; (c) when subdividing the coarse mesh, edge vertices are snapped onto the associated stroke.

image-based distance transform [FCTB08] to approximate this distance. Given a binary image I, the distance transform $T(I)$ is an image in which the value of a pixel encodes the distance to the nearest black pixel. If $I_{i,j}$ is black, $T(I_{i,j}))=0$, pixels adjacent to black pixels have the value 1, and so forth. Since the goal is to compute the distance from interior vertices to the boundary, I should contain only the boundary strokes, but not region boundaries or features.

This creates a displacement map whose usage is similar to Williams' 3D Paint [Wil90], but derived automatically from the input strokes rather than acquired from a range scanner or extra user input. Then, for any vertex in the planar mesh, the distance d to the boundary can be found with a lookup in $T(I)$. That yields a linear measure; to get a smooth result, d is passed through a circular mapping function $X(d)$ (Figure 7a).

A benefit of this approach is that different shapes can be achieved by using different mapping functions. Figure 7b shows a few examples. The cross-section annotation also allows the user to draw their own function directly. However, because the distance field is not smooth at the boundary, this approach can produce sharpness along the object's cross-section.

C. Applying Annotations

From an interface perspective, annotations must be distinguishable from the sketch. In a paper sketch, such mark-ups might be made in a different color of ink, or they may be clearly distinguishable based on context. Since our system is context-free (that is, no particular modeling context or domain is assumed), it would be infeasible to unambiguously distinguish between a sketch and annotations; for example, a cross could be interpreted as a feature on an object or as a hole-cutting annotation. Therefore, the Annotation pen is a separate modality that the user must explicitly select before marking up their sketch.

The supported annotations are shown in Figure 8. The cross-section annotation applies to an object boundary, and allows the user specify the distance transform mapping by sketching. Several region-based operations can be annotated, including hole-cutting, extrusion, and bumps. Finally, annotations can be used to emboss feature lines in an object.

The shapes of the annotations were chosen because their interpretation is clear both to the user and to the system. The context of each annotation – starting on a stroke or region, or contained within a region – is found from the region hierarchy. In our implementation, a hash-table is used to match a region or stroke color in the connected components image to the corresponding region or stroke.

By design, there is little ambiguity between the annotations once context is considered. For instance, the cross-section annotation is the only one that relates to an object boundary stroke. Thus, any annotation beginning on an object boundary can reasonably be classified as a cross-section. Similarly, any annotation beginning on a region boundary is either an extrusion or a bump, so it only remains to classify the stroke as a line or an arc. If a cross is found, then it can only be a hole. Finally, the emboss annotation is the only one to begin on a feature stroke.

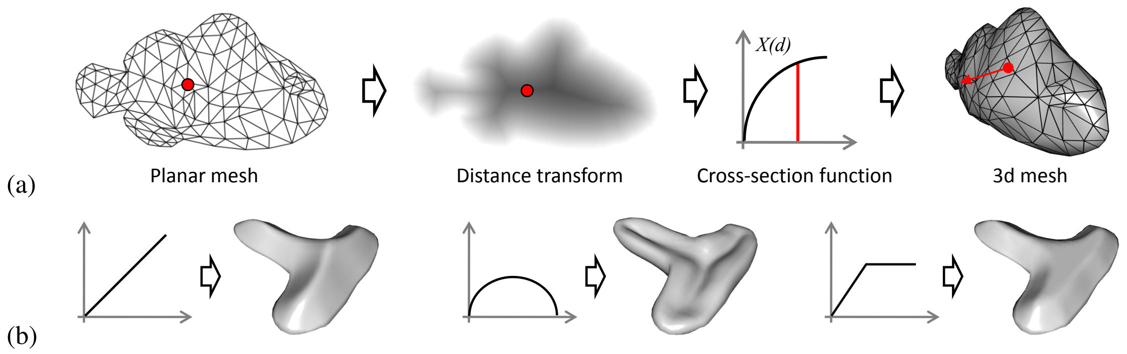


Fig. 7. (a) Inflation uses a distance transform $T(I)$ to displace vertices in M_p; (b) the 3D shape can be altered by using different cross-section functions.

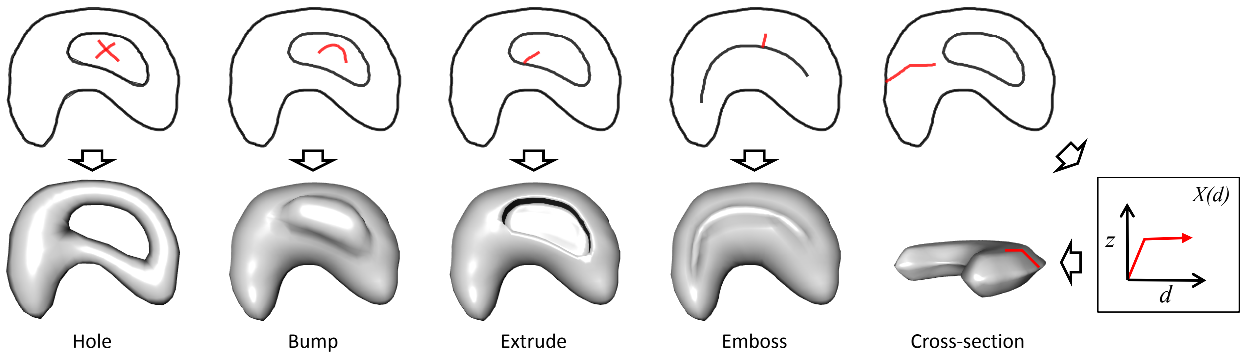


Fig. 8. The set of annotations supported in NaturaSketch (annotation strokes are shown in red).

V. Results

NaturaSketch runs interactively on modest hardware, including a 1.6 GHz tablet PC. Tracing and classifying strokes typically takes about 0.5 seconds for a 512x512 raster size, with stroke thinning taking the bulk of that time. Creating the planar mesh and inflating the surface each require less than 100ms. Interactivity is important in a sketch-based system, so that the user can easily make changes to their sketch and see the result immediately.

We have used this software tool to create a variety of models from photographs. The combination of an input image and user sketch provides a model-image correspondence that can be used to automatically texture-map the objects. As seen in Figure 9, the resulting models are visually appealing and suitable for applications such as interactive games or animations. For multi-part objects such as the lobster or dragon, the parts were assembled manually.

We conducted an informal study to observe how new users interacted with the system. Out of the eight participants (7 male, 1 female), five were graphics researchers, though only two considered themselves to be experienced users of modeling software tools (either sketch-based or traditional). Of the remaining three participants, there was one experienced 2D pencil artist, while the other two were not experienced with either drawing or modeling.

Participants in the study were given a brief ten-minute overview of the system and its capabilities and then asked to create an object. Some user-created objects are shown in Figure 9e-f, each created in 6-8 minutes.

We observed that subjects would generally draw much better when tracing an image versus freehand sketching. Those with an artistic background could draw well without assistance, but also appreciated the ability to have an image guide. The subjects also found the annotations intuitive and easy to remember. The cross-section annotation was one exception, as some users found it unintuitive to draw half of the function. One possible remedy would be to accept the entire function and only retain half, but this could also lead to unexpected results for asymmetric functions. The other annotations (holes, extrusion, bumps) were grasped very quickly by users, since the shape and style of the annotation is tied directly to the result. The only pitfall was the requirement that some annotations begin on a stroke: due to inaccuracies in drawing, annotations were occasionally not recognized correctly because they did not begin exactly on a stroke.

Some areas for improvement were identified. The magnetic pen tool, for instance, did not always give the expected result for images with weak edges. Some participants also felt that support for multiple cross-sections on a single object would be desirable – for example, a sharp-edged sword blade and a rounded hilt.

A. Application: 3D Conversion

Many of the 3D films released today are "3D conversions" of regular 2D source material. The 3D effect is added in post-production via a process of per-frame segmentation and in-painting of occluded regions. This is painstaking work, requiring the efforts of hundreds of artists over several months. Such conversions are generally considered inferior to true 3D

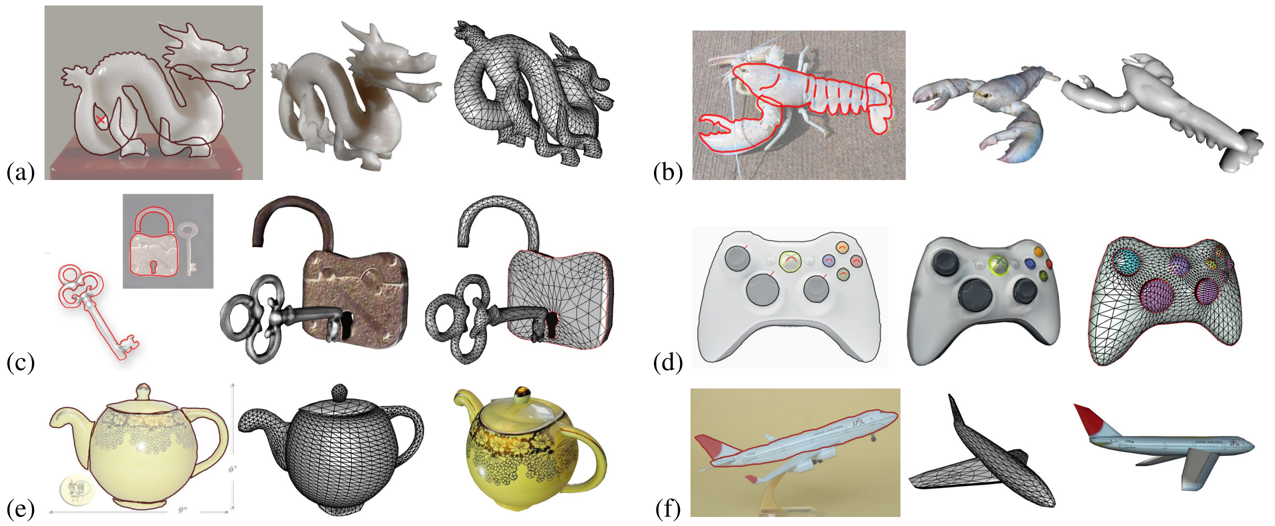


Fig. 9. Some objects created with the NaturaSketch system. The (e) teapot and (f) plane were created by participants in our user study. Each object was created in less than 10 minutes.

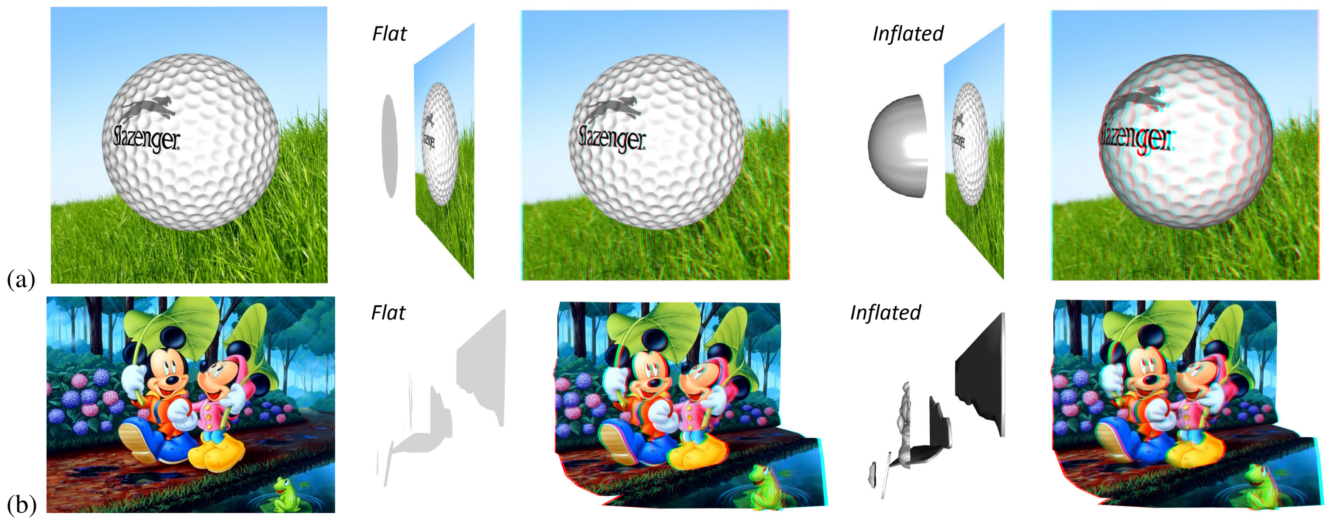


Fig. 10. 3D conversion: (a) a golf ball from a 3D model; (b) of a frame of Disney animation. *(Viewable with red-cyan glasses.)*

source material because the depth is only inter-object, not intra-object.

Based on these observations, we considered how NaturaSketch could be used to create more immersive 3D conversions. Using our system, rather than just segmenting each frame into different depth planes, we can segment and inflate different parts of an image to create in-plane depth variance. In this way, there is intra-object depth because each eye sees a different view of each object. Sýkora et al. [SSJ*10] explored a similar concept in their work on cartoon pop-ups using a small number of user-specified relative depths.

Figure 10 shows a couple of examples. Figure 10a shows a 2D rendering of a 3D golf ball, followed by both flat and inflated 3D conversions. The original 3D scene provides a ground-truth expectation, and because of the shading and depth the inflated version more effectively creates a perception of 3D.

In Figure 10b, because the scene is filled with complex shapes and textures the differences are more subtle. The in-plane depth created by inflation, along with the shading cues, do seem to increase the perception of 3D. However, the benefits would be easier to judge by converting an entire scene.

VI. Conclusion

The NaturaSketch represents several contributions to SBIM. The stroke extraction, classification, and parameter tuning methods allow our system to support and handle natural sketched input with many overlapping strokes and regions. This works in conjunction with a robust mesh construction algorithm that embeds all of the input strokes into the mesh geometry, and a set of sketch-based annotations for editing and deforming a sketched object. Each of these components are tied together by an image-assisted interface in which images can be used for drawing assistance and automated texturing.

The main limitation of our system is that the reconstructed shapes are plane-symmetric. Complex objects such as the dragon in Figure 9 can be created via annotations and by creating multi-part objects, but a direction for future work is to extend the expressive range of the system. This could be done in several ways, such as supporting shapes drawn from off-axis views (such as a car drawn from a three-quarter view) or supporting sketches from multiple viewpoints and combining them during reconstruction.

References

[CSSJ05] Cherlin J. J., Samavati F., Sousa M. C., Jorge J. A.: Sketch-based modeling with few strokes. In *Proc. of Spring Conference on Computer Graphics (SCCG '05)* (2005), pp. 137–145.

[FCTB08] Fabbri R., Costa L., Torelli J., Bruno O. M.: 2d Euclidean distance transform algorithms: A comparative survey. *ACM Computing Surveys 40*, 1 (2008), 1–44.

[Hof00] Hoffman D. D.: *Visual Intelligence: How We Create What We See*. W. W. Norton & Company, 2000.

[IMT99] Igarashi T., Matsuoka S., Tanaka H.: Teddy: A sketching interface for 3d freeform design. In *Proc. of SIGGRAPH'99* (1999).

[MB95] Mortensen E., Barrett W.: Intelligent scissors for image composition. In *Proc. of SIGGRAPH '95* (1995), pp. 191–198.

[NISA07] Nealen A., Igarashi T., Sorkine O., Alexa M.: Fibermesh: Designing freeform surfaces with 3d curves. In *ACM Transactions on Graphics (Proc. of SIGGRAPH '07)* (2007), ACM Press.

[OS10a] Olsen L., Samavati F. F.: Image-assisted modeling from sketches. In *Proc. of Graphics Interface (GI '10)* (2010).

[OS10b] Olsen L., Samavati F. F.: Stroke extraction and classification for mesh inflation. In *Proc. of Eurographics Symposium on Sketch-Based Interfaces and Modeling (SBIM '10)* (2010).

[RH08] Rajan P., Hammond T.: From paper to machine: Extracting strokes from images for use in sketch recognition. In *Proc. of Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM '08)* (2008).

[SSJ*10] Sýkora D., Sedlacek D., Jinchao S., Dingliana J., Collins S.: Adding depth to cartoons using sparse depth (in)equalities. In *Proc. of Eurographics 2010* (2010).

[SvdP06] Sharon D., van de Panne M.: Constellation models for sketch recognition. In *Proc. of Eurographics Workshop on Sketch Based Interfaces and Modeling (SBIM '06)* (2006).

Luke Olsen received his PhD in computer graphics from the University of Calgary in 2011. His research interests lie in sketch-based interfaces, subdivision and multiresolution, and the intersection of computer vision and graphics. He is now working on exciting projects at Microsoft.

Faramarz F. Samavati is an associate professor in the Department of Computer Science, University of Calgary. Prof. Samavatis research interest is Computer Graphics, Visualization and 3D imaging. He has authored more than 70 research papers in these areas and edited a recent book in Sketch-based Modeling. He is also a network investigator of GRAND NCE (Networks of Centres of Excellence of Canada in Graphics, Animation and New Media).

Joaquim A. Jorge coordinates the VIMMI research group at INESC-ID and is a Professor at Instituto Superior Tcnico, Technical University of Lisbon. He is Editor-in-Chief of the Computers and Graphics Journal, a Fellow of the Eurographics Association and a Senior Member of ACM and IEEE. He helped organize over 30 scientific events and served on over 130 program committees, (co)authored over 190 publications in international refereed journals, conferences and edited a recent book on Sketch-Based Modeling.