Computers & Graphics 33 (2009) 85–103

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

Sketch-based modeling: A survey

Luke Olsen [a,*], Faramarz F. Samavati [a], Mario Costa Sousa [a], Joaquim A. Jorge [b]

[a] Department of Computer Science, University of Calgary, Calgary, AB, Canada
[b] Departamento de Engenharia Informática, Instituto Superior Técnico, Lisbon, Portugal

ARTICLE INFO

Article history:
Received 22 May 2008
Received in revised form
12 September 2008
Accepted 30 September 2008

Keywords:
Sketch-based modeling
Interface design
Perception

ABSTRACT

User interfaces in modeling have traditionally followed the WIMP (Window, Icon, Menu, Pointer) paradigm. Though functional and very powerful, they can also be cumbersome and daunting to a novice user, and creating a complex model requires considerable 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 sketches—hasty freehand drawings—to be used in the modeling process, from rough model creation through to fine detail construction. Mapping a 2D sketch to a 3D modeling operation is a difficult task, rife with ambiguity. To wit, we present a categorization based on how a 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. Additionally, in this paper we introduce a survey of sketch-based interfaces focused on 3D geometric modeling applications. The canonical and recent works are presented and classified, including techniques for sketch acquisition, filtering, and interpretation. The survey also provides an overview of some specific applications of SBIM and a discussion of important challenges and open problems for researchers to tackle in the coming years.

1. Introduction

"Everyone can draw" may not be a strictly accurate statement, but there seems to be a universal capacity for visual communication. It is why primitive men told stories through hieroglyphs, and why every meeting room has a whiteboard adorning the wall. Sketching is a natural way to communicate ideas quickly: with only a few pencil strokes, complex shapes can be evoked in viewers.

In computer modeling, sketching on paper is often used in the early prototyping stages of a design, before the depicted design is manually converted into a 3D model by a trained 3D artist (Fig. 1). Because of this, model creation is a major bottleneck in production pipelines, requiring human effort to create the complex and diverse shapes and intricate inter-relationships. Current high-end modeling systems such as Maya [1], SolidWorks [2], and CATIA [3] 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.

A recent research direction in modeling interfaces is to automate or assist the sketch-to-3D translation process. This trend, known as sketch-based interfaces for modeling (SBIM), is motivated by the ease of sketching and the ability of human viewers to imbue so much meaning into a sketch. The guiding research question, then, is *How can computers understand and interpret sketches in three dimensions*?

Scientists have been pondering this question for many decades. The human visual system is able to understand complex shapes from single images or sketches, even from simple line drawings devoid of any shading cues, but the effortlessness of perception makes it a difficult process to formalize. 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 artificial intelligence (AI). Though research efforts have thus far been driven primarily by computer modeling researchers, the emergence of powerful commodity computer hardware and cooperative research is pushing the field to exciting results.

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 sketching. This would allow users to construct and edit models

* Corresponding author.
E-mail addresses: olsenl@cpsc.ucalgary.ca (L. Olsen), samavati@cpsc.ucalgary.ca (F.F. Samavati), mario@cpsc.ucalgary.ca (M.C. Sousa), jorgej@ist.utl.pt (J.A. Jorge).

0097-8493/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cag.2008.09.013

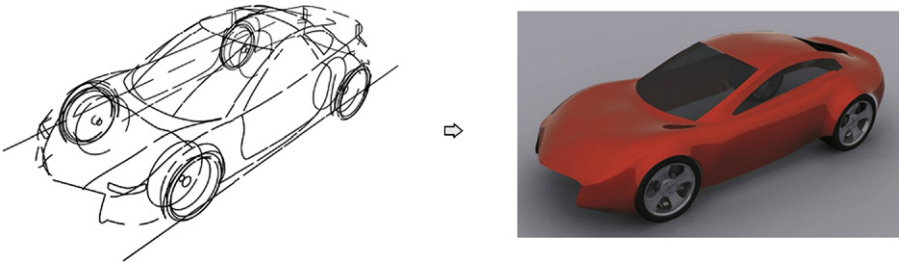

Fig. 1. Creating a 3D model from a sketch requires complex software and an expert user to understand what the sketch depicts and translate that to 3D. SBIM attempts to simplify or automate the process.

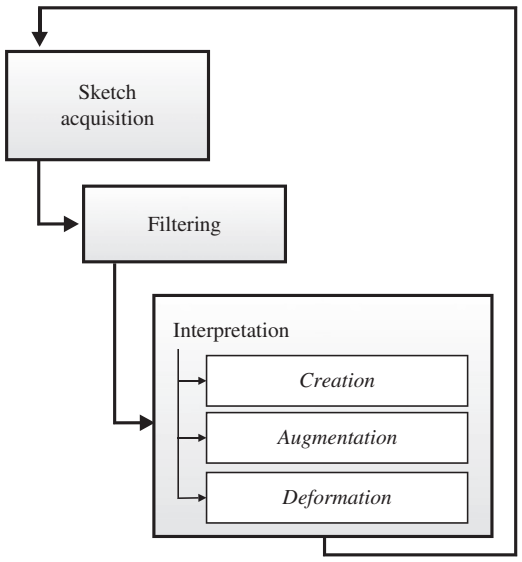

Fig. 2. The SBIM pipeline: after acquiring and filtering the input sketch, the sketch can be interpreted as an operation in 3D.

in a progressive way, from an initial concept to a detailed and accurate final model. While there is a lot of research interest in SBIM, it has not yet gained full support in industry because SBIM systems are not complete replacements of all functionality. Commercial modeling packages that support freehand sketches include Archipelis Designer [4] and Sunny3D [5], while programs like ZBrush [6] and MudBox [7] allow modelers to paint details onto a surface with brush strokes.

Sketch-based interfaces date back to Sutherland's SketchPad system [8], which used a light-pen input device to directly create and 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, and represent the resulting object. Where modern systems primarily improve upon SketchPad is in automation: a SketchPad user must explicitly specify all geometry, whereas modern systems can leverage better algorithms and increased computing power to automatically infer 3D shapes from 2D input. Sketch-based techniques have found utility in a wide range of modeling tasks, some of which are discussed in Section 7.

In this paper (extended and thoroughly revised from [9]), we survey the state of sketch-based interfaces for 3D geometric modeling applications. The main challenge in SBIM is sketch interpretation, of which we identify 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. 2. The first stage is to acquire a sketch from the user (Section 3), followed by a filtering stage to clean and transform the sketch (Section 4). In the final stage of the pipeline, the sketch is interpreted as the specification of or operation on a 3D model (Section 5).

This survey is organized as follows. After briefly discussing the role of perception in SBIM (Section 2), each stage of the SBIM pipeline is described in detail in Sections 3–5, including a discussion of two critical areas in application design: surface representation (Section 5.4) and interface design (Section 6). We conclude with a discussion of challenges and open problems (Section 8).

2. The role of perception

The human visual system is vastly complex, yet taken for granted because it works so effortlessly throughout our lives. While a thorough discussion of cognitive science is beyond the scope of this paper and our expertise, notions from this area have already influenced the design of SBIM systems (explicitly and implicitly) and will no doubt continue to do so in the future. After all, a person's perception of shape informs how they draw: perception and communication are dual sides of our visual intelligence.

The fundamental problem that our visual system must deal with is that "the image at the eye has countless possible interpretations" [10]. 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. Fig. 3 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.

So how do we interpret Fig. 3 as a cube, rather than the infinitely many other choices? We might observe that of the three plausible models shown in Fig. 3 whose contour lines project to a cube-like object, only the cube itself conforms to the our visual rules. Hoffman [10] calls the other candidates "accidental views", since any slight change in viewpoint would reveal them to be non-cubes. Put another way, accidental views are *unstable*, but most views of an object are stable under slight changes. Our visual system, therefore, heavily favors the stable interpretations.

Now consider an artist who wants to sketch 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

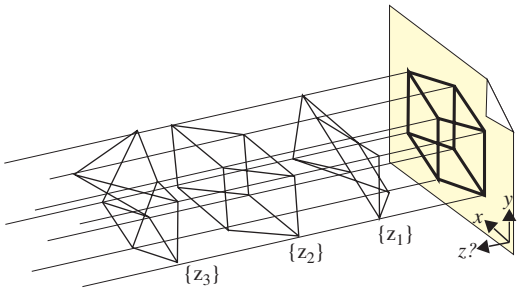

Fig. 3. Ambiguous interpretation of a 2D sketch in 3D: there are infinitely many objects that project to the same 2D input. Reproduced with permission from [11].

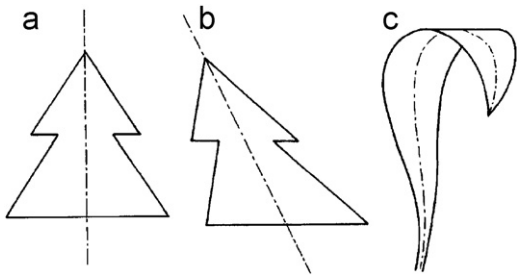

Fig. 4. Three types of symmetry: (a) real; (b) skewed; and (c) generalized (adapted from [14]).

ways to reconstruct a drawing, "your visual system is biased. It constructs only those 3D worlds that conform to its rules ... [and prunes] the possible depths you can see from infinity down to one or two" [10].

Visual rules allow us to make sense of images we have never seen before, but they are 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 [12], imbuing them with unseen complexity. For example, when shown an image or even just a silhouette of a sports car, 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 [13]. *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.

Symmetry is another important property of many objects; being able to detect or infer symmetry in a 2D form provides invaluable information for reconstruction of the 3D form. This includes not just "true" symmetry, but also arbitrary projections of symmetric 3D objects. Tanaka et al. [14] discuss three kinds of symmetry: real symmetry, in which the axis of symmetry is a line in the image plane; skewed symmetry, in which the axis is a line passing through the image plane; and generalized symmetry, in which the axis is a free-form line in 3D. Fig. 4 illustrates these ideas.

The notions of perception can help us to understand the challenges and design decisions made in SBIM. As we will see in Section 5, the ways in which SBIM systems deal with the ambiguity of single images relate to visual memory and rule systems. And, as discussed in Section 8, understanding our own perception also suggests ways to improve the software-based perception required for SBIM.

3. Sketch acquisition

Let us now return to the SBIM pipeline. The most basic operation shared between all SBIM systems is, of course, obtaining a sketch from the user. The key characteristic of a sketch-based input device is that it allows freehand input. The standard mouse fits this definition, but 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. 5) are particularly suited to natural interaction.

Real 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 aspects 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. Some devices also report the pen orientation. Haptic devices [15] 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 [16] and even 3D virtual reality devices [17].

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 feedback force, decreasing the device's pen-like attributes. As such hardware becomes more compact, less costly, and truly immersive, their 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 [18], but to date there has been little investigation in this direction for 3D reconstruction tasks.

Off-line scanning of sketches is another option, but such an approach would be more akin to the single-image recognition problem in computer vision. This might work in a domain-specific

Fig. 5. Input to a sketch-based system is acquired from pen-based or free-form devices such as a tablet display. Pictured: Wacom Cintiq (www.wacom.com).

application, such as scanning architectural drawings. For general modeling tasks, however, this approach is very difficult and currently lacking robust solutions. Interactive systems are generally more feasible, providing more information to the application (drawing order, speed, etc.) and constant feedback to the user. 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 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. 6b). Note that the samples are spaced irregularly, depending 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 [19,20].

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 one or more strokes. The basic stroke information can be augmented by additional information, such as pressure or pen orientation, depending on the target application and available hardware.

Due to the large body of work in image processing, some SBIM applications choose to use an image-based stroke representation, in which the stroke is approximated with a pixel grid (Fig. 6c). As the input device is moved over the virtual paper, it leaves an "ink trail" behind. An image-based representation has the advantage of fixed memory usage, as well as automatic blending of multiple strokes. However, the temporal nature of sketching is lost, along with any auxiliary information that is available.

The notion of a "drawing canvas" [21,22] is used in SBIM systems to embed a sketch into 3D world coordinates. The simplest way to define a canvas is to specify a particular plane, such as the x–y plane or a user-specified 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 (though the depth is still unconstrained). A unique, symmetric 3D curve may be recoverable by assuming the input strokes are plane-symmetric and inverting the viewing projection [23]. A final variation is to project the sketch onto an existing 3D model based on the current viewpoint (Fig. 7).

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 upon which the user draws [24–27]. The image can also be used for assisted sketching, where the input strokes are "snapped" to edges in the image [25].

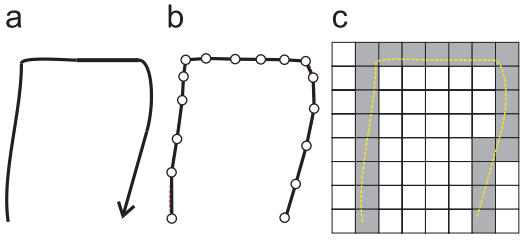

Fig. 6. An input stroke (a) is provided to the application as (b) a sequence of point samples; (c) some applications choose to use an image-based representation.

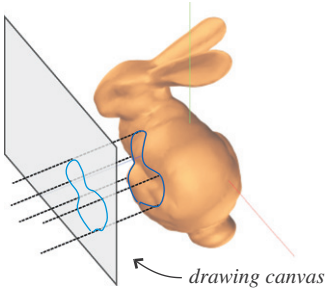

Fig. 7. Sketches are embedded into 3D by projecting onto a drawing canvas, or perhaps onto existing geometry.

4. Sketch filtering

Before attempting to interpret a sketch, it is necessary to perform some filtering. One motivating factor is that the input will invariably contain some noisy or erroneous samples. Sezgin and Davis [28] 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 100 Hz only few points per inch can be sampled" [28].

Even with careful drawing, device errors and sampling issues remain. Therefore, the input to a sketch-based 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 tasks easier. Below we present some commonly used filtering methods in SBIM.

4.1. Resampling and smoothing

The spacing between samples in a raw input stroke varies among devices as well as with the drawing speed of the user. One way to reduce the noise in an input stroke is to *resample* the data. Resampling can be done on-the-fly by discarding any sample within a threshold distance of earlier samples, and by interpolating between samples separated by more than a threshold. It can also be done after the stroke is finished. Depending on the needs of the application, linear or smooth interpolation can be used. See Fig. 8a.

An extreme form of resampling is *polyline* (*or polygon*) *approximation*, which reduces the complexity of a stroke to just a few samples (Fig. 8b). For example, Teddy [29] 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. These approaches are best suited to smooth inputs, and otherwise may give unsatisfactory results because their sample distribution is not based on local stroke features such as corners.

In the general case, a robust 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 [30], for instance, minimizes the maximum distance of any point to the straight-line approximating line. There are rigorous computational geometry approaches [31] for tackling this problem, but they are intended to operate on

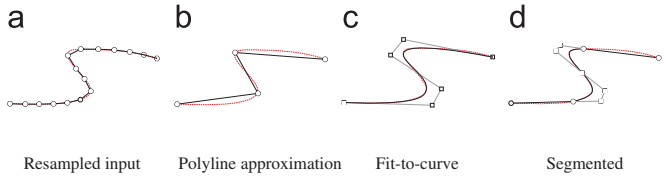

Fig. 8. Filtering operations: (a) smooth uniform resampling; (b) coarse polyline approximation; (c) fit to a spline curve; (d) segmented into straight and curved sections. In each figure, circles denote endpoints of straight-line segments, while squares represent curve control points.

positional information; with sketched input, there is additional temporal information that can be used to identify perceptually important points in a stroke, such as corners (gradual direction changes) and darts (abrupt changes). For example, Saga [19] uses drawing speed to identify "partition points," prompting the user to confirm uncertain partitions; Sezgin et al. [20] use curvature (maxima) and drawing speed (minima) to identify corner points.

Even after resampling, there will be some noisy samples. Smoothing operators can reduce noise, at the expense of possibly obscuring real discontinuities in the input. Some techniques include applying a local averaging filter to each sample (i.e. replace each sample with the average of neighboring points) [32] or Gaussian filtering (center-weighted averaging) [33].

4.2. Fitting

After resampling or smoothing, a sketch still contains a large number of sample points with little meaning. Fitting the sketch to other representations has the dual advantages of simplifying the input and making it easier to compare against each other. In fact, curve fitting is necessary in some SBIM systems in which the reconstructed surface is based on constructive curves (such as a surface of revolution).

Curve fitting is a simplification approach that yields lower errors relative to polygon approximation, at the cost of more computation. Least-squares polynomial fitting [34] is an option, but parametric forms such as Bézier [35,36] and B-spline [37–39] curves are preferable in graphics. Fig. 8c illustrates spline curve fitting.

More recently, subdivision and variational implicit curves have been employed in SBIM systems. Alexe et al. [32] use a Haar wavelet transformation to get a multi-scale stroke representation. Cherlin et al. [40] fit a subdivision curve to a stroke by applying reverse Chaikin subdivision to the raw stroke samples, effectively de-noising the data. Schmidt et al. [16] infer geometric constraints from the input sketch to fit a variational implicit curve.

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 [20,36,41,42]. Sezgin et al. [20], 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. 8d.

Yu [43] argues that because splines are difficult to compare at a high level, it is better to fit primitive shapes such as squares, ellipses, and arcs. This is the approach taken by Saga [19] for fitting shapes in a freehand CAD system, although his system required the user to verify and correct the labelings. The fit-to-primitive approach has been used in several SBIM systems [44,45].

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. [46]) 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-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 [46–48] or as a batch process after a sketch is finished [49], although offline processing is more difficult due to constraint propagation.

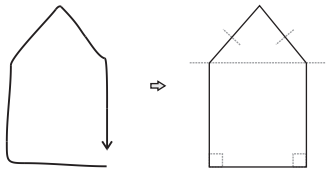

Fig. 9. Beautification infers global geometric constraints between strokes, such as parallelism, symmetry, and perpendicularity.

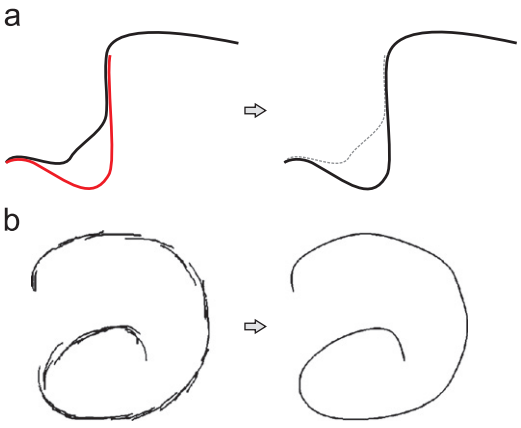

Fig. 10. (a) Oversketching is a quick and effective way to interactively correct a sketch; (b) oversketched strokes can be blended in a batch process after sketching is complete (reproduced with permission from [50]).

4.3. Oversketching

Fitting approaches are most suitable for applications where precision is desirable or assumed, such as engineering drawings. In applications that wish to support free-form sketching and make few assumptions about the user's intention, however, fitting a "nice" representation may inadvertently destroy some important feature of the sketch. In this case, the user has to be able to sketch exactly what they want, and correct themselves when a mistake is made.

Oversketching is a commonly used interface element, for allowing a user to carefully sketch over the offending region when a mistake is made during sketching. 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. 10a). Oversketching can be supported in 2D before interpretation [20,51,52], the system can retain the original sketch for constrained 3D oversketching later in the pipeline (see Section 5.3).

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. 10b). Some SBIM systems allow for this type of multi-stroke input, automatically blending the strokes together [27,50,53,54]. In a stroke-space approach, the geometric relationships between strokes are used to blend them; for example, Pusch et al. [50]

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" as the user draws. Finally, a semi-automatic approach may be used, in which the user identifies which strokes can be blended together [16].

5. Sketch interpretation in SBIM

After a sketch has been sufficiently filtered, the final stage of the pipeline is interpret the sketch, mapping it to a 3D modeling operation. We use the term "interpret" in the literal sense, i.e. to interpret a sketch is to assign meaning to it. Unlike a command selected from a menu, freehand input is inherently ambiguous and open to multiple interpretations. What has the user intended to draw? Is the input valid and consistent? How can the sketch be mapped to a modeling operation? These are the questions that an SBIM system needs to answer.

There are many different approaches to answering these questions, but some common elements can be identified. We propose a categorization of SBIM systems based on the types of modeling operations considered. The most important category includes systems that *create* fully 3D models automatically from input sketches (Section 5.1). Other important tasks include using input strokes to *augment* existing models with details (Section 5.2) and to *deform* an existing model (Section 5.3). There are a variety of surface representations that are used to model the 3D objects, each having strengths and weaknesses (Section 5.4). Finally, a carefully designed interface is necessary to choose the correct interpretation at the correct time, as discussed in Section 6.

A complete SBIM system can be used in all aspects of the modeling pipeline, from prototyping to fine-tuning, by providing each type of interpretation. As with any subjective categorization, there are some examples that do not fit neatly into a particular category. In the following subsections, we offer a category-centric view of sketch interpretation and discuss the relevant works within. Table 1 presents a system-centric view of the major works in SBIM that support at least model creation, from early (SKETCH [44], Teddy [29]) to state-of-the-art (SmoothSketch [55], FiberMesh [56]) 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

A model creation system attempts to reconstruct a 3D model from the 2D sketched input. We divide the gamut of creation systems into two categories, *evocative*, and *constructive*. The distinction is that in a constructive system, the input strokes are somehow mapped directly to the output model, while in an evocative system a sketch is used to instantiate built-in model types similar to the input.

This is just one of the possible classifications one could apply to SBIM, but one that neatly aligns with the classical distinction between reconstruction and recognition. Evocative systems first recognize a sketch against a set of templates, 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, evocative systems are akin to visual memory, whereas constructive systems are more rule-based.

Because evocative 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 evocative systems. This is mostly embodied by evocative systems that deform the template objects to match the input sketch [26], or constructive systems that exploit domain-specific knowledge.

5.1.1. Evocative systems

Evocative systems are characterized by the fact that they have some "memory" of 3D shapes built in, which guides their interpretation of input sketches. If a system is designed for character creation, for example, the shape memory can be chosen to identify which parts of a sketch correspond to a head, torso, and so forth. Then the conversion to 3D is much easier, because the shapes and relative proportions of each part is known a priori.

Within this category, we identify two main approaches: iconic systems, and template retrieval systems.

5.1.1.1. Iconic systems. In this approach, the system extrapolates a final 3D shape based on only a few iconic strokes [44,48,60]. A classical example is the SKETCH system of Zeleznik et al. [44], 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. 11). Iconic systems are not far removed from WIMP systems, in the sense that the stroke groups are used to initiate commands rather than buttons or menus.

The GIDeS system of Jorge et al. [45] follows a similar design, providing templates for a broader range of primitive objects, as well as some parameterized templates for engineering design (see Section 5.1.2).

The Chateau system of Igarashi and Hughes [47] also extrapolates shape from a few strokes, although it is not truly free form. 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.

5.1.1.2. Template retrieval systems. The second main approach in evocative systems is to retrieve template objects from a database of template objects [26,63,74,78]. Rather than simple primitive objects, the templates are more complete and complex objects. And from the user's perspective, they must provide a complete and meaningful sketch of the desired object, rather than just a few evocative strokes.

This approach is more extensible than extrapolation, because 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 more complex, it may be impossible to attain a specific result by combining the template objects.

The increased complexity on both the input and output sides is reflected in the underlying matching algorithms. A retrieval-based system faces the problem of matching 2D sketches to 3D templates. To evaluate their similarity in 3D would require reconstruction of the sketch, which is the ultimate problem to be solved. Therefore, comparison is typically done by extracting a 2D form from the 3D template object (although other approaches, such as graph matching [78] have been proposed).

Funkhouser et al. [63] use the projected contour from 13 different viewpoints to define the shape descriptor of an object, based on their observations that "people tend to sketch … [from]

Table 1
Taxonomy of sketch-based modeling systems, including creation mode (Section 5.1), surface representation (Section 5.4), editing operations (Sections 5.2 and 5.3), and interface type (Section 6).

		Creation method					Surface type				Editing operations						Interface	
		Iconic	Template	Engineering	Free-form	Multi-view	Parametric	Mesh	Implicit	Fair	Surficial Aug.	Additive Ang.	Cut/ tunnel	Oversketch	Bend/ twist	CSG/ Boolean	Suggestive	Gestural
1989	Tanaka et al. [14]				•		•											
1992	Kanai et al. [57]				•		•							•				
1996	Lipson [58]			•				•										
	SKETCH [44]	•						•									•	
1997	Quicksketch [36]				•		•	•			•				•		•	
1998	Digital Clay [49]			•				•										
1999	Teddy [29,59]				•			•			•		•		•			•
2000	GIDeS [60]	•		•		•		•								•		•
2001	Chateau [47]	•						•									•	
2002	Karpenko et al. [61]				•				•			•				•		
	3d Sketch [62]			•				•										
2003	CIGRO [48]			•				•										•
	BlobMaker [51]				•				•			•		•		•	•	•
	3D Search [63]		•					•									•	
	Piquer et al. [64]			•				•										
2004	Alexe et al. [32]				•				•			•			•	•		
	Karpenko et al. [65]				•				•					•				
	Smartpaper [53]			•				•				•	•			•		•
	ConvMo [66]				•			•				•				•		
	Ribald [67]			•				•										
2005	Cherlin et al. [40]				•		•							•	•			
	Das et al. [21]					•		•										
	Masry and Lipson [11]			•				•						•				
	ShapeShop [16]				•				•		•	•	•			•	•	•
	Varley et al. [68]			•				•										
	Yang et al. [26]		•				•				•							
2006	Kara et al. [27]		•				•				•			•				
	SmoothSketch [55]				•			•										
	Kara and Shimada [39]					•		•						•				•
	Owada et al. [69]				•				•			•	•			•		•
2007	Cordier and Seo [70]				•				•									
	Hui and Lai [71]					•		•										
	Plushie [72]				•			•				•	•					•
	FiberMesh [56]				•	•				•	•	•	•	•				•
	Rose et al. [73]					•				•							•	
	Magic Canvas [74]		•					•									•	•
	Wang and Markosian [75]				•		•	•						•		•		•
2008	ShapeShop v2 [76]			•	•			•			•	•	•			•	•	•
	Lee and Funkhouser [77]		•					•				•				•	•	

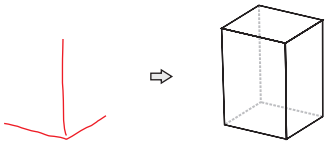

Fig. 11. An iconic evocative-stroke system extrapolates a 3D form from only a few evocative strokes.

a remarkably consistent set of view directions". Object templates are created by applying image-based transformations to each contour, extracting a fixed-length, rotation-invariant feature. Input sketches are then matched to an object by applying the same image transformations and comparing against the stored templates. Also, to improve the recognition rate the user can sketch up to three different views of an object.

This problem—matching 2D shapes—has been studied in computer vision, mostly comparing silhouettes and contours via image-based methods (see Veltkamp [79] for a good introduction). Image-based techniques in SBIM discard the potentially important temporal and auxiliary (pressure, etc.) information available in a sketch, but benefit from the large body of work in shape matching. Funkouser et al. [63] also argue for the use of image-based matching since it allows the user to provide "fragmented sketch marks" (as opposed to some stroke-based systems that require long continuous strokes). This is less of a problem in light of recent work in batch oversketching to blend fragmented sketches into a single contour.

Shin and Igarashi's Magic Canvas system [74] uses template retrieval for scene construction (Fig. 12). They also extract several (16) contours from each template object, but use a Fourier-based method for sketch 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 sub-sketches within the scene, and also to choose appropriate objects among several candidates.

Yang et al. [26] propose a similar template-based system, but rather than mesh-based templates, they use procedurally described models. Instead of having a mug's 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.

A recent approach by Lee and Funkhouser [77] diverges from the concept of template models and moves toward template *parts*; for example, rather than a template for an entire airplane, the system contains template for wings, engines, missiles, and so on. Using a sketch-based interface, a user can add parts to an existing model by sketching the contour of a part in its approximate location on the model. The system finds matching parts and after the user selects an appropriate match, the part is composited into the existing model, automatically placed relative to other elements.

5.1.2. Constructive systems

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. 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: mechanical objects, smooth objects, and objects drawn from multiple viewpoints.

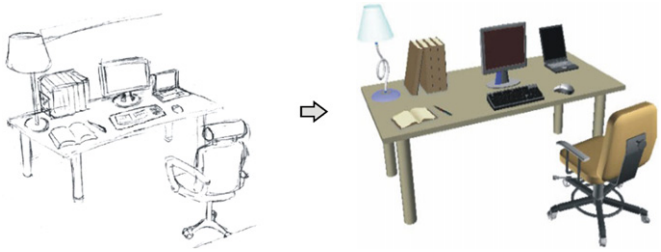

Fig. 12. A template retrieval system matches sketches to 3D models, useful for applications such as scene construction. Reproduced with permission from [74].

5.1.2.1. Engineering design systems. Our visual system can reconstruct mechanical (hard-edged) and smooth objects—or anything in between—with equal aplomb. Sketch-based modeling applications are typically targeted toward one or the other, however, because choosing either a smooth or non-smooth interpretation at the design level reduces the number of possible interpretations of a sketch.

The design and specification of engineered (i.e. mostly planar) objects is an important industrial application of computer modeling. As such, it attracted attention early in the life of SBIM [80]. The optimization-based approach of Lipson and Shpitalni [58] encapsulates many of the techniques seen in later work. Each input stroke is assumed to represent an edge of a 3D wireframe model, and each coincident endpoint a vertex in the model. The sketch is also assumed to represent the object in a parallel projection. These constraints place little burden on the user, but greatly simplify the system. After detecting important relationships in the 2D sketch graph—planarity, corners, isometries, orthogonality, and so forth—reconstruction is performed by optimizing a linear equation in which the depth of each vertex are the unknowns.

Reconstruction of 3D geometry from line drawings has been studied in computer vision for some time. Line labeling [81] is an algorithm for classifying line segments in an image as either concave, convex, or contour edges, which define constraints on the geometry for reconstruction. It is possible, of course, to apply such algorithms directly on sketched input by using an stroke-based representation.

A difficult task in line drawing reconstruction is identifying the locations of vertices, corners, and edges the object. In an interactive system, this can be determined as the user draws the strokes, after which reconstruction can be done in a "batch" process [11,17,49,68].

Symmetry is a common and often desirable property of engineered objects. Though it is not trivial to detect, knowledge of symmetry can be exploited during reconstruction to reduce the search space [62,64].

A limitation of line-labeling approaches is that they have limited support for curved segments, although some recent systems have supported this. Varley et al. [67] 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 [11] also use a two-stage approach, but theirs is hidden from the user: the system automatically extracts a straight-line representation via segmentation.

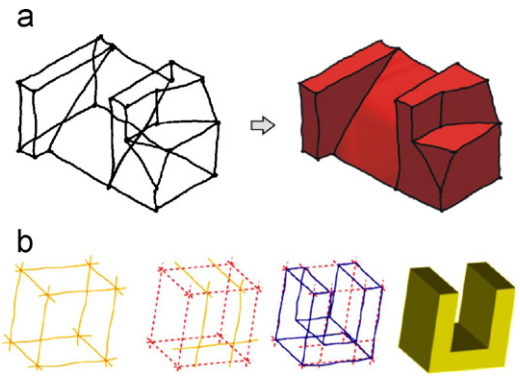

Fig. 13. Engineering design systems exploit domain-specific knowledge to reconstruct quite complex sketches: (a) a batch reconstruction system (reproduced from [11]); (b) an interactive system (adapted from [48]).

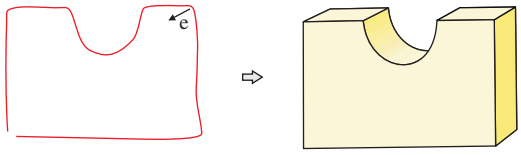

Fig. 14. Extrusion is a simple method for reconstructing a contour, by sweeping it along an extrusion vector *e*.

An alternative to batch systems is to interactively reconstruct the object as the user sketches (Fig. 13). This allows the user to immediately see the result and possibly correct or refine it, and also allows the system to employ more simple reconstruction rules. The most common approach is *extrusion*, a term for creating a surface by "pushing" a profile curve through space along some vector (or curve) [16,36,53,60,75]; see Fig. 14 for an illustration. This technique is well-suited to creating models with hard edges, such as cubes (extruded from a square) and cylinders (from a circle).

The extrusion approach overlaps somewhat with evocative systems, since the user only needs to sketch the profile curve and extrusion vector. 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.

5.1.2.2. Free-form design systems. Though some engineering design systems support curved strokes, reconstruction is still based on a straight-line representation. Reconstructing smooth, natural objects requires a different approach.

It has been observed that our visual system prefers to interpret smooth line drawings as 3D contours [10]. Accordingly, the majority of constructive SBIM systems choose to interpret strokes as contour lines [16,29,32,36,40,55,56,66]. (The *contour* is 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 (Fig. 17). The contour includes not only the silhouette outline, but also reveals interior features like the chin and nose in the example.)

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.

Skeleton-based approaches are a prevalent method for creating a 3D model from a contour sketch [16,29,32,36,40,51,66,72]. The skeleton is defined as the line from which the closest contour

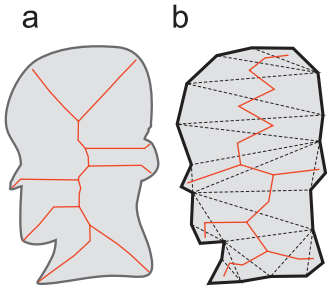

Fig. 15. The skeleton of a contour is often used to create a smooth 3D object: (a) the true skeleton; (b) the approximated skeleton (chordal axis) from Delaunay triangulation.

points are equidistant (Fig. 15), providing a distance field that helps to determine a surface in 3D unambiguously (such that the distance from surface to skeleton is related to the distance of contour points to the skeleton).

Finding an accurate skeleton can be expensive, but can be approximated from the Delaunay triangulation (DT) [82] of a closed polygon by connecting the center points of adjacent non-boundary triangles (Fig. 15); this is known as the chordal axis transform. Recently, Levet and Granier [83] have proposed a skeleton extraction method that yields a smoother skeleton with fewer spurious branches.

How can the skeleton be used to generate a 3D model? There are many approaches, depending on the complexity of the skeleton's structure. The simplest non-trivial skeleton is a straight line. In a symmetric sketch, the skeleton is a straight line aligned with the axis of symmetry. To generate a surface, the sketch can be rotated around the skeleton, creating a surface of revolution [16,36]. A single stroke can also specify the contour, with either a fixed or user-sketched rotation axis to define the surface.

Cherlin et al. [40] extend this idea to a generalized surface of revolution, in which the skeleton is given by the medial axis between two strokes (the authors refer to this construction as *rotational blending surfaces*); see Fig. 16a. Their system also allows the user to provide a third stroke that defines a free-form cross-section, increasing the expressiveness of this construction.

These constructions assume that the input curves lie in the same (drawing) plane, and generate objects with symmetry about this plane. A more challenging approach is to view the input strokes as the projection of possibly symmetric 3D curves—that is, the drawn strokes exhibit skewed or generalized symmetry. In an early approach, Tanaka et al. [14] assume that two input strokes are symmetric in 3D, and determine the axis of symmetry with some additional user input to identify symmetric vertices on each stroke; they then reconstruct the surface as a general cylinder connecting the strokes. Kanai et al. [57,84] later proposed a more robust symmetry-detecting sketch system capable of detecting and reconciling symmetry from multiple views; B-spline patches are used to reconstruct the surface.

An unfortunate aspect of these parametric constructions is the limited topology. The resulting object can always be parameterized over a 2D plane, and the skeletons contain no branches. For contours with branching skeletons, a more robust method is required.

For simple (i.e. non-intersecting) closed contours, inflation is an unambiguous way to reconstruct a plausible 3D model. The Teddy system [29], for instance, inflates a contour by pushing vertices away from the chordal axis according to their distance from the contour; see Fig. 16b for a typical result.

The skeletal representation of a contour also integrates naturally with an implicit surface representation. In the approach of Alexe et al. [32], spherical implicit primitives are placed at each

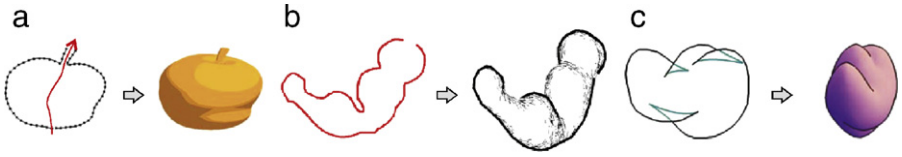

Fig. 16. Free-form model creation from contour sketches: (a) rotational blending surfaces have non-branching skeletons [40]; (b) Teddy inflates a sketch about its chordal axis (reproduced with permission from [29]); (c) SmoothSketch infers hidden contour lines (green lines) before inflation (reproduced from [55]).

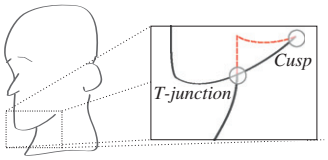

Fig. 17. The contour of an object conveys a lot of shape information. Cutout: *T*-junctions and cusps imply hidden contour lines (red).

skeleton vertex; when the primitives are blended together, the result is a smooth surface whose contour matches the input sketch. Other systems [16,51,61] instead use variational implicit surfaces [85], which use the sketched contour to define constraints in the implicit function.

For non-simple contours, such as ones containing self-intersections, a simple inflation method will fail. Recall that 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. Fig. 17 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; Williams [86] has proposed a method for using these to infer hidden contour lines in an image.

Cordier and Seo [70] use Williams' contour completion algorithm to support complex contour sketches containing *T*-junctions. The hidden contours can be sorted by relative depth, allowing the sketch to be positioned in 3D such that it can be inflated without self-intersections. To reconstruct a surface, the authors use a method similar to Alexe et al.'s implicit surface method. Karpenko and Hughes [55] also use Williams' algorithm, including support for not only *T*-junctions but also cusps (Fig. 15c). They take a different approach to reconstruction: 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 finding a smooth equilibrium state. Unfortunately, the mass-spring optimization requires careful parameter tuning and does not guard against self-intersections.

A final way to reconstruct a contour sketch is to fit a surface that is as smooth as possible. Surface fitting interpret input strokes as geometric constraints of the form, "the surface passes through this contour." The outside normal of the contour also constrains the surface 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 [86] push the system toward a solution. Nealen et al.'s FiberMesh system [56] uses a non-linear optimization technique to generate smooth meshes while also supporting sharp creases and darts.

5.1.3. Multi-view systems

An advantage of the surface fitting technique used in FiberMesh is that additional strokes can be added to define more constraints on the surface—even in different drawing planes than the initial contour. That is, the user can sketch in 3D to define a network of strokes, which together define a surface. This is an example of multi-view sketching.

While Fibermesh is interactive—the surface is immediately visualized after each sketch input—there has also been some work in batch multi-view sketch systems. In multi-view sketching systems [17,21,25,71,73], the strokes are typically interpreted as object boundaries. Das et al. [21], 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. [73] also use 3D boundary sketches, to define smooth planar deformations known as developable surfaces.

Karpenko et al. [65] propose an iterative sketching system based on "epipolar lines." After drawing a stroke in the drawing plane, the user can rotate the view and see lines extending along the depth axis—visualization of the depth ambiguity. Further input strokes are projected onto these lines, thereby fixing the depth component. In this way complex 3D curves can be sketched, although the authors admit the process is "cognitively difficult."

Multi-view sketching has also been explored in a more literal sense, in systems that allow the user to provide several sketches of an object from different viewpoints (such as front, side, and top views) [60,63]. Reconstruction in this case requires the system to find correspondences between each viewpoint to construct the 3D curve network.

Sketching in 3D without interactive feedback is difficult, 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 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. 7). 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) [87]. 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 [56,87,88]. 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. 18a). The sketched lines may also be treated as new geometric constraints in surface optimization approaches [56].

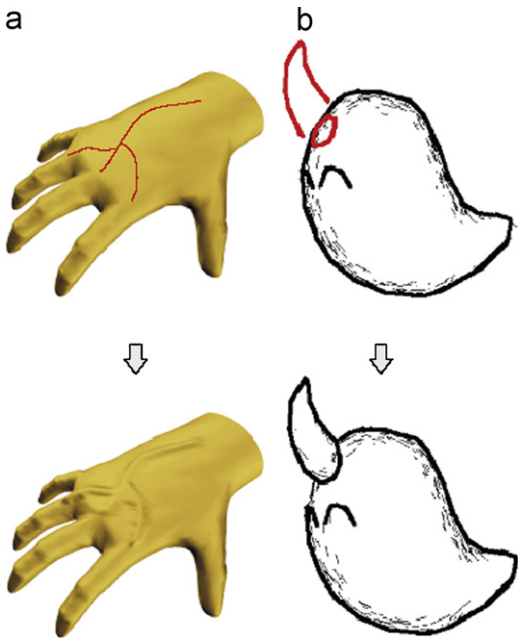

Fig. 18. Sketch-based augmentations: (a) surficial augmentation displaces surface elements to create features (from [87]); (b) additive augmentation joins a new part with an existing model (reproduced with permission from [29]). The latter figure also includes surficial features (the eyes).

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 [87,89].

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 [29,56]. For example, the extrusion operator in Teddy [29] 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. 18b). Schmidt et al. [16] 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 [90], 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 [56,91,92], bending [29,39,40,75,92], twisting [93], tunneling (creating a hole) [16,56] contour oversketching [40,89,94], segmentation [92,95], free-form deformation (FFD) [96], and affine transformations [97]. And, like augmentation, sketch-based deformations typically 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. 19a). 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 oversketching. For example, bending and twisting deform an object by matching a *reference* stroke to a *target* stroke, as shown in Fig. 19b. 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. 19c.

Nealen et al. [56] 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.

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 [96] 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.

Kara et al. [27] propose a template-deformation system (Fig. 20), in which the user provides a concept sketch and then manually selects an appropriate template matching the sketch. Using computer vision techniques, the template is aligned with the input sketch, which the user then oversketches to deform the template interactively.

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 [16,36], and rotational blending surfaces [40]. They are a well-studied representation, easily integrated into an application or exported to other modeling software. However, due to a simple 2D 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 extend parametric surfaces to general topology, and are often used in SBIM [21,29,71,83,89]. The main drawback of meshes is that some editing operations are difficult to implement, such as blending two objects together. Mesh quality is also an issue [59,75,83], as irregular faces can lead to unstable lighting and surface property calculations. Though a mesh-like representation is generally necessary for rendering an object to the display, more flexible representations can be used in the background.

Implicit surfaces have several advantageous properties from a modeling perspective, including support for hierarchical modeling, blending, and boolean operations. However, they are naturally

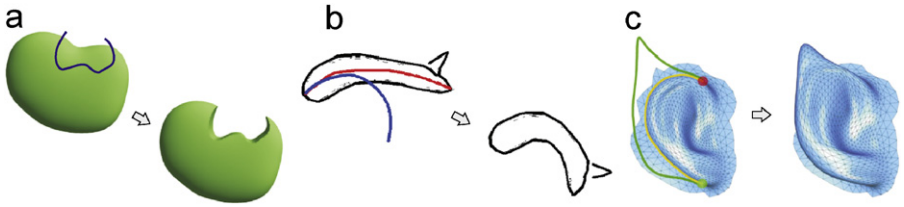

Fig. 19. Sketch-based deformations: (a) cutting strokes (blue) define a cutting plane along the view direction (from [91]); (b) bending a model so that a reference stroke (left) is aligned with a target stroke (right) [40]; (b) contour oversketching matches object contours (yellow) to target strokes (green) (reproduced with permission from [89]).

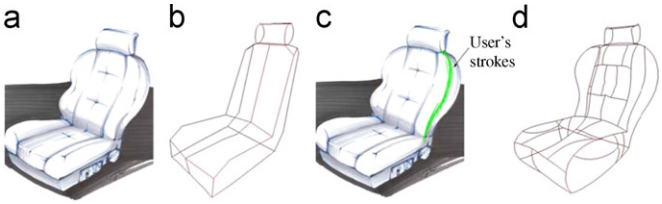

Fig. 20. Kara et al. propose a system that registers a chosen template object (b) to an input sketch (a). The user then interactively deforms the template by tracing over the sketch (c); the resulting model (d). Reproduced from [27].

smooth and blobby, and introducing sharp or hard-edged features is difficult. Another drawback is that implicits do not allow direct surface manipulation, so the grab-and-drag modeling metaphor is precluded. Finally, 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 [16,32,51,61], and may also be used as an intermediate representation from which to extract a mesh [66,83].

Implicit surfaces are more correctly defined to as isosurfaces extracted from an implicit volume. The volumetric representation can be used to model a broader variety of topologies, as well as simplifying the implementation of operations such as cutting [69]. The drawbacks of this representation are similar to implicit surfaces: rendering requires discretization and polygonization of a surface, and direct manipulation is infeasible.

Fair surfaces are meshes that result from solving a constrained optimization problem [56,73,86]. 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 difficult to anticipate from the user's perspective.

Finding a surface representation that is suitable for all modeling tasks is an important challenge in SBIM, and modeling in general (Section 8).

6. Interface design

A complete modeling system must simultaneously support many operations such as creation, augmentation, and deformation, plus viewing and rendering controls. Each operation represents a mode or a state; a traditional "modal" interface design would require explicit switching between modes, via buttons, menus, or keyboard shortcuts and modifiers. 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.

A common approach found in SBIM systems is the use of *gestural* interfaces to simplify common operations. Moving away from menu-based command specification, a gestural interface uses simple free-form stroke input to specify commands and manipulate objects (directly or indirectly). Though the lack of menus may be less intimidating to a novice user, remembering the correct stroke-operation mapping still requires training and cognitive effort on the user's part. Some examples of gestural commands are cutting and deleting strokes [25], object grouping [98], erasing and local smoothing [56], and stroke blending [16].

Using gestures to specify commands that require user-specified parameter values is more complicated, but there have been some novel approaches in this area. Severn et al. [97] describe a direct manipulation approach called *transformation strokes*. In their system, the user can quickly scale, rotate, and translate an object with a single U-shaped gesture. The width and height specify the aspect ratio, while the placement and orientation of the stroke specify translation and rotation. Depth is always a problem, but other objects in the scene can be used to disambiguate the transformed object's position. In this way, objects composed of different parts can be assembled very rapidly.

Schmidt et al. [99], meanwhile, use gestures not to manipulate an object directly, but simply to initiate an operation widget. The user can then interact with the widget to manipulate the object interactively. For example, a simple linear stroke that crosses an object initiates a translation widget, which is an arrow that can be dragged back and forth to translate the object. Again, the ease of initiating and performing a transformation enables rapid object assembly.

While sketches can be used in many facets of a modeling interface, a purely gestural sketch-based interface causes modality problems. That is, a given stroke or gesture can have different meanings in different modes of the system. As an example, the ShapeShop system of Schmidt et al. [16,76,99] uses gestures to initiate widgets, but also allows surficial augmentation strokes—what happens if an augmentation stroke is the same as a widget gesture? Only the user can truly know the intended meaning in this case. Therefore, when designing a gestural and sketch-based system, a critical issue is how to provide a consistent and predictable interface without modality problems.

There are two ways to avoid modality problems in a sketch-based interface. One is to design the system so that the inputs across all the modes are mutually exclusive (i.e. input stroke *A* only appears in mode *X*). In systems that allow free-form input, this constraint is very difficult to satisfy (i.e. how can any particular stroke be assumed to not exist in sketched input?) A more feasible solution is to detect the ambiguous inputs—inputs that represent valid input for several modes—and prompt the user for some clarifying action. This approach is commonly called a *suggestive interface* (Fig. 21).

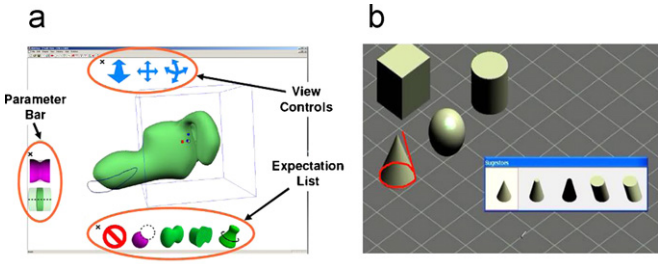

Fig. 21. Suggestive interfaces use expectation lists to disambiguate input: (a) interface of ShapeShop [16], showing an expectation list for operations. (b) Interface of GIDeS [45], showing expectation lists with different shapes for user confirmation.

In a suggestive interface, the system identifies all possible interpretations (often called an "expectation list") of a given input stroke or sketch, from which the user can choose the intended action. Expectation lists can be used not just to clarify gestural inputs [45,76,100], but also in other stages of the system such as model creation [16,45]. For example, consider Fig. 21b, which shows a number of plausible interpretations of the line-and-circle input.

This highlights a couple of important issues to consider in suggestive interfaces. First, there can be many interpretations of a given input, but presenting all possible choices to the user can be overwhelming. Instead, a system should try to rank the likelihood of each interpretation and offer only the few most likely choices to the user. This approach may also be combined with a learning system to adapt to individual usage patterns [39,54]. Second, and more difficult to manage, is that as a sketch-based system increases in functionality, the frequency of these expectation lists popping up and demanding user attention can be very intrusive and annoying. Striking the proper balance between a system's autonomy and the user's control is a difficult and important challenge.

6.1. Gesture recognition

In a gestural interface, the system needs to recognize the input. That is, if the user sketches gesture A, the system can recognize it by comparing against a *template* $\hat{A}$. We define a template as any comparable description of an object; for sketches, this can range from low-level representations such as point sequences or bitmaps, or higher-level embeddings such as a normalized count of angular activity [101]. Generally there will be a set of possible templates, necessitating sketch recognition algorithms that search the set to find the best match.

A fundamental aspect to gesture-based interfaces is the design of a robust and consistent gesture "vocabulary" (Fig. 22). 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" [102]. Care should be taken to design a good set of gestures—that is, a set that is distinct, memorable, and easy to draw.

Many approaches have been proposed for recognizing gestures. An early approach by Rubine [103] 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 [104]. Other methods exploit domain-specific knowledge to derive higher-level understanding of strokes, such as building a diagrammatic representation [105] or identifying and labeling different elements [106]. Hammond and Davis [107] 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.

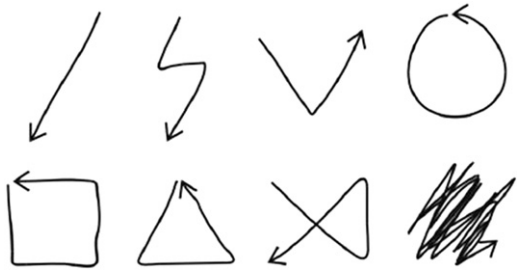

Fig. 22. Sample gestures recognized in [101].

Gesture matching borrows conceptual elements from trajectory analysis, in that both deal with the behavior of moving objects. In the case of a sketch, each stroke captures the trajectory of the input device. Fourier analysis is perhaps the most common technique in trajectory analysis [108,109]. 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 [110] attempt to address this issue by using smaller waves, but suffer from signal length restrictions.

Gesture recognition has become the focus of user interface (HCI) research. A recent popular approach is known as the $1 recognizer [111]. The input gesture is first resampled to remove drawing speed variation, then aligned along an "indicative angle" to provide rotation invariance. Finally, the gesture is scaled non-uniformly into a unit square. Templates undergo the same transformations, and point-wise distance is used to compare two transformed strokes. This approach offers both strong performance with minimal training and low computational overhead, making it well-suited to gestural interfaces.

Compared to general shape matching, the demands of a gesture recognizer are unique: it must only distinguish a limited number of distinct inputs, but it must do it quickly. Therefore, sketch-based systems often sacrifice rigor for speed. For instance, the Teddy system [29] 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 Section 5.3). Yang et al. [26] 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, which are used as a stroke "signature" for recognition. More recently, Olsen et al. [101] propose a method for describing a stroke by quantizing its angular distribution, and within the context of SBIM are able to outperform classical methods, including Fourier. Each of these approaches would likely give poor results for general shape matching, but perform well within the target SBIM applications.

7. Applications

Thus far, we have focused on the use of SBIM in fundamental modeling tasks. There are many specific applications in which free-form sketch input is a very useful and powerful interface paradigm, some of which are discussed below. This section is

intended merely to direct the reader toward interesting applications of sketch-based interfaces in computer graphics, as an in-depth review is beyond the scope of this survey.

The applications can be classified in two groups. *Computer-aided design* (CAD; Section 7.1) applications are targeted at modeling 3D objects that will eventually have a physical manifestation. Therefore, input sketches need to be complemented with constraints to address manufacturing limitations. *Content creation* (Section 7.2) applications, meanwhile, are intended for modeling 3D objects that will exist solely in the digital world, for use in computer animation, interactive computer games, film, and so on. In this domain, geometric precision is less important than allowing the artist to create free-form surfaces from freehand input.

7.1. SBIM in CAD

Existing CAD tools focus on representing design ideas and models which are nearly complete. Currently, concept sketches are developed in 2D and manually translated to a 3D representation using traditional CAD tools, a process that can take many weeks. This is one of the causes of long production cycles in the design industry. SBIM tools have the potential to enable faster and more natural exploration of ideas, allowing creation and systematic refinement of 3D models from early concept sketches to finished designs (Fig. 23).

Content creation for *industrial design* has proved an elusive target due to the difficulty in expressing precise NURBS-type surfaces with free-form entities. Typical approaches involve sketching and manipulating construction curves [112–114] or character lines [117] to deform 3D templates.

The design of *mechanical engineering* objects encounters similar issues with sketch-based specification of precise part placement and geometric constraints. These have been addressed by systems such as GIDeS [115,116], which combines iconic input methods with constraints and dynamic menus to support the creation of complex mechanical parts from sketched input. GIDeS allows for precise placement of objects by using constraints and expectation lists, both for 2D and 3D constructs as well as implicit CSG operators. A similar system is CEGROSS from Contero et al. [118], which combines a constraint satisfaction engine with sketches and reconstruction, allowing engineers to specify mechanical parts in a perspective drawing.

Architectural drawings and building depictions have been the subject of much work since Gross et al.'s Cocktail Napkin system [119,120]. Leclercq et al.'s Esquisse system [121,122] provides a comprehensive sketch-based architectural modeling package, allowing architects to develop 3D building models from conceptual floorplan sketches. Dorsey et al.'s Mental Canvas [22] is targeted more toward the creation of conceptual drawings than full 3D reconstruction, offering an interface for sketching on multiple planes and then "pushing" the sketches through 3D space onto other canvases to create a quasi-3D representation.

7.2. SBIM in digital content creation

With the ubiquity of computer-generated images in films and television, as well as the emergence of interactive computer gaming, the demand for digital content creation is extremely high. As such, techniques such as SBIM that can increase the efficiency of the production pipeline are being heavily explored (Fig. 24).

To construct a digital world, Cohen et al.'s Harold system [123] allows users to sketch arbitrary elements such as terrain elevation, trees, buildings, and characters. World geometry is approximated with $2\frac{1}{2}$-D billboard models; that is, all groups of planar strokes are reoriented in a view-dependent way as the camera moves through the world to give the impression of three dimensions.

A similar quasi-3D approach is taken by Bourguignon et al. [126]. In their system, their goal is visual communication rather than surface reconstruction; drawn strokes are "promoted" to 3D. The user can draw from multiple perspectives by rotating the viewpoint to construct sketchy 3D objects or create annotations of

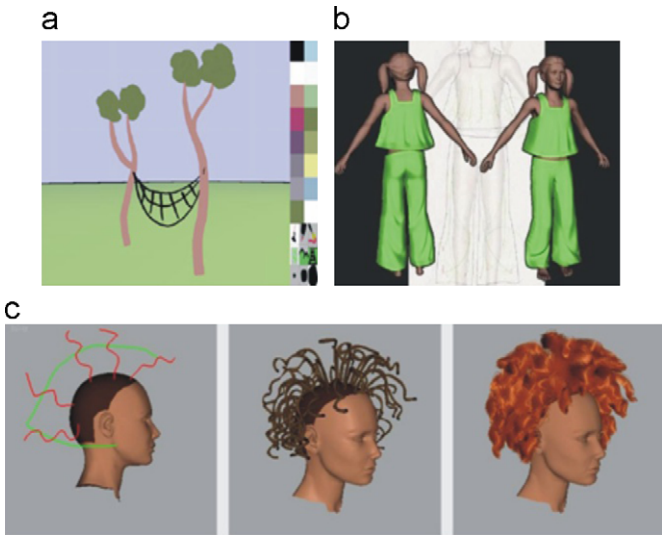

Fig. 24. Applications of SBIM in content creation: (a) interactive worlds [123]; (b) garment design [124]; (c) hair modeling [125].

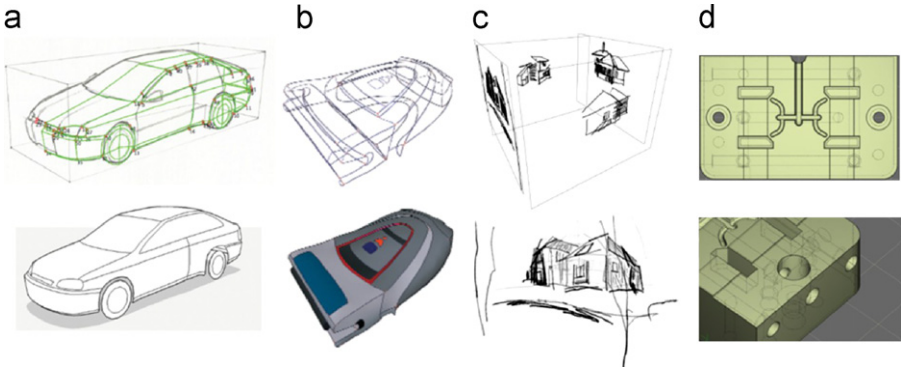

Fig. 23. Applications of SBIM in CAD: (a) automotive design [112]; (b) industrial design [113,114]; (c) architecture [22]; (d) mechanical engineering [115,116].

imported objects. The system renders the result non-photorealistically, to stress the imprecise and informal nature.

Sketch-based interfaces have also been used for virtual garment design. In Turquin et al.'s system [127], users draw an outline of the front and back of the garment, and the system makes geometric inferences about the overall shape of the garment. Both the garment's shape and the way the character is wearing it are determined at once.

Sketch-based interfaces have also been explored for character animation, using hand-drawn sketches to specify key poses or positions in an animation sequence [128–130]. Davis et al. [128], for instance, extract joint positions from a stick-figure sketch via image-processing techniques, apply geometric and physical constraints to rule out implausible poses, and then deform the character model to match the sketched pose. Thorne et al. [129] instead allow the user to sketch character motion using a set of sketch gestures that are mapped to pre-defined motions such as walking and jumping.

Hair is notoriously difficult to model, due to the sheer number of elements. Using Wither et al.'s system [125] users sketch example hair strands over a side view of the character's head. Geometric and mechanical properties of the hair strands are inferred to adjust the shape of the scalp and generate an adequate hair volume. Malik [131] presented a sketching interface for modeling and editing hairstyles to mimic hairdressing operations such as cutting, combing, frizzing and twisting using a 3D scalp model.

Plant modeling is another laborious and time-consuming process. Plants have intricate instances of branching and organ structures with varied postures and spatial distributions. There is of course an incredible diversity in the plant world, with trees, flowers, and single-stem plants having similar yet distinct geometric characters. SBIM tools allow faster and more natural description of the plant posture, branching structures, and organ geometry and positioning.

Okabe and Igarashi's system [132] infers the geometry of trees from 2D sketches of the branching structures. Ijiri et al. [133] use gestural sketches to control the shape of the main trunk of a recursively defined branching structure. Zakaria and Shukri [134] start by sketching an initial tree structure, and then "spray" leaf surfaces around crown regions, so that tree branches grow toward the sprayed leaves.

In Ijiri et al.'s system [135], sketched flower petals and other elements are composed into complete flowers using positioning rules from *floral diagrams*. These flowers are then organized into complete arrangements using *inflorescences diagrams*. In a follow-up work [136], individual flower organs are sketched on drawing planes positioned at different orientations across the plant structure.

Anastacio et al. [137] use concept sketches to constructs 3D plant arrangements using phyllotactic patterns. A more recent work [138] proposes a method that translates the concept sketches to positional functions as input to L-systems for procedurally generating the final plant structure.

8. Challenges and open problems

Though recently we have seen substantial advances in SBIM, there remain many important open problems and challenges in this area. Indeed, human-like shape perception of 2D sketches by computers remains largely an elusive target. And while SBIM systems indeed offer an improvement over traditional systems in terms of accessibility, they are not yet complete replacements. Regardless of the approach, current SBIM systems can model only a limited range of objects with low complexity, and there remains much work to be done to bridge this gap.

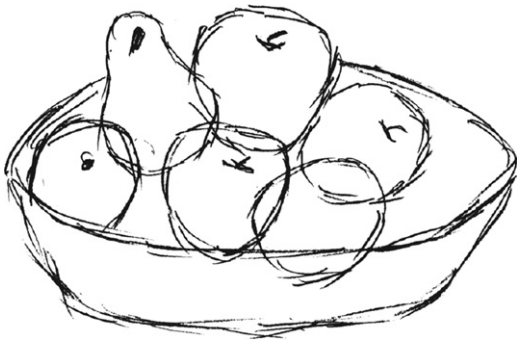

Fig. 25. Support for natural sketching, with overlapping or extraneous strokes, is lacking in SBIM systems.

8.1. Interface

One of the main goals in SBIM has been to provide a more natural interface that mimics the feel of traditional media. With real pencil-and-paper sketching, in the initial stages of the design process an artist will often faintly sketch primitive shapes to define the overall form of an object, and then use many small strokes to complete the sketch (Fig. 25). While these strokes define the general sweep for a final pleasant curve, they are drawn freely in an arbitrary order, with possible intersections, discontinuities, and even extraneous strokes. However, most sketch-based interfaces are far from natural—many require the user to draw in very specific ways to function properly, which reduces the immersion and ease of use.

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" [98]. Thus, designing SBIM systems with the right combination of algorithmic and interface elements to provide stable and predictable interaction is a large 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 [139]. 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 another challenge for SBIM researchers.

8.2. Shape cues

Most of the efforts in SBIM have made use of contour lines as constructive curves for modeling and deformation. This is reasonable since contour lines are so perceptually meaningful, but it is not always enough. Traditional art employs other shapes

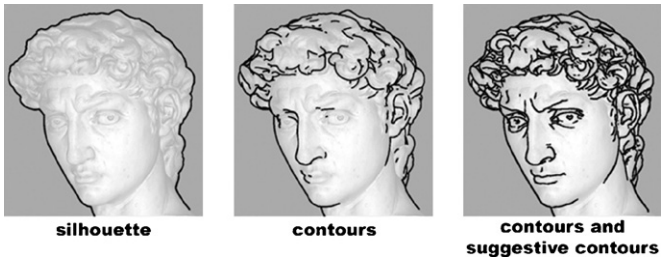

Fig. 26. Shape cues beyond contour lines could be used to sketch more complex and less ambiguous shapes (image source: www.cs.rutgers.edu/decarlo/contour.html).

cues such as hatching, scribble lines, stippling, shading, and suggestive contours [140] to convey 3D forms (Fig. 26). Including these cues into sketch interpretation is non-trivial, but has the potential to drastically improve modeling, augmentation and deformation techniques. While there has been some progress toward extracting shape information from other shape cues, i.e. shape from shading in single images [141,142], and using other important curves [40,56,89], more elaborate research is still needed and it remains an open and challenging area in SBIM.

This research direction, in some ways, parallels the development of non-photorealistic rendering (NPR). NPR asks the question, "How can a 3D model be rendered artistically and economically in a way that accurately and clearly reveals its shape?" NPR approaches found contour lines to be critical for shape perception, but have advanced beyond them to include various other artistic shape cues such as hatching and suggestive contours. Perhaps SBIM—which has been referred to as "inverse NPR" [89]—can learn from these developments and *extract* shape information from artistic cues. The current works of Wu [143] and Gingold and Zorin [144] are a step in this direction.

8.3. Visual memory versus visual rules

Human perception relies on both visual memory and visual rules for 3D reconstruction. However, most SBIM systems are based on only one of these two skills, and have a long way to go before approaching the versatility of human perception. An evocative system is primarily limited by the size of its 'memory,' and techniques for measuring the similarity between sketches and 3D shapes are still poor replacements for human visual memory. Designing a successful general purpose shape retrieval system remains an important challenge. As evidence of this challenge, face recognition—perhaps the most studied object detection area—has been steadily investigated and improved over the last decade [145].

On the other hand, the ambiguity problem is very challenging for constructive systems. Without some form of visual memory, it is difficult or impossible to resolve the depth and occlusion ambiguities (Fig. 27). Therefore, constructive systems can only build rough prototypes or cartoony-looking models, while evocative systems can produce more precise, but limited, models from the template set. As Karpenko and Hughes [55] 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. The work of Yang et al. [26] is an example toward such a system, though their template definition is difficult to extend.

8.4. Model quality

Improving the model quality in SBIM is another important challenge. Parametric surfaces, such as rotational blends [40], can create high-quality results from sketches, and are easily transported to professional modeling software. In addition, they feature an efficient multi-resolution representation and simple texture mapping. However, the limited topology necessitates careful alignment of multiple patches to create complex models.

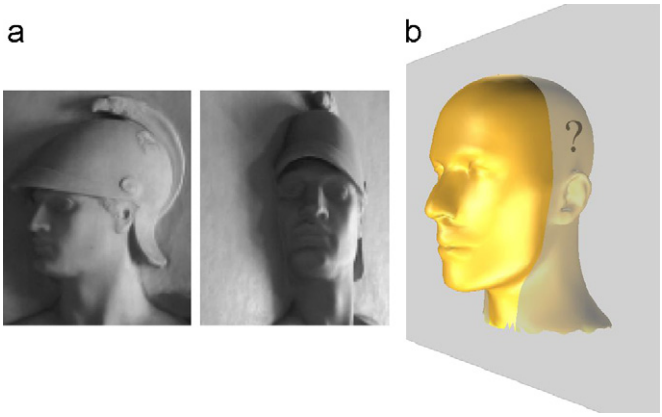

Fig. 27. Ambiguities in 2D: both (a) the depth (from [146]) and (b) occluded surfaces are difficult to recover by rules alone.

Polygonal meshes does not have the issue of topology limitation, and, as shown in work such as FiberMesh [56], it is possible to specify complex shapes using stroke-based operations. However, current approaches to mesh generation are slow for high-quality meshes due to the (sometimes non-linear) energy minimization stage. In addition, stroke-based operations may globally distort the entire mesh which usually is not a desirable effect. The challenge is to find a high quality but efficient surface representation that also supports local stroke-based operations, bridging the gap between quality and ease of specification.

Additionally, models created by SBIM systems tend to have a blobby appearance. Adding high quality details and sharp features is another aspect that requires more investigation. Several recent works [56,76,87] allow the specification of sharp creases and darts, which is a step in the right direction. In the future, to support more complex features, modeling concepts such as multi-resolution editing could dramatically increase the utility of sketch-based systems (Fig. 28).

8.5. Precision

A lack of precision has often been cited as one of the weakness of SBIM, compared with a control-point paradigm that allows to accurately select and modify a surface. Specifying or inferring geometric constraints in an SBIM system—such as parallelism, perpendicularity, dimension equality, and horizontal–vertical alignment—makes it possible to introduce precision, at least in engineering design systems (e.g. GIDeS [45,115]). The interface necessarily becomes more complex, however, and so precision is often sacrificed in favor of simplicity, particularly in free-form design systems.

It is arguable whether control point manipulation is in fact an ideal mechanism for precise surface manipulation. For example, assume that we want to displace a surface at a point P on the surface by d units along normal direction. It is not obvious that which control point must be displaced and even if there are some good candidates, how much displacement is needed for them. An ideal SBIM system can think outside of the control-point box, and perhaps find a better paradigm that fits with human experiences and conventions.

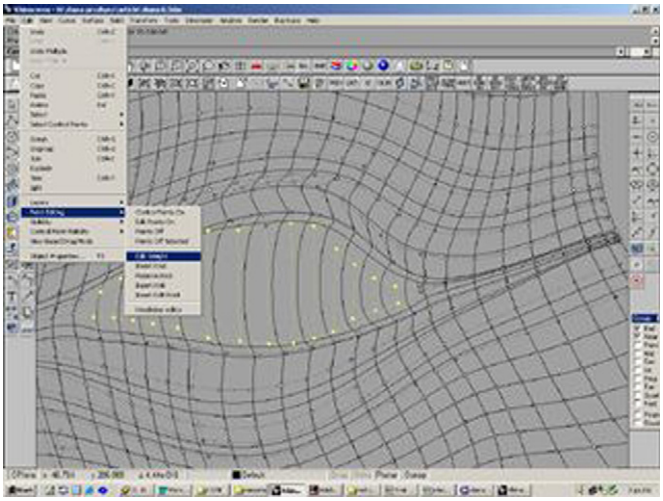

Fig. 28. Precision is often lacking in SBIM systems, making it a difficult sell for industrial use. Screenshot from Maya [1].

9. Conclusion

Sketch-based systems have a reputation of being suitable only for "quick-and-dirty" [26] 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 has the potential to be used for a wide range of modeling tasks.

Perhaps we should simply embrace the ambiguous nature of sketched input. Art is 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 drawn across a piece of paper, or a paint brush across a canvas. Traditional modeling applications also support iterative design through tools such as subdivision.

This should inspire SBIM to pursue stable and predictable interfaces that naturally support an iterative methodology, rather than pure reconstruction. As Nealen et al. [89] 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."

A hybrid system that contains a substantial shape memory, robust creation rules, and perhaps even a capacity to learn new shapes, hold the most potential for approaching human-like sketch understanding. The diversity of disciplines involved in realizing such a system—modeling, vision, HCI, perception—will ensure that sketch-based modeling remains an exciting and challenging topic for years to come.

Each passing year brings new and exciting advances in the field. In addition to broader publication venues such as computer graphics or HCI journals and conferences, there are several venues catering specifically to SBIM. These include the annual Eurographics Workshop on sketch-based interfaces and modeling, the AAAI Symposium on Sketch Understanding, and the 2007 SIGGRAPH course on sketch-based interfaces.

References

[1] Autodesk Inc., Maya ⟨www.autodesk.com/maya⟩.
[2] Dassault Systèmes, Solidworks ⟨www.solidworks.com⟩.
[3] Dassault Systemes, Catia ⟨www.catia.com⟩.
[4] Archipelis, Archipelis designer ⟨www.archipelis.com⟩.
[5] E-Frontier, Sunny3d ⟨www.e-frontier.co.jp/sunny3d⟩.
[6] Pixologic, Inc., Zbrush ⟨www.pixologic.com⟩.
[7] Autodesk Inc., Mudbox ⟨www.mudbox3d.com⟩.
[8] Sutherland I. Sketchpad: a man-machine graphical communication system. In: AFIPS conference proceedings, vol. 23, 1963.
[9] Olsen L, Samavati FF, Sousa MC, Jorge JA. A taxonomy of modeling techniques using sketch-based interfaces. In: Eurographics 2008 state of the art report, 2008.
[10] Hoffman DD. Visual intelligence: how we create what we see. W.W. Norton & Company; 2000.
[11] Masry M, Lipson H. A sketch-based interface for iterative design and analysis of 3d objects. In: Proceedings of the eurographics workshop on sketch-based interfaces and modeling (SBIM '05), 2005.
[12] Hoffman D, Singh M. Salience of visual parts. Cognition 1997;63(1):29–78.
[13] Company P, Piquer A, Contero M. On the evolution of geometrical reconstruction as a core technology to sketch-based modeling. In: Proceedings of eurographics workshop on sketch-based interfaces and modeling (SBIM '04), 2004.
[14] Tanaka T, Naito S, Takahashi T. Generalized symmetry and its application to 3d shape generation. The Visual Computer 1989;5(1–2):83–94.
[15] Hayward V, Astley OR, Cruz-Hernandez M, Grant D, deLaTorre G. Haptic interfaces and devices. Sensor Review 2004;24(1):16–29.
[16] Schmidt R, Wyvill B, Sousa MC, Jorge JA. Shapeshop: sketch-based solid modeling with blobtrees. In: Proceedings of eurographics workshop on sketch-based interfaces and modeling (SBIM '05), 2005.
[17] Fleisch T, Brunetti G, Santos P, Stork A. Stroke-input methods for immersive styling environments. In: Proceedings of the international conference on shape modeling and applications (SMI '04), 2004.
[18] LeapFrog Enterprises, Fly fusion pentop computer ⟨http://www.flyworld.com/whatis/index.html⟩.
[19] Saga S. A freehand interface for computer aided drawing systems based on the fuzzy spline curve identifier. In: Proceedings of the IEEE international conference on systems, man and cybernetics, 1995.
[20] Sezgin TM, Stahovich T, Davis R. Sketch based interfaces: early processing for sketch understanding. In: Proceedings of workshop on perceptive user interfaces (PUI '01), 2001.
[21] Das K, Diaz-Gutierrez P, Gopi M. Sketching free-form surfaces using network of curves. In: Proceedings of eurographics workshop on sketch-based interfaces and modeling (SBIM '05), 2005.
[22] Dorsey J, Xu S, Smedresman G, Rushmeier H, McMillan L. The mental canvas: a tool for conceptual architectural design and analysis. In: Proceedings of Pacific conference on computer graphics and applications (PG'07), 2007.
[23] Bae S-H, Kijima R, Kim W-S. Digital styling for designers: 3D plane-symmetric freeform curve creation using sketch interface, Lecture notes in computer science, vol. 2669/2003. Berlin: Springer; 2003. p. 701–10.
[24] Branco V, Costa A, Ferreira FN. Sketching 3d models with 2d interaction devices. Computer Graphics Forum 1994;13(3):489–502.
[25] Tsang S, Balakrishnan R, Singh K, Ranjan A. A suggestive interface for image guided 3d sketching. In: Proceedings of the SIGCHI conference on human factors in computing systems (CHI '04), ACM, 2004.
[26] Yang C, Sharon D, van de Panne M. Sketch-based modeling of parameterized objects. In: Proceedings of eurographics workshop on sketch-based interfaces and modeling (SBIM '05), 2005.
[27] Kara L, D'Eramo C, Shimada K. Pen-based styling design of 3d geometry using concept sketches and template models. In: Proceedings of ACM solid and physical modeling conference (SPM '06), 2006.
[28] Sezgin TM, Davis R. Scale-space based feature point detection for digital ink. In: Making pen-based interaction intelligent and natural, AAAI fall symposium, 2004.
[29] Igarashi T, Matsuoka S. Tanaka H. Teddy: a sketching interface for 3d freeform design. In: Proceedings of the SIGGRAPH'99, 1999.
[30] Kurozumi Y, Davis W. Polygonal approximation by the minimax method. Computer Graphics and Image Processing 1982;19:248–64.
[31] Saykol UGE, Gülesir G, Ulusoy O. KiMPA: a kinematics-based method for polygon approximation. Lecture notes in computer science, vol. 2457/2002. Berlin/Heidelberg: Springer; 2002. p. 186–94.
[32] Alexe A, Gaildrat V, Barthe L. Interactive modelling from sketches using spherical implicit functions. In: Proceedings of international conference on computer graphics, virtual reality, visualisation and interaction in Africa (AFRIGRAPH '04), 2004.
[33] Taubin G. Curve and surface smoothing without shrinkage. In: ICCV '95: proceedings of the fifth international conference on computer vision, 1998. IEEE Computer Society; 1995.
[34] Koenig H. Modern computational methods. London: Taylor & Francis; 1998.
[35] Piegl L. Interactive data interpolation by rational bezier curves. IEEE Computer Graphics and Applications 1987;7:45–58.
[36] Eggli L, Ching-Yao H, Bruderlin B, Elber G. Inferring 3d models from freehand sketches and constraints. Computer-Aided Design 1997;29(2):101–12.
[37] Rogers DF. Constrained b-spline curve and surface fitting. Computer-Aided Design 1989;21(10):641–8.
[38] Banks M, Cohen E. Real time spline curves from interactively sketched data. In: Proceedings of the SIGGRAPH'90, vol. 24, no. 2, 1990. p. 99–107.
[39] Kara L, Shimada K. Construction and modification of 3d geometry using a sketch-based interface. In: Proceedings of eurographics workshop on sketch-based interfaces and modeling (SBIM '06), 2006.

[40] Cherlin JJ, Samavati F, Sousa MC, Jorge JA. Sketch-based modeling with few strokes. In: Proceedings of spring conference on computer graphics (SCCG '05), 2005.
[41] Samavati F, Mahdavi-Amiri N. A filtered b-spline model of scanned digital images. Journal of Science 2000;10:258–64.
[42] Kasturi R, O'Gorman L, Govindaraju V. Document image analysis: a primer. Sadhana 2002;27(1):3–22.
[43] Yu B. Recognition of freehand sketches using mean shift. In: Proceedings of the international conference on intelligent user interfaces (IUI '03), 2003.
[44] Zeleznik R, Herndon K, Hughes J. SKETCH: an interface for sketching 3d scenes. In: Proceedings of the SIGGRAPH '96, 1996.
[45] Jorge JA, Silva FN, Cardoso DT. GIDeS++. In: Proceedings of the 12th annual Portuguese CG meeting, 2003.
[46] Igarashi T, Matsuoka S, Kawachiya S, Tanaka H. Interactive beautification: a technique for rapid geometric design. In: Proceedings of ACM symposium on user interface software and technology (UIST '97), 1997.
[47] Igarashi T, Hughes JF. A suggestive interface for 3d drawing. In: Proceedings of ACM symposium on user interface software and technology (UIST '01), 2001.
[48] Contero M, Naya F, Jorge J, Conesa J. CIGRO: a minimal instruction set calligraphic interface for sketch-based modeling. Lecture notes in computer science, vol. 2669/2003. Berlin, Heidelberg: Springer; 2003. p. 989.
[49] Schweikardt E, Gross MD. Digital clay: deriving digital models from freehand sketches. In: Digital design studios: do computers make a difference? (ACADIA '98), 1998.
[50] Pusch R, Samavati F, Nasri A, Wyvill B. Improving the sketch-based interface: forming curves from many small strokes. In: Proceedings of computer graphics international (CGI 2007), 2007.
[51] DeAraujo B, Jorge J. Blobmaker: free-form modelling with variational implicit surfaces. In: Proceedings of 12 Encontro Português de Computacão Grafica, 2003.
[52] Fleisch T, Rechel F, Santos P, Stork A. Constraint stroke-based oversketching for 3d curves. In: Proceedings of eurographics workshop on sketch-based interfaces and modeling (SBIM '04), 2004.
[53] Shesh A, Chen B. SMARTPAPER: an interactive and user friendly sketching system. In: Proceedings of eurographics 2004, 2004.
[54] Kara L, Stahovich T. An image-based trainable symbol recognizer for hand drawn sketches. Computers and Graphics 2005;29(4):501–17.
[55] Karpenko OA, Hughes JF. Smoothsketch: 3d free-form shapes from complex sketches. In: Proceedings of SIGGRAPH '06, 2006.
[56] Nealen A, Igarashi T, Sorkine O, Alexa M. Fibermesh: designing freeform surfaces with 3d curves. In: ACM transactions on graphics (proceedings of the SIGGRAPH '07). ACM Press, 2007.
[57] Kanai S, Furushima S, Takahashi H. Generation of free-form surface models by understanding geometric and topological constraints on rough sketches. In: Proceedings of IEEE international conference on systems engineering, 1992.
[58] Lipson H, Shpitalni M. Optimization-based reconstruction of a 3d object from a single freehand line drawing. In: ACM SIGGRAPH 2007 courses, 2007.
[59] Igarashi T, Hughes JF. Smooth meshes for sketch-based freeform modeling. In: Proceedings of ACM symposium on interactive 3D graphics, 2003.
[60] Pereira JP, Jorge JA, Branco V, Ferreira FN. Towards calligraphic interfaces: sketching 3d scenes with gestures and context icons. In: Proceedings of WSCG '00, 2000.
[61] Karpenko O, Hughes JF, Raskar R. Free-form sketching with variational implicit surfaces. In: Proceedings of eurographics 2002, 2002.
[62] Mitani J, Suzuki H, Kimura F. 3d Sketch: sketch-based model reconstruction and rendering. From geometric modeling to shape modeling, 2002. p. 85–98.
[63] Funkhouser T, Min P, Kazhdan M, Chen J, Halderman A, Dobkin D, et al. A search engine for 3d models. ACM Transactions on Graphics (Proceedings of SIGGRAPH '03) 2003;22(1):83–105.
[64] Piquer A, Martin RR, Company P. Using skewed mirror symmetry for optimisation-based 3d line-drawing recognition. In: Proceedings of IAPR international workshop on graphics recognition, 2003.
[65] Karpenko O, Hughes JF, Raskar R. Epipolar methods for multi-view sketching. In: Proceedings of eurographics workshop on sketch-based interfaces and modeling (SBIM '04), 2004.
[66] Tai C-L, Zhang H, Fong JC-K. Prototype modeling from sketched silhouettes based on convolution surfaces. Computer Graphics Forum 2004;23: 71–83.
[67] Varley P, Takahashi Y, Mitani J, Suzuki H. A two-stage approach for interpreting line drawings of curved objects. In: Proceedings of eurographics workshop on sketch-based interfaces and modeling (SBIM '04), 2004.
[68] Varley P, Martin R, Suzukia H. Frontal geometry from sketches of engineering objects: is line labelling necessary? Computer-Aided Design 2005;37:1285–307.
[69] Owada S, Nielsen F, Nakazawa K, Igarashi T. A sketching interface for modeling the internal structures of 3d shapes. In: ACM SIGGRAPH 2006 courses, 2006.
[70] Cordier F, Seo H. Free-form sketching of self-occluding objects. IEEE Computer Graphics and Applications 2007;27(1):50–9.
[71] Hui K, Lai Y. Generating subdivision surfaces from profile curves. Computer-Aided Design 2007;39(9):783–93.
[72] Mori Y, Igarashi T. Plushie: an interactive design system for plush toys. In: Proceedings of SIGGRAPH '07, 2007.
[73] Rose K, Sheffer A, Wither J, Cani M-P, Thibert B. Developable surfaces from arbitrary sketched boundaries. In: Proceedings of the eurographics symposium on geometry processing (SGP '07), 2007.
[74] Shin H, Igarashi T. Magic canvas: interactive design of a 3-d scene prototype from freehand sketches. In: Proceedings of graphics interface (GI '07), 2007.
[75] Wang H, Markosian L. Free-form sketch. In: Proceedings of eurographics workshop on sketch-based interfaces and modeling (SBIM '07), 2007.
[76] Schmidt R, Singh K. Sketch-based procedural surface modeling and compositing using surface trees. In: Proceedings of eurographics 2008, 2008.
[77] Lee J, Funkhouser T. Sketch-based search and composition of 3d models. In: Proceedings of eurographics workshop on sketch-based interfaces and modeling (SBIM '08), 2008.
[78] Fonseca MJ, Ferreira A, Jorge JA. Towards 3d modeling using sketches and retrieval. In: Proceedings of eurographics workshop on sketch-based interfaces and modeling (SBIM '04), 2004.
[79] Veltkamp R. Shape matching: similarity measures and algorithms. In: Proceedings of international conference on shape modeling and applications (SMI '01), 2001.
[80] Pugh D. Designing solid objects using interactive sketch interpretation. In: Proceedings of symposium on interactive 3D graphics (I3D '92), 1992.
[81] Malik J. Interpreting line drawings of curved objects. International Journal of Computer Vision 1987;1:73–103.
[82] de Berg M, van Kreveld M, Overmars M, Schwarzkopf O. Computational geometry: algorithms and applications. 2nd ed. Berlin: Springer; 2000.
[83] Levet F, Granier X. Improved skeleton extraction and surface generation for sketch-based modeling. In: Graphics interface 2007, 2007.
[84] Shusaku, Super5 ⟨www.shusaku.co.jp/www/product_S5M.html⟩.
[85] Turk G, O'Brien J. Variational implicit surfaces. Technical Report, Georgia Institute of Technology; 1999.
[86] Williams LR. Perceptual completion of occluded surfaces. PhD thesis, University of Massachusetts; 1994.
[87] Olsen L, Samavati F, Sousa MC, Jorge J. Sketch-based mesh augmentation. In: Proceedings of the 2nd eurographics workshop on sketch-based interfaces and modeling (SBIM), 2005.
[88] Biermann H, Martin I, Zorin D, Bernardini F. Sharp features on multi-resolution subdivision surfaces. Graphics Models 2001;64(2):61–77 [Proceedings of Pacific Graphics'01].
[89] Nealen A, Sorkine O, Alexa M, Cohen-Or D. A sketch-based interface for detail-preserving mesh editing. In: Proceedings of SIGGRAPH '05, 2005.
[90] Zelinka S, Garland M. Mesh modeling with curve analogies. In: Proceedings of Pacific graphics '04, 2004.
[91] Wyvill B, Foster K, Jepp P, Schmidt R, Sousa MC, Jorge J. Sketch based construction and rendering of implicit models. In: Proceedings of eurographics workshop on computational aesthetics in graphics, visualization and imaging, 2005.
[92] Ji Z, Liu L, Chen Z, Wang G. Easy mesh cutting. Computer Graphics Forum 2006;25(3):283–91 [Proceedings of eurographics '06].
[93] Kho Y, Garland M. Sketching mesh deformations. In: ACM SI3DG: symposium on interactive 3D graphics and games 2005, 2005.
[94] Zimmermann J, Nealen A, Alexa M. Silsketch: automated sketch-based editing of surface meshes. In: Proceedings of eurographics workshop on sketch-based interfaces and modeling (SBIM '07), 2007.
[95] Yuan X, Xu H, Nguyen MX, Shesh A, Chen B. Sketch-based segmentation of scanned outdoor environment models. In: Proceedings of eurographics workshop on sketch-based interfaces and modeling (SBIM '05), 2005.
[96] Draper G, Egbert P. A gestural interface to free-form deformation. In: Proceedings of graphics interface 2003, 2003.
[97] Severn A, Samavati F, Sousa MC. Transformation strokes. In: Proceedings of eurographics workshop on sketch-based interfaces and modeling (SBIM '06), 2006.
[98] Landay J, Myers B. Sketching interfaces: toward more human interface design. Computer 2001;34(3):56–64.
[99] Schmidt R, Singh K, Balakrishnan R. Sketching and composing widgets for 3d manipulation. In: Proceedings of eurographics 2008, 2008.
[100] Igarashi T, Hughes JF. Clothing manipulation. In: Proceedings of ACM symposium on user interface software and technology (UIST '02), 2002.
[101] Olsen L, Samavati F, Sousa MC. Fast stroke matching by angle quantization. In: Proceedings of the first international conference on immersive telecommunications (ImmersCom 2007), 2007.
[102] Long ACJ, Landay J, Rowe L, Michiels J. Visual similarity of pen gestures. In: Proceedings of the SIGCHI conference on human factors in computing systems (CHI '00), 2000.
[103] Rubine D. Specifying gestures by example. In: Proceedings of SIGGRAPH '91, 1991.
[104] Lee W, Kara L, Stahovich T. An efficient graph-based symbol recognizer. In: Proceedings of eurographics workshop on sketch based interfaces and modeling (SBIM '06), 2006.
[105] Alvarado C, Davis R. Sketchread: a multi-domain sketch recognition engine. In: Proceedings of ACM symposium on user interface software and technology (UIST '04), 2004.
[106] Sharon D, van de Panne M. Constellation models for sketch recognition. In: Proceedings of eurographics workshop on sketch based interfaces and modeling (SBIM '06), 2006.
[107] Hammond T, Davis R. Ladder, a sketching language for user interface developers. Computers and Graphics 2005;28:518–32.

[108] Agrawal R, Faloutsos C, Swami A. Efficient similarity search in sequence databases. In: Proceedings of international conference of foundations of data organization and algorithms, 1993.
[109] Naftel A, Khalid S. Motion trajectory learning in the dft-coefficient feature space. In: Proceedings of IEEE international conference on computer vision systems (ICVS '06), 2006.
[110] Chan K-P, Fu A. Efficient time series matching by wavelets. In: Proceedings of 15th international conference on data engineering, 1999.
[111] Wobbrock JO, Wilson AD, Li Y. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In: Proceedings of the 20th ACM symposium on user interface software and technology (UIST'07), 2007.
[112] Das K, Diaz-Gutierrez P, Gopi M. Example-based conceptual styling framework for automotive shapes. In: Proceedings of eurographics workshop on sketch-based interfaces and modeling (SBIM '07), 2007.
[113] Kara LB, Shimada K. Sketch-based 3d shape creation for industrial styling design. IEEE Computer Graphics & Applications 2007;27(1):60–71.
[114] Kara LB, Shimada K, Marmalefsky SD. An evaluation of user experience with a sketch based 3d modeling system. Computers & Graphics 2007;31(4): 580–97.
[115] Pereira JP, Jorge JA, Branco VA, Silva NF, Cardoso TD, Ferreira FN. Cascading recognizers for ambiguous calligraphic interaction. In: Proceedings of the eurographics workshop on sketch-based interfaces and modeling, Grenoble, France, 2004.
[116] Fonseca MJ, Ferreira A, Jorge JA. Towards 3d modeling using sketches and retrieval. In: Proceedings of the eurographics workshop on sketch-based interfaces and modeling, Grenoble, France, 2004.
[117] Pernot J-P, Guillet S, Lion J-C, Giannini F, Catalano CE, Falcidieno B. A shape deformation tool to model character lines in the early design phases. In: Proceedings of shape modeling international, 2002.
[118] Contero M, Naya F, Company P, Saorn JL, Conesa J. Improving visualization skills in engineering education. IEEE Computer Graphics and Applications 2005;25(5):24–31.
[119] Gross MD. The cocktail napkin, the fat pencil, and the slide library. In: Proceedings of association for computer aided design in architecture (ACADIA '94), 1994.
[120] Gross M, Do E. Drawing on the back of an envelope: a framework for interacting with application programs by freehand drawing. Computers and Graphics 2000;24(6):835–49.
[121] Leclercq P. Invisible sketch interface in architectural engineering. Lecture notes in computer science, vol. 3088/2004. BErlin: Springer; 2004. p. 353–363.
[122] Juchmes R, Leclercq P, Azar S. A freehand-sketch environment for architectural design supported by a multi-agent system. Computers and Graphics 2005;29(6):905–15.
[123] Cohen JM, Hughes JF, Zeleznik RC. Harold: a world made of drawings. In: Proceedings of the first international symposium on non-photorealistic animation and rendering (NPAR '00), 2000.
[124] Turquin E, Cani M-P, Hughes JF. Sketching garments for virtual characters. In: Proceedings of first eurographics workshop on sketch-based interfaces and modeling (SBIM '04), 2004.
[125] Wither J, Bertails F, Cani M-P. Realistic hair from a sketch. In: Shape modeling international, 2007.
[126] Bourguignon D, Cani M-P, Drettakis G. Drawing for illustration and annotation in 3D. Computer Graphics Forum 2001;20(3):114–22.
[127] Turquin E, Wither J, Boissieux L, Cani M-P, Hughes J. A sketch-based interface for clothing virtual characters, IEEE Computer Graphics & Applications 2007;27(1):72–81.
[128] Davis J, Agrawala M, Chuang E, Popović Z, Salesin D. A sketching interface for articulated figure animation. In: Proceedings of the 2003 ACM SIGGRAPH/ eurographics symposium on computer animation, 2003.
[129] Thorne M, Burke D, van de Panne M. Motion doodles: an interface for sketching character motion. In: Proceedings of SIGGRAPH '04, 2004.
[130] Chen B-Y, Ono Y, Nishita T. Character animation creation using hand-drawn sketches. The Visual Computer 2005;21(8–10):551–8 [Pacific graphics 2005 conference proceedings].
[131] Malik S. A sketching interface for modeling and editing hairstyles. In: Proceedings of the third EUROGRAPHICS workshop on sketch-based interfaces and modeling, Dublin, Ireland, 2005.
[132] Okabe M, Owada S, Igarashi T. Interactive design of botanical trees using freehand sketches and example-based editing. Computer Graphics Forum (Eurographics '05) 2005;24(3):487–96.
[133] Ijiri T, Owada S, Igarashi T. The sketch L-system: global control of tree modeling using free-form strokes. In: Proceedings of smart graphics '06. Lecture notes on computer science, vol. 4073, 2006. p. 138–46.
[134] Zakaria M, Shukri S. A sketch-and-spray interface for modeling trees. In: Proceedings of Smart Graphics '07. Lecture notes on computer science, vol. 4569, 2007. p. 23–35.
[135] Ijiri T, Owada S, Okabe M, Igarashi T. Floral diagrams and inflorescences: interactive flower modeling using botanical structural constraints. ACM Transactions on Graphics (SIGGRAPH '05) 2005;24(3):720–6.
[136] Ijiri T, Owada S, Igarashi T. Seamless integration of initial sketching and subsequent detail editing in flower modeling. Computer Graphics Forum Eurographics 2006;25(3):617–24.
[137] Anastacio F, Sousa MC, Samavati F, Jorge J. Modeling plant structures using concept sketches. In: Proceedings of 4th international symposium on non-photorealistic animation and rendering (NPAR '06), 2006.
[138] Anastacio F, Prusinkiewicz P, Sousa MC. Sketch-based parameterization of l-systems using illustration inspired construction lines. In: Proceedings of 5th eurographics workshop on sketch-based interfaces and modeling (SBIM '08), 2008.
[139] Joseph J, LaViola J. Sketching and gestures 101. In: ACM SIGGRAPH 2007 courses. ACM, 2007.
[140] DeCarlo D, Finkelstein A, Rusinkiewicz S, Santella A. Suggestive contours for conveying shape. ACM Transactions on Graphics 2003;22(3):848–55 [Proceedings of SIGGRAPH'03].
[141] Zhang R, Tsai P-S, Cryer J, Shah M. Shape-from-shading: a survey. IEEE Transactions on Pattern Matching and Machine Intelligence 1999;21(8):690–706.
[142] Prados E, Faugeras O. Shape from shading. In: Paragios YCN, Faugeras O, editors. Handbook of mathematical models in computer vision. New York: Springer; 2006. p. 375–88 [chapter 23].
[143] Wu T-P, Tang C-K, Brown MS, Shum H-Y. Shapepalettes: interactive normal transfer via sketching. ACM Transactions on Graphics 2007;26(3):44.
[144] Gingold Y, Zorin D. Shading-based surface editing. ACM Transactions on Graphics 2008: 27(3) [Proceedings of SIGGRAPH 2008].
[145] Sahami M, Mittal V, Baluja S, Rowley H. The happy searcher: challenges in web information retrieval. In: Proceedings of the 8th Pacific rim conference on artificial intelligence (PRICAI), 2004.
[146] Belhumeur PN, Kriegman DJ, Yuille AL. The bas-relief ambiguity. International Journal of Computer Vision 1999;35(1):33–44.