A Taxonomy of Modeling Techniques using Sketch-Based Interfaces

L. Olsen[†1] F.F. Samavati[1] M. Costa Sousa[1] and J. Jorge[2]

[1]Dept. of Computer Science, University of Calgary, Canada
[2]Departamento de Engenharia Informática, Instituto Superior Tècnico, Lisbon, Portugal

Abstract

Traditional user interfaces in modeling have followed the WIMP (Window, Icon, Menu, Pointer) paradigm. While functional and powerful, they can also be cumbersome and daunting to a novice user; creating a complex model requires much expertise and effort. A recent trend is toward more accessible and natural interfaces, which has lead to sketch-based interfaces for modeling (SBIM). The goal is to allow hand-drawn sketches to be used in the modeling process, from rough model creation through to fine detail construction. Mapping 2D sketches to a 3D modeling operation is a difficult and ambiguous task, so our categorization is based on how an SBIM application chooses to interpret a sketch, of which there are three primary methods: to create a 3D model, to add details to an existing model, or to deform and manipulate a model. In this STAR, we present a taxonomy of sketch-based interfaces focused on geometric modeling applications. The canonical and recent works are presented and classified, including techniques for sketch acquisition, filtering, and interpretation. The report also includes a discussion of important challenges and open problems for researchers to tackle in the coming years.

1. Introduction

Creating 3D models is often a hard and laborious task due to the complexity and diversity of shapes involved, the intricate relationships between them, and the variety of surface representations. Current high-end modeling systems such as Maya [Auta], AutoCAD [Autb], and CATIA [Das] incorporate powerful tools for accurate and detailed geometric model construction and manipulation. These systems typically employ the WIMP (Window, Icon, Menu, Pointer) interface paradigm, which are based on selecting operations from menus and floating palettes, entering parameters in dialog boxes, and moving control points.

Research in modeling interfaces has recently explored alternate paradigms such as pencil-and-paper sketching to enable fast, approximate reconstruction and manipulation of 3D models, drawing inspiration from the cartoon and animation industries. This paradigm is known as Sketch-based Interfaces for Modeling (SBIM), and the goal is to use freehand drawings and sketches as a way to create and edit 3D geometric models in expedited ways, mimicking the dynamism of traditional media. Already there are commercial modeling packages that explore sketch- and brush-based metaphors, allowing a modeler to draw objects (SketchUp [Goo]) or paint details onto a surface (ZBrush [Pix], MudBox [Autc]).

The trend and ultimate goal of SBIM research is to converge modeling systems, integrating the expressive power and control of WIMP-based systems with the expeditious and natural interaction of sketch-based paradigms. This would allow users to construct and edit models in a progressive way, from an initial concept to a detailed and accurate final model.

To design a computer application capable of understanding sketched input, it helps to first consider how a human perceives the world around them. Though our visual system uses two retinal images and shading cues to interpret what we see in three dimensions, we are nonetheless able to understand shapes from single images, even from simple line drawings devoid of any shading cues. The effortlessness of shape perception makes it difficult to understand and formal-

[†] Corresponding author: olsenl@cpsc.ucalgary.ca

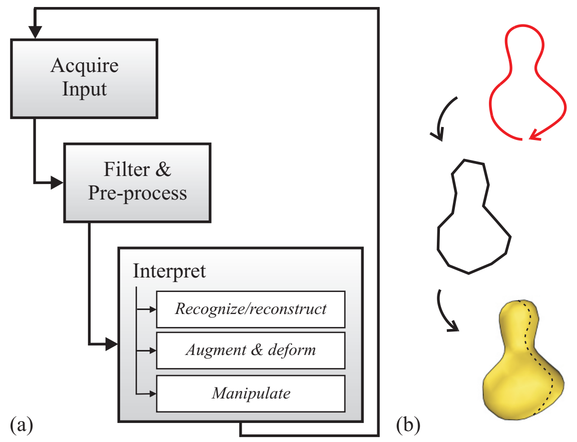

Figure 1: *(a) The SBIM pipeline: after acquiring an input sketch and applying some filters to it, the sketch is interpreted as an operation in 3D. (b) a simple example (3D model created with Teddy [IMT99]).*

ize, but developing an SBIM system that behaves intuitively from the user's perspective requires consideration of perceptual and cognitive issues. In fact, SBIM stands at the intersection of several diverse domains, including computer vision, human-computer interaction (HCI), and cognitive science. Though research efforts have thus far been propelled primarily by computer graphics researchers, the emergence of powerful commodity computer hardware is creating many exciting opportunities for interdisciplinary work to drive the field to exciting results.

Sketch-based systems date back to Sutherland's SketchPad system [Sut63], which used a light-pen input device to directly manipulate on-screen objects, preceding the ubiquitous mouse by several years. SketchPad anticipated many challenges that SBIM would encounter in the future, including how to accept and process user input, interpret that input as an object or operation on an object, and represent the resulting object. Sketch-based techniques have since found utility in a wide range of modeling applications. Some examples include animation [DAC*03, TBvdP04, CON05], clothing design [IH02, DJW*06], data visualization [CSSM06], plant modeling [IOOI05, ASSJ06, OMKK06, ZS07], image deformation [ESA07], architecture and engineering [HU90, VMS05, YXN*05], 3D model databases [FMK*03], mathematical sketching [LZ04], and user interface design [CGFJ02, HD05].

In this report, we present a taxonomy of sketch-based interfaces, focused on geometric modeling applications. Our categorization is based on how an SBIM application interprets a sketch, of which there are three primary methods: to create a 3D model, to add details to an existing model, or to deform and manipulate a model. The pipeline of an SBIM application is summarized in Fig. 1a. The first stage is to acquire a sketch from the user (Sec. 3), followed by a filtering stage in which the sketch is cleaned up and possibly transformed into a more usable form (Sec. 4). In the final stage of the pipeline, the sketch is interpreted as the specification of or operation on a 3D model (Sec. 5). Figure 1b depicts a typical result from a model-creation system.

This STAR is organized as follows. After introducing some notions of perception in Sec. 2, each stage of the SBIM pipeline is described in detail in Sections 3–5. Section 5 also includes a discussion of two critical areas in application design, surface representation (Sec. 5.4) and interface (Sec. 5.5). We conclude with a discussion of challenges and open problems (Sec. 6). The report is targeted at readers with an interest in computer-assisted sketch recognition and reconstruction; some familiarity with computer graphics concepts is helpful but not necessary.

2. Basics of Perception

Distance of it self, is not to be perceived; for 'tis a line (or a length) presented to our eye with its end toward us, which must therefore be only a *point*, and that is *invisible*.
– *William Molyneux, as quoted in [Hof00]*

The human visual system is vastly complex, yet we take it for granted because it works so effortlessly throughout our lives. Only in the last few decades has cognitive science really started to pay attention to how our 'visual intelligence' works. While a thorough discussion is beyond the scope of this paper and our expertise, some notions from this area have already influenced the design of SBIM systems and will no doubt continue to do so in the future. After all, a person's perception of shape informs how they draw that shape: perception and communication are dual sides of our visual intelligence.

The fundamental problem that our visual system must deal with, according to Hoffman [Hof00], is that "the image at the eye has countless possible interpretations." Consider the trivial case of a sketch containing only a single point. Even if the 2D coordinates of the point are known exactly, the sketch could represent any subset of points lying on the line passing through it and the viewer's eye. Figure 2 illustrates the problem with a non-trivial line drawing, depicting three of the infinitely many objects that project to a cube-like image.

Though we can convince the logical part of our brain that the drawing could represent something other than a cube, the same cannot be said for the visual part. Try as we might, it will always be seen as a cube. This interpretation emerges as the result of relatively simple rules that govern our visual system. To understand the rules, we first have to introduce some terminology.

A *silhouette* is a filled outline segmenting an object from

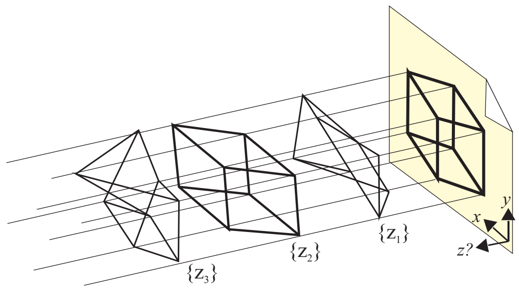

Figure 2: *Ambiguous interpretation of a 2D sketch in 3D: there are infinitely many objects that project to the same 2D input. Reproduced with permission from [ML05].*

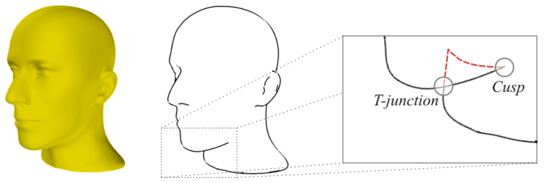

Figure 3: *The contour of an object conveys a lot of shape information. Cutout: T-junctions and cusps imply hidden contour lines (red).*

its background, and is very meaningful for shape recognition, providing a "useful first index into [a person's] memory of shapes" [HS97]. In computer graphics, a related notion is the *contour*, defined as the projection of all points on an object whose surface normal is perpendicular to the view direction, dividing visible parts of the object from the invisible. As illustrated in Fig. 3, the contour includes not just the silhouette outline, but also reveals interior features (like the chin and nose in the example). As our viewpoint of the object changes, the contour also will change to reveal different interior features. DeCarlo et al. [DFRS03] note that near-contours – regions of an object where a contour would appear after a small viewpoint change – are also useful for shape perception.

The contour of an object separates those parts of the object facing toward the viewer from those facing away. In non-trivial objects, there may be parts of the surface that are facing the viewer, yet are not visible to the viewer because it is occluded by a part of the surface nearer to the viewer. Figure 3 shows an example of this: the contour of the neck is occluded by the chin. Note that where the neck contour passes behind the chin, we see a *T* shape in the projected contour (called a *T-junction*), and the chin contour ends abruptly (called a *cusp*). T-junctions and cusps indicate the presence of a hidden contour, providing useful shape information. Williams [Wil94] has proposed a method for using this observation to infer hidden contour lines in an image.

So how do we interpret Fig. 2 as a cube, rather than the infinitely many other choices? We again turn to Hoffman [Hof00], who lists ten visual rules that are used to perceive 3D shape from 2D images, based on the shape cues provided by contours, T-junctions, and cusps.

1. Always interpret a straight line in an image as a straight line in 3D.
2. If the tips of two lines coincide in an image, then always interpret them as coinciding in 3D.
3. Always interpret lines collinear in an image as collinear in 3D.
4. Interpret elements nearby in an image as nearby in 3D.
5. Always interpret a curve that is smooth in an image as smooth in 3D.
6. Where possible, interpret a curve in an image as the contour of a surface in 3D.
7. Where possible, interpret a T-junction in an image as a point where the full contour conceals itself.
8. Interpret each convex point on a bound as a convex point on a contour.
9. Interpret each concave point on a bound as a saddle point on a contour.
10. Construct surfaces in 3D that are as smooth as possible.

Since Fig. 2 contains only straight lines, the first three or four rules apply, leading us to see the drawing as a cube. In fact we can see two cubes, but we cannot force our eyes to see it as anything else. In general, when trying to reconstruct an image, "your visual system is biased. It constructs only those 3D worlds that conform to its rules. . . . They prune the possible depths you can see from infinity down to one or two."

Visual rules allow us to make sense of images we've never seen before, but they are also limited in that they force us to see the simplest object. We also have a vast memory of shapes that is used to interpret images [HS97], imbuing them with unseen complexity. When given an image or even just a silhouette of a sports car, for example, we can quickly determine that the object belongs to the automobile class and infer its approximate geometry, symmetry, and scale. This highlights an important distinction between recognition or reconstruction [CPC04]. *Reconstruction* is the task of creating a complete description of the 3D geometry of an object based on a 2D representation. A similar but distinct task is *recognition*, or identifying which class of object an image represents based on shape memory. In other words, if visual memory can recognize a shape, we can more easily reconstruct it. Otherwise, reconstruction falls back on the visual rule system.

The notions of perception introduced here can help us to understand the challenges and design decisions made in SBIM. As we will see in Sec. 5, the ways in which SBIM

Figure 4: *Input to a sketch-based system is acquired from pen-based or free-form devices such as a tablet display (reproduced from [YXN*05]).*

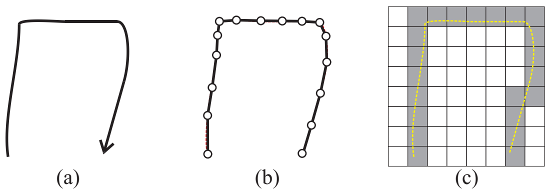

Figure 5: *An input stroke (a) is provided to the application as (b) a sequence of point samples; (c) some applications choose to use an image-based representation.*

systems deal with the ambiguity of single images relate to visual memory and rule systems. Understanding our own perception also suggests ways to improve the software-based perception required for SBIM.

3. Sketch Acquisition

The most basic operation shared between all SBIM systems is, of course, obtaining a sketch from the user. Input devices that closely mimic the feel of freehand drawing on paper, such as tablet displays, are better able to exploit a user's ability to draw. Devices in which the display and input device are coupled (Fig. 4) are particularly suited to natural interaction. However, decoupled tablets and even a traditional mouse can meet the basic input requirements of a sketch-based interface.

Pencil and paper is a very rich medium for communication. An artist can convey information not just with the overall form of the drawing, but also by varying drawing pressure and stroke style. From the artist's perspective, the medium itself provides feedback via the texture of the paper, as they feel their pencil scraping across the surface – drawing on a napkin, for instance, has a different tactile response than regular paper.

Some efforts have been made to transfer these elements to the digital domain. Many tablet devices are now pressure sensitive, providing not just positional information about the pen tip, but also a measure of how hard the user is pressing the pen into the tablet. Haptic devices [HACH*04] are a more recent development that provide active feedback to the user through the pen device itself, such as low-frequency vibration to simulate friction between the (virtual) pencil and paper. Other possible input devices include tabletop displays [SWSJ05] and even 3D virtual reality devices [FBSS04].

Such devices are intended to increase the user's feeling of immersion, although they are often cumbersome and may actually decrease immersion. For instance, a haptic pen is attached to an arm that provides the feedback force, decreasing the device's pen-like attributes. As such hardware becomes more compact and less costly, its adoption should increase.

It should be noted that the ultimate verisimilitudinous interface would be real pencil-and-paper combined with some sort of active digitization. There are commercial products that offer automatic digitization of text and figures [Lea], but to date there has been little investigation in this direction for 3D reconstruction tasks. Off-line scanning of sketches is another option, but in this report we limit our focus to interactive systems.

3.1. Sketch Representation

At the bare minimum, a pen-based input device will provide positional information in some 2D coordinate system, usually window coordinates. The interrupt-driven nature of modern operating systems means that positional updates are usually generated only when the device is moved. The sampling rate varies from one device to the next, but in any case the sampled positions represent a piecewise-linear approximation of continuous movements (Fig. 5b). Note that the samples are spaced irregularly, dependent on the drawing speed. Samples tend to be spaced more closely near corners as the user draws more carefully, a fact which can be exploited to identify 'important' parts [SSD01].

We will refer to a time-ordered sequence of points as a *stroke* $S = \{p_1, p_2, \ldots, p_n\}$, where $p_i = [x_i\ y_i\ t_i]$ contains a 2D coordinate and a time stamp, and the beginning and end of a stroke are demarcated by pen-down and pen-up actions. A sketch is comprised of one or more strokes.

The basic stroke information can be augmented by additional information, such as pressure or pen angle, depending on the target application and available hardware. Not all SBIM systems choose to use all available information. For example, one system may require a pressure-sensitive device, while another may ignore everything but the positional information.

Due to the large body of work on image processing techniques, some SBIM applications prefer an image-based

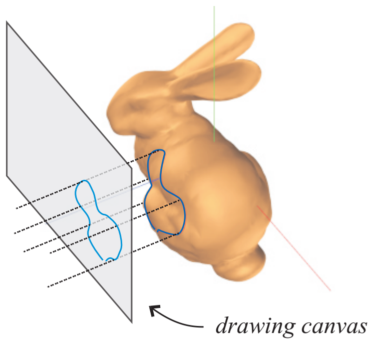

Figure 6: *Sketches are embedded into 3D by projecting onto a drawing canvas, or perhaps onto existing geometry.*

stroke representation, in which the stroke is approximated with a pixel grid (Fig. 5c). A sketch image can be obtained either by copying the frame buffer directly from the application, or by rendering a sketch to an off-screen buffer.

3.2. The Drawing Canvas

As Das et al. [DDGG05] note, the notion of a "drawing canvas" is often useful in SBIM systems to transform a sketch from meaningless window coordinates to 3D world coordinates. The simplest way to define a canvas is to choose an axis-aligned plane, such as the x-y plane, and project the sketch onto that plane (by setting the depth or z component to zero, for instance). The active view-plane also works well as a canvas, allowing the user to draw from multiple angles as they change the viewpoint. A third variation is to project the sketch onto an existing 3D model based on the current viewpoint (Fig. 6).

Some SBIM systems are tailored toward casual or novice users rather than design professionals. To assist a novice with the sketching process, the canvas can be replaced with an image that the user draws on top of [TBSR04, YSvdP05]. An intelligent application can also extract information from the image and, for instance, 'snap' the sketch to edges in the image [TBSR04].

4. Sketch Filtering & Pre-Processing

Before attempting to interpret a sketch, it is usually necessary to perform some filtering. One motivating factor is that the input will invariably contain some noisy or erroneous samples. Sezgin & Davis [SD04] identify two main culprits: user and device error. Poor drawing skills or slight jitter in a user's hand results in not-quite-straight line segments and not-quite-smooth curves. The second source of error is "digitization noise" caused by spatial and temporal quantization of the input by the mechanical hardware: "a traditional digitizing tablet . . . may have resolution as low as 4-5 dpi [dots per inch] as opposed to scanned drawings with up to 1200-2400 dpi resolution. This is because sometimes users draw so fast that even with high sampling rates such as 100Hz only few points per inch can be sampled" [SD04].

Even with careful drawing, device errors and sampling issues must be dealt with. Therefore, the input to a sketch system is generally considered to be an imperfect representation of user intention and is "cleaned up," or filtered, before interpretation. This serves to both reduce noise and to attain a form that makes subsequent recognition and interpretation easier. Below we present some filtering methods commonly used in SBIM; Sec. 4.1 then discusses techniques for sketch recognition.

Resampling The spacing between samples in a raw input stroke varies among devices as well as with the drawing speed of the user. A uniform sampling rate can be beneficial at later stages in a system, so resampling the input stroke is common [BMZB01, DJ03, KS06, OSS07]. Resampling can be done on-the-fly by discarding any sample p_k within a threshold ε of p_{k-1}, and by interpolating between samples separated by more than ε. It can also be done after the sketch is finished. Depending on the needs of the application, linear or smooth interpolation can be used. See Fig. 7a.

Polyline approximation An extreme form of resampling is polyline approximation (or polygon, in the case of closed curves), which drastically reduces the complexity of a stroke to just a few samples (Fig. 7b). For example, Igarashi's Teddy system [IMT99] constructs a closed polygon by connecting a stroke's first and last point, and resampling the stroke so that all edges are a uniform, predefined length. Another simple approach is to simply retain every n-th sample in a stroke, although these approaches can give unsatisfactory results because their sample distribution is not based on local stroke features.

Instead, a good algorithm will place some bounds on the amount of error introduced by approximation, retaining few samples in flat regions and more in regions with lots of detail. The minimax method [KD82], for instance, minimizes the maximum distance of any point to the straight-line approximating line. There are rigorous computational geometry approaches [ESU02] for tackling this problem, but they are intended to operate on positional information; with sketched input, there is additional temporal information that can be used to identify perceptually important points (corners, darts, etc.) in a stroke.

Curve Fitting Polygon approximation works well when the input sketch is intended to be piecewise-linear, but leads to large approximation errors when applied to smooth input. Curve fitting is an alternative approach that yields lower approximation errors at the cost of more computation. Least-squares polynomial fitting [Koe98] is an option, but parametric forms such as Bézier [Pie87, ECYBE97] and B-spline [Rog89, BC90, KS06] curves are preferable in graphics. Figure 7c illustrates spline curve fitting. Yu [Yu03] ar-

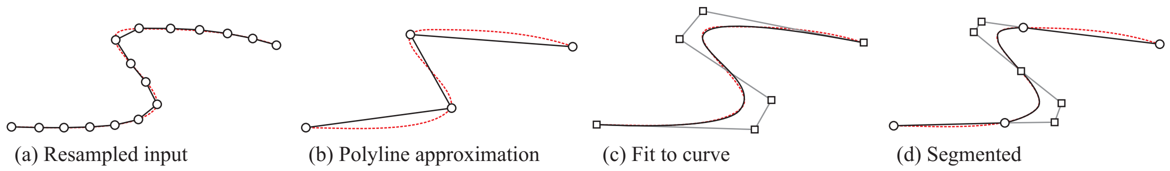

Figure 7: *Filtering operations: (a) smooth uniform resampling; (b) coarse polyline approximation; (c) fit to a spline curve; (d) segmented into straight and curved sections. In each figure, circles denote endpoints of straight line segments, while squares represent curve control points.*

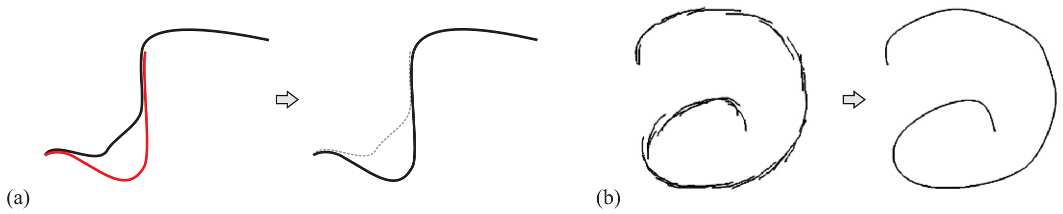

Figure 8: *(a) Oversketching is a quick and effective way to interactively correct a sketch; (b) Oversketched strokes can be blended in a batch process after sketching is complete (reproduced with permission from [PSNW07]); (c) Beautification infers global geometric constraints between strokes, such as parallelism, alignment, and perpendicularity.*

gues that because splines are difficult to compare at a high level, it is better to fit primitive shapes such as circles, ellipses, and arcs; this approach has not caught on in SBIM though, perhaps because it assumes too much about the user's intention.

More recently, subdivision and variational implicit curves have been employed in SBIM systems. Alexe et al. [AGB04] use a Haar wavelet transformation to get a multi-scale stroke representation. Cherlin et al. [CSSJ05] fit a subdivision curve to a stroke by applying reverse subdivision to the raw stroke samples, effectively de-noising the data. Schmidt et al. [SWSJ05] infer geometric constraints from the input sketch to fit a variational implicit curve.

Segmentation There are many examples of sketched input that contain both piecewise-linear and smooth sections. Often it is beneficial to explicitly segment straight and curved sections of a sketch, fitting polylines to the former and smooth curves to the latter [ECYBE97, SMA00, SSD01, KOG02]. Sezgin et al [SSD01], for instance, use speed and curvature data extracted from an input stroke to construct a polyline approximation, and then fit cubic Bézier curves to line segments that have a high approximation error. See Fig. 7d. Segmentation may also refer to the identification of distinct or meaningful sub-sketches within a sketch [LZ04].

Oversketching If a user makes a mistake while drawing a stroke, or wants to redo a portion of an input sketch, he or she can carefully sketch over the offending region. The system can then update the sketch by finding the region affected by the secondary stroke, splicing in the new portion, and smoothing the transition between the old and new segments (Fig. 8a). To facilitate oversketching, an SBIM system usually requires the user to indicate when the sketch is finished [SSD01, DJ03, FRSS04]; alternatively, the system can retain the original sketch and allow for (possibly constrained) 3D oversketching later in the pipeline [CSSJ05, KS06, KDS06, NISA07, WM07].

There is another form of oversketching used by artists in which a drawing is made up of several overlapping strokes, such that the strokes are collectively perceived as a single object (Fig. 8b). Some SBIM systems allow for this type of multi-stroke input, automatically blending the strokes together [SC04, KS05, PSNW07]. In a stroke-space approach, the geometric relationships between strokes are used to blend them; for example, Pusch et al. [PSNW07] use hierarchical space partitioning to divide many strokes into locally orientable segments, and then fit a B-spline curve passing through the segments. In image-based approaches, strokes are blended together 'for free' when the strokes are rendered to an image [KS05]. Finally, a semi-automatic approach may be used, in which the user identifies which strokes can be blended together [SWSJ05].

Beautification The techniques discussed above can all be considered to operate on a local, or per-stroke, level. Beautification (we borrow the term from Igarashi et al. [IMKT97]) is a technique for inferring geometric constraints between strokes on a global level, such as linearity, co-location, parallelism, perpendicularity, and symmetry (Fig. 9). For instance, when drawing a square, the system could fit straight

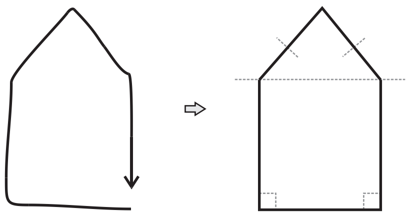

Figure 9: *Beautification infers global geometric constraints between strokes, such as parallelism, symmetry, and perpendicularity.*

line segments to each edge, but also infer that adjacent edges should be at right angles to each other. Beautification can be done either interactively [IMKT97, IH01, CNJC03] or as a batch process after a sketch is finished [SG98].

4.1. Sketch Recognition

During sketch interpretation (Sec. 5), it is often helpful to have a high-level description of a stroke or sketch. That is, if the user has sketched object A, the system may be able to *recognize* it by comparing against a description, or *template*, $\hat{A}$. Generally there will be a set of possible templates, so, much like our visual memory system, sketch recognition algorithms are necessary to search through a "memory" of templates and find the best match.

Many approaches have been proposed for recognizing sketched input. An early approach by Rubine [Rub91] uses geometric properties to compare strokes, such as the initial angle and bounding box size. Graph-based techniques judge similarity from the spatial relationships between strokes in a sketch, such as crossings and shared endpoints [LKS06]. Other methods exploit domain-specific knowledge to derive higher-level understanding of strokes, such as building a diagrammatic representation [AD04] or identifying and labelling different elements [SvdP06]. Hammond and Davis [HD05] propose a sketch recognition "language" in which the template objects are described by their component primitives and geometric constraints between them; for example, a stick figure consists of a circle connected to a line, which is itself connected to four other lines.

Stroke matching borrows conceptual elements from trajectory analysis, the latter dealing with the behavior of moving objects. In the case of a sketch, each stroke captures the trajectory of an input device. Fourier analysis is perhaps the most common technique in trajectory analysis [AFS93, NK06]. A trajectory (equivalently stroke) of variable length is converted to a fixed-length "feature" by separating the 2D positional information into two signals, applying the Fourier transformation to each signal, and retaining a fixed number of the most-significant Fourier coefficients. In this way, the Fourier features can easily be compared with an element-wise distance measure. One drawback of the Fourier transform is that it loses locality of features in the input due to signal-sized waves. Wavelet methods [CF99] attempt to address this issue by using smaller waves, but suffer from signal length restrictions. The Fourier method has been used in the context of SBIM by Shin & Igarashi's Magic Canvas system [SI07] (discussed in more detail in Sec. 5.1.1).

Shape matching is another existing field with parallels to sketch recognition, mostly based on comparing silhouettes and contours (see Veltkamp [Vel01] for a good introduction). There are important distinctions to make between sketch recognition, however. First, the underlying representation used in shape matching is typically an image or pixelmap, while sketch-based systems are based on stroke input. Image-based techniques have been explored in the context of sketch recognition [FMK*03, KS05], but this conversion loses the potentially important temporal and auxiliary (pressure, etc.) information available in a sketch. A second distinction is that sketch-based systems strive for interactive performance, and will often trade rigor for computational efficiency. Funkouser et al. [FMK*03] argue for the use of image-based matching in applications in which the user is free to provide "fragmented sketch marks," because contour matching methods will likely fail on such input. This is less of a problem in light of recent work in batch oversketching to blend fragmented sketches into a single contour.

Though much can be learned from the literature in other domains, the best results are likely to be obtained from methods targeted at the particular problem of sketch recognition. For example, the Teddy system [IMT99] uses simple geometric properties of strokes, such as the ratio of a stroke's length to its convex hull perimeter, to match input strokes to operations (see Sec. 5.3). Yang et al. [YSvdP05] similarly extract a few simple measurements from a stroke, including straight-line length, angle, free-form arc length, and the area between the stroke and its straight-line approximation; these values are used as a stroke "signature" for recognition. More recently, Olsen et al. [OSS07] propose a method of describing a stroke based on quantizing its angular distribution, and within the context of SBIM were able to outperform classical methods, including Fourier. Each of these approaches would likely give poor results for general shape matching, but perform well within their target application.

5. Sketch Interpretation in SBIM

After a sketch has been sufficiently filtered, converted, and recognized, the next stage of the pipeline is interpret the sketch. We propose a categorization of SBIM systems based on how input strokes are interpreted. The most important category includes systems that *create* fully 3D models automatically from input sketches (Sec. 5.1). Other important tasks include using input strokes to *augment* existing models with details (Sec. 5.2) and to *deform* an existing model (Sec. 5.3).

	Creation Mode					*Surface Type*				*Editing Operations*						*Interface*	
	Extrapolate	Template	CAD	Contours	3D/Multi	Parametric	Mesh	Implicit	Fair	Surficial Aug.	Additive Aug.	Cut/Tunnel	Oversketch	Bend/Twist	CSG/Boolean	Suggestive	Gestural
SKETCH [ZHH96]	●						●									●	
Quicksketch [ECYBE97]				●		●	●			●				●		●	
Digital Clay [SG98]			●				●										
Teddy [IMT99,IH03]				●			●			●		●		●			●
GIDeS [PJBF00]	●		●		●		●								●		●
Chateau [IH01]	●						●									●	
CIGRO [CNJC03]			●				●										●
BlobMaker [DJ03]				●				●			●		●		●	●	●
3D Search [FMK*03]		●					●									●	
[AGB04]				●				●			●			●	●		
Smartpaper [SC04]			●				●				●	●			●		●
ConvMo [TZF04]				●			●				●				●		
Ribald [VTMS04]			●				●										
[CSSJ05]				●		●							●	●			
[DDGG05]					●		●										
[ML05]							●						●				
ShapeShop [SWSJ05]				●				●		●	●	●			●	●	●
[VMS05]			●				●										
[YSvdP05]		●				●				●							
SmoothSketch [KH06]				●			●										
[KS06]					●		●						●				●
[CS07]				●				●									
[HL07]					●		●										
Plushie [MI07]				●			●				●	●					●
FiberMesh [NISA07]				●					●	●	●	●	●				●
[RSW.07]					●				●							●	
Magic Canvas [SI07]		●					●									●	●
[WM07]				●		●	●						●		●		●

Table 1: *Taxonomy of sketch-based modeling systems, including creation mode (Sec. 5.1), surface representation (Sec. 5.4), editing operations (Sec. 5.2 and 5.3), and interface type (Sec. 5.5).*

There are a variety of surface representations available to an SBIM system, each having strengths and weaknesses (Sec. 5.4). Finally, choosing the correct interpretation at the correct time requires a carefully designed interface, as discussed in Sec. 5.5.

A classification of 'complete' SBIM systems – that is, systems that offer both model creation and editing tools via a sketching metaphor – is presented in Table 1, from early (SKETCH [ZHH96], Teddy [IMT99] to state-of-the-art (SmoothSketch [KH06], FiberMesh [NISA07]) systems. The table summarizes the main techniques used and features offered in each system, and also indicates the surface representation and interface design choices when such information is available.

5.1. Model Creation Systems

To make a reasonable attempt at reconstruction, an SBIM system must somehow perceive a 3D shape from the 2D input. We divide the gamut of creation systems into two categories: *suggestive*, and *constructive*. This aligns with the classical distinction between reconstruction and recognition. Suggestive systems first recognize which template a sketch corresponds to, and then use the template to reconstruct the geometry. Constructive systems forgo the recognition step, and simply try to reconstruct the geometry. In other words, suggestive systems are akin to visual memory, whereas constructive systems are more rule-based.

Because suggestive systems use template objects to interpret strokes, their expressiveness is determined by the richness of the template set. Constructive systems, meanwhile, map input sketches directly to model features; therefore, their expressiveness is limited only by the robustness of the reconstruction algorithm and the ability of the system's interface to expose the full potential.

Of course, there is some overlap between constructive and suggestive systems. This is mostly embodied by suggestive systems that deform the template objects to match the in-

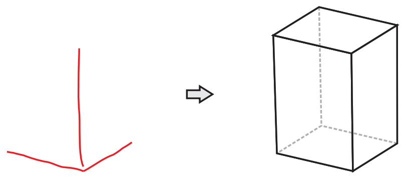
Figure 10: *A suggestive-stroke system extrapolates a 3D form from only a few evocative strokes.*

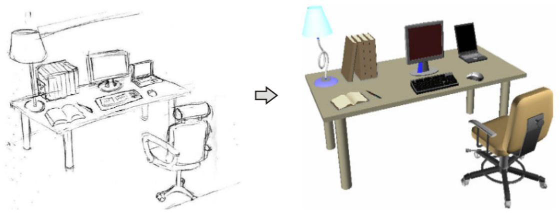
Figure 11: *A template retrieval system matches sketches to 3D models, useful for applications such as scene construction. Reproduced with permission from [SI07].*

put sketch [YSvdP05], or constructive systems that exploit domain-specific knowledge.

5.1.1. Suggestive Strokes

Suggestive-stroke systems are characterized by the fact that they have some "memory" of 3D shapes built in, which guides their interpretation of input sketches. Reconstruction of a sketch depends on being able to recognize it. This correlates with the concept of visual memory, but transferring the entire visual experience of a human to a computer is a daunting task. Therefore, suggestive systems are only as good as their shape memory allows them to be. Within this category, we identify two main approaches: extrapolation, and template retrieval.

Extrapolation In this approach, the system extrapolates a final 3D shape based on only a few meaningful strokes [ZHH96, PJBF00]. A classical example is the SKETCH system of Zeleznik et al. [ZHH96], which uses simple groups of strokes to define primitive 3D objects. Three linear strokes meeting at a point, for instance, are replaced by a cuboid whose dimensions are defined by the strokes (see Fig. 10).

The Chateau system of Igarashi and Hughes [IH01] also extrapolates shape from a few strokes. Limiting their system to architectural forms allows it to make assumptions about the input such as planarity, symmetry, orthogonality, and so forth. The interactive nature of the system also keeps the recognition tasks simple and concise, avoiding many problematic cases since the user can see immediately how the system has or will interpret their action.

Template Retrieval The second main approach is to retrieve template objects from a database or set of template objects based on the input sketch [FMK*03, YSvdP05, SI07]. This approach is more extensible than extrapolation: adding new behavior to the system is as easy as adding a new object to the database. Conversely, because the "building blocks" – the shape templates – are generally more than just primitive objects, it may be impossible to attain a specific result.

A retrieval-based system faces the problem of matching 2D sketches to 3D templates. To compare their similarity in 3D would require reconstruction of the sketch, which is the ultimate problem to be solved. Thus, comparison is typically done by projecting the 3D model into 2D. Funkhouser et al. [FMK*03], for example, extract the contour from 13 different viewpoints to define the shape of an object. Input sketches are then compared against each of these contours by an image-based matching method. To improve the recognition rate, the user can sketch three different views of an object. In a user study, they found that "people tend to sketch objects with fragmented boundary contours and few other lines, they are not very geometrically accurate, and they use a remarkably consistent set of view directions." These observations allowed them to extract meaningful 2D representations of the 3D templates.

Shin and Igarashi's Magic Canvas system [SI07] uses template retrieval for scene construction (Fig. 11). They too extract several (16) contours from each template object, but use a stroke-based Fourier method for contour matching. Constructing a scene with several objects requires not just template retrieval, but also correct placement of each object within the scene. Thus, Magic Canvas rotates and scales the objects to match the input sketch orientation, and also infers simple geometric relationships (such as a lamp resting on top of a desk). User intervention is required to initiate the retrieval process on subsketches within the scene, and also to choose appropriate objects among several candidates.

Yang et al. [YSvdP05] propose a similar template-based system, but rather than mesh-based templates, they use procedurally-described models. Instead of having a mug mesh for a template, for instance, they have a template that describes how to *make* a mug out of simple primitives. This approach has the benefit of allowing the template to be deformed to match the input sketch, rather than just replaced with an instance of the template. However, the procedural template definition makes adding new templates more difficult than mesh-based approaches.

5.1.2. Constructive Strokes

Pure reconstruction is a more difficult task than recognize-then-reconstruct, because the latter uses predefined knowledge to define the 3D geometry of a sketch, thereby skirting the ambiguity problem to some extent (ambiguity still exists in the recognition stage). Constructive-stroke systems must reconstruct a 3D object from a sketch based on rules alone.

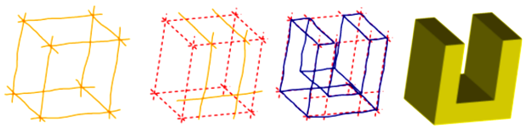
Figure 12: *CAD systems exploit domain-specific knowledge to reconstruct quite complex sketches. Adapted with permission from [CNJC03].*

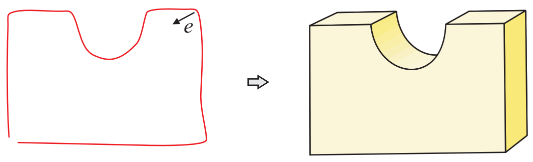

Figure 13: *Extrusion is a simple method for reconstructing a contour, by sweeping it along an extrusion vector e.*

Because reconstruction is such a difficult and interdisciplinary problem, there have been many diverse attempts at solving it. We identify three main interpretations in constructive systems: CAD drawings, contour lines, and 3D boundaries.

Computer-Assisted Design (CAD) Design and specification of mechanical (i.e. mostly planar) objects is an important industrial application of computer modeling. As such, it attracted attention early in the life of SBIM [Pug92, SG98,PJBF00,CNJC03,CPC04,FBSS04,VTMS04,VMS05, ML05]. Focusing on a particular domain helps to constrain the reconstruction problem. In CAD systems, the input is usually assumed to be a line drawing that defines an object in an orthogonal frame, roughly in line with the first three visual rules proposed by Hoffman. This allows unambiguous reconstruction of drawings such as the cube in Fig. 2; the choice of orthogonal frame eliminates one of the two possible cube orientations.

Reconstruction of CAD drawings depends on identifying the locations of vertices, or corners, of the object. In an interactive system, vertex positions and connections can be determined as the user draws the strokes; reconstruction from an image alone would be more difficult, as vertex connections would have to be determined algorithmically.

Early approaches left most of the burden on the user. For example, Pugh's system [Pug92] started from a primitive object like a cube in order to provide a 3D reference, and required the user to explicitly specify geometric constraints. Later approaches to CAD reconstruction fall into one of two classes: interactive, or batch. Interactive approaches [PJBF00,CNJC03] reconstruct the geometry incrementally as the user sketches (Fig. 12), allowing the user to see the result and possibly correct or refine it. Batch systems [FBSS04, VMS05, ML05] wait until the user has provided a complete sketch and attempt to reconstruct it according to internal rules or vertex-connection templates.

Recent systems have included support for curved segments in CAD drawings. Varley et al. [VTMS04] use a two-stage approach: in the first stage, the user draws an overall model structure with only straight lines; in the second stage, the model is re-drawn with curved segments, and the reconstructed model from the first stage acts as a template for reconstruction. Masry and Lipson [ML05] also use a two-stage approach, but theirs is hidden from the user: the system automatically extracts a straight-line representation via segmentation.

The CAD approach overlaps somewhat with suggestive systems, in that expressiveness is limited by the domain in question. However, reconstruction is not based on or limited by recognition: the user is free to create an almost limitless variety of objects within that domain, unhindered by any template set (at least in the sense of object templates used in suggestive systems – some CAD systems do in fact use templates for vertices of different orientation and valence [Pug92]).

Contour Lines Though some CAD systems support curved strokes, reconstruction is still based on a straight-line representation. Reconstructing smooth, natural objects requires a different approach. As Hoffman's sixth rule indicates, our visual system interprets smooth line drawings as 3D contours when possible. Fittingly, the majority of constructive SBIM systems choose to interpret strokes as contour lines [ECYBE97,IMT99,AGB04,TZF04,CSSJ05,SWSJ05, KH06, NISA07]. There are still many objects that correspond to a given contour, so further assumptions must be made to reconstruct a sketch. A key idea in constructive systems is to choose a simple shape according to some internal rules, and let the user refine the model later.

To justify this approach, we might observe that of the three plausible models shown in Fig. 2 whose contour lines project to a cube-like object, only the cube itself conforms to the our visual rules. Hoffman [Hof00] calls the other candidates "accidental views", since any slight change in viewpoint would reveal them to be non-cubes. Because of this, they are eliminated as candidates by our visual system. Now consider a user of an SBIM system who wants to draw one of the non-cubes. Would they choose to draw the object from the accidental viewpoint? Not likely, because their own visual rules would see it as a cube. So although there are infinitely many ways to reconstruct a drawing, the number of corresponding non-accidental shapes is substantially lower.

Perhaps the most simplistic method of reconstruction is *extrusion* – that is, creating a surface by "pushing" the contour through space along some direction *e* [ECYBE97,SC04, SWSJ05, WM07]; see Fig. 13 for an illustration. This technique, also known as linear sweep, is useful for creating

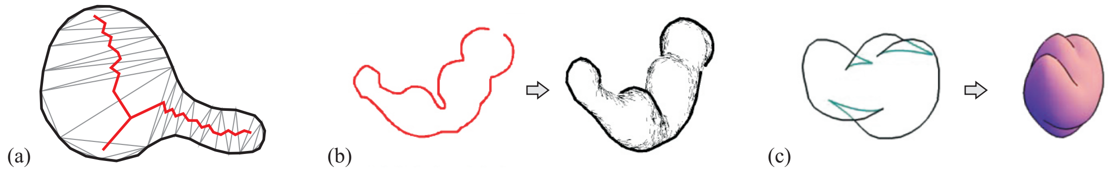

Figure 14: *Inflation of contour sketches: (a) the Delaunay triangulation (interior lines) and chordal axis (red line) are useful during reconstruction; (b) Teddy inflates a contour drawing about its chordal axis (reproduced with permission from [IMT99]); (c) SmoothSketch infers hidden contour lines (green lines) before inflation (reproduced from [KH06]).*

models with hard edges, such as cubes (extruded from a square) and cylinders (from a circle).

Extrusion results in models with a quadrilateral cross-section, which is inappropriate for many natural objects. To attain models with smoother cross-sections, surfaces of revolution are a common alternative [ECYBE97, CSSJ05, SWSJ05]. Cherlin et al. [CSSJ05], for instance, constructing a generalized surface of revolution about the medial axis between two strokes (the authors refer to this construction as *rotational blending surfaces*). Their system also allows the user to provide a third stroke that defines a free-form cross-section.

Circular cross-sections can also be achieved by *inflating* a contour sketch, akin to blowing up a balloon [IMT99, DJ03, KH06]. For simple (i.e. non-intersecting) closed contours, inflation is an unambiguous way to reconstruct a plausible 3D model. The Teddy system [IMT99] inflates a closed contour by creating a polygon approximation, computing the Delaunay triangulation (DT) [dBvKOS00], extracting the chordal axis, and then inflates the object by pushing vertices away from the chordal axis according to their distance from the contour; see Fig. 14b for a typical result.

The "skeleton" of a closed contour defined by the chordal axis transform (or other methods [LG07]) is a useful piece of information to an SBIM system [IMT99, DJ03, AGB04, TZF04]. A skeleton consists of all points within the contour that are equidistant from at least two points on the contour, and can be approximated from the DT of a closed polygon by connecting the center points of adjacent non-boundary triangles (Fig. 14a). As Teddy has shown, the chordal axis and DT can be used to generate a triangle mesh model quickly and easily. The skeletal representation of a contour also integrates naturally with an implicit surface representation. In the approach of Alexe et al. [AGB04], spherical implicit primitives are placed at each skeleton vertex; when the primitives are blended together, the result is a smooth surface whose contour matches the input sketch.

For non-simple contours, such as ones containing self-intersections, a simple inflation method will fail. Karpenko and Hughes [KH06] also use an inflation approach, but they use the contour completion of Williams to infer the hidden contours in a sketch (Fig. 14c). This information allows the sketch to be positioned in 3D such that it can be inflated without self-intersections. A smooth shape is attained by first creating a "topological embedding" and then constructing a mass-spring system (with springs along each mesh edge) and letting the system reach a smooth equilibrium state. Cordier and Seo [CS07] also use Williams' contour completion algorithm, but propose a different method of reconstructing a model from an input sketch. Rather than using mass-spring mesh relaxation – which requires careful parameter tuning and does not guard against self-intersections – the authors use Alexe et al.'s implicit surface reconstruction method.

A final way to reconstruct a contour sketch is to fit a surface that is, as Hoffman's tenth rule suggests, "as smooth as possible". Surface fitting approaches operate on the premise that the input strokes represent geometric constraints of the form, 'the surface passes through this contour, with surface normals defined by the contour's normal.' These constraints define an optimization problem: of the infinite number of candidates, find one suitable candidate that satisfies the constraints. Additional constraints such as smoothness and thin-plate energy [Wil94] are required to push the system toward a solution. This approach has been used to reconstruct variational implicit surfaces [TO99, DJ03, SWSJ05]. More recently, Nealen et al. [NISA07] have used a non-linear optimization technique to generate smoother meshes while also supporting sharp creases and darts.

3D Sketching The reconstruction methods for 2D contours lead to fairly bland-looking models, due to the simplistic rules governing them. Other ways to disambiguate sketched input are to sketch with 3D strokes [FBSS04, DDGG05, RSW*07], or to sketch from multiple viewpoints [PJBF00, FMK*03, TBSR04, HL07]. These strokes are typically interpreted as object boundaries. Das et al. [DDGG05], for example, use a 3D network of curves to define the boundaries of an object, smoothly interpolating between them to reconstruct a model. Rose et al. [RSW*07] also use 3D boundary sketches, to define smooth planar deformations known as developable surfaces.

However, sketching in 3D is arguably less intuitive since our visual system is built around 2D stimuli. Thus most systems are content to implement simple reconstruction within an iterative modeling paradigm. That is, rather than the user

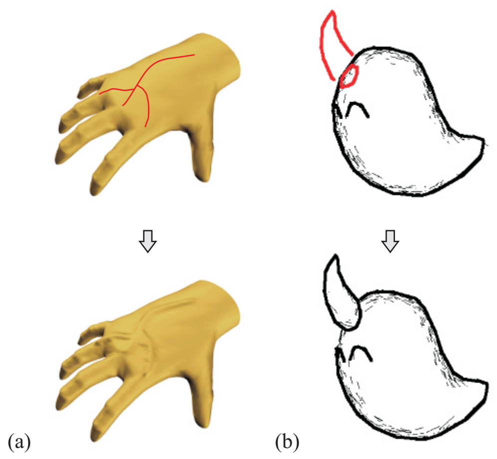

Figure 15: *Sketch-based augmentations: (a) surficial augmentation displaces surface elements to create features (from [OSSJ05]); (b) additive augmentation joins a new part with an existing model (reproduced with permission from [IMT99]). The latter figure also includes surficial features (the eyes).*

creating 3D or multiple sketches of an object, they can reconstruct a single sketch, rotate the model, sketch a new part or a deformation, ad infinitum until the desired result is achieved. The editing components of such a system are the topic of the following two sections.

5.2. Augmentation

As the previous section illustrated, creating a 3D model from 2D sketches is a difficult problem whose only really feasible solutions lead to simplistic reconstructions. Creating more elaborate details on an existing model is somewhat easier however, since the model serves as a 3D reference for mapping strokes into 3D (Fig. 6). Projecting a stroke onto a model relies on established graphical techniques, such as ray-casting (cast a ray from eye position through stroke point on the drawing plane) or unprojection (invert the view matrix, then use z-buffer to find depth) [OSSJ05]. Augmentations can be made in either an *surficial* or *additive* manner.

Surficial augmentation allows users to sketch features on the surface of the model, such as sharp creases [BMZB01, OSSJ05, NISA07] or curve-following slice deformations [ZG04]. After a sketch has been projected onto a surface, features are created by displacing the surface along the sketch. Usually the surface is displaced along the normal direction, suitable for creating details like veins (Fig. 15a). The sketched lines may also be treated as new geometric constraints in surface optimization approaches [NISA07].

Surficial augmentations can often be done without changing the underlying surface representation. For example, to create a sharp feature on a triangle mesh, the existing model edges can be used to approximate the sketched feature and displaced along their normal direction to actually create the visible feature [NSACO05, OSSJ05].

Additive augmentation uses constructive strokes to define a new part of a model, such as a limb or outcropping, along with additional stroke(s) that indicate where to connect the new part to the original model [IMT99, NISA07]. For example, the extrusion operator in Teddy [IMT99] uses a circular stroke to initiate the operation and define the region to extrude; the user then draws a contour defining the new part, which is inflated and attached to the original model at the connection part (Fig. 15b). Schmidt et al. [SWSJ05] exploit the easy blending afforded by an implicit surface representation to enable additive augmentation, with parameterized control of smoothness at the connection point. Their system does not require explicit specification of the connection point, since implicit surfaces naturally blend together when in close proximity. Additive augmentation only affects the original model near the connection point.

The somewhat subjective difference between the two types of augmentation is one of scale: surficial augmentations are small-scale and require only simple changes to the underlying surface, whereas additive augmentations are on the scale of the original model. The distinction can become fuzzy when a system allows more pronounced surficial augmentations, such as Zelinka and Garland's curve analogy framework [ZG04], which embeds 2D curve networks into arbitrary meshes, and then displaces the mesh along these curves according to a sketched curve.

5.3. Deformation

Besides augmentation, there have been many SBIM systems that support sketch-based editing operations, such as cutting [WFJ*05, JLCW06, NISA07], bending [IMT99, CSSJ05, JLCW06, KS06, WM07], twisting [KG05], tunneling (creating a hole) [SWSJ05, NISA07] contour oversketching [NSACO05, ZNA07], segmentation [YXN*05, JLCW06], free-form deformation [DE03], and affine transformations [SSS06]. And, like augmentation, sketch-based deformations have a straightforward and intuitive interpretation because the existing model or scene anchors the sketch in 3D.

To cut a model, the user simply needs to rotate the model to an appropriate viewpoint and draw a stroke where they want to divide the model. The stroke can then be interpreted as a cutting plane, defined by sweeping the stroke along the view direction (Fig. 16a). Tunneling is a special case of cutting, in which the cutting stroke is a closed contour contained within a model – everything within the projected stroke is discarded, creating a hole.

Other deformations are based on the idea of oversketch-

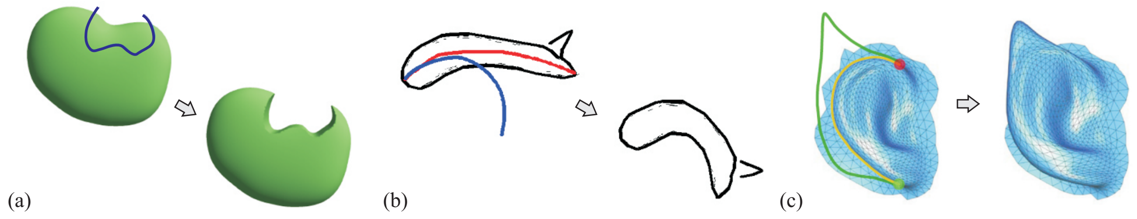

Figure 16: *Sketch-based deformations: (a) cutting strokes (blue) define a cutting plane along the view direction (from [WFJ*05]); (b) bending a model so that the reference stroke (red) is aligned with the target stroke (blue) [IMT99]; (b) contour oversketching matches object contours (yellow) to target strokes (green) (reproduced with permission from [NSACO05]).*

ing. For example, bending and twisting deform an object by matching a *reference* stroke to a *target* stroke, as shown in Fig. 16b. Contour oversketching is also based on matching a reference to a target stroke, but in this case, the reference is a contour extracted from the model itself, as in Fig. 16c.

Nealen et al. [NISA07] support a handle-based deformation, allowing object contours to be manipulated like an elastic. When a stroke is "grabbed" and dragged, the stroke is elastically deformed orthogonal to the view plane, thereby changing the geometric constraint(s) represented by the stroke. As the stroke is moved, their surface optimization algorithm recomputes a new fair surface interactively.

Free-form deformation (FFD) is a generalized deformation technique based on placing a control lattice around an object or scene. Objects within the lattice are deformed when the lattice points are moved, akin to manipulating a piece of clay. Draper and Egbert [DE03] have proposed a sketch-based FFD interface that extends the functionality of Teddy, allowing bending, twisting, and stretching. Both local and global deformations can be specified with FFD.

Model assembly – typically an arduous task, as each component must be translated, rotated, and scaled correctly – is another editing task that can benefit from a sketch-based interface. Severn et al [SSS06] propose a technique for applying affine transformations to a model with a single stroke. From a U-shaped transformation stroke, their method determines a rotation from the stroke's principal components, a non-uniform scaling from the width and height, and a translation from the stroke's projection into 3D. By selecting components and drawing a simple stroke, the model assembly task is greatly accelerated.

Sketch-based interfaces have also been explored for character animation, using hand-drawn sketches to specify key poses or positions in an animation sequence [DAC*03, TBvdP04, CON05]. Davis et al. [DAC*03], for instance, extract joint positions from a stick-figure sketch via image-processing techniques such as erosion; geometric and physical constraints are then applied to rule out implausible poses, and the character model is deformed to match the sketched pose. Thorne et al. [TBvdP04] instead allow the user to sketch character motion in 2D or 3D, using a set of sketch "gestures" (see Sec. 5.5) that can be mapped to pre-defined motions such as walking and jumping.

5.4. Surface Representation

Choosing an appropriate surface representation is an important design decision. Each has benefits and drawbacks that must be weighed to suit the needs of the intended application. Below we discuss the main surface types.

Parametric Surfaces include NURBS patches, surfaces of revolution, and rotational blending surfaces, and are parameterized over a 2D space [ECYBE97, CSSJ05]. They are a well-studied representation and easy to integrate into an application. However, because of the simple parameter space, the topology of a single surface is limited to shapes homeomorphic to a plane. Building more interesting shapes with branching structures or complex topology requires either crude patch intersections or careful alignment of several patches.

Meshes are another well-studied surface representation used in SBIM [IMT99, DDGG05, NSACO05, HL07, LG07]. Unlike parametric surfaces, meshes support arbitrary topology. The drawback of meshes is that some editing operations are difficult to implement, such as blending two objects together. Mesh quality is also an issue [IH03, LG07, WM07]. A mesh-like representation is necessary for rendering an object to the display, but more flexible representations can be used in the background.

Implicit surfaces naturally support hierarchical modeling, blending, and boolean operations. However, attaining interactive performance is technically challenging because the surface must be discretized to a mesh representation before rendering. Nevertheless, with careful implementation implicit surfaces have been shown to be a viable surface representation for SBIM [DJ03, AGB04, SWSJ05], and are also used as an intermediate representation from which to extract a mesh [TZF04, LG07].

Fair surfaces are meshes that result from solving a constrained optimization problem [Wil94, NISA07, RSW*07].

As the user sketches, new constraints are defined and the solution is re-computed. This is a very flexible representation and well-suited to SBIM, but has a couple of important drawbacks. First, the fitted surfaces are generally very smooth, even with sharp-feature constraints, limiting the expressiveness. Second, because the surface results from a global optimization, the resulting surface is sometimes unexpected from the user's perspective.

5.5. Interface Design

With all of the possible ways to interpret an input stroke or sketch – creation, augmentation, deformation – how does the system decide what to do at any given moment? Traditional modeling applications use modal interfaces, requiring the user to explicitly choose an operation mode from a menu or toolbar before performing the operation. Modal interfaces have also been used in SBIM systems. For example, to initiate the bending operation in Teddy the user must click a button after drawing the reference stroke; this informs the system to interpret the next stroke as a target stroke and perform a bending operation.

There are two main approaches to building sketch-based interfaces: *suggestive* and *gestural*. In a *suggestive* interface, the system will consider all possible interpretations (or modes), and let the user decide which one is intended [ECYBE97, IMKT97, IH01, TBSR04, SWSJ05]. Returning to the bending operation in Teddy, the system would realize that the first stroke could either be a surficial detail *or* a reference stroke for bending, and somehow notify the user of the multiple interpretations. Figure 17 depicts a suggestive interface in action.

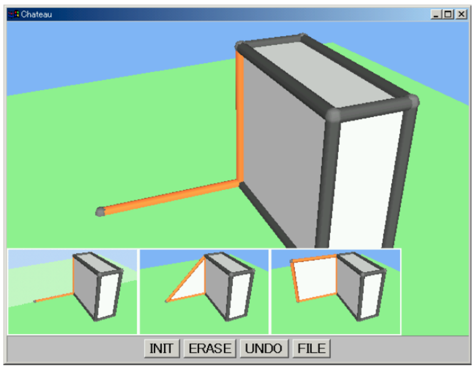

Figure 17: *Suggestive interfaces: when ambiguous user input is provided, the user can be given a choice among several possible interpretations (insets). Reproduced with permission from [IH01].*

The problem with modal interfaces is that the user may become frustrated by errors that result from being in the incorrect mode. Non-modal interfaces attempt to avoid mode errors by designing non-redundant inputs [Ras00], and have been used in SBIM systems that seek a unified sketch-based interface.

In a *gestural* interface, an operation is specified by a simple, distinct gesture stroke [PJBF00, LLRM00, LM01, DE03, TBvdP04, KS06, OSS07]. Mapping a gesture to an operation requires some sort of sketch recognition, such as the methods discussed in Sec. 4.1. However, the complexity of recognition can be reduced by the gesture vocabulary: often, quite simple recognition will suffice for a well-defined set of gestures, since "perceptual similarity of gestures is correlated with . . . computable features such as curviness" [LLRM00]. Care should be taken to design a 'good' set of gestures – that is, a set that is distinct, memorable, and easy to draw.

When an interface relies on successful recognition, there is some probability that an input will be mis-recognized; this can become frustrating for a user if it happens too often. A simple way to guard against this is to combine gestural and suggestive interfaces: the recognized gesture is chosen as the default operation, but the user is able to override it. Another option is to train the gesture recognizer with user input, tailoring the system according to how each individual user tends to draw the gestures [KS05, KS06].

6. Challenges & Open Problems

Despite the incredible advances in SBIM, there remain several important open problems. Reconstruction and recognition algorithms still have a long way to go before approaching the power of human perception, and the inherent ambiguity of 2D images suggests that flawless reconstruction of arbitrary sketches is impossible. A recognition-based approach is primarily limited by the size of the available shape memory. Therefore, as Karpenko & Hughes [KH06] suggest, a hybrid system "in which the user's sketch is both inflated and matched against a large database of known forms" could be very powerful. A hybrid approach would also temper the fact that while constructive systems are generally suited to building rough prototypes or cartoony-looking models, suggestive systems can produce more precise models from the template set.

The development of SBIM, in some ways, parallels the development of non-photorealistic rendering (NPR). NPR asks the question, "How can a 3D model be rendered artistically in a way that accurately and clearly reveals its shape?" Early approaches found that contour lines alone are very evocative of the visual memory, similar to the use of contour lines for reconstruction in most early SBIM systems. NPR approaches advanced beyond contour lines to include suggestive contours, hatching lines, pen-and-ink, stippling, and various other artistic ways to provide shape cues. Perhaps SBIM – which has been referred to as "inverse NPR" [NSACO05] – can learn from these developments and *extract* shape information from artistic cues. While there has been some progress toward extracting shape information from shading in single images (shape-from-shading) [ZTCS99, PF06], it remains an open and challenging area in SBIM.

Model quality remains another important challenge in SBIM. Because the user does not have precise control over the surface as they would in a traditional modeling package, and due to the typical "as smooth as possible" reconstruction approach, models created by SBIM systems tend to have a blobby appearance. Recent work such as FiberMesh [NISA07], which allows the specification of sharp creases and darts, is a step in the right direction. In the future, modeling concepts such as multiresolution editing and topology-changing operations could dramatically increase the utility of sketch-based systems.

Designing the interfaces such that there is a noticeable and worthwhile increase in utility compared to a traditional interface is another challenge. While navigating through three levels of menu items to find the desired operation in Maya may be cumbersome, once it has been found and activated the result of the operation is predictable and deterministic. A sketch-based system, on the other hand, is largely built around elaborate guesswork and inference, of classifying input as being more like Operation A than Operation B. When a system makes the wrong choice, it can be very frustrating for the user. As Landay and Myers note about their system, "failure to provide sufficient feedback about its recognition was the source of most of the confusion" [LM01]. Thus, designing SBIM systems with the right combination of algorithmic and interface elements to provide stable and predictable interaction is a huge challenge for ongoing research. This includes the ability to recognize troublesome inputs and smoothly guide the user to a resolution.

Sketch-based interfaces also suffer from the problem of self-disclosure [JJL07]. Traditional WIMP interfaces are discoverable, in the sense that a user can look at the menu titles, icons, buttons, and dialog boxes, and garner some idea of what the application can do and how to use it. An SBIM system, on the other hand, may simply provide the user with a blank window representing virtual paper, with no buttons or menus whatsoever. Though it may be more usable and efficient for someone who has been given a tutorial, such an interface does not disclose any hints about how to use it. Devising elegant solutions to this problem is a large challenge for SBIM researchers.

Finally, sketch-based or gestural interfaces have the potential to improve other modeling tasks. Already there have been some novel uses of gestural interfaces, such as clothing manipulation [IH02, DJW*06], finite element analysis [ML05], procedural plant modeling [OMKK06, ZS07] and even stuffed toy design [MI07]. In each case, sketched input is used to define an initial state of a complex physical simulation or procedural model, domains that are typically encumbered with many parameters and initial settings to define. Mori and Igarashi [MI07] provide an intriguing example of how SBIM techniques could be integrated with physical simulation: "if one can run an aerodynamic simulation during the interactive design of an airplane model, it might be helpful to intelligently adjust the entire geometry in response to the user's simple deformation operations so that it can actually fly." Exploring the output space of a procedural or physical model can be much more natural and efficient with free-form gestures, a notion that needs to be explored more fully in the future.

7. Conclusion

Sketch-based systems have a reputation of being suitable only for "quick-and-dirty" [YSvdP05] modeling tasks, an image that must be shed if the field wants to be a viable alternative to high-end modeling packages. This report has shown a tremendous diversity of techniques and applications, illustrating that SBIM is suitable to a wide range of modeling tasks.

We simply have to embrace the ambiguous nature of sketched input. Art is often an iterative process, progressing from a rough outline to a highly detailed product – a character animator will first draw the form of a character with ellipses and other primitive shapes, then slowly add layers of complexity. The key is that the medium is predictable: an artist knows exactly what will happen when a pencil is ran across a piece of paper, or a paint brush across a canvas. This should inspire SBIM to pursue stable and predictable interfaces that naturally support an iterative methodology, rather than blind reconstruction. As Nealen et al. [NSACO05] argue, though "our capability to derive a mental model from everyday shapes around us is well developed, we fail to properly communicate this to a machine. This is why we have to model in a loop, constantly correcting the improper interpretation of our intentions."

The confluence of so many disciplines – graphics, vision, HCI, cognition – will ensure sketch-based modeling remains an exciting and challenging topic for years to come.

References

[AD04] Alvarado C., Davis R.: Sketchread: a multi-domain sketch recognition engine. In *Proc. of ACM symposium on User interface software and technology (UIST '04)* (2004), pp. 23–32.

[AFS93] Agrawal R., Faloutsos C., Swami A.: Efficient similarity search in sequence databases. In *Proc. of Intl. Conference of Foundations of Data Organization and Algorithms* (1993).

[AGB04] Alexe A., Gaildrat V., Barthe L.: Interactive modelling from sketches using spherical implicit functions. In *Proc. of Intl. Conference on Computer graphics, virtual reality, visualisation and interaction in Africa (AFRIGRAPH '04)* (2004), pp. 25–34.

[ASSJ06] ANASTACIO F., SOUSA M. C., SAMAVATI F., JORGE J.: Modeling plant structures using concept sketches. In *Proc. of the Symposium on Non-Photorealistic Animation and Rendering (NPAR '06)* (2006).

[Auta] AUTODESK, INC.: Maya. www.autodesk.com/maya.

[Autb] AUTODESK, INC.: Maya. www.autodesk.com/autocad.

[Autc] AUTODESK, INC.: Mudbox. www.mudbox3d.com.

[BC90] BANKS M., COHEN E.: Real time spline curves from interactively sketched data. *Proc. of SIGGRAPH '90 24*, 2 (1990), 99–107.

[BMZB01] BIERMANN H., MARTIN I., ZORIN D., BERNARDINI F.: Sharp features on multiresolution subdivision surfaces. *Graphics Models (Proc. of Pacific Graphics '01) 64*, 2 (2001), 61–77.

[CF99] CHAN K.-P., FU A.: Efficient time series matching by wavelets. In *Proc. of 15th International Conference on Data Engineering* (1999).

[CGFJ02] CAETANO A., GOULART N., FONSECA M., JORGE J. A.: Sketching user interfaces with visual patterns. In *Proc. of 1st Ibero-American Symposium on Computer Graphics (SIACG'02)* (2002).

[CNJC03] CONTERO M., NAYA F., JORGE J., CONESA J.: *CIGRO: A Minimal Instruction Set Calligraphic Interface for Sketch-Based Modeling*, vol. 2669/2003 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2003, p. 989.

[CON05] CHEN B.-Y., ONO Y., NISHITA T.: Character animation creation using hand-drawn sketches. *The Visual Computer 21*, 8-10 (2005), 551–558. Pacific Graphics 2005 Conference Proceedings.

[CPC04] COMPANY P., PIQUER A., CONTERO M.: On the evolution of geometrical reconstruction as a core technology to sketch-based modeling. In *Proc. of Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM '04)* (2004).

[CS07] CORDIER F., SEO H.: Free-form sketching of self-occluding objects. *IEEE Computer Graphics and Applications 27*, 1 (2007), 50–59.

[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.

[CSSM06] CHEN J., SAMAVATI F., SOUSA M. C., MITCHELL R.: Sketch-based volumetric seeded region growing. In *Proc. of Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM '06)* (2006).

[DAC*03] DAVIS J., AGRAWALA M., CHUANG E., POPOVIĆ Z., SALESIN D.: A sketching interface for articulated figure animation. In *Proc. of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2003), pp. 320–328.

[Das] DASSAULT SYSTEMES: Catia. www.3ds.com/products-solutions/brands/CATIA.

[dBvKOS00] DE BERG M., VAN KREVELD M., OVERMARS M., SCHWARZKOPF O.: *Computational Geometry: Algorithms and Applications*, 2nd ed. Springer, 2000.

[DDGG05] DAS K., DIAZ-GUTIERREZ P., GOPI M.: Sketching free-form surfaces using network of curves. In *Proc. of Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM '05)* (2005).

[DE03] DRAPER G., EGBERT P.: A gestural interface to free-form deformation. In *Proc. of Graphics Interface 2003* (2003), pp. 112–120.

[DFRS03] DECARLO D., FINKELSTEIN A., RUSINKIEWICZ S., SANTELLA A.: Suggestive contours for conveying shape. *ACM Transactions on Graphics (Proc. of SIGGRAPH '03) 22*, 3 (2003), 848–855.

[DJ03] DEARAUJO B., JORGE J.: Blobmaker: Free-form modelling with variational implicit surfaces. In *Proc. of 12 Encontro Português de Computacão Grafica* (2003), pp. 320–328.

[DJW*06] DECAUDIN P., JULIUS D., WITHER J., BOISSIEUX L., SHEFFER A., CANI M.-P.: Virtual garments: A fully geometric approach for clothing design. *Computer Graphics Forum (Proc. of Eurographics '06) 25*, 3 (2006).

[ECYBE97] EGGLI L., CHING-YAO H., BRUDERLIN B., ELBER G.: Inferring 3d models from freehand sketches and constraints. *Computer-Aided Design 29*, 2 (1997), 101–112.

[ESA07] EITZ M., SORKINE O., ALEXA M.: Sketch-based image deformation. In *Proc. of Vision, Modeling, and Visualization (VMV)* (2007).

[ESU02] E. SAYKOL G. GÜLESIR U. G., ULUSOY O.: *KiMPA: A Kinematics-Based Method for Polygon Approximation*, vol. 2457/2002 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2002, pp. 186–194.

[FBSS04] FLEISCH T., BRUNETTI G., SANTOS P., STORK A.: Stroke-input methods for immersive styling environments. In *Proc. of the International Conference on Shape Modeling and Applications (SMI '04)* (2004), pp. 275–283.

[FMK*03] FUNKHOUSER T., MIN P., KAZHDAN M., CHEN J., HALDERMAN A., DOBKIN D., JACOBS D.: A search engine for 3d models. *ACM Transactions on Graphics (Proc. of SIGGRAPH '03) 22*, 1 (2003), 83–105.

[FRSS04] Fleisch T., Rechel F., Santos P., Stork A.: Constraint stroke-based oversketching for 3d curves. In *Proc. of Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM '04)* (2004).

[Goo] Google, Inc.: Sketchup. sketchup.google.com.

[HACH*04] Hayward V., Astley O. R., Cruz-Hernandez M., Grant D., deLaTorre G.: Haptic interfaces and devices. *Sensor Review 24*, 1 (2004), 16–29.

[HD05] Hammond T., Davis R.: Ladder, a sketching language for user interface developers. *Elsevier Computers and Graphics 28* (2005), 518–532.

[HL07] Hui K., Lai Y.: Generating subdivision surfaces from profile curves. *Computer-Aided Design 39*, 9 (2007), 783–793.

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

[HS97] Hoffman D., Singh M.: Salience of visual parts. *Cognition 63*, 1 (1997), 29–78.

[HU90] Hwang T., Ullman D.: The design capture system: Capturing back-of-the-envelope sketches. *Journal for Engineering Design 1*, 4 (1990), 339–353.

[IH01] Igarashi T., Hughes J. F.: A suggestive interface for 3d drawing. In *Proc. of ACM symposium on User interface software and technology (UIST '01)* (2001), pp. 173–181.

[IH02] Igarashi T., Hughes J. F.: Clothing manipulation. In *Proc. of ACM symposium on User interface software and technology (UIST '02)* (2002), pp. 91–100.

[IH03] Igarashi T., Hughes J. F.: Smooth meshes for sketch-based freeform modeling. In *Proc. of ACM Symposium on Interactive 3D Graphics* (2003), pp. 139–142.

[IMKT97] Igarashi T., Matsuoka S., Kawachiya S., Tanaka H.: Interactive beautification: a technique for rapid geometric design. In *Proc. of ACM symposium on User interface software and technology (UIST '97)* (1997), pp. 105–114.

[IMT99] Igarashi T., Matsuoka S., Tanaka H.: Teddy: A sketching interface for 3d freeform design. In *Proc. of SIGGRAPH'99* (1999), pp. 409–416.

[IOOI05] Ijiri T., Owada S., Okabe M., Igarashi T.: Floral diagrams and inflorescences: interactive flower modeling using botanical structural constraints. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers* (2005), pp. 720–726.

[JJL07] Joseph J. LaViola J.: Sketching and gestures 101. In *ACM SIGGRAPH 2007 courses* (2007), ACM, p. 2.

[JLCW06] Ji Z., Liu L., Chen Z., Wang G.: Easy mesh cutting. *Computer Graphics Forum (Proc. of Eurographics '06) 25*, 3 (2006), 283–291.

[KD82] Kurozumi Y., Davis W.: Polygonal approximation by the minimax method. *Computer Graphics and Image Processing 19* (1982), 248–264.

[KDS06] Kara L., D'Eramo C., Shimada K.: 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).

[KG05] Kho Y., Garland M.: Sketching mesh deformations. In *ACM SI3DG: Symposium on Interactive 3D Graphics and Games 2005* (2005).

[KH06] Karpenko O. A., Hughes J. F.: Smoothsketch: 3d free-form shapes from complex sketches. In *Proc. of SIGGRAPH '06* (2006), pp. 589–598.

[Koe98] Koenig H.: *Modern Computational Methods*. Taylor & Francis, 1998.

[KOG02] Kasturi R., O'Gorman L., Govindaraju V.: Document image analysis: a primer. *Sadhana 27*, 1 (2002), 3–22.

[KS05] Kara L., Stahovich T.: An image-based, trainable symbol recognizer for hand-drawn sketches. *Computers & Graphics 29*, 4 (2005), 501–517.

[KS06] Kara L., Shimada K.: Construction and modification of 3d geometry using a sketch-based interface. In *Proc. of Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM '06)* (2006).

[Lea] LeapFrog Enterprises: Fly fusion pentop computer. http://www.flyworld.com/whatis/index.html.

[LG07] Levet F., Granier X.: Improved skeleton extraction and surface generation for sketch-based modeling. In *Graphics Interface 2007* (2007).

[LKS06] Lee W., Kara L., Stahovich T.: An efficient graph-based symbol recognizer. In *Proc. of Eurographics Workshop on Sketch Based Interfaces and Modeling (SBIM '06)* (2006).

[LLRM00] Long A. C. J., Landay J., Rowe L., Michiels J.: Visual similarity of pen gestures. In *Proc. of the SIGCHI conference on Human factors in computing systems (CHI '00)* (2000), pp. 360–367.

[LM01] Landay J., Myers B.: Sketching interfaces: toward more human interface design. *Computer 34*, 3 (2001), 56–64.

[LZ04] LaViola J., Zeleznik R.: Mathpad2: A system for the creation and exploration of mathematical sketches. *ACM Transactions on Graphics (Proc. of SIGGRAPH '04) 23*, 3 (2004), 432–440.

[MI07] Mori Y., Igarashi T.: Plushie: An interactive design system for plush toys. In *Proc. of SIGGRAPH '07* (2007).

[ML05] MASRY M., LIPSON H.: A sketch-based interface for iterative design and analysis of 3d objects. In *Proc. of Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM '05)* (2005), pp. 109–118.

[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.

[NK06] NAFTEL A., KHALID S.: Motion trajectory learning in the dft-coefficient feature space. In *Proc. of IEEE International Conference on Computer Vision Systems (ICVS '06)* (2006).

[NSACO05] NEALEN A., SORKINE O., ALEXA M., COHEN-OR D.: A sketch-based interface for detail-preserving mesh editing. In *Proc. of SIGGRAPH '05* (2005), pp. 1142–1147.

[OMKK06] ONISHI K., MURAKAMI N., KITAMURA Y., KISHINO F.: *Modeling of Trees with Interactive L-System and 3D Gestures*, vol. 3853/2006 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2006, pp. 222–235.

[OSS07] OLSEN L., SAMAVATI F., SOUSA M. C.: Fast stroke matching by angle quantization. In *Proc. of the First Intl. Conference on Immersive Telecommunications (ImmersCom 2007)* (2007).

[OSSJ05] OLSEN L., SAMAVATI F., SOUSA M. C., JORGE J.: Sketch-based mesh augmentation. In *Proc. of the 2nd Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM)* (2005).

[PF06] PRADOS E., FAUGERAS O.: Shape from shading. In *Handbook of Mathematical Models in Computer Vision*, N. Paragios Y. C., Faugeras O., (Eds.). Springer, 2006, ch. 23, pp. 375–388.

[Pie87] PIEGL L.: Interactive data interpolation by rational bezier curves. *IEEE Computer Graphics and Applications 7* (1987), 45–58.

[Pix] PIXOLOGIC, INC.: Zbrush. www.pixologic.com/home.php.

[PJBF00] PEREIRA J. P., JORGE J. A., BRANCO V., FERREIRA F. N.: Towards calligraphic interfaces: Sketching 3d scenes with gestures and context icons. In *Proc. of WSCG '00* (2000).

[PSNW07] PUSCH R., SAMAVATI F., NASRI A., WYVILL B.: Improving the sketch-based interface: Forming curves from many small strokes. In *Proc. of Computer Graphics International (CGI 2007)* (2007).

[Pug92] PUGH D.: Designing solid objects using interactive sketch interpretation. In *Proc. of Symposium on Interactive 3D graphics (I3D '92)* (1992), pp. 117–126.

[Ras00] RASKIN J.: *The Humane Interface: New Directions for Designing Interactive Systems*. Addison-Wesley Professional, 2000.

[Rog89] ROGERS D. F.: Constrained b-spline curve and surface fitting. *Comput. Aided Design 21*, 10 (1989), 641–648.

[RSW*07] ROSE K., SHEFFER A., WITHER J., CANI M.-P., THIBERT B.: Developable surfaces from arbitrary sketched boundaries. In *Proc of Eurographics Symposium on Geometry Processing (SGP '07)* (2007).

[Rub91] RUBINE D.: Specifying gestures by example. In *Proc. of SIGGRAPH '91* (1991), pp. 329–337.

[SC04] SHESH A., CHEN B.: SMARTPAPER: An interactive and user friendly sketching system. In *Proc. of Eurographics 2004* (2004).

[SD04] SEZGIN T. M., 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.

[SG98] SCHWEIKARDT E., GROSS M. D.: Digital clay: Deriving digital models from freehand sketches. In *Digital Design Studios: Do Computers Make A Difference? (ACADIA '98)* (1998), pp. 202–211.

[SI07] SHIN H., IGARASHI T.: Magic canvas: interactive design of a 3-d scene prototype from freehand sketches. In *Proc. of Graphics Interface (GI '07)* (2007), pp. 63–70.

[SMA00] SAMAVATI F., MAHDAVI-AMIRI N.: A filtered b-spline model of scanned digital images. *Journal of Science 10* (2000), 258–264.

[SSD01] SEZGIN T. M., STAHOVICH T., DAVIS R.: Sketch based interfaces: early processing for sketch understanding. In *Proc. of Workshop on Perceptive user interfaces (PUI '01)* (2001), pp. 1–8.

[SSS06] SEVERN A., SAMAVATI F., , SOUSA M. C.: Transformation strokes. In *Proc. of Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM '06)* (2006).

[Sut63] SUTHERLAND I.: Sketchpad: A man-machine graphical communication system. In *AFIPS Conference Proceedings 23* (1963), pp. 323–328.

[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).

[SWSJ05] SCHMIDT R., WYVILL B., SOUSA M. C., JORGE J. A.: Shapeshop: sketch-based solid modeling with blobtrees. In *Proc. of Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM '05)* (2005).

[TBSR04] TSANG S., BALAKRISHNAN R., SINGH K., RANJAN A.: A suggestive interface for image guided 3d sketching. In *Proc. of the SIGCHI conference on Human factors in computing systems (CHI '04)* (2004), ACM, pp. 591–598.

[TBvdP04] THORNE M., BURKE D., VAN DE PANNE M.: Motion doodles: an interface for sketching character motion. In *Proc. of SIGGRAPH '04* (2004), pp. 424–431.

[TO99] TURK G., O'BRIEN J.: *Variational Implicit Surfaces*. Tech. rep., Georgia Institute of Technology, 1999.

[TZF04] TAI C.-L., ZHANG H., FONG J. C.-K.: Prototype modeling from sketched silhouettes based on convolution surfaces. *Computer Graphics Forum 23* (2004), 71–83.

[Vel01] VELTKAMP R.: Shape matching: similarity measures and algorithms. In *Proc. of Intl. Conference on Shape Modeling and Applications (SMI '01)* (2001), pp. 188–197.

[VMS05] VARLEY P., MARTIN R., SUZUKIA H.: Frontal geometry from sketches of engineering objects: is line labelling necessary? *Computer-Aided Design 37* (2005), 1285–1307.

[VTMS04] VARLEY P., TAKAHASHI Y., MITANI J., SUZUKI H.: A two-stage approach for interpreting line drawings of curved objects. In *Proc. of Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM '04)* (2004).

[WFJ*05] WYVILL B., FOSTER K., JEPP P., SCHMIDT R., SOUSA M. C., JORGE J.: Sketch based construction and rendering of implicit models. In *Proc. of Eurographics Workshop on Computational Aesthetics in Graphics, Visualization and Imaging* (2005).

[Wil94] WILLIAMS L. R.: *Perceptual Completion of Occluded Surfaces*. PhD thesis, University of Massachusetts, 1994.

[WM07] WANG H., MARKOSIAN L.: Free-form sketch. In *Proc. of Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM '07)* (2007).

[YSvdP05] YANG C., SHARON D., VAN DE PANNE M.: Sketch-based modeling of parameterized objects. In *Proc. of Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM '05)* (2005).

[Yu03] YU B.: Recognition of freehand sketches using mean shift. In *Proc. of Intl. conference on Intelligent user interfaces (IUI '03)* (2003), pp. 204–210.

[YXN*05] YUAN X., XU H., NGUYEN M. X., SHESH A., CHEN B.: Sketch-based segmentation of scanned outdoor environment models. In *Proc. of Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM '05)* (2005), pp. 19–26.

[ZG04] ZELINKA S., GARLAND M.: Mesh modeling with curve analogies. In *Proc. of Pacific Graphics '04* (2004), pp. 94–98.

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

[ZNA07] ZIMMERMANN J., NEALEN A., ALEXA M.: Silsketch: Automated sketch-based editing of surface meshes. In *Proc. of Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM '07)* (2007).

[ZS07] ZAKARIA M., SHUKRI S. R. M.: *A Sketch-and-Spray Interface for Modeling Trees*, vol. 4569/2007 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2007, pp. 23–35.

[ZTCS99] ZHANG R., TSAI P.-S., CRYER J., SHAH M.: Shape-from-shading: a survey. *IEEE Transactions on Pattern Matching and Machine Intelligence 21*, 8 (1999), 690–706.