Joaquim Jorge
Faramarz Samavati
Editors

Sketch-based Interfaces and Modeling

Springer

Sketch-based Interfaces and Modeling

Joaquim Jorge • Faramarz Samavati
Editors

Sketch-based Interfaces and Modeling

Springer

Editors
Joaquim Jorge
Depto. Engenharia Informática
Instituto Superior Técnico
Universidade Técnica de Lisboa
Avenida Rovisco Pais
Lisboa 1049-001
Portugal
jaj@vimmi.inesc-id.pt

Faramarz Samavati
Dept. Computer Science
University of Calgary
University Drive NW 2500
Calgary, Alberta T2N 1N4
Canada
samavati@ucalgary.ca

ISBN 978-1-84882-811-7 e-ISBN 978-1-84882-812-4
DOI 10.1007/978-1-84882-812-4
Springer London Dordrecht Heidelberg New York

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Cover design: deblik

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Foreword

The field of sketch-based interfaces and modeling (SBIM) has had a long history. Since the early 1960s, which saw the birth of interactive computer graphics through Ivan Sutherland's Sketchpad and Jacks' DAC-1 system at General Motors, we have seen researchers developing methods and techniques to let users interact with a computer through sketching, a simple, yet highly expressive medium. Initially, SBIM was not a field in and of itself, but a set of distinct areas where researchers from different backgrounds worked in isolation, without a real community to share ideas. Areas within SBIM included sketch-based modeling, where the goal was to easily create 3D models, and sketch-based interfaces, where the goal was to develop systems for recognizing, for example, hand-writing, command gestures, 2D diagrams, and mathematics. Today, SBIM has emerged as a subfield of computer science that blends concepts from computer graphics, human-computer interaction, artificial intelligence, and machine learning and has brought the two areas of sketching—interface and model specification—together. This synergy was spearheaded by Joaquim Jorge and John Hughes, who started the first SBIM conference in 2004.

Over the years, SBIM has had some great successes (e.g., hand-printing and more recently cursive hand-writing recognition) as well as notable failures where the problem is still intractable in the general case (e.g., 3D sketch understanding). As with most of promising technology, it may take multiple decades for the technology to become mature enough to become viable. Speech recognition is a classic example of this, having taken more than four decades of research and productization before becoming commoditized, and SBIM is just starting to be mature enough to enable us to see that it can be used mainstream. Hand-writing and mathematical expression recognition and simple modeling tools like Google's SketchUp are some examples.

It is interesting to look at the history of using sketching to create graphical models and have the computer recognize hand-written text, mathematics, and diagrams. Any 2D visual language lends itself to sketch-based input, given that it is much easier to enter such languages (e.g., musical scores, mathematics or chemical molecule diagrams) by simply entering them with a pen or stylus than having to convert the language into a encoded 1D form entered on the keyboard. SBIM can trace its roots

not just to Ivan Sutherland's seminal Sketchpad but also to his brother Bert Sutherland's system for sketching out logic diagrams and to Robert Anderson's Ph.D. research at Harvard in the late 1960s on mathematics recognition and subsequent evaluation using the RAND tablet, the earliest predecessor to the digitizing tablets of today. It is interesting to note that the areas pioneered by the Sutherlands and Anderson still represent significant research problems today in both recognition and modeling. Fontaine Richardson's Applicon CAD modeling tool was the first commercial product to feature gesture recognition for model elements and commands using a digitizing tablet. There was relatively little research, let alone commercial exploitation, during the 1970s, although Negroponte's Architecture Machine Group at MIT did do some important work on recognizing architectural diagrams. In particular, in 1976 the SIGGRAPH papers by Weinzapfel on Architecture-By-Yourself and by Herot on the HUNCH system began to explore how computers could interpret hand-drawn diagrams and what inference mechanisms and domain knowledge were needed to do so.

In the 1980s and 1990s, we began to see a number of pen-based forerunners to TabletPCs and PDAs, as well as pen-based PC software appear in the market place, commercial implementations inspired by Alan Kay's Dynabook vision of the late sixties. These included Wang FreeStyle, Microsoft Pen Windows, Go's Penpoint, and Apple's Newton. The new devices showed that the commercial sector was starting to see the potential benefits of pen input and gesture-based interfaces. Unfortunately, essentially all these commercial efforts failed for various reasons such as inadequate computing speed and memory, insufficient battery life, and lack of sophisticated recognition technology. Despite these too-early attempts, digitizing tablets continued to be routinely used by artists and designers to create digital ink that remained uninterpreted (e.g., in painting systems) or as a substitute for the mouse with standard WIMP GUIs—robust character, symbol and gesture recognition, let alone sketch understanding, had to wait for more powerful hardware and recognition algorithms.

In the late 1990s we saw two seminal contributions in sketch-based interfaces for 3D modeling. The SKETCH system, developed by Zeleznik et al. in 1996, used a gestural interface and inferencing mechanisms to create 3D objects out of standard 3D geometric primitives such as cuboids, cylinders, and cones for conceptual 3D modeling. In 1999, the Teddy system, developed by Igarashi et al., let users make more free-form, organic 3D models. Both of these interfaces showed that sketch-based interfaces for this type of task is a very natural one since users could make rough drawings of the models they are interested in and have the computer interpret them to generate the 3D geometry. These systems led to a significant amount of new work on sketch-based interfaces for creating and manipulating 3D models.

In the last decade, we have seen an explosion of both sketch-based interfaces and pen-based computing devices. Better and faster hardware coupled with new machine learning techniques for more accurate recognition and more robust depth inferencing techniques for sketch-based modeling have enabled SBIM to enter a new era in research and development. This is one of the main reasons why this is a timely book: it provides us with a very useful collection of state-of-the-art technology from leaders of the SBIM field.

Although great strides have been made, there is still a lot to do to bring SBIM to the mainstream. Faster CPUs, better digitization technology, better battery life are just some of the areas that must be improved from a hardware perspective. More robust recognition algorithms that can handle subtle variability in user hand-writing as well as better depth inferencing in sketch-based modeling are still unsolved problems. Integration with other interaction modalities such as multi-touch and speech recognition to create multi-modal interfaces is now an important research area. Usability analysis of these interfaces is also critically important to advancing SBIM. The current book presents a snapshot of the state of the art in the area. I look forward to the advances that will be made in SBIM in the coming years and I hope that the readers will find inspiration in the valuable collection of articles gathered herein to stimulate their endeavors and advance this important field.

Brown University Andries van Dam

Preface

Sketch-based interfaces date back to Ivan Sutherland's pioneering work. SketchPad, a pen-based system, preceded the ubiquitous mouse by several years. However, SketchPad was too advanced for its day. For many years, this seminal work has remained more of a source of inspiration and awe than a trend to be followed.

Personal computers became sufficiently powerful in the nineties to support research in sketched-based interfaces in Interactive Computer Graphics and Human-Computer Interaction (HCI). People can now interact with drawings, editing and augmenting sketches in many different ways. Indeed, electronic drawings can be parsed and converted to digital objects such as pictures, diagrams and 3D models. Sketching can also be used for editing and animating these objects, a feature not possible on paper. Advances in this area provide the possibility of giving virtual life to simple sketches and effectively use computers to enhance creative thinking. Yet, for all its deceptive simplicity, sketching remains a hard challenge to meet for computer scientists. This is because sketches engage human intellect and abilities in ways that are difficult to approach with machines.

Thus, sketch-based interfaces are the subject of much lively research in recent years. Researchers from many disciplines have contributed to the body of knowledge on sketch-based interfaces. As such, it is difficult to gather a completely inclusive compilation of work done on this topic. Notably, sketching has become a recurring theme at many HCI conferences such as CHI, UIST, IUI, and AVI, and the IEEE Symposium on Visual Languages and Computing (VL/HCC). The Association for the Advancement of Artificial Intelligence (AAAI) held symposia on diagrammatic representations and reasoning, and sketch understanding. Additionally, the graphics community has usually published papers from this area, in conferences such as SIGGRAPH, Eurographics and the SMARTGRAPHICS symposium. Most notably, since 2004, the Eurographics Association has held a series of annual symposia on Sketch-Based Interfaces and Modeling (SBIM).

This book provides an overview of the topics covered in this emerging area of Interactive Computer Graphics, in two main parts. The first part contains chapters related to sketch-based interfaces and pen-based computing. The second part includes chapters about creation and modification of 3D models, covering the use of sketches in graphical and geometrical modeling. We aim to present a collection of

works representing recent developments in this area, within the scope of interfaces and modeling, hoping that this book proves to be a valuable resource for students, researchers and academics.

We would like to gratefully thank and acknowledge the many people who have assisted in the preparation of this book and reviewing its chapters.

Joaquim Jorge
Faramarz Samavati

Contents

Chapter 1
Introduction

Faramarz F. Samavati, Luke Olsen, and Joaquim A. Jorge

1.1 Sketch-based Interfaces

Sketch-based interfaces have come a long way since Ivan Sutherland's seminal work. Indeed, Sketchpad [24] spearheaded many techniques in both Computer-Science and Human-Computer Interaction, a system that not only improved on its predecessors but was also an improvement on many of its successors, to quote C.A.R. Hoare [10]. Each generation of sketch-based interfaces can be traced to different hardware devices that shaped their inception and evolution: the lightpen, the digitizing tablet and stylus combination, later the mouse, more recently tablet PCs and multitouch surfaces. These, in combination with the available platform computing power, largely shaped both research and commercial products.

Sketchpad featured an interactive system that allowed users to create engineering drawings using a lightpen, as shown on Fig. 1.1. The calligraphic interface combined sketched input with graphical constraints which were solved by the system to beautify the drawing without the need for explicit commands entered by the user. The GRAIL system, developed by RAND corporation [7] to work with the first tablet digitizing input device, allowed engineers to enter flowcharts using a combination of drawings, text recognition and pen gestures. GRAIL (depicted in Fig. 1.2) used

F.F. Samavati · L. Olsen
Dept. Computer Science, University of Calgary, University Drive NW 2500, Calgary, Alberta T2N 1N4, Canada

F.F. Samavati
e-mail: samavati@ucalgary.ca

L. Olsen
e-mail: ljolsen@ucalgary.ca

J.A. Jorge (✉)
Depto. Engenharia Informática Instituto Superior Técnico, Universidade Técnica de Lisboa, Avenida Rovisco Pais, Lisboa 1049-001, Portugal
e-mail: jaj@vimmi.inesc-id.pt

J. Jorge, F. Samavati (eds.), *Sketch-based Interfaces and Modeling*,
DOI 10.1007/978-1-84882-812-4_1,

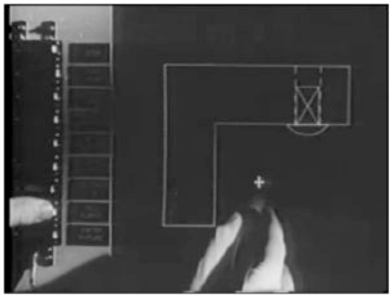

Fig. 1.1 The Sketchpad system in use. It is possible to see the lightpen and button pad on the left. Reproduced from http://www.archive.org

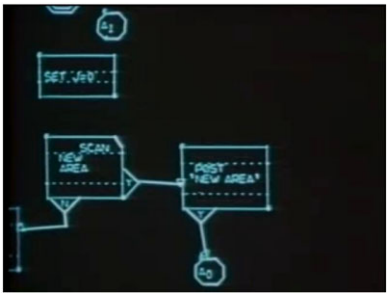

Fig. 1.2 A screenshot of the GRAIL system. Reproduced from http://www.archive.org

a combination of domain knowledge and contextual information to largely do away with the need for explicit commands, illustrating the power of Calligraphic User Interfaces.[1]

While the mouse was invented by Douglas Englebart in the mid-sixties, it did not see widespread use until the Apple Macintosh adopted it as a key component to its desktop user interface two decades later.

The topic of Sketch-Based Interfaces was largely dormant until the early nineties when the first pen-based computers appeared on the market. However, these early platforms were significantly underpowered to tackle handwriting and sketch recognition. By the end of the decade most tablet PC companies had gone out of business. Pen-based interfaces survived in the commercial marketplace thanks to PDAs such as the Apple Newton and the Palm Pilot.

More recently, the advent of multi-touch displays and tablet PCs that combine tactile and pen input has spurred new interest in Calligraphic Interfaces. Also, there is emerging research in three-dimensional applications of sketching in virtual immersive environments, especially in combination with other modalities.

[1] Calligraphic Interfaces, also known as Calligraphic User Interfaces, designate a family of computer applications organized around human-created drawings whether they are used to depict shapes, prepare designs, generate ideas, or simply to enter commands or depictions into a computer [14].

1.1.1 Sketching Issues and Research Topics in HCI

Sketching appears to be a concise, powerful and fast alternative to many conventional input modalities. Indeed, its strengths and weaknesses as a computer input modality come from the very same features. Through sketching people can express, interpret and modify shapes and relationships among drawing elements without much regard to neatness, alignment or precise measurement. However, the ambiguity and imprecision characteristic of free-hand drawings are also a major impediment to developing effective recognizers. Indeed, while other modalities, such as speech recognition and natural language, seem to be making significant strides, sketch recognition has yet to see widespread adoption as an unconstrained input technique. This is probably because the terseness and expressive power of sketches come at the cost of significant human higher cognitive abilities being involved, even more so than for other challenging recognition-based modalities. In their literature survey of sketching in design, Johnson et al. [13] indicate four main challenges to developing both useful and usable general-purpose sketch-based interfaces:

- Native hardware support for pen-based interaction
- Comprehensive robust toolkits for sketch-based systems
- User-friendly methods to train and model recognizable input
- Better interaction techniques for sketch-based systems

The forthcoming sections directly address many of these challenges with the possible exception of developing hardware support for Calligraphic User Interfaces. While hardware support is important, we have deemed it out of scope for this book, which focuses primarily on interfaces and applications proper.

Toolkits are important, but as a software engineering construct they are largely tied to GUI-style interfaces from the late eighties. It may be argued that novel software engineering techniques, methods and artifacts need to emerge to support the new generation of Calligraphic User Interfaces [14].

The third challenge is by and large being addressed by current and ongoing work in the community. Indeed, even if successful at first, hardwired recognizers are hardly the ultimate approach to recognizing sketch input. This is because the endless variations, rich vocabulary and inherent imprecision to user's input make it very difficult to devise simple techniques that satisfactorily handle most cases. Therefore extensible approaches are needed to augment the vocabulary and constructs of a recognition-based interface in powerful yet usable manners.

As for the fourth challenge, recognition-based user interfaces pose interesting problems to HCI researchers. Because sketch input can be ambiguous, the interface should approach it in a different way from the discrete, deterministic techniques so successfully applied to handling mouse and keyboard input. Further, resolving ambiguity can and should be delegated to humans, requiring good interaction design techniques to be brought to bear on the problem. Another very interesting issue in HCI for Calligraphic Interfaces is handling errors which cannot be ascribed to users. Handling these in a graceful if not creative manner is still a vibrant research topic to be addressed by the community.

While most research has focused on the early stages of design and modeling, there is a need for significant progress in terms of interfacing with existing CAD systems, applications and at the final stages of design where more detailed information is entered. Indeed, most of the current applications focus on ideation, whereas little thought has been given to make these ideas and shapes manufacturable.

1.1.2 Recognition

Sketch Recognition is central to Calligraphic Interfaces. This is because it allows applications to become organized around what users draw in a quasi-declarative way, instead of focusing on commands or constructive sequences as many traditional interfaces do. Sketch Recognition is related to both handwriting and gesture recognition, in the sense that it supersedes both. Plamondon and Srihari provide a comprehensive survey on handwriting recognition [21].

Contrary to handwriting or textual recognition, sketches have a non-linear syntax, in that meaning is ascribed to a drawing by looking at shapes and spatial relations rather than sequences, which is the main organizing principle in linear languages. Diagrammatic notations provide excellent means for expressing concepts due to the descriptive power of graphical symbols and spatial arrangements. Graphics Recognition is the subfield of Document Image Analysis and Recognition concerned with interpreting non-textual information present in document images. This field is important because many documents use a diagrammatic notation: i.e., architectural floor plans, mechanical drawings, electronic circuit diagrams, musical scores, flow charts, etc. In particular *sketches*, which roughly express abstract concepts, are a kind of diagram consisting of freehand line drawings. Because of their concise and expressive nature, sketches are a very effective communication mechanism. Thus, online recognition for sketching interfaces has elicited a growing interest among the HCI community [18].

Gesture recognition has been a heavily researched related area. While Ivan Sutherland's work was mostly concerned with a simple set of primitives, Rubine's recognizer allowed single stroke gestures to be learned and later recognized via a simple linear classifier [22], in one of the most cited papers in the field of gesture/symbol recognition. This is because recognition is at the heart of any system that handles sketching. In their comprehensive survey on sketching, Johnson et al. [13] identify some key issues in sketch recognition. These include when to recognize sketches (recognition can distract users from the task at hand); what symbols to recognize; how much of a drawing needs to be recognized; how to segment input—many drawings contain overlapping symbols or strokes. A key issue is how to group strokes in order to recognize what the user meant to draw. Recognizers need this information to perform adequately. Another important issue is what recognition strategy to apply. Many have been proposed over the years with varying degrees of success.

Training recognizers is also an important area. Most interfaces resorting to diagrammatic representations require a rather large vocabulary of symbols (or spatial

arrangements of strokes) that needs to be described if we are to develop algorithms or techniques to handle these. Instead of hard-coding these symbols as some have done successfully [8], many systems use a trainable recognizer. However, these need to be provided with good examples of what to recognize.

Another important issue is how to handle recognition errors. Recognition results are often inaccurate or ambiguous; or they return unwanted symbols. This comes both as a curse and an opportunity. Indeed, traditional interfaces assume that errors are somehow the user's fault and worry about how to best convey the appropriate messages to the user. In recognition interfaces, however, it is often best to let the user disambiguate or handle recognition errors in a constructive way to either correct the drawing or retrain the recognizer. Much research still needs to be done in this area.

Finally, two other important areas not addressed above are context and visual languages. Indeed symbols often change meaning depending on the context (other symbols surrounding them) or semantic domain. Both are related to visual languages. Whereas many syntax-driven approaches use some form of visual languages and parsing to extract meaning from diagrams [17] domain-specific knowledge often needs to be specifically coded and addressed at several stages of recognition. Indeed, providing multi-domain recognizers remains an interesting challenge.

A good example of recognition-level research that addresses many of these problems appears in Chap. 2, in which Christine Alvarado approaches multi-domain hierarchical free-sketch recognition using graphical models. We have selected this work because it touches many of the issues in sketch recognition discussed above.

1.1.3 Modes

According to WikiPedia[2] a mode is a "distinct setting within a computer program or any physical machine interface, in which the same user inputs will produce perceived different results than it would in other settings". Thus one important problem in most sketching interfaces is mode switching. Sketching user interfaces interpret input differently depending on which mode the program is in, for instance, drawing applications have input modes such as select, edit object, or input drawing, GUI programs often show which mode they are in by redundant means (cursor, selected entries on a palette, etc.). For example, the cursor will change shape to a pencil to indicate that users can draw when the pencil tool is active. Or it may change to a ruler to indicate that users may enter the corners of a rectangle. In both cases users can press a mouse button and drag the cursor. But the drawing program will parse user input in terms of the active tool. In sketch-based user interfaces sometimes users may not be aware of which mode the program is in, or may be unsure of how to activate the desired mode. Managing modes often distracts people from the task at hand by forcing them to focus on the syntax of the tool they are using rather than their work. This is a significant problem in Calligraphic User Interfaces whose functionality is not as self-disclosing as that of conventional desktop applications.

[2] http://en.wikipedia.org/wiki/Mode_(computer_interface), accessed October 2010.

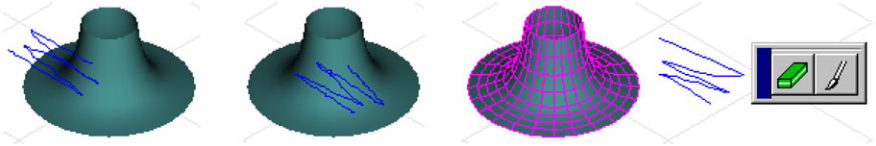

Fig. 1.3 Example of ambiguous input handling in GiDeS. On the left, a stroke which overlaps the outline of a screen object is interpreted as a delete command. In the middle, a stroke totally contained inside a screen object signifies recolor. On the right a scratch gesture totally outside a screen object becomes ambiguous: a menu appears on screen asking the user which of the two meanings should be assumed. Reproduced with permission from [15]

Applications resort to two different approaches to mode switching. In Sketchpad, users changed mode by operating physical buttons with the left hand, while entering drawings with the right hand. In contrast, GRAIL would infer the correct mode by looking at ink drawn by users in the context in which it was drawn. For example, a crossing gesture over a graphical entity would erase it. Text entered inside a box would become a label and a rectangle drawn on an empty area would be recognized as a box. This is fine as long as there are no ambiguous interpretations to a gesture. Some Calligraphic applications such as GiDES [15] handle this by exposing the ambiguity to users and letting them make choices, as can be seen in Fig. 1.3.

In Chap. 3, Eric Saund and Edward Lank approach the problem of minimizing modes in sketch interfaces by using an Inferred Mode protocol. They try to automatically recognize what mode the application should be in according to the user's input in the context of what has been drawn, to the extent that an action can be unambiguously determined.

1.1.4 Sketch-based Applications

There are many research applications that illustrate the power of Sketch-Based Interfaces. Among these we have selected four which address many of the issues highlighted above.

Many sketch-based interfaces parse diagrams to develop simulations of physical or abstract entities. In Chap. 4 Tom Stahovich describes Pen-based user interfaces for engineering and educational applications, in the mechanical and electrical engineering domains. The chapter contributes work on three fundamental sketch understanding problems. The first is pen stroke segmentation, to decompose a pen stroke into geometric primitives. The second, sketch parsing, clusters pen strokes or geometric primitives into groups representing individual symbols. Last, symbol recognition classifies symbols once they have been located by a parser. This chapter provides excellent insights ranging from the low-level details of stroke/ink processing to the high-level issues of symbol recognition and semantic analysis.

In Chap. 5, Joseph LaViola describes MathPad, a system for Mathematical sketching. Diagrams and illustrations are often used to help explain mathematical

concepts. Moreover, they are commonplace in math and physics textbooks and provide intuition into abstract principles. Unfortunately, static diagrams generally assist only in the initial formulation of a mathematical problem, not in its analysis or visualization. MathPad describes how to combine on-line recognition with a gestural interface to recognize mathematical formulas and associate variables and values with a physical simulation in order to enter, solve and visualize mathematical expressions. The author describes how to address modes by an ingenuous combination of location-aware gestures, imperative gestures and context in order to make modes largely invisible.

In Chap. 6 Michiel van de Panne and Dana Sharon present an interesting approach to Sketch recognition using flexible parts-based spatial templates. In automatic recognition of drawings for modeling, it is often difficult to describe what is to be recognized in terms of two-dimensional depictions of three-dimensional entities, especially if we want to afford a degree of flexibility to end-users. Their key insight is to use a 2D template for each class of object to be modeled. Templates provide explicit descriptions for optional parts, and thus constitute a compact and scalable approach for modeling many classes of objects as particular layouts of a collection of parts. This helps to avoid the combinatorial explosion that would otherwise occur, by explicitly modeling all possible combinations of parts that might constitute an object. The template structure also provides context for recognizing the parts themselves, making it easier to recognize those parts. Their system matches key points on a sketch to aid in top–down reconstruction, using a branch-and-bound search to identify the template (and corresponding three-dimensional model) that most closely matches a two-dimensional sketch. While their technique looks at first more limited than parsing and is constrained to the template database it provides a seemingly scalable approach to an open-ended problem, in that new templates can in principle easily be learnt from examples.

Finally, in Chap. 7 Manuel João Fonseca and Joaquim Jorge discuss Sketch-based retrieval of vector drawings, describing a Calligraphic User Interface to retrieve clip-art media using a structural approach. Their approach uses topological and geometric information automatically extracted from drawings to derive a multilevel description, which affords a coarse-to-fine comparison between simple sketches and complex vector drawings using a relational graph. Their approach avoids graph matching by using graph spectra as features in a scalable manner. They also show how this retrieval mechanism can be integrated into a 3D sketch-based modeling tools (GiDeS system) applying a paradigm of implicit retrieval, whereby sketched objects are automatically used as queries and returned results are presented as modeling suggestions at the user interface.

1.2 Creation and Modification of 3D Models

Model creation is a major bottleneck in production pipelines, involving complex and diverse shapes with intricate inter-relationships. User interfaces in modeling have traditionally followed the WIMP (Windows, Icons, Menus, Pointer) paradigm.

Though functional and very powerful, they can also be cumbersome and daunting to a novice user, due to numerous commands hidden under layers of functionality. Thus, creating complex models using computers can require considerable expertise and effort. Sketch-based interfaces have also been explored in this context, with the goal of allowing hand-drawn sketches to be used in the modeling process, from rough model creation through fine detail construction. However, mapping 2D sketches to 3D modeling operations is a difficult task, rife with ambiguity. SBIM applications for 3d modeling can be categorized according to how they 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.

A model creation system attempts to reconstruct a 3D model from the 2D sketched input. The gamut of creation systems can be divided into two categories, *suggestive* and *constructive*, based on whether or not the input strokes are mapped directly to the output model (in a suggestive system, they are not).

This aligns with the classical distinction between reconstruction and recognition. Suggestive 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, suggestive systems are akin to visual memory, whereas constructive systems are more rule-based [11].

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

1.2.1 Suggestive Systems

Suggestive-stroke systems are characterized by the fact that they have some "memory" of 3D shapes built in, which guides their interpretation of input sketches. 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 the suggestive-stroke category, two main approaches can be uses. In the first approach, the system extrapolates a final 3D shape based on only a few iconic strokes. A classical example is the SKETCH system of Zeleznik et al. [27], 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.

In the second approach of suggestive systems, a template objects from a database of template objects [9, 23] is retrieved. 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 the 3D template object.

1.2.2 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. All the techniques in the second part of this book propose constructive systems.

A common approach for constructive systems is to interactively reconstruct the object as the user sketches. 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 for creating "manufactured objects" is *extrusion*, a term for creating a surface by "pushing" a profile curve through space along some vector (or curve); see Fig. 1.4 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).

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 [11]. Accordingly, the majority of constructive SBIM systems choose to interpret strokes as contour lines (see Chaps. 8 to 12).

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. The skeleton is defined as the line from which the closest contour points are equidistant.

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 (see Chap. 10). A single stroke can also specify the contour, with either a fixed or user-sketched rotation axis to define the surface.

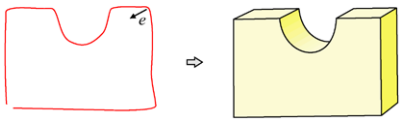

Fig. 1.4 Extrusion is a simple method for reconstructing a contour, by sweeping it along an extrusion vector *e*

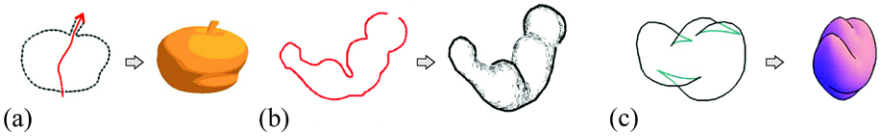

Fig. 1.5 Free-form model creation from contour sketches: **a** rotational blending surfaces have non-branching skeletons [5]; **b** Teddy inflates a sketch about its chordal axis (reproduced with permission from [12]); **c** SmoothSketch infers hidden contour lines (*green lines*) before inflation (reproduced from [16])

In Chap. 10, this idea is extended 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*). The system also allows the user to provide a third stroke, which defines a free-form cross-section, increasing the expressiveness of this construction.

Unfortunately, parametric surfaces—including surfaces of revolution—suffer from topological limitations. 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 (Chap. 8), for instance, inflates a contour by pushing vertices away from the chordal axis according to their distance from the contour; see Fig. 1.5b 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. [1], spherical implicit primitives are placed at each skeleton vertex; when the primitives are blended together, the result is a smooth surface whose contour matches the input sketch. ShapeShop (Chap. 11) instead uses variational implicit surfaces [25], which use the sketched contour to define constraints in the implicit function.

A different way to reconstruct a contour sketch is to fit a surface that is as smooth as possible. Surface fitting interprets 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 [26] push the system toward a solution. The FiberMesh system (Chap. 9) uses a non-linear optimization technique to generate smooth meshes while also supporting sharp creases and darts.

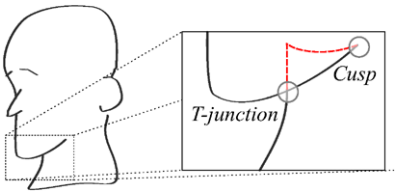

Fig. 1.6 The contour of an object conveys a lot of shape information. *Cutout*: T-junctions and cusps imply hidden contour lines (*red*)

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. Figure 1.6 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 [26] has proposed a method for using these to infer hidden contour lines in an image.

Karpenko and Hughes in Chap. 12 use Williams' algorithm, including support for not only T-junctions but also cusps. They take this 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.

1.2.3 Augmentation

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. 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 [4, 20] (see also Chap. 8) or curve-following slice deformations [28]. 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. 1.7a). The sketched lines may also be treated as new geometric constraints in surface optimization approaches.

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 [19, 20].

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

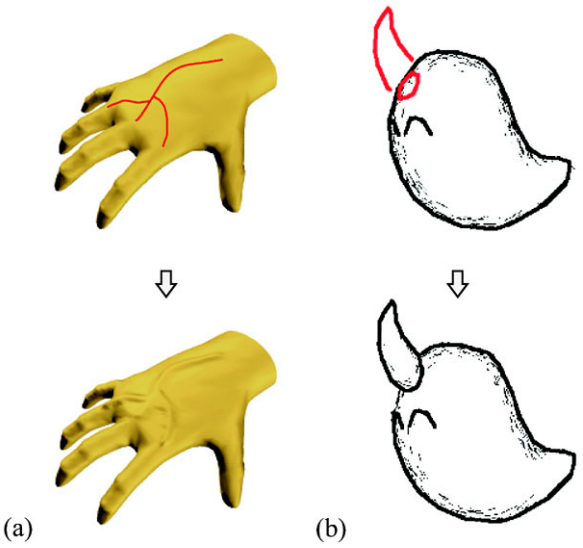

Fig. 1.7 Sketch-based augmentations: **a** surficial augmentation displaces surface elements to create features (from [20]); **b** additive augmentation joins a new part with an existing model (reproduced with permission from [12]). The latter figure also includes surficial features (the eyes)

to connect the new part to the original model (see Chaps. 8 and 9). For example, the extrusion operator in Teddy 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. 1.7b). ShapeShop (Chap. 11) exploits the easy blending afforded by an implicit surface representation to enable additive augmentation, with parameterized control of smoothness at the connection point. The 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 [28], which embeds 2D curve networks into arbitrary meshes and then displaces the mesh along these curves according to a sketched curve.

1.2.4 Deformation

Besides augmentation, there have been many SBIM systems (including those that have been described in this book) that support sketch-based editing operations, such as cutting, bending, tunneling (creating a hole), contour oversketching, segmentation, and affine transformations. And, like augmentation, sketch-based deformations

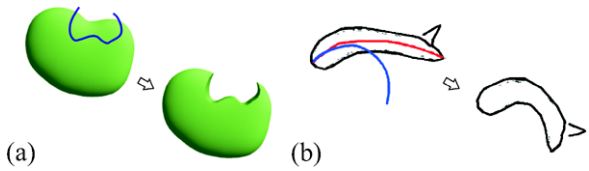

Fig. 1.8 Sketch-based deformations: **a** cutting strokes (*blue*) define a cutting plane along the view direction (Chap. 13); **b** bending a model so that the reference stroke (*red*) is aligned with the target blue stroke (Chap. 8)

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. 1.8a). 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. 1.8b. 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 [19].

In Chap. 9, a handle-based deformation is supported, 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.

Model assembly—typically an arduous task, as each component must be translated, rotated, and scaled correctly—is another editing task that can benefit from a sketch-based interface. In Chap. 10, a technique is proposed for applying affine transformations to a model with a single stroke. From a *U*-shaped transformation stroke, their method determines a rotation from the stroke's principal components, a non-uniform scaling from the width and height, and a translation from the stroke's projection into 3D. By selecting components and drawing a simple stroke, the model assembly task is greatly accelerated.

1.2.5 Modeling Applications

Knowing the nature of the model and the target application helps to infer the third dimension better and enhances the usability of the interface and the quality of the models in SBIM. There are many specific applications in which free-form sketch input is a very useful and powerful interface paradigm. The applications can be classified in two groups. *Computer-aided design* 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. In Chap. 13, a SBIM system is described for industrial product design. *Content creation* applications, meanwhile, are intended for modeling 3D objects that will exist usually 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. Dressing and hair-styling are two difficult examples in this area. In Chap. 14, several SBIM techniques are presented for designing cloth and hair for a virtual character.

This book provides an overview of the areas covered by Sketch-Based Interfaces and Modeling. We hope that through the discussions and examples provided herein readers will be motivated to further contribute to the research motives in this rapidly evolving and very exciting multidisciplinary area.

References

1. Alexe, A., Gaildrat, V., Barthe, L.: Interactive modelling from sketches using spherical implicit functions. In: Proc. of Int. Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa (AFRIGRAPH '04), pp. 25–34 (2004)
2. Autodesk Inc.: AutoCAD. http://www.autodesk.com/autocad
3. Autodesk Inc.: Maya. http://www.autodesk.com/maya
4. Biermann, H., Martin, I., Zorin, D., Bernardini, F.: Sharp features on multiresolution subdivision surfaces. Graphics Models (Proc. of Pacific Graphics '01) **64**(2), 61–77 (2001)
5. Cherlin, J.J., Samavati, F., Sousa, M.C., Jorge, J.A.: Sketch-based modeling with few strokes. In: Proc. of Spring Conference on Computer Graphics (SCCG '05), pp. 137–145 (2005)
6. Dassault Systemes: Catia. http://www.catia.com
7. Ellis, T.O., Heafner, J.F., Sibley, W.L.: The grail project: An experiment in man-machine communications. Tech. rep., RAND Corporation (1969)
8. Fonseca, M.J., Pimentel, C., Jorge, J.A.: Cali: An online scribble recognizer for calligraphic interfaces. Tech. Rep. SS-02-08, AAAI (2002)
9. Funkhouser, T., Min, P., Kazhdan, M., Chen, J., Halderman, A., Dobkin, D., Jacobs, D.: A search engine for 3d models. ACM Transactions on Graphics (Proc. of SIGGRAPH '03) **22**(1), 83–105 (2003)
10. Hoare, C.A.R.: Hints on programming language design. SAIL Computer Science Department TR CS-403, October 1973
11. Hoffman, D.D.: Visual Intelligence: How We Create what We See. Norton, New York (2000)
12. Igarashi, T., Matsuoka, S., Tanaka, H.: Teddy: A sketching interface for 3d freeform design. In: Proc. of SIGGRAPH'99, pp. 409–416 (1999)
13. Johnson, G., Gross, M.D., Hong, J., Do, E.Y.L.: Computational support for sketching in design: A review. In: Foundations and Trends in Human Computer Interaction, vol. 2. Now, Hanover (2009). doi:10.1561/1100000013
14. Jorge, J.A.P.: Parsing adjacency languages for calligraphic interfaces. Ph.D. thesis, Rensselaer Polytechnic Institute (1995)
15. Jorge, J.A., Silva, F.N., Cardoso, D.T.: Gides++. In: Proc. of the 12th Annual Portuguese Computer Graphics Meeting, pp. 167–171 (2003)
16. Karpenko, O.A., Hughes, J.F.: Smoothsketch: 3d free-form shapes from complex sketches. In: Proc. of SIGGRAPH '06, pp. 589–598 (2006)
17. Mas, J., Llados, J., Sanchez, G., Jorge, J.: A syntactic approach based on distortion-tolerant adjacency grammars and a spatial-directed parser to interpret sketched diagrams. Pattern Recognition (2010)

18. Narayanan, N.H., Hübscher, R.: Visual language theory: Towards a human computer interaction perspective. In: Visual Language Theory, pp. 85–127. Springer, Berlin (1998)
19. Nealen, A., Sorkine, O., Alexa, M., Cohen-Or, D.: A sketch-based interface for detail-preserving mesh editing. In: Proc. of SIGGRAPH '05, pp. 1142–1147 (2005)
20. Olsen, L., Samavati, F., Sousa, M.C., Jorge, J.: Sketch-based mesh augmentation. In: Proc. of the 2nd Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM) (2005)
21. Plamondon, R., Srihari, S.: On-line and off-line handwriting recognition: A comprehensive survey. IEEE Transactions on Pattern Analysis and Machine Intelligence **22**(1), 63–84 (2000)
22. Rubine, D.H.: Specifying gestures by example. In: Proceedings of the 18th Annual SIGGRAPH Conference on Computer Graphics and Interactive Techniques, pp. 329–337. ACM, New York (1991)
23. Shin, H., Igarashi, T.: Magic canvas: interactive design of a 3-d scene prototype from freehand sketches. In: Proc. of Graphics Interface (GI '07), pp. 63–70 (2007)
24. Sutherland, I.: Sketchpad: A man-machine graphical communication system. In: AFIPS Conference Proceedings, vol. 23, pp. 323–328 (1963)
25. Turk, G., O'Brien, J.: Variational implicit surfaces. Tech. rep., Georgia Institute of Technology (1999)
26. Williams, L.R.: Perceptual completion of occluded surfaces. Ph.D. thesis, University of Massachusetts (1994)
27. Zeleznik, R., Herndon, K., Hughes, J.: SKETCH: An interface for sketching 3d scenes. In: Proc. of SIGGRAPH '96. pp. 163–170 (1996)
28. Zelinka, S., Garland, M.: Mesh modeling with curve analogies. In: Proc. of Pacific Graphics '04, pp. 94–98 (2004)

Part I
Sketch-based Interfaces

Chapter 2
Multi-domain Hierarchical Free-Sketch Recognition Using Graphical Models

Christine Alvarado

2.1 Introduction

Consider the following physics problem:

> An 80-kg person is standing on the edge of a 3.6-m cliff. A 3-meter rope is attached to a point directly above his head, and on the end of the rope is a 40-kg medicine ball. The ball swings down and knocks the person off the cliff. Fortunately, there is a (padded) cart at the bottom. How far away from the cliff must the cart be placed in order to catch the person?

The above problem illustrates the central role of pictures and diagrams in understanding. It is almost impossible to read this problem without picturing the scenario in your head, and, for most people, the diagram is essential in solving the problem.

Because of the power of diagrams in thinking and design [8, 12, 56], people rely heavily on hand-sketched diagrams as a quick, lightweight way to put their ideas on paper and help them visualize solutions to their problems. Students, designers, scientists and engineers use sketches in a wide variety of domains, from physical (e.g., mechanical and electrical engineering designs) to conceptual (e.g., organizational charts and software diagrams).

Diagrams drawn on paper are just static pictures, but when drawn on a tablet computer, diagrams have the potential to be interpreted by the computer, and then made interactive. With the rise of pen-based technologies, the number of sketch-based computer tools is increasing. Sketch recognition-based computer systems have been developed for a variety of domains including (but not limited to) mechanical engineering [2, 21, 51], electrical engineering [16], user interface design [9, 34, 42], military course of action diagrams [11, 13], mathematical equations [33, 41], physics [39], musical notation [7, 14], software design [24, 36], note taking (Microsoft OneNote), and image editing [46]. In addition, a few multi-domain recognition toolkits have been proposed [3, 25, 35, 40].

C. Alvarado (✉)
Harvey Mudd College, 301 Platt Blvd., Claremont, CA, USA
e-mail: alvarado@cs.hmc.edu

J. Jorge, F. Samavati (eds.), *Sketch-based Interfaces and Modeling*,
DOI 10.1007/978-1-84882-812-4_2,

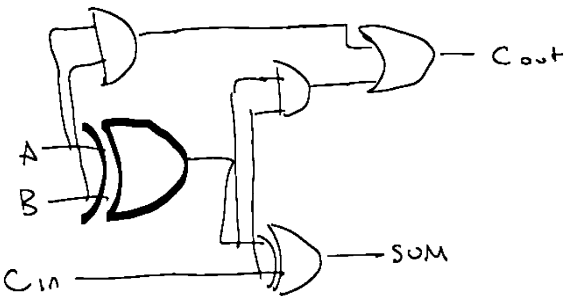

Fig. 2.1 A diagram drawn by a student in a digital design class (stroke thickness altered for illustration)

The problem of two-dimensional sketch recognition is to parse the user's strokes to determine the best set of known patterns to describe the input. This process involves solving two interdependent subproblems: stroke segmentation and symbol recognition. *Stroke segmentation* (or just *segmentation*) is the process of determining which strokes should be grouped to form a single symbol. *Symbol recognition* is the process of determining what symbol a given set of strokes represents.

Despite the growing number of systems, this two-dimensional parsing problem remains a challenging problem for a real-time system. Sketched symbols rarely occur in their canonical form: both noise in the sketch and legal symbol variations make individual symbols difficult to recognize. Furthermore, segmentation and symbol recognition are inherently intertwined. In the sketch in Fig. 2.1, if the system could correctly group the three bold strokes in this sketch, it likely could identify those strokes as an XOR gate using a standard pattern matching technique. Unfortunately, simple spatial and temporal grouping approaches do not work: the three strokes that form the XOR gate are not all touching each other, but they are touching the input and output wires. If the computer somehow can find the correct grouping, it probably will be able to match the strokes to a shape in its library. However, naïvely trying all combinations of stroke groups is prohibitively time-consuming.

Researchers have employed different techniques to cope with these challenges. Some of the systems listed above perform only limited recognition by design. ScanScribe, for example, uses perceptual guidelines to support image and text editing but does not attempt to recognize the user's drawing [46]. Similarly, the sketch-based DENIM system supports the design of web pages but recognizes very little of the user's sketch [42]. Systems of this sort are powerful for their intended tasks, but they do not support a the creative sketch-based design process in more complex domains.

Other recognition systems place restrictions on the user's drawing style in order to make recognition easier. We list four common drawing style restrictions that address these challenges, ordered from most restrictive to least restrictive, and give examples of systems that use each:

1. Users must draw each symbol using a pre-specified pattern or gesture (e.g., Palm Graffiti®, ChemPad [54]).
2. Users must trigger recognition after each symbol (or pause notably between symbols) (e.g., HHreco [28], QuickSet [11]).
3. Users must draw each symbol using temporally contiguous strokes (e.g., ACSPARC [16]).

4. Some systems place few restrictions on the way users draw, but rely on user assistance or specific domain assumptions to aid recognition. To trigger recognition in MathPad^2, for example, the user must circle pieces of the sketch [39]. The approach presented by Kara and Stahovich performs robust recognition of feedback control system diagrams, but relies on the assumption that the diagram consists of a number of shapes linked by arrows, which is not the case in many other domains [31].

While these previous systems have proven useful for their respective tasks, we aim to create a general sketch recognition system that does not rely on the drawing style assumptions of any one domain. This chapter describes a general-purpose recognition engine that can be applied to a number of symbolic domains by inputting the shapes and commonly occurring combinations of shapes using a hierarchical shape description language, described below. Based on these descriptions, we use a constraint-based approach to recognition, evaluating potential higher-level interpretations for the user's strokes by evaluating their subcomponents and the constraints between them. To achieve recognition robustness and efficiency, we use a combined bottom–up and top–down recognition algorithm that generates the most likely (possibly incomplete) interpretations first (bottom–up) and then actively seeks out lower-level parts of those interpretations that are still missing (top–down).

This chapter presents a synthesis of work presented in [3] and [4], as well as recent work that builds on this prior work. We begin by exploring the challenges of recognizing real-world sketches. Next, we present our approach to recognition, including how we represent knowledge in our system, how we manage uncertainty, and our method of searching for possible interpretations of the user's sketch. Next we analyze our system's performance on real data in two domains. We conclude with a discussion of the major remaining challenge for multi-domain sketch recognition revealed by our evaluation: the problem of efficient and reliable sketch segmentation. We present an emerging technique that attempts to solve this problem.

2.2 The Challenges of Free-Sketch Recognition

Like handwriting and speech understanding, sketch understanding is easy for humans, but difficult for computers. We begin by exploring the inherent challenges of the task.

Figure 2.2 shows the beginning of a sketch of a family tree, with the strokes labeled in the order in which they were drawn. The symbols in this domain are given in Fig. 2.3. This sketch is representative of drawing patterns found in real-world data [5], but it has been redrawn to illustrate a number of challenges using a single example. The user started by drawing a mother and a father, then drew three sons. He linked the mother to the sons by first drawing the shafts of each arrow and then drawing the arrowheads. (In our family tree diagrams, each parent is linked to each child with an arrow.) He will likely continue the drawing by linking the father to the children with arrows and linking the two parents with a line.