Sketching Interfaces

28

Tong Lu and Liu Wenyin

Contents

T. Lu (✉)
Department of Computer Science and Technology, State Key Laboratory for Novel Software Technology at Nanjing University, Nanjing, China
e-mail: lutong@nju.edu.cn

L. Wenyin
College of Computer Science and Technology, Shanghai University of Electric Power, Shanghai, China
e-mail: liuwenyin@gmail.com

D. Doermann, K. Tombre (eds.), *Handbook of Document Image Processing and Recognition*, DOI 10.1007/978-0-85729-859-1_31,
© Springer-Verlag London 2014

Abstract

This chapter introduces the problems and technologies involved for implementing the sketching interfaces. A number of typical applications of sketching interface are discussed. Conclusion is drawn by summarizing the current issues and further research perspectives.

Keywords

Annotation tool • Graphic document understanding • Online graphics recognition • Pen input • Sketch recognition • Sketching interface • Stroke segmentation • Symbol recognition

Introduction

Sketching on paper is a natural way of externalizing thought and is often used in early prototyping stages to express and record design ideas. Its popularity has not decayed even after computer-aided design becomes popular. The commoditization of increasingly numbered pen-based input devices indicates the trend against traditional WIMP (window, icon, menu, and pointer) paradigm. The recent multi-touch interactive displays also enable users to interact with complex models through multiple points of control and natural hand gestures. The models can be directly activated and tracked on the display. The multi-touch control provides not only a continuous freehand experience but also a more direct visual feedback. The traditional single-point click-drag operations are slowly being replaced by continuous strokes and freehand gestures. Touch Screen Palm PDA, Blackberry, iPhone, Amazon Kindle 3, Sony Reader PRS 700, and a variety of tablet and table PCs such as the Microsoft Coffee Table make accessing information faster, easier, and more natural to interact with. In such trend, sketching interfaces play an important role in human-computer interaction.

The input in such sketching interfaces is either the user's command or the graphical content. Both require interpretation and understanding by the computer. A command of the user is expressed by a sketching gesture and used to operate the computer. The content is the data the user wishes to input as part of the whole graphic document. Both gesture and content are composed of strokes, the basic format of sketch input. These strokes usually undergo a preprocessing procedure, including noise filtration and/or stroke segmentation, before further processing. Then, the computer should distinguish between gesture input and content input before applying specific recognition procedures. Such distinction can be done either with an explicit mode switch between the gesture mode and the content mode or with an automatic classification. Besides, the whole graphic document can be analyzed and understood after its completion or partial completion. In this chapter, we present related problems and technologies involved for implementing the sketching interfaces together with selected instances of sketching interface applications.

Online sketches, or more formally, online sketching documents, are a special kind of graphical documents. Their analysis usually undergoes a similar technical framework as other graphical documents, including offline graphical documents, CAD drawings/diagrams, and geographic maps. However, sketches have both online/dynamic features (with known trajectories and temporal information of the graphic objects) and sketchy features (in irregular/free-form shapes). In this chapter, we focus on these specialties. Readers interested in techniques for analysis and interpretation of other offline or vector-form graphical documents at a broader context are recommended to refer to ▸Chap. 17 (Analysis and Interpretation of Graphical Documents) of this handbook for more details.

Sketching System Overview

Sketching interfaces are usually supported by specific hardware systems with direct pointing devices. The typical devices and software systems which support sketching interface applications include personal digital assistants (PDAs, e.g., Apple Newton), graphic tablets (e.g., Wacom tablet), tablet PCs, and now the most popular iPads and iPhones. Such devices accept input using either special pens/styluses or just fingers.

Similar to other online recognition systems introduced in ▸Chap. 26 (Online Handwriting Recognition), the fundamental input format of sketching interfaces is a so-called stroke, which is a temporal sequence of $X - Y$ coordinates representing the pen trajectory, captured using a digitizer that senses the pen-tip position, speed, etc. Without loss of generality, let us first examine several common scenarios of applications of sketching interfaces.

1. While a user is drawing a stroke, it is continuously morphed to the predicted primitive shape as in Arvo and Novins's work [1]. We refer to this type of recognition as simultaneous recognition since the recognition is simultaneously and continuously done while the user is inputting. Limited by its computational cost, this type of recognition systems is suitable for simple graphics input. On the other stream, primitive shape recognition is performed on the completion of strokes. We refer to this type of recognition as immediate recognition since recognition is only done right after the stroke is completely drawn. These systems include QuickDiagram [2], and CALI [3]. Moreover, Fonseca et al. [3] also refer to this kind of interface for graphics input as calligraphic interfaces.
2. After a user draws a freehand stroke, the stroke is segmented into a few fragments. Each of them is fitted and/or recognized as a primitive geometric shape. This step is called preprocessing. In some systems, this step is omitted and the original strokes simply undergo certain subsequent procedures for recognition and understanding.
3. The primitive shape segments resulted from scenario 2 or even the original strokes are then categorized to either gesture input or content input. There are two genres of methods for this purpose: implicit mode detection and explicit mode switching. The explicit approach requires the user to specify the category

of his/her next input by designated operations, such as pressing a certain button. The implicit approach does not require the user to specify his/her intended input category but automatically determines it.

4. If the input is determined as a part of the graphic object/document, the system proceeds to online graphics recognition step. Various features (either statistical or syntactic/structural or combined) of the strokes are usually extracted for classification. Graphic objects are usually recognized online, which means the temporal (or even others, e.g., pressure and speed) information of strokes is utilized. If a stroke is determined as part of a previously drawn graphic object, it is combined with the previously inputted components according to their spatial relationship to form a query of similar instances in the model database. The retrieved candidates are ranked according to their similarity to the query and displayed for the user to select. The object selected by the user then replaces the input strokes on the screen. In this way, the user does not need to finish the entire object and can therefore save some of his/her time. This technique is particularly useful for graphics input on small screen devices, and it is also a type of simultaneous recognition. On the other hand, it is also possible to recognize a composite object after its completion, as in immediate recognition.
5. If the input is classified as a gesture, the system proceeds to gesture recognition procedure. Choice of features can be different from those of graphic object recognition. A typical example of gesture recognition is drawing an "X" gesture or some doodle lines (like using an eraser on a sheet of paper) on top of an existing graphic object which triggers delete operation of the graphic object. This is known as gesture recognition for editing. Many sketching systems, such as SILK [4] and CALI [3], embed such functions and gestures for common operations, namely, "delete," "move," and "copy." A well-known application of graphical gesture recognition is a pen-based interface for interactive editing [5] on both text and graphics.
6. After all the gestures and graphic objects are recognized, the system may perform understanding of the whole graphic documents in certain domain and output desired results or response.

As we can see from these scenarios, the most common, basic, and important application of sketching interfaces is inputting and editing graphic objects. Online graphics recognition is the key enabling technique for such kind of sketching interfaces. The overall workflow of the tasks in a sketching interface system is shown in Fig. 28.1. The remaining of this chapter focuses on introducing researches for each key procedure of the workflow. Then, we list the current advances of sketching interface applications. We conclude by delivering a thorough discussion on the current hindrances and perspectives of sketching interfaces.

Preprocessing

In online sketching interfaces, the strokes are normally sampled as time-stamped points. There are methods, e.g., Rubine's [6], which directly recognize the input

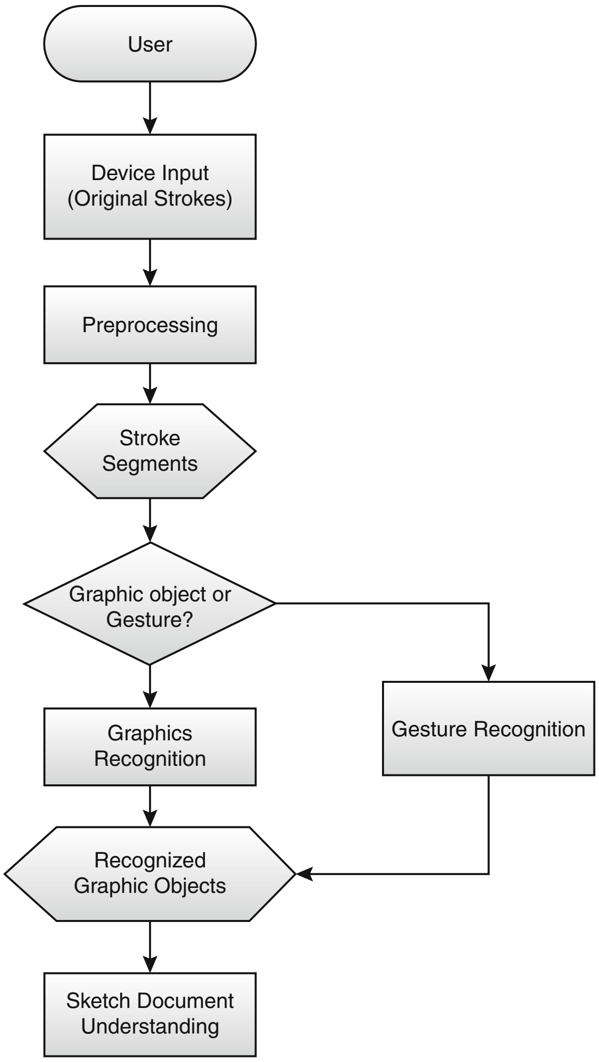

User
Device Input (Original Strokes)
Preprocessing
Stroke Segments
Graphic object or Gesture?
Graphics Recognition
Gesture Recognition
Recognized Graphic Objects
Sketch Document Understanding

Fig. 28.1 The framework of the procedures in a typical sketching interface system

stroke from the time-stamped points. These methods may work well for simple gesture recognition which will be discussed in section "Direct Sample Point Matching." However, direct recognition from time-stamped points is inefficient for composite online graphics, e.g., circuit diagrams. In this section, we will focus on existing methods for preprocessing input strokes into higher level primitive representations (i.e., denoising techniques, stroke segmentation techniques, and primitive fitting techniques) for further recognition and understanding processes.

Denoising

In order to enhance the robustness of recognition, noncritical points, agglomerate points (caused by shaky operations), and end points need to be removed, filtered, and refined, respectively. These extra points or noise points are detected by analyzing the density of the stroke points or the distance from the points to the desired/intended line segment. Denoising is essential if the recognition method used relies on accurate stroke segmentation, for example, hidden Markov model (HMM) proposed by Sezgin and Davis [7].

Stroke Segmentation

After noise reduction process introduced above, input strokes are normally segmented into sub-strokes and fitted into primitive shapes before conducting recognition. Existing methods of stroke segmentation can be categorized into local optimal approaches and global optimal approaches.

Local optimal approaches attempt to find breakpoints of a stroke, which is also referred to as splitting points. Most of these approaches first determine the support region for a stroke point. Then, within the support region, they calculate the significance of the point based on designated heuristics and use it as an indicator to identify splitting points. The heuristic for calculating the significance of a stroke point has been researched intensively. The local information, such as speed and curvature, of stroke points are utilized to detect splitting points more accurately [8]. The advantages of local optimal methods are that they are usually very fast and sometimes even parameter-free. However, their accuracy is limited due to the local optimality.

The second category is called globally optimal approach, which is also referred to as edge approximation. Subjective judgment criteria for perceptual significance, which are used to define a measure of perceptual error, are often introduced in this type of methods [9]. For every fragment of a stroke, a primitive shape with the minimal perceptual error is chosen to fit the original stroke. The advantage of this approach is that their accuracy is relatively high. However, almost all of them are not parameter-free; for example, some methods require the number of segments to be defined. Furthermore, the computational complexities of these methods are usually high.

Primitive Shape Fitting

The variety of primitive elements used for fitting the user sketches is also a well-studied topic. Ray and Ray [10] fit strokes using only line segments. This approach is usually fast in computation but may produce a large number of segments for a cursive stroke. Therefore, higher order curves such as circular arcs are used.

They can produce a more accurate representation at a cost of increased computational complexity. With the development of ellipse fitting [11], elliptical arcs are used to produce a more accurate representation. Many other approaches finish the process of curve fitting using a combination of line segments and conic arcs.

Gesture Recognition

In this section, we survey on the techniques used for distinguishing whether an input stroke(s) is a gesture (user command) or a part of the graphic document and what specific command it is for.

Methods for Distinguishing Between Gestures and Content Inputs

The methods used for distinguishing between gestures and content inputs can be classified into two major classes, namely, explicit methods and implicit methods. Explicit methods require users to explicitly indicate the type of the next input stroke. In implicit methods, the distinguishing task is done by the computer. The reason for this classification is that the former simplifies the recognition complexity at cost of degrading user experience, while the latter attempts to improve user experience at cost of increasing computational complexity.

Explicit Methods

Sketching interfaces developed using explicit mode-switching methods usually support two distinct modes: gesture mode and content mode. In content mode, the strokes are taken as content input. In gesture mode, the strokes are taken as commands. There are many ways of implementing the mode switch, for instance, a button. However, users may have to frequently switch between these modes, and an inefficient mode-switching technique may become a major bottleneck for user experience of the interface. As a consequence, simple and effective explicit mode-switching techniques could provide users with consistent mechanisms that are applicable across a wide variety of applications. Here we list the most commonly used explicit mode-switching techniques:

1. Barrel Button: This is the standard and frequently used technique in many existing pen-based applications [12]. Holding the barrel button (on the pen as shown in Fig. 28.2) down while drawing on the panel indicates a gesture input. Drawing without holding the barrel button indicates content input.
2. Press and Hold: This method requires a user to press the tip of the pen onto the tablet, hold it still until some mode change feedback appears, and then, while the mode change feedback still appears on the screen, the user can either lift the pen to bring up a pop-up menu or move the pen to draw a gesture. This technique leverages temporal information for switching modes, which is useful on devices where few input devices (including buttons) are available, e.g., a PDA or mobile phone. Once the pen touches the tablet, it enters the hold detection phase in which

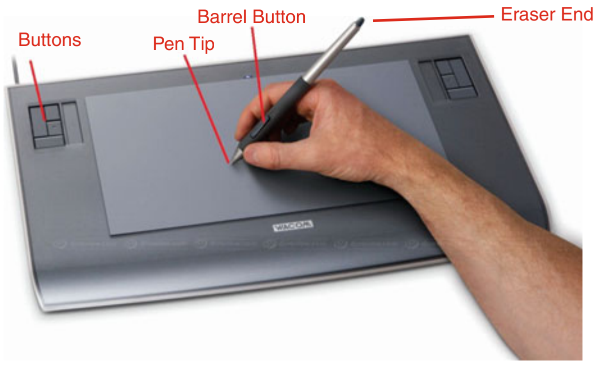


Fig. 28.2 A sample tablet

the bounding box size of the pen movement must be kept less than a pen travel threshold to be considered "still". The system developed by Wenyin and Kong [2] includes this feature.

3. Non-preferred Hand: This method requires two-handed interactions. The preferred hand (e.g., the right hand) is used for drawing strokes, and the non-preferred hand (e.g., the left hand) switches modes by pressing the corresponding buttons (e.g., the button on the top left of the tablet shown in Fig. 28.2 can be used for switching modes using the left hand, while the right hand is used for holding the pen). This reduces the task completion time by simultaneously carrying out two subtasks. Many researches (e.g., [13]) have been done to examine the efficiency of two-handed interactions. Also, Ruiz et al. [14] have proposed a model for non-preferred hand model switching.
4. Pressure-Based Mode Switching: Pen pressure sensitive inputs are available on many tablet devices. The LEAN system proposed by Ramos et al. uses pressure as a feature for gesture recognition. Furthermore, Ramos et al. indicate that dividing the pressure space into six levels or less produces the best user performance [15]. Under the assumption that content input occupies most of the user's time in a pen-based interface, they preserve the normal (middle) pressure space for content input and heavy spectrum for gesturing. They suggested a practical method for obtaining a suitable threshold value by using a small sample. They also suggested using personalized pressure setting to improve performance.
5. Using Pen with Eraser End: Some pens are designed with an eraser end (as shown in Fig. 28.2), and a user can use the eraser end of the pen for indicating gesture strokes [16].
6. Repeated Strokes: Since there are only two types of inputs, content and gesture, an alternative method for specifying input type is to repeat the strokes for one type of inputs. For example, the sketching interface proposed by Bae et al., EverybodyLovesSketch [17], uses the convention that repeated strokes are recognized as content input while otherwise a single stroke is recognized as a gesture. Alternatively, repeated strokes can be recognized as gestures for the interfaces where gesture inputs are less frequent.

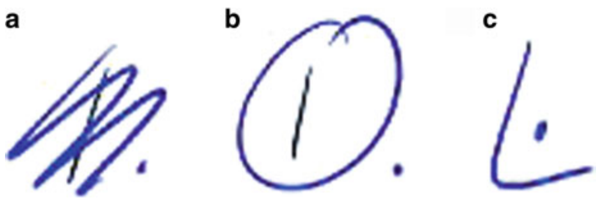


Fig. 28.3 Terminal punctuation. The *blue* strokes in (**a**), (**b**), and (**c**) correspond to two-stroke gestures (delete, select, and paste) (From [20])

The experiment carried out by Li et al. [18] shows that non-preferred hand method offers the fastest performance. Press and hold is significantly slower than other methods and prone to error. Pressure-based mode switching did not perform as well as non-preferred hand method but can be improved using a personalized pressure space setting. Explicit methods are suitable for professional users, like designers or artists, since the working efficiency can be achieved heavily depends on the users' experience on the mode selection methods. The mode selection method using non-preferred hand is an optimal choice in this category for enhancing working efficiency. However, it has a limitation of requiring both hands to corporate while working. Furthermore, explicit methods are usually hardware dependent as discussed above.

Implicit Methods

Implicit methods do not require mode switching during user's input. The computer is responsible for determining whether an input pattern (can be formed by multiple strokes) is a gesture or a content input. There are still no standard methods for implementing implicit mode switching since they often require more complex recognition algorithms and architectural design. Furthermore, the effectiveness of implicit mode switching is limited due to the limited computational power a PC can provide. However, as the processing power of a PC is rapidly growing, implicit mode switching for generic sketching interfaces will eventually become feasible. The major aspects we need to consider for implementing implicit mode switching are:

1. Terminal punctuation gesture: Saund and Lank suggested to use terminal punctuation (additional strokes after an input pattern) to indicate a gesture input [19]. For example, as shown in Fig. 28.3, the dots are drawn after main gesture patterns to indicate gesture input. Terminal punctuation gestures can also be used to parameterize the main gesture. For example, drawing a cross sign inside a lasso immediately after drawing the lasso or drawing an acute angle crossing the drawn lasso indicates cut operation on the region selected by the lasso (as shown in Fig. 28.4). However, this method is not commonly used due to two major challenges: (a) how to distinguish punctuated gestures from regular content stroke without special hardware and (b) How to provide feedback of the state of the system given that until punctuation is specified, the preceding input may or may not be one of many gestures.
2. Gestural shortcuts: Zeleznik and Miller [20] suggest to use "/" like stroke followed by a mnemonic word or letter to perform meta-functions such as displaying a menu or widget. For example, drawing "/" with a letter "U"

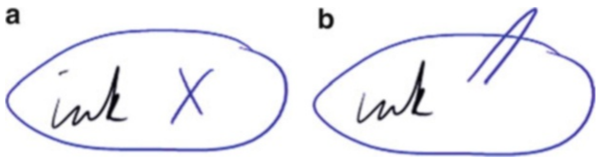


Fig. 28.4 Terminal punctuations indicating cut operation (From [20])

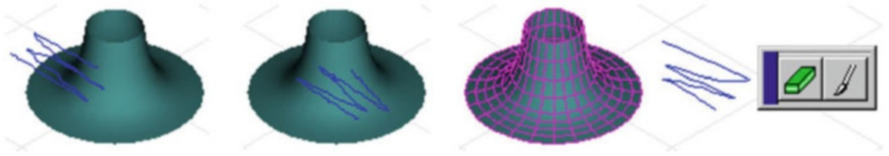


Fig. 28.5 Context-based recognition example [21]

brings up an undo button on the screen. However, we face similar challenges as discussed in terminal punctuation gestures.

3. Context-based recognition: In order to use the gesture vocabulary efficiently, the same gesture can be interpreted differently under different circumstances. As shown in Fig. 28.5, a doodle stroke which crosses the edge of a screen object is interpreted as a delete command on the leftmost picture. 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.
4. Multi-touch gestures and uni-touch content input: This approach is rarely implemented or experimented. The key advantage of using multi-touch gestures for command input and uni-touch gestures for content input is that the distinction between command input and content input can be made at the very beginning of the gesture. Therefore, more useful functions can be achieved; for example, touching the screen using two fingers immediately triggers drag command and the content displayed on the screen moves along with the fingers. For instance, Kurihara et al. [22] developed a whiteboard system which applies similar concept. Although many multi-touch devices have been commoditized and a considerable amount of effort has been made for developing multi-touch gesture recognition technology, researches based on empirical evaluation and theoretical support are still rarely seen. There have been attempts of formulating the description of multi-touch gestures [23] and the interaction techniques using multi-touch gestures [24]. Furthermore, a few approaches, e.g., HMM [25] and neural network [26], have been proposed for recognizing multi-touch gestures. However, there is still space for further research studies.
5. Other information such as speed may also be used to distinguish a gesture from a graphic object input. For example, quickly drawing a line across a graphical object may be interpreted as deleting the graphical object. Whereas, drawing the line slowly may be interpreted as additional graphical input. However, not many researches have been done on this topic.

Table 28.1 Comparison of the gesture recognition approaches

Type of approaches	Complexity of recognizable gestures	Training	Preprocessing	Relative recognition accuracy	Recognition speed
Hand-coded rules	Simple gestures	Not required	Not required	Acceptable for trivial cases	Quick for trivial cases but not acceptable for complex rules
Feature-based recognition	Dependent on the feature set	Required	Not required	Often high given enough samples	Dependent on the feature extraction and feature comparison time cost
Direct sample point matching	Simple uni-stroke gestures	Not required	Resizing and resampling of the input gestures	High for trivial cases	Quick
Probabilistic model	Accept complex multi-stroke gestures	Required and often computationally expensive	Stroke segmentation and primitive fitting are required	High given enough samples	Dependent on the complexity of the gestures

Recognition Algorithms

In this section, the commonly used approaches for gesture recognition are discussed. Table 28.1 demonstrates a comparison among the recognition approaches with respect to the metrics: (1) complexity of recognizable gestures, (2) training, (3) preprocessing, (4) relative recognition accuracy, and (5) recognition speed.

Hand-Coded Rules

Sketching interface systems recognize gestures using a great amount of different methods. Among these methods, rule-based ad hoc algorithms occupy the largest proportion of all the recognition methods at the early age. Those systems often empirically set several thresholds on the properties of the input gestures to perform simple hand-coded recognition. For example, in the work [27], the sketch system allows the user to draw a lasso surrounding a drawn object and then draw a “pigtail delimiter” to specify the desired operation among “cut,” “move,” “copy,” etc. As shown in Fig. 28.6, “pigtail delimiter” is detected by searching current pen stroke backwards over a 2,000 ms time window to look for an intersection line segment. If there is an intersection, the area and the perimeter of the polygon formed by the intersection of the stroke are calculated. Thresholds for those two parameters are empirically set to filter noises. Systems with gesture recognition method that lies in this category are usually complicated and difficult to implement, maintain, and modify.

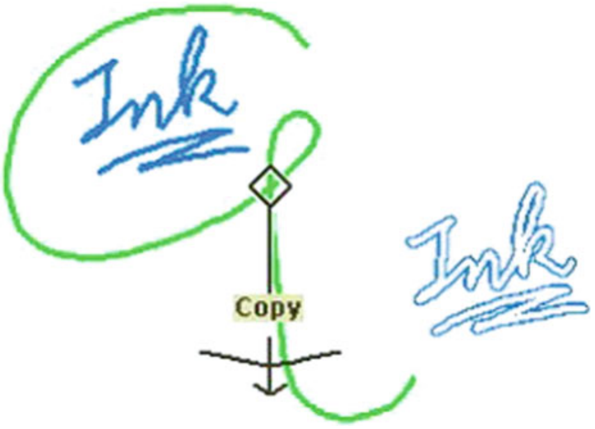


Fig. 28.6 A gesture called "pigtail delimiter" specifies "copy" command [27]

Feature-Based Recognition

A better way of recognizing gestures other than hand-coded methods is to extract a set of features and train a gesture recognizer. The merit of a recognizer is that it could enable users or developers to extend the gesture commands easily. Unlike rule-based hand-coded recognizers, well-designed feature extraction recognizers are more stable and more error tolerating. It can also shift the burden of correcting sketch recognition from the user to the system.

Rubine developed a gesture manipulation interface called "GRANDMA," which is short for "Gesture Recognizers Automated in a Novel Directed Manipulation Architecture" [6]. This is in literature the first attempt of developing a generic and extensible gesture management interface, enabling users to rapidly add gestures by training the single-stroke recognizer of the interface. The recognizer of the interface is feature-based single-stroke learner. The author decided to use 13 features extracted from the inputted training single-stroke gesture together with a linear discriminator for learning and classification. Normally, the recognizer requires 15–20 examples for training each gesture class. Then, given an input, the recognizer decides to which gesture class the input belongs. A reference implementation for Rubine's recognizer is developed by Myers et al. [28]. There are several sketch systems, such as [29], which incorporate Rubine's gesture recognizer.

Willems et al. [30] explored a large number of features for multi-stroke gesture recognition, and they presented a famous g-48 feature set. The detailed information of the feature set can be found in the appendix part of their work. With this feature set, the accuracy of tested classifiers (including multilayered perceptron, SVM, and DTW) can be significantly improved.

Direct Sample Point Matching

Online gestures are often sampled as time-stamped points as discussed in the preprocessing section. There are methods which match an input gesture with a defined gesture from point to point, for example, Protractor by Li [31]. This kind of methods first resamples the sample points of a raw gesture input as a fixed number, say N, of points which are equally spaced along the trajectory of the gesture. Then, the resampled gesture is rotated and scaled according to the designated heuristics of each specific method. The final step is to search for or compute from a closed form solution the optimal similarity between two gestures with respect to a defined similarity measure. This approach is easy to implement and often produce acceptable recognition accuracy for simple uni-stroke gestures. However, the computational cost grows rapidly as N increases which is inevitable for recognizing complex gestures.

Probabilistic Model

If we make the basic assumption that the nature of pen-based gestures is a stochastic process, we can model and recognize them using the probabilistic modeling scheme. Unlike the linear discriminator used in GRANDMA, which only accepts single-stroke gestures, probabilistic model allows much more intelligent search from the space of all possible interpretations of sketches. However, this is at the cost of increasing the computational complexity since the size of the interpretations grows exponentially as the number of strokes increases. The most popular probabilistic model is HMM, which is largely used in speech recognition and symbol recognition. Its improved versions are also successfully applied in many domains, like variable duration Markov model (VDHMM) used in handwriting recognition which is discussed in ▸Chap. 26 (Online Handwriting Recognition). An HMM assumes that the process being learned is composed of a finite sequence of discrete and stochastic hidden states. The states are assumed to be hidden but observable through features that the hidden states emit [32]. Anderson et al. [32] also state three classical problems associated with HMMs:

1. Classification problem: Given an observation sequence, which model is the most likely to have emitted particular sequence?
2. Most likely state sequence problem: What is the most likely sequence of hidden states which are believed to have emitted a given observation sequence?
3. Training or learning problem: Given one or more observation sequences, what are the model parameters associated which maximizing the likelihood of the observations?

There are quite a number of successful approaches for sketch gesture system using HMM or its variations as recognition scheme. Sezgin and Davis [7] presented semantic-based stroke recognizer modeled with HMMs. The recognizer views the process of sketching a gesture/symbol as an incremental, interactive process. By incremental it means strokes are put on the sketching surface of the sketch device one at a time. Its HMMs incorporate the sequential information of the strokes so that the sketching is highly stylized. For example, when sketching iconic people, most people tend to start from the head rather than other parts. This information is utilized and can drastically affect the interpretation of sketches.

Online Graphics Recognition

Online graphics recognition is referred to as the entire procedure of recognizing the desired graphic object while sketching is taking place. It may also be called real-time or dynamic symbol recognition. The recognition approaches introduced for online gesture recognition can be adapted to online graphics recognition. However, differing from online gesture recognition, online graphics recognition can be interactive. For example, a user does not need to complete the drawing before choosing the desired symbol from a candidate list suggested by the system [2]. A user may also make corrections after the system gives immediate feedback and thus avoid accumulating too many errors. Furthermore, the immediate feedback may also yield user adaptation. If a particular symbol keeps being wrongly recognized, the user can alter his/her sketching style to improve recognition accuracy. On the other hand, some recognizers are capable of adapting itself to suit a particular drawing style [33].

The procedure of online graphics recognition can be activated at different levels of detail, such as at the primitive shape level, and most commonly, at the composite graphics object level [34]. At the primitive shape level, the predicted primitive shape can be continuously morphed while a user is drawing a stroke [1]. This type of recognition is also referred to as simultaneous recognition since recognition is simultaneously and continuously done along with the user's inputting progress. However, it is mainly suitable for simple graphics input. This is because the recognition gets updated whenever a new sample point is inputted, resulting in high computational cost and slow response time for complex strokes. Hence, it is more common that the primitive shape is recognized only after a stroke is completely drawn.

In most situations, online graphics recognition is performed at the level of composite graphics objects, in which the basic processing unit is a primitive. After recognizing and regularizing the current stroke into primitives (each primitive corresponds to a segment of the stroke), it is desired to combine together the latest primitives with the ones recognized from previous strokes based on both their spatial and temporal relationships, determine or predict the type and parameters of the composite graphics object that the user is intending to input, and then redisplay (replace on the screen the freehand one) it in its regular form. The composite graphic objects are usually predefined in a model database.

Depending on whether the composite graphic object is recognized after it is completed or not, the recognition methods are grouped into two categories: partial graphics recognition and complete graphics recognition.

Partial Graphic Object Recognition

Partial graphic object recognition is referred to as the process that the system returns potential matches before the graphic object is completely drawn. In order to achieve this, recognition must take place while the graphic object is being drawn or, more

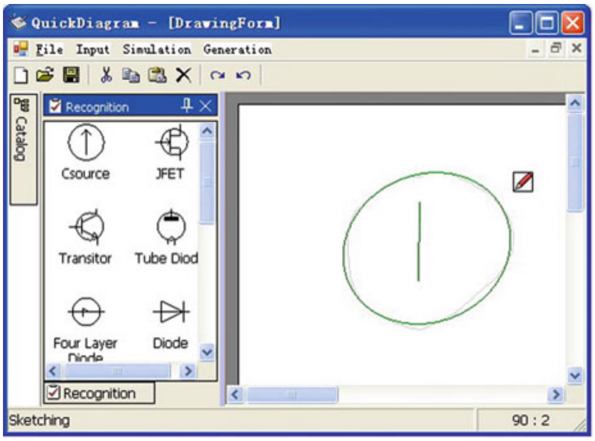


Fig. 28.7 Partial symbol recognition of QuickDiagram. Partial graphic input (composed by a *circle* and a *vertical straight line*) on the *right* is recognized as the potential matches on the *left*

precisely, after each intermediate stroke is drawn. There are some sketch interfaces supporting partial graphics recognition. Partial similarity is calculated to search for the user-intended object. In order to achieve efficient recognition and prediction of the partially drawn object, the search space is usually reduced by temporal or spatial constraints.

A number of partial graphic object recognition systems rely on temporal constraints to prune the search space. Lank et al. [35] develop a UML system, assuming that objects are drawn using consecutive strokes. Other systems limit the search space by employing greedy recognition strategies [36]. An example of this continuous prediction process from QuickDiagram [2] is shown in Fig. 28.7. While these greedy recognition methods do not limit the graphic objects to being drawn consecutively, they may create undesired interdependencies among objects. Besides, dynamic information is also used to aid recognition; e.g., Sezgin and Davis [37] rely on preferred stroke orders and build an HMM to recognize each symbol.

Many partial graphics recognition systems use spatial constraints to limit their search. Most of the works consider only groups of strokes that locate within a spatial bound to be the same object [38, 39]. To further prune the search space, many systems set a hard threshold on constraints between sub-shapes [38, 40]. Few combinations need to be considered for recognition. However, it is difficult to select a suitable threshold.

Existing researches on online graphics recognition focus more on complete graphic object recognition and only assess the recognition accuracy for completed graphic objects. Partial graphic object recognition has drawn much less attentions, let alone its recognition accuracy or prediction performance or efficiency.

Complete Graphic Object Recognition

There are mainly three categories of approaches to this problem: rule-based approaches, machine learning approaches, and similarity-based approaches.

1. Rule-Based Approaches

   An intuitive approach to graphics recognition is to build a decision tree where each leaf represents a class of graphics objects and each edge represents a rule to classify to branches. However, this approach suffers from its poor extensibility. The tree structure must be updated or redesigned when adding new classes of graphic objects. There are some improvements of these rule-based approaches to enhance the adaptability. Sun et al. [41] also adopt an incremental decision tree induction method to construct dynamic user models to provide user adaptation.

2. Machine Learning Approaches

   The recognition of graphic object can be regarded as a classification problem and solved by machine learning algorithms, such as neural networks (NN) [42], HMM [7], and genetic algorithm (GA) [43]. Hybrid methods are also used, e.g., NN/HMM approaches. The advantage of these machine learning approaches is that they are robust to noise and can be easily extended, while the disadvantage is that pretraining is required before the classification process. Also, the performance of the classifiers is greatly affected by the size of the training set and the selection of the training samples.

3. Similarity-Based Approaches

   In similarity-based approaches, similarities between two graphic objects are defined and calculated. Usually, the graphic object is represented in the form of graph [44]. In general, these graphs are matched and the most similar graphic object is selected from the database as the recognition result. These approaches are easy to extend and do not require a large amount of training data. However, they usually require a high computational cost and have been proven to be NP complete. Furthermore, these approaches are sensitive to the noise and, hence, not robust.

   ▸Chapters 17 (Analysis and Interpretation of Graphical Documents) and ▸16 (An Overview of Symbol Recognition) in this handbook deal with the analysis and interpretation of general graphical documents including CAD drawings, electronic diagrams, and geographic maps, most of which are offline data. These kinds of data undergo a similar recognition framework as sketching data does, though detail recognition methods may be different.

Graphic Document Understanding

Understanding of a sketchy graphic document is similar to those regular graphic documents after the sketchy graphic objects and their relations are recognized. A graphic document (or a diagram/drawing) is usually represented using a graph. Symbols (or graphic objects) are represented by nodes and are linked by their relations. Properties of the graphic objects are represented by the attributes of the nodes. Links (edges in the graph) can also have certain properties/attributes.

The key issue in graphic document understanding is how to represent and store such graph configuration and relevant domain knowledge in a sketching system. There are quite a number of categories of domain knowledge representation, and in this section, we only present a few common examples.

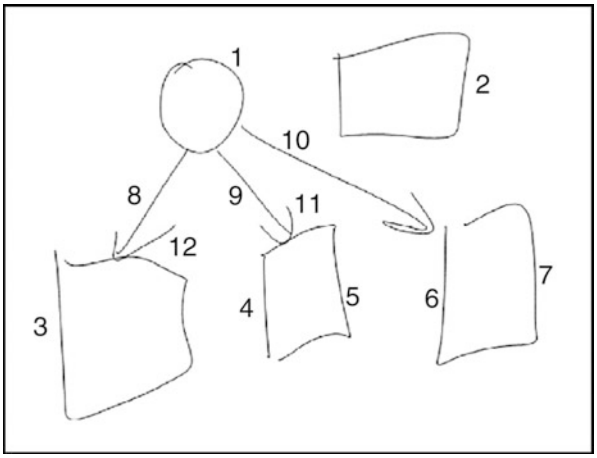


Fig. 28.8 Snapshot of the family tree understanding in [47]

1. Rule-based hand coding: This category may have the most number of applications. These systems usually combine the graphic document representation with domain knowledge using hand coding. This is the most intuitive way of document understanding since it only needs a number of hand coded rules to represent the domain knowledge and configuration of the graph representation. Although the systems that adopt such method are easy to design, there is almost no room for extension, e.g., to learn a new domain of symbols.
2. XML knowledge representation and understanding: One example of such category is [2]. In this work, not only the graph representation of the document but also the constraints for circuit and the symbol configuration are defined and stored using XML knowledge representation. When the user would like the system to understand a new set of symbols, he/she only needs to define the symbols' XML representation using symbol developing interface. The user can also conduct certain analyses of the graphic documents, with the help of the XML representation of the graphic document.
3. Constraint programming language: Generally, constraint-based techniques enable the user to program the interactive behavior of a graphic document. In computer-aided design, a constraint-based graphic editor has a distinct advantage over a conventional drawing program: a user can state desired relations and the system ensures that they are maintained. As early as "Sketchpad" [45], constraint programming language is used to maintain spatial relationship among the drawn symbols. Alvarado and Davis [46] represent a shape's definition in a language called Ladder. As Fig. 28.8 shows, Ladder descriptions list the basic components making up the shape and the geometric constraints that must hold between them. Recognizing individual shapes is then a process of matching the strokes drawn against a set of shape descriptions for a domain. The description is significant both for what it constrains and for what it does not constrain. For example, to be an arrow the shaft must be longer than the heads. It does not specify the lines'

orientation or length, letting the system recognize arrows of any size and in any orientation.

In the next section, we will also show the graphic document understanding aspects of these systems.

The techniques for understanding sketchy graphic document fall into the category of "hybrid approaches for interpretation" in section "Hybrid Approaches for Graphics Analysis" of ▸Chap. 17 (Analysis and Interpretation of Graphical Documents) in this handbook, where both components/elements recognition and domain knowledge are used to reach an overall understanding of the whole graphical document. Readers may also refer to ▸Chap. 17 (Analysis and Interpretation of Graphical Documents) for a broader discussion of analysis and interpretation of other kinds of graphical documents, including CAD drawings, electronic diagrams, geographic maps, and mechanic charts, most of which are offline data.

Applications

With the rise of pen-based and touch-based technologies, the number of sketch-based tools and interfaces is increasing. A variety of domains are benefiting from the sketch interfaces, including annotation and editing, user interface design, sketch-based modeling, diagram recognition, and mathematical equations. In addition, a few multi-domain recognition toolkits are also reported.

Sketch-Based Annotation

Annotation is one of the most natural applications of sketching interface, since sketching is popularly used as an annotation tool to facilitate information understanding. For example, they can help clarify basic assumptions or specify technical details. This section gives a brief overview of sketch-based annotation tools and the corresponding techniques.

A typical sketch-based annotation system needs to provide all the aspects of a sketching system (including both recognition and understanding) and should provide means for the following: (1) freely selecting contents to annotate, such as graphic symbols or textual words; (2) adaptively adding sketchy annotations to the selected contents; and (3) conveniently establishing relationships among the selected elements and the sketchy annotations.

2D Annotation

Kirchhoff's Pen [48] is a study system that provides tutoring for Kirchhoff's voltage law (KVL) and current law (KCL) using sketching technologies. The circuit annotations are shown in Fig. 28.9. The goal of Kirchhoff's Pen is to figure out the circuit formulas related to KVL or KCL and verify them with students' answers. As shown in Fig. 28.9, the annotations are sketched freely by users, making it difficult

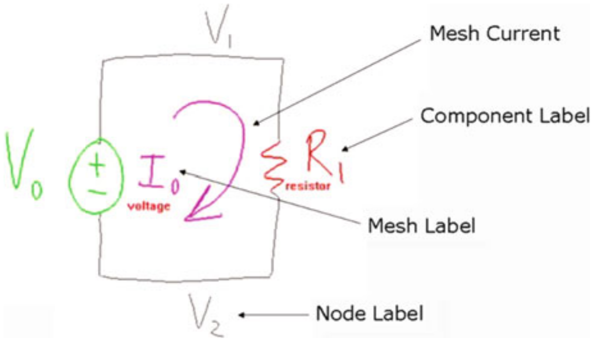


Fig. 28.9 Typical circuit annotations in Kirchhoff's Pen

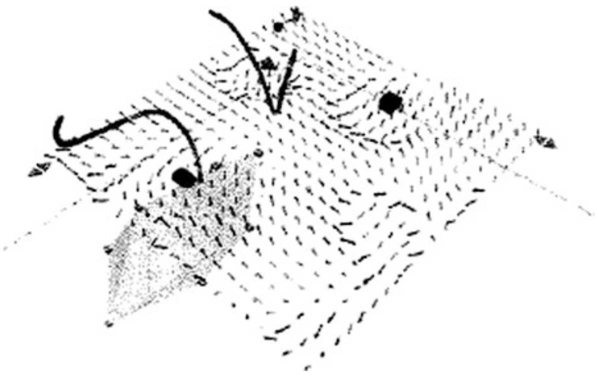


Fig. 28.10 The example of 3D annotation [49]

to automatically identify the associated object for each annotation or extract the relationships among the annotations.

In the identification process, Kirchhoff's Pen translates the sketched text annotations into characters by a handwriting recognizer and a special-purpose recognizer, simultaneously avoiding the misclassifications of special pairs such as "I"-"1" or "\"-"/." The handwriting recognizer produces a ranked set of alternative interpretations for each character, and the highest-ranked choice that is consistent with their problem domain is selected. After identifying the annotated text characters, Kirchhoff's Pen associates them with their neighboring circuit symbols to obtain annotated labels.

3D Annotation

How to annotate in a 3D scene is an interesting problem. Loughlin and Hughes [49] proposed a system that supports annotation as an integrated part of a fluid flow visualization system, as shown in Fig. 28.10. Unlike typical annotations on static 2D images, this system embeds annotations in 3D data space. This immersion makes it easy to associate user comments with the features they describe. To avoid clutter and data hiding, annotations are represented by graphical annotation markers that have associated information. Therefore, graphical attributes of the markers, such as size and color, can differentiate annotations with different functions, authors, etc.

Annotations can be easily added, edited, and deleted in the 3D space. Moreover, annotations can be simultaneously loaded into a visualization approach. This allows scientists, collaborating on a dataset, to use annotations as a form of communication, as well as a history of data analysis sessions. Annotation markers also aid scientists in navigating through the data space by providing landmarks at interesting positions.

The annotations in their system are actually represented by small geometric markers in the 3D data space. Each marker has an associated content which the user can edit at any time. The geometry of a marker gives visual feedback on the content of the annotation. Interactive shadows could be added to the markers on the planes defined by the principal axes. Each shadow is constrained to move in the plane in which it lies. If a user moves a shadow, the marker moves in a parallel plane. This constrained translation helps to precisely position a marker. Shadows are used to highlight specific zones interested by annotators.

The features of the points highlighted by geometric markers are later recognized and stored. Since the features of unsteady fluid flows change over time, the annotations are relocated by recognizing the features of the annotated points. The time-varying annotations are supported by feature-characterization code to improve the annotation effects. The feature-characterization code is the specifications of the features of annotations found.

Sketch-Based Diagramming

Sketch-aided diagramming is used in different domains, such as circuit diagram, and UML diagram, to provide a natural and intuitive interaction environment. In this section, we will briefly talk about several typical examples of sketch-based diagramming.

Circuit Diagram Analysis

Sketch-aided circuit diagram drawing and analysis is not a new subject to researchers. There are several systems and applications that can be found in literature.

AC-SPARC

AC-SPARC is developed by Gennari et al. [50]. The name of this system is the abbreviation for analog circuit sketch pArsing, recognition, and error correction. As Fig. 28.11 shows, users are expected to draw network-like diagrams consisting of symbols, using mouse or digitizing tablet and stylus, and link them together. A parser automatically extracts geometric information from continuous stream of strokes. Candidate symbols of recognition are pruned using domain knowledge, while they are classified with domain independent, probabilistic, feature-based symbol recognizer. Error correction carried out automatically is aided by the domain knowledge and the context within the sketched diagram. For electronic diagrams, SPICE code can be generated for further analysis.

Following the standard procedures of sketch recognition system, AC-SPARC starts with the "ink segmentation" in its architecture of interpreter. As the user

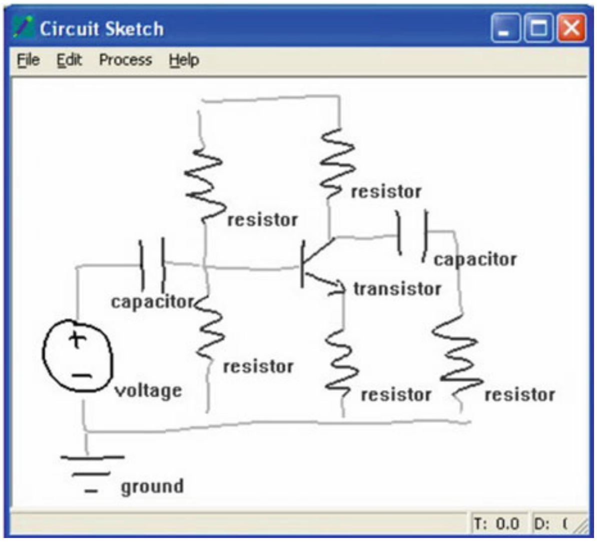
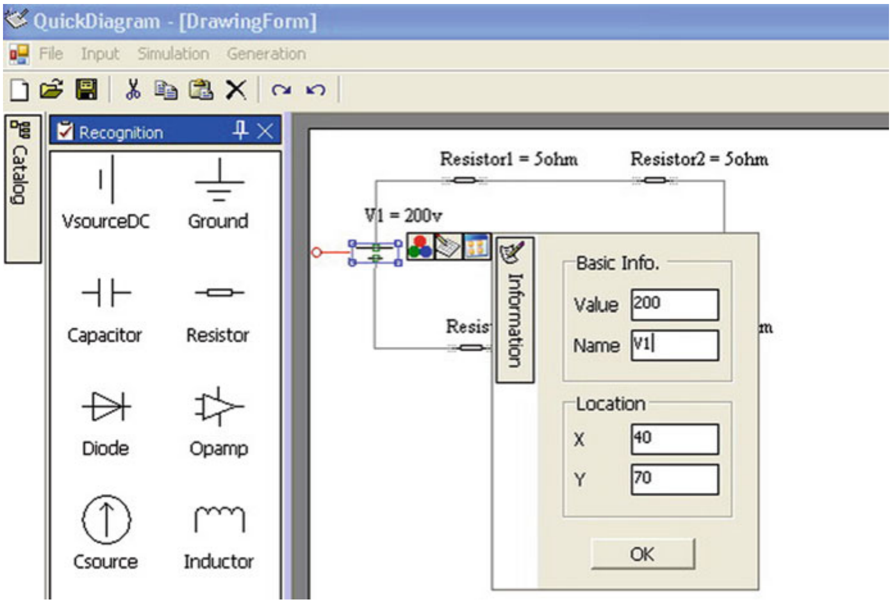


Fig. 28.11 AC-SPARC sketch interface and interpretation result

draws, pixels of the sketches are segmented into lines and arcs, which serve as the geometric "primitives" for recognition. Stahovich's algorithm for segmentation [8] is chosen in AC-SPARC.

Given a set of primitives, it is necessary to combine the right ones to form candidate symbols while distinguishing the link primitives. AC-SPARC does this by employing two locators and identifying the starting and ending of one symbol's drawing. One locator is the "ink density locator" which calculates the density of each primitive among its bounding box. Drawn time of the primitives is also recorded in order to include the imaginative link. Density series are calculated by accumulating primitives and the indication of the end point is that the addition of other segments causes decrease of density. This step is repeated but is started from the end point to find the starting point. The other locator is the segment difference locator. It calculates four characteristics of each primitive, and if two primitives have more than two (or a larger number as a predefined threshold) different characteristics, they are considered as segment difference point. A symbol is likely to be bounded by two-segment difference point, having more than two but less than the user-defined maximum number of primitives.

Candidate symbols are pruned according to domain-based knowledge. For electronic symbols, particular rules can be developed, such as the ink density is high, two primitives touching the candidate symbol are more or less collinear, enough primitives or at least a full circle is generally contained, and so on. After pruning, symbol recognition is performed. Symbol recognition in AC-SPARC involves two stages as training and classification. Firstly, it extracts nine geometric features of symbols, and secondly, it extracts nine types of geometric features and performs classification using a naive Bayesian framework. The classifier is invariant to rotation, scaling, translation, and order of strokes.



Value: 200
Name: V1
X: 40
Y: 70

Fig. 28.12 Property dialog box for a device where its name and parameter value can be modified

Finally, automated error correction and some predefined and hard-coded rules for the circuit are used; for example, the number of the connection of a symbol should be 2 or 3. The system will either automatically correct the detected errors or give the user a warning.

QuickDiagram

QuickDiagram [2] is a system for quick diagramming input and understanding. With a user sketching a (complete or partial) component/symbol or a wire (connecting two components) of the diagram, the system can recognize and beautify it immediately. After the entire diagram is finished, certain understandings can be obtained. Especially, the following two methods are used to interpret the recognized diagram: (1) nodal analysis on resistive circuits and (2) generation of PSpice codes from the recognized diagrams.

QuickDiagram can understand certain circuit diagrams, in which each device is modeled in its regular form with its connection points also marked. After recognition, each device is assigned with a default name automatically numbered and displayed beside the device in the diagram. The default value of its parameters, e.g., R1 = 5 Ω, is also assigned and displayed next to its name. Its name and value can also be modified in its property dialog box, as shown in Fig. 28.12. The connection points of the devices are also numbered such that each connection point in the entire diagram has a unique name. After all the individual devices and their connections are recognized, the entire diagram is displayed in a neat form prior to further analysis and understanding.

UML Diagramming

Another popular domain for sketch-based diagramming is UML diagramming.

Dachselt et al. [51] present the usage of digital pens and paper in conjunction with sketch-based UML tools to provide a flexible spontaneous sketching. Their work deals with the seamless integration of pen-based and digital UML sketching. It can be extended to use with tabletops.

Chen et al. [52] present a UML tool called SUMLOW that allows user to draw UML constructs, mix different UML elements, and annotate diagrams and input text. A key novelty of the tool is the preservation of hand-drawn diagrams and support for manipulation of these sketches using pen-based actions. Sketched diagrams can be automatically "formalized" into computer-recognized and beautified UML diagrams and then exported to a third-party CASE tool for further extension and use. They describe the motivation for SUMLOW, illustrate the use of the tool to sketch various UML diagram types, describe its key architecture abstractions and implementation approaches, and report on two evaluations of the toolset. Their experiences are useful for others developing sketching-based design tools or those looking to leverage pen-based interfaces in software applications.

Sketch-Based Design and Modeling

There are various fields of designs that can be improved both on performance and efficiency if they are aided by sketching interfaces or systems.

User Interface Design

Benefiting from the flexible nature of sketch drawing, commands that are made using sketches for computer control are more intuitive and more comfortable. This is particularly evident during the interaction between human and robot, especially under urgent circumstances, such as search-and-rescue scenarios. Shah et al. [53] developed a probabilistic command interface for controlling robots using multi-strokes sketch command. Their system allows the user to draw ten gestures to indicate his/her commands, as Fig. 28.13a illustrates. For example, a user can draw a circle around a robot to active it for further control. An X is referred to as a waypoint that the robot has to pass through during the movement and an arrow indicates the robot's travel route and the final orientation (indicated by the orientation of the head of the arrow). A zone specifies an area that the robot must explore, and the robot would randomly sample waypoints within the area and pass through. The user can simply draw the sketches to control the robot using a stylus and a tablet. Figure 28.13b presents a snapshot of the interface of the system.

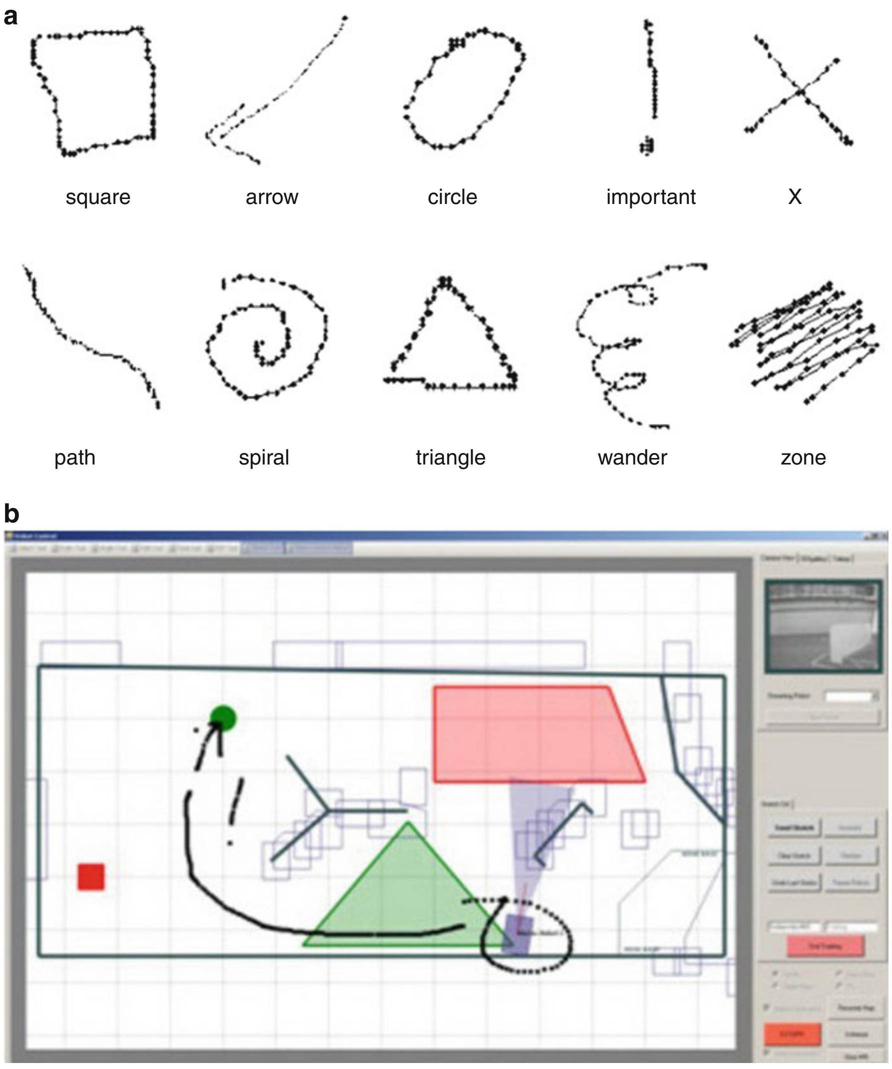


Fig. 28.13 (**a**) Ten sketches of command and (**b**) interface snap shot. Four sketch commands are used: *circle*, *arrow*, *zone*, and *triangle*

2.5D Modeling

Zhu et al. [54] propose a sketch-based dynamic illustration method of fluid system. They present a lightweight sketching system that enables interactive design, editing, and illustration of complex fluid systems. The modeling is performed on a 2.5D canvas to design the object shapes and connections of a fluid circuit. The input sketches are automatically analyzed and abstracted. Fluid model is required to enhance the illustration in the background. Rich operations are provided by the

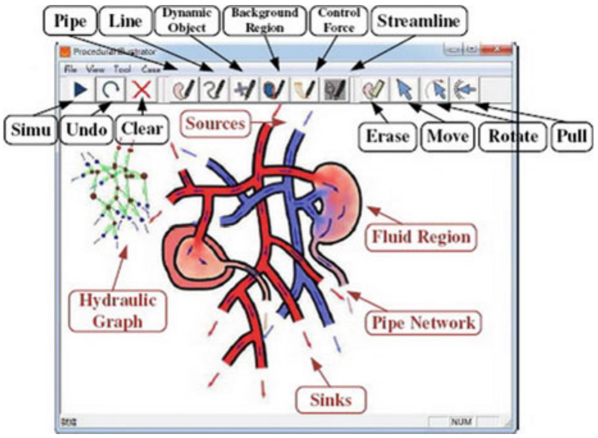


Fig. 28.14 A screen snapshot of the sketch-based fluid illustration system [54]

system for users to edit the fluid system incrementally, and the new internal flow pattern is modeled in real time. This work is tested in domains of medicine, biology, and engineering. Figure 28.14 shows a screen snapshot of their system.

3D Modeling

Sketch-based 3D construction or understanding uses similar techniques at the 2D level of recognition and understanding. However, more techniques specific to the 3D level are required. Especially, Zeleznik et al. [55] invent an interface to input 3D sketchy shapes by recognizing the predefined patterns of some 2D shapes that represent certain sketchy solid shapes.

Sketch-Based Recognition and Simulation

Chemical Drawing Recognition

Shi and Zhang [56] propose a SVM-HMM-based classifier for online chemical symbol recognition. They modify the PSR algorithm [57] into the MPSR algorithm to enhance the processing efficiency and user-friendliness.

Given the size of the symbol set and the distinction between organic ring structure (ORS) and non-ring structure (NRS), their method cannot use only a single-stage classifier to perform classification for all symbols. Therefore, they design a double-stage classification architecture, in which the SVM-based classifier roughly classifies the symbols into ORS and NRS at the first stage, then the HMM-based classifier performs fine classification at the second stage. The input of the whole classification architecture is an isolated handwritten chemical symbol and the outputs are the top three candidates.

Handwritten chemical symbol is a temporal sequence of X–Y points captured using a digitizer that senses the pen-tip position while writing. Due to arbitrary handwriting and various writing styles, noises, broken, and conglutinate strokes are inevitable and make the recognition more difficult. However, stochastic models can effectively deal with noises and variations caused by handwriting. As a stochastic model, HMMs are suitable for handwritten recognition for many reasons. Therefore, HMM is applied to model handwritten chemical symbols at the second stage [56].

Mathematical Expression

Mathematical expression analysis is a frequently studied area and a more complete survey can be found in ▸Chap. 20 (Processing Mathematical Notation) of this handbook. In this section, we only focus on mathematical expression analysis in the context of sketching interfaces.

Typical online mathematical expression recognition approaches [58] follow two steps. First, strokes are segmented and grouped to form single symbols, i.e., digits, and operators. Second, structural analysis is employed to determine the mathematical relationship among the symbols. This step is particularly challenging due to different writing styles and irregularity of the layout of the writing. If there are more than two symbols, their relationship can only be determined globally. That is, the first symbol has to be located before any other decisions to be made. To overcome such problems, Tapia and Rojas [59] use a minimum spanning tree and symbol dominance analysis to handle overlapping and controversial association of objects and horizontal layout irregularities. Using contextual information to solve the ambiguities of interpretation is also discussed in the work from Belaid et al. [60]. However, better solution is provided by adopting some context-free frameworks and probabilistic models. Koschinski et al. [61] proposed a system to recognize online handwritten mathematic expressions on a tablet.

Mechanical Engineering

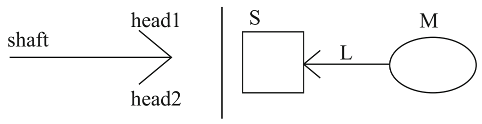
ASSIST, a Shrewd Sketch Interpretation and Simulation Tool [38], can understand mechanical engineering sketches and then interpret their mechanisms using simulation. It is particularly interesting that physical gravity and contact between the objects can be well simulated, like a floating object can fall down to the ground without supporting.

EsQUIsE [62] captures and understands architectural sketches and then builds 3D models based on the recognition results. It can also carry evaluation of the project like the real-time walk time through assessment of the building's energy needs, building and functioning cost, etc.

Multi-domain Applications

Most sketch systems are domain dependent and most attempts on domain-independent systems are proved to be unsuccessful [47]. To specify a certain domain field, graphical lexicon, syntax, and semantics must be well modeled. Attempts on the combination of domain specification and sketch recognition can be found in the work [46]. In their work, a system called SketchREAD is developed. In this system, a novel descriptive language for shape's structural organization

Fig. 28.15 The LADDER syntax for the shape "arrow." The domain pattern "mother–son" is also indicated



```
DEFINE ARROW
(Subshapes
L1,L2,L3: (Line shaft head1 head2))
(Constraints
C1: (coincident                 C3: (equal-length head1 head2)
        shaft.p1 head1.p1)      C4: (shorter head1 shaft)
C2: (coincident                 C5: (acute-angle head1 shaft)
        shaft.p1 head2.p2)      C6: (acute-angle head2 shaft)
DEFINE MOTHER-SON
(Subshapes
M,S,L: (Female M) (Male S) (Child-link L)
(Constraints
C1:     (touches L.head S)
C2:     (touches L.tail M)
```

called LADDER is used. Under its description, no training data or programming is necessary. Figure 28.15 illustrates an example description on an arrow and some other hierarchy information used in the matching step.

Conclusion

In this chapter, we present the problems, technologies, and applications of sketch interface systems, which take freehand sketching input as either command gestures or graphic objects. The main procedures of sketch processing are the following: preprocessing, recognition, and understanding.

Preprocessing techniques in this context include denoising, stroke segmentation, and primitive fitting. Classical recognition methods usually omit the preprocessing stage, while modern recognition approaches employ preprocessing techniques to enhance their performance.

Methods proposed for recognizing gestures and graphic objects are presented. As demonstrated, the initial stage of the recognition process distinguishes between gestures and graphic objects. Two approaches for gestures/graphic objects distinction, namely, explicit mode switching and implicit mode detection, are elaborated. Once the distinction is finished, corresponding recognition processes are carried out for gestures or graphic objects. Gestures are often easy to recognize due to their simplicity, whereas recognition of graphic objects requires more sophisticated mathematical models and interactive techniques for obtaining better recognition accuracy. In sections "Gesture Recognition" and "On-Line Graphics Recognition," we brief the researches done for both gesture recognition and graphic object recognition, particularly for sketch interfaces, stressing on their online natures. The major approaches can be summarized as follows:

1. Hand-coded or rule-based methods
2. Feature extraction methods
3. Direct sample point matching methods
4. Probabilistic model-based methods

However, as discussed in sections "Partial Graphic Object Recognition" and "On-Line Graphics Recognition," none of these methods is perfect for all situations. They either suffer from performance limitations or are computationally expensive. Furthermore, the machine learning approaches which often produce accurate recognition rely on having a cumbersome training stage. However, there are quite a few ready-to-use API's and toolkits for gesture recognition with moderate performance, as listed in the next section.

Graphic document understanding is discussed in section "Complete Graphic Object Recognition." This processing stage is domain dependent. Graphic document understanding provides the user with more intuitive understanding of the sketch inputs and, hence, enhances user experience. Three mainstream approaches, namely, rule-based hand-coding, XML knowledge representation and understanding, and constraint programming language, for graphic document understanding are discussed.

In section "Applications," we look at the sketch interface applications for different purposes: annotation, diagramming, design and modeling, recognition and simulation, and multi-domain applications. We think that domain-specific sketch interface will still be a major research direction. In other words, the sketch recognition and understanding methods should base on a specific application in order to achieve more domain-specific functions. On the other hand, although the major aim of recognition methods is still improving recognition accuracy with low computational cost, recognition methods with specialties, such as partial symbol recognition, start to attract more attentions.

Description of Consolidated Software and/or Reference Datasets

QuickDiagram (previously, *SmartSketchpad*) is a sketching interface for quick diagram input and understanding. It involves quite a number of methods for stroke processing, symbol recognition, and syntax and semantic. With a user sketching a (complete or partial) symbol or wire (connecting two symbols) of the diagram, the system can recognize and beautify it immediately. After the entire diagram is complete, it can be analyzed and understood via nodal analysis and generation of PSpice codes. The *QuickDiagram* system is released as an open-source project at http://code.google.com/p/quickdiagram/.

de Silva R, Bischel DT Lee WS, Peterson EJ, Calfee RC and Stahovich TF (2007) Kirchhoff's Pen: A pen-based circuit analysis tutor. University of California, Riverside.

$1 gesture recognizer is an open-source simple gesture recognition toolkit which has been implemented in various programming languages. The original version only supports uni-stroke gestures. $N gesture recognizer is the extended version

for uni-touch and multi-stroke gestures (http://depts.washington.edu/aimgroup/proj/dollar/).

Android gesture package is the open-source gesture recognition package provided by google for android SDK. Its current version (4.0) implements an offline gesture recognition approach which describes each gesture using a statistical descriptor and retrieves a predefined gesture by comparing the descriptors. It supports uni-touch and multi-stroke gestures (http://developer.android.com/reference/android/gesture/package-summary.html).

Apple iOS gesture recognition package is the open-source gesture recognition package provided by apple for iOS SDK. It supports simple multi-touch gestures (http://developer.apple.com/library/ios/#documentation/UIKit/Reference/ UIGestureRecognizer_Class/ Reference/Reference.html#//apple_ref/occ/cl/UIGestureRecognizer).

Silverlight for Windows phone toolkit also includes API's for touch input and gesture recognition package (http://create.msdn.com/en-US/education/quickstarts/Touch_Input).

Multi-touch systems I have known and loved is an online overview of historical multi-touch system written by Bill Buxton (http://billbuxton.com/multitouchOverview.htm).

Cross-References

▸An Overview of Symbol Recognition
▸Analysis and Interpretation of Graphical Documents
▸Processing Mathematical Notation

References

1. Arvo J, Novins K (2000) Fluid sketches: continuous recognition and morphing of simple hand-drawn shapes. In: Proceedings of the 13th annual ACM symposium on user interface software and technology 2000, San Diego. ACM, pp 73–80
2. Wenyin L et al (2010) QuickDiagram: a system for online sketching and understanding of diagrams. In: Ogier J-M, Liu W, Lladós J (eds) Graphics recognition. Achievements, challenges, and evolution. Springer, Berlin/Heidelberg, pp 130–141
3. Fonseca MJ, Pimentel C, Jorge JA (2002) CALI: an online scribble recognizer for calligraphic interfaces. In: AAAI 2002 spring symposium – sketch understanding, Palo Alto, pp 51–58
4. Landay JA, Myers BA (1995) Interactive sketching for the early stages of user interface design. In: Proceedings of the SIGCHI conference on human factors in computing systems 1995, Denver. ACM/Addison-Wesley, pp 43–50
5. Saund E et al (2002) Perceptual organization as a foundation for intelligent sketch editing. In: AAAI 2002 spring symposium – sketch understanding 2002, Palo Alto
6. Rubine D (1991) Specifying gestures by example. ACM SIGGRAPH Comput Graph 25(4):329–337
7. Sezgin TM, Davis R (2005) HMM-based efficient sketch recognition. In: Proceedings of the 10th international conference on intelligent user interfaces 2005, San Diego. ACM, pp 281–283

8. Herold J, Stahovich TF (2011) SpeedSeg: a technique for segmenting pen strokes using pen speed. Comput Graph 35(2):250–264
9. Horng J-H, Li JT (2001) A dynamic programming approach for fitting digital planar curves with line segments and circular arcs. Pattern Recognit Lett 22(2):183–197
10. Ray BK, Ray KS (1994) A non-parametric sequential method for polygonal approximation of digital curves. Pattern Recognit Lett 15(2):161–167
11. Fitzgibbon A, Pilu M, Fisher RB (1999) Direct least square fitting of ellipses. IEEE Trans Pattern Anal Mach Intell 21(5):476–480
12. Liao C et al (2008) Papiercraft: a gesture-based command system for interactive paper. ACM Trans Comput Hum Interact 14(4):1–27
13. Kabbash P, Buxton W, Sellen A (1994) Two-handed input in a compound task. In: Proceedings of the SIGCHI conference on human factors in computing systems: celebrating interdependence 1994, Boston. ACM, pp 417–423
14. Ruiz J, Bunt A, Lank E (2008) A model of non-preferred hand mode switching. In: Proceedings of graphics interface 2008, Windsor. Canadian Information Processing Society, pp 49–56
15. Ramos G, Boulos M, Balakrishnan R (2004) Pressure widgets. In: Proceedings of the SIGCHI conference on human factors in computing systems 2004, Vienna. ACM, pp 487–494
16. Forsberg A, Dieterich M, Zeleznik R (1998) The music notepad. In: Proceedings of the 11th annual ACM symposium on user interface software and technology 1998, San Francisco. ACM, pp 203–210
17. Bae S-H, Balakrishnan R, Singh K (2009) EverybodyLovesSketch: 3D sketching for a broader audience. In: Proceedings of the 22nd annual ACM symposium on user interface software and technology 2009, Victoria. ACM, pp 59–68
18. Li Y et al (2005) Experimental analysis of mode switching techniques in pen-based user interfaces. In: Proceedings of the SIGCHI conference on human factors in computing systems 2005, Portland. ACM, pp 461–470
19. Saund E, Lank E (2003) Stylus input and editing without prior selection of mode. In: Proceedings of the 16th annual ACM symposium on user interface software and technology 2003, Vancouver. ACM, pp 213–216
20. Zeleznik R, Miller T (2006) Fluid inking: augmenting the medium of free-form inking with gestures. In: Proceedings of graphics interface 2006, Quebec. Canadian Information Processing Society, pp 155–162
21. Samavati FF, Olsen L, Jorge JA (2011) Introduction (Chapter 1). In: Jorge J, Samavati F (eds) Sketch-based interfaces and modeling. Springer-Verlag London, pp 1–15
22. Kurihara K et al (2011) Toward localizing audiences' gaze using a multi-touch electronic whiteboard with sPieMenu. In: Proceedings of the 16th international conference on intelligent user interfaces 2011, Palo Alto. ACM, pp 379–382
23. Wang D-x, Xiong Z-h, Zhang M-j (2011) An application oriented and shape feature based multi-touch gesture description and recognition method. Multimed Tools Appl 23(2):1–23
24. Yee W (2009) Potential limitations of multi-touch gesture vocabulary: differentiation, adoption, fatigue human-computer interaction. In: Jacko J (ed) Novel interaction methods and techniques. Springer, Berlin/Heidelberg, pp 291–300
25. Damaraju S, Kerne A (2008) Multitouch gesture learning and recognition system. In: Extended abstracts of IEEE workshop on tabletops and interactive surfaces 2008, Amsterdam, pp 102–104
26. Li W, He H (2010) Fingertip tracking and multi-point gesture recognition. In: International symposium on intelligent signal processing and communication systems, Chengdu
27. Hinckley K et al (2005) Design and analysis of delimiters for selection-action pen gesture phrases in scriboli. In: Proceedings of the SIGCHI conference on human factors in computing systems 2005, Portland. ACM, pp 451–460
28. Myers BA et al (1997) The Amulet environment: new models for effective user interface software development. IEEE Trans Softw Eng 23(6):347–365
29. Ramos G, Balakrishnan R (2003) Fluid interaction techniques for the control and annotation of digital video. In: Proceedings of the 16th annual ACM symposium on user interface software and technology 2003, Vancouver. ACM, pp 105–114

30. Willems D et al (2009) Iconic and multi-stroke gesture recognition. Pattern Recognit 42(12):3303–3312
31. Li Y (2010) Protractor: a fast and accurate gesture recognizer. In: Proceedings of the SIGCHI conference on human factors in computing systems 2010, Atlanta. ACM, pp 2169–2172
32. Anderson D, Bailey C, Skubic M (2004) Hidden Markov model symbol recognition for sketch-based interfaces. In: AAAI 2004 Fall symposium, Arlington
33. Tappert CC, Suen CY, Wakahara T (1990) The state of the art in online handwriting recognition. IEEE Trans Pattern Anal Mach Intell 12(8):787–808
34. Wenyin L (2004) On-line graphics recognition: state-of-the-art. In: Lladós J, Kwon Y-B (eds) Graphics recognition. Recent advances and perspectives. Springer, Berlin/Heidelberg, pp 291–304
35. Lank E et al (2001) On-line recognition of UML diagrams. In: Proceedings of the 6th international conference on document analysis and recognition, Seattle
36. Lin J et al (2001) DENIM: an informal tool for early stage web site design. In: Proceedings of the SIGCHI conference on human factors in computing systems 2001, Seattle. ACM, pp 205–206
37. Sezgin TM, Davis R (2008) Sketch recognition in interspersed drawings using time-based graphical models. Comput Graph 32(5):500–510
38. Alvarado C, Davis R (2006) Resolving ambiguities to create a natural computer-based sketching environment. In: ACM SIGGRAPH 2006 courses, Boston. ACM, p 24
39. Hammond T, Davis R (2006) Tahuti: a geometrical sketch recognition system for UML class diagrams. In: ACM SIGGRAPH 2006 courses, Boston. ACM, p 25
40. Mahoney JV, Fromherz MPJ (2002) Three main concerns in sketch recognition and an approach to addressing them. In: AAAI 2002 Spring symposium – sketch understanding, Palo Alto
41. Sun Z et al (2004) User adaptation for online sketchy shape recognition. In: Lladós J, Kwon Y-B (eds) Graphics recognition. Recent advances and perspectives. Springer, Berlin/Heidelberg, pp 305–316
42. Lee S-W, Kim Y-J (1995) A new type of recurrent neural network for handwritten character recognition. In: Proceedings of the 3rd international conference on document analysis and recognition, Montreal
43. Chen K-Z et al (2003) Recognition of digital curves scanned from paper drawings using genetic algorithms. Pattern Recognit 36(1):123–130
44. Llados J, Marti E, Villanueva JJ (2001) Symbol recognition by error-tolerant subgraph matching between region adjacency graphs. IEEE Trans Pattern Anal Mach Intell 23(10): 1137–1143
45. Sutherland IE (2003) Sketchpad: a man-machine graphical communication system. Technical reports, University of Cambridge
46. Alvarado C, Davis R (2007) SketchREAD: a multi-domain sketch recognition engine. In: ACM SIGGRAPH 2007 courses, San Diego. ACM, p 34
47. Davis R (2007) Magic paper: sketch-understanding research. Computer 40(9):34–41
48. de Silva R et al (2007) Kirchhoff's pen: a pen-based circuit analysis tutor. In: Proceedings of the 4th Eurographics workshop on sketch-based interfaces and modeling 2007, Riverside. ACM, pp 75–82
49. Loughlin MM, Hughes JF (1994) An annotation system for 3D fluid flow visualization. In: Proceedings of IEEE conference on visualization, Washington, DC
50. Gennari L et al (2005) Combining geometry and domain knowledge to interpret hand-drawn diagrams. Comput Graph 29(4):547–562
51. Dachselt R, Frisch M, Decker E (2008) Enhancing UML sketch tools with digital pens and paper. In: Proceedings of the 4th ACM symposium on software visualization 2008, Ammersee. ACM, pp 203–204
52. Chen Q, Grundy J, Hosking J (2008) SUMLOW: early design-stage sketching of UML diagrams on an E-whiteboard. Softw Pract Exp 38(9):961–994
53. Shah D, Schneider J, Campbell M (2010) A robust sketch interface for natural robot control. In: IEEE/RSJ international conference on intelligent robots and systems, Taipei

54. Zhu B et al (2011) Sketch-based dynamic illustration of fluid systems. ACM Trans Graph 30(6):1–8
55. Zeleznik RC, Herndon KP, Hughes JF (2007) SKETCH: an interface for sketching 3D scenes. In: ACM SIGGRAPH 2007 courses, San Diego. ACM, p 19
56. Shi G, Zhang Y (2010) An improved SVM-HMM based classifier for online recognition of handwritten chemical symbols. In: Chinese conference on pattern recognition, Chongqing
57. Yang Z, Guangshun S, Kai W (2010) A SVM-HMM based online classifier for handwritten chemical symbols. In: 20th international conference on pattern recognition, Istanbul
58. Chan K-F, Yeung D-Y (2000) Mathematical expression recognition: a survey. Int J Doc Anal Recognit 3(1):3–15
59. Tapia E, Rojas R (2004) Recognition of on-line handwritten mathematical expressions using a minimum spanning tree construction and symbol dominance. In: Lladós J, Kwon Y-B (eds) Graphics recognition. Recent advances and perspectives. Springer, Berlin/Heidelberg, p 329–340
60. Belaid A, Haton J-P (1984) A syntactic approach for handwritten mathematical formula recognition. IEEE Trans Pattern Anal Mach Intell PAMI-6(1):105–111
61. Koschinski M, Winkler HJ, Lang M (1995) Segmentation and recognition of symbols within handwritten mathematical expressions. In: International conference on acoustics, speech, and signal processing, Detroit
62. Juchmes R, Leclercq P, Azar S (2005) A freehand-sketch environment for architectural design supported by a multi-agent system. Comput Graph 29(6):905–915

Further Reading

Buxton2007. Buxton B (2007) Sketching user experiences: getting the design right and the right design, 1st edn. Morgan Kaufmann, San Francisco, p 448, ISBN-13 978-0123740373

Jorge2011. Jorge J, Samavati F (eds) (2011) Sketch-based interfaces and modeling, vol XII, 1st edn. Springer, London/New York, p 402. ISBN 978-1-84882-811-7