PERGAMON

Computers & Graphics 24 (2000) 835–849

COMPUTERS & GRAPHICS

www.elsevier.com/locate/cag

Calligraphic Interfaces

Drawing on the Back of an Envelope: a framework for interacting with application programs by freehand drawing

Mark D. Gross*, Ellen Yi-Luen Do

Design Machine Group, Department of Architecture, University of Washington, Seattle, WA 98195-5720, USA

Abstract

The Back of an Envelope project demonstrates how a calligraphic interface (one that employs a pen or stylus to input freehand drawing marks) can be used in a wide variety of domains, from databases to simulation programs, to 3D modeling, from mostly symbolic diagrams to freeform sketches. The wide variety of drawing types and domains calls for a diverse range of approaches. We describe some of the functionality of our systems, including contextual recognition of symbols and configurations and emergent shape recognition, and some of the calligraphic interfaces we've built. © 2000 Elsevier Science Ltd. All rights reserved.

Keywords: Design; Diagramming; Freehand sketching; Human–computer interface; Knowledge-based design systems; Recognition

1. Introduction

People often use drawings and diagrams to communicate with one another—at whiteboards, formally in technical papers, and informally on scraps of paper. Although these visual explanations need not stand alone, they are valuable additions to information conveyed using other means and modalities. People could benefit from computers that communicate with diagrams and drawings in addition to speech, text, and structured window–mouse interactions.

Exciting new work is being done in the multidisciplinary area of diagram-based reasoning: cognitive scientists are examining how diagrams aid human thinking and artificial intelligence researchers are looking at the implications of machine reasoning with diagrammatic representations [1]. Increasing interest in diagrammatic reasoning underscores the need to research and develop interactive, pen-based, or "calligraphic" interfaces for a potentially wide range of application domains.

Drawings range from the informal quick sketch to the carefully constructed illustration. Some drawings convey mostly symbolic information, for example, a molecular diagram. Others convey mostly shape, but hardly any symbolic, information, for example, a sketch for an automobile body. Some drawings are in between: they convey both symbolic and geometric information, for example, a sketch map to help a guest find the way to your home. In the Back of an Envelope project we have been investigating the uses of sketches, diagrams, and drawings in various contexts and constructing calligraphic interfaces for diverse purposes.

Most of the computer-based tools that support the creation and editing of graphic data fall into one of two categories: highly structured interfaces for creating accurate and specific, drawings and models; and interfaces that allow freeform input but that do not attempt to structure it. For example, CAD drawing and modeling programs belong to the first category, and paint programs belong to the second. Structured interfaces are useful when the designer is prepared to make precise and specific commitments (for example, as to shape, dimen-

*Corresponding author. Tel.: +1-206-616-2817; fax: +1-206-616-4992.

E-mail addresses: mdg@cs.washington.edu (M.D. Gross), ellendo@u.washington.edu (E.Y.-L. Do).

PII: S0097-8493(00)00087-X

sion, and position of elements), but for many tasks (such as conceptual design) the designer is not. On the other hand, freeform interfaces typically do not capture data in a format that can be operated on semantically. For example, if the designer draws a circle in a paint program, it may be difficult to select the circle and perform geometric operations on it. Our approach seeks to combine the virtues of these two interface styles, to provide the freedom and flexibility of freehand drawing, yet be able to capture the semantic content of input by recognizing and parsing it. In this way designers may be able to take advantage of the symbolic and numeric processing of computers during early, conceptual, thinking, without having to falsely commit to highly precise representations.

Despite recent improvements in recognition technologies (both in pen input and in speech), accurate recognition rates remain imperfect, and domains that demand perfect recognition in a single mode of input are likely to frustrate users: the current generation of continuous speech recognizers attests to this. However, user acceptance of pen interfaces depends on more than purely the recognition rate [2]; and systems that support multimodal interaction may be able to make up for imperfect modal recognition rates [3].

We describe in brief some of the functionality of the systems we have built to support and experiment with calligraphic interfaces. As the title of this paper indicates, the emphasis of our work is to explore and demonstrate how freehand drawing can be used to interact with a diverse array of applications, and to understand the configuration of recognition, parsing, and graphic manipulation that these applications suggest. Although we have constructed our own recognizer and parser (outlined below in Section 4), this is not the focus of our work. In principle, other recognizers and parsers could be used in their place. With the exception of the NetDraw group drawing manager (which is written in Java), all our prototypes are implemented in Macintosh Common Lisp (MCL).

The paper is organized as follows. Section 2 reviews related work in diagram recognition and visual language parsing and the application of these techniques in systems to support specific tasks. Section 3 presents an overview of the structure of the Back of an Envelope system, and Section 4 goes into some detail about the system's major components. Section 5 shows how the system has been used to build prototype interfaces in several application areas: information retrieval (query by diagram), simulation, and construction of three-dimensional models, as well as a scheme for invoking applications appropriate to the task at hand, based on an examination of the user's graphic marks. The paper concludes in Section 6 with a brief discussion and identifies the tasks that we are presently working on.

2. Related work

A look at Ivan Sutherland's pen-based "Sketchpad" program [4] reminds us that interacting with computers through a pen-based interface is not a new idea. As early as the 1970s, interest in machine processing of hand drawn diagrams led to some experimental sketch recognition systems [5,6]. However, with the widespread acceptance of mouse and menu (WIMP style) interfaces in the 1980s, general interest in pen-based input waned, and did not revive until inexpensive and cordless digitizing technologies became widely available in the 1990s. Today, pocket sized pen-based computers such as the 3Com Palm Pilot are extremely popular, although most applications that run on these machines (personal calendars, notepads, etc.) use the pen only for pointing and entering text, not for graphical input. The development of new display technologies such as e·ink [www.eink.com] encourage the development of calligraphic, or pen-based freehand drawing interfaces.

Researchers in a diverse array of fields have examined sketching and cognition. Goel [7], for example, argues that sketches enable designers to engage a kind of thinking that is different from the symbolic reasoning that "traditional" cognitive science tries to account for. Related artificial intelligence and cognitive science research focuses on diagrammatic reasoning in problem solving (e.g. [8,9]). Robbins [10] examines the role of drawing in the production of designs through a series of case study interviews with well-known architects. Others have studied the use of drawings generally [11,12] as well as in specific domains such as mechanical design [13], graphic and interaction design [14], and physics problem solving [15].

These studies of diagram and sketch notations as aids to cognition combined with the development of early visual programming languages such as PICT [16] made clear the need for "visual grammars for visual languages" [17]. Beginning with Kirsch's picture grammars [18], much work has been done on visual language grammars and parsing [19–25]. Much of this work assumes that the visual language expressions to be parsed will be well formed, a reasonable assumption when structured editing tools are used to construct diagrams, and for domains such as visual programming languages in which the rules of diagramming are stated clearly a priori. For design domains, as Lakin [17] pointed out, these assumptions may not hold: design diagrams can be messy, vague, and incomplete. Systems are needed that tolerate these characteristics, handling parts of the diagram that can be immediately recognized and parsed while delaying evaluation of those parts of the diagram that may be filled in later or whose interpretation may be clarified by contextual information.

Recognition of raw input is a basic element of any system that uses freehand pen-based input for symbolic

processing. Accurate recognition of arbitrary freehand drawing input has been only moderately successful, leading for example to the adoption (on some palmtop computers) of the stylized Graffiti alphabet for text input tasks. Recognizers can be distinguished as stroke recognizers that analyze the pen's movement and ink recognizers that analyze the graphic mark the pen makes. A recognizer may use a fixed algorithm to recognize a restricted number of shapes [26]. A trainable recognizer may work by statistically matching features of input data [27]; or it may use neural network models to identify items in the input data. Recognition rates can benefit by feeding back higher level contextual information to low-level glyph recognizers [28], an approach we follow in our own system.

Surprisingly, with a growing emphasis on research on sketch recognition, and visual language parsing, still relatively few systems employ these techniques in application interfaces. Landay's [29] SILK system allowed designers to sketch and simulate the performance of user interface designs. Egenhofer's spatial-query-by-sketch [30] offers a pen-based interface to query a geographic information system (GIS). Moran's group's Tivoli whiteboard support system [31,32] uses ink-based (as opposed to stroke-based) recognition technologies in enabling highly specific meeting support tasks. Stahovich's Sketch·It system [33] applies qualitative reasoning simulations to generate mechanical design variants; despite its name it does not employ a pen-based interface. The XLibris system [34] integrates pen-based input with a lightweight display for annotating text; it uses an ink-based recognition scheme. The idea of "intelligent paper" or merging physical and computational drawing environments has been explored in projects like Clearboard [35] and the Digital Desk [36–39], though in much of this work recognition and interpretation of drawings has not played an important role.

In constructing the BoE we have learned from many of these earlier systems, and many BoE components are modeled on them. BoE plays a mediating role: it combines recognition and parsing in an end-user modifiable scheme, and it provides a simple interface for an end user to extend the visual vocabulary. It is not built to serve a single application domain (as for example is SILK, Tivoli, or spatial-query-by-sketch); rather it is meant to adapt to various specific domains. Unlike other general-purpose systems such as Digital Desk or Clearboard, however, the BoE emphasizes recognition and interpretation of graphic input, which is intended to be customized to the domain and to the individual end user.

3. System overview

An overview of the BoE system architecture is shown in Fig. 1. The project began with the Electronic Cocktail Napkin, which comprises routines for recognizing and parsing diagrams symbolically. We have added routines for working with drawings as geometric entities as well. Drawings are used to communicate with external applications such as visual databases, simulation programs, and three-dimensional modeling environments, and with other people.

Symbolic diagram processing in the Back of an Envelope (BoE) involves recognizing input glyphs drawn on a drawing pad, and parsing visual language expressions.

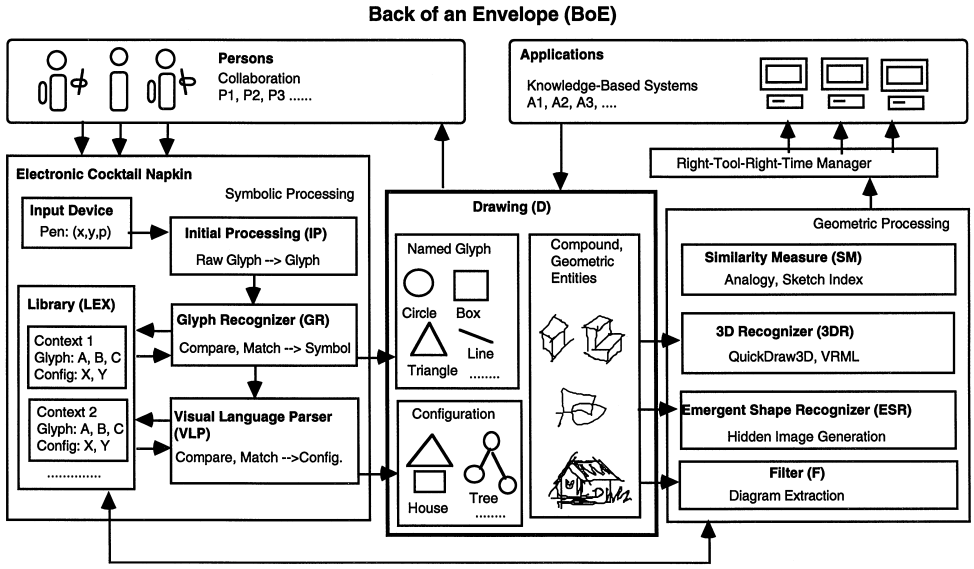

Fig. 1. Back of an Envelope system overview.

Both these recognition steps use a library of stored graphic symbol templates and visual parsing rules, hierarchically organized as a set of contexts. Initial processing (IP) of stroke information obtained from the tablet identifies each glyph's features. Next, the glyph recognizer (GR) matches features of the input glyph against previously stored symbol templates in a library (LEX). The visual language parser (VLP) searches for configurations in the drawing.

The system learns new graphic symbols and configurations on the fly, from examples presented by end users, so the interface can accommodate idiosyncratic drawing styles and preferences. The system is tolerant of ambiguous and unrecognizable glyphs, opportunistically using information as it becomes available to identify previously indeterminate glyphs and to recognize configurations.

Geometric processing routines for working with non-symbolic properties of drawings include machinery to recognize emergent shapes (shapes that the designer did not draw but may be perceptually present in the drawing) (ESR), to process 2D projections of 3D forms (3DR), and filters (F) to simplify the drawing. These geometry routines may interact with the symbolic recognizers. For example, the emergent shape recognizer (discussed further in Section 4.2) calls on the glyph recognizer to determine whether a candidate emergent shape should be culled or kept.

The BoE interface consists of a drawing pad and sketchbook windows, with buttons that augment gestural commands. The designer can place simulated tracing paper on the drawing and move it around, bring in underlay images, copy and paste drawings into and from the sketchbook, and post drawings on a bulletin board. In most of our interface applications, the drawing pad stands apart from the application windows, we have also experimented with overlaying the BoE drawing window as a transparent layer on top of the application, and interacting with the application through the drawing layer.

4. System architecture

4.1. Symbolic processor

4.1.1. Initial processing

The BoE system captures stroke data from the tablet, making use of pen path and other stroke features to identify symbols that the designer draws. The moving pen generates a stream of (x, y, p) data (p is pressure). A glyph begins when the designer puts pen to tablet, and it ends when the designer lifts the pen for longer than a given time-out duration. A raw glyph consists of a sequence of strokes each of which consists of a sequence of (x, y, p) triples. The input processor examines these data to identify features: corners, pen path through a 3×3 grid, overall size and aspect ratio, axial direction of the glyph and the speed with which the glyph was drawn. Fig. 2 displays the features for several simple glyphs. The result of initial processing is stored as a glyph data structure, with a slot for each feature and a pointer to the raw glyph.

4.1.2. Recognizing graphic symbols

The input processor passes the resulting glyph to the glyph recognizer (GR), which determines if the glyph represents a previously defined symbol. The recognizer compares the unidentified input glyph against a library of symbol templates (LEX) and produces a list of candidate symbols paired with recognition certainty values from 0 to 5.

Each template in the library corresponds to the glyph data structure, but in each feature slot the template records all previously trained feature values, accumulated as the symbol is trained. For example, if the system has been shown triangles drawn with one, two, or three strokes then the `number_of_strokes` slot in the template contains the set (1 2 3). If the slot value in the input glyph is a member of the set in the template, then that feature of the input glyph is deemed to match the

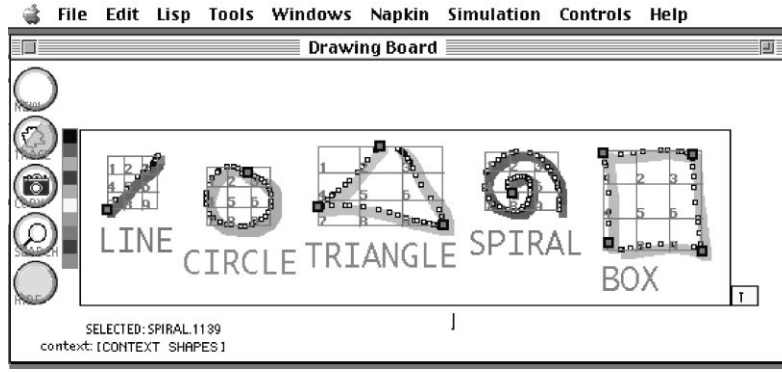

Fig. 2. Glyph features used for recognition include pen path and corner location in a 3×3 grid, number of corners, and aspect ratio.

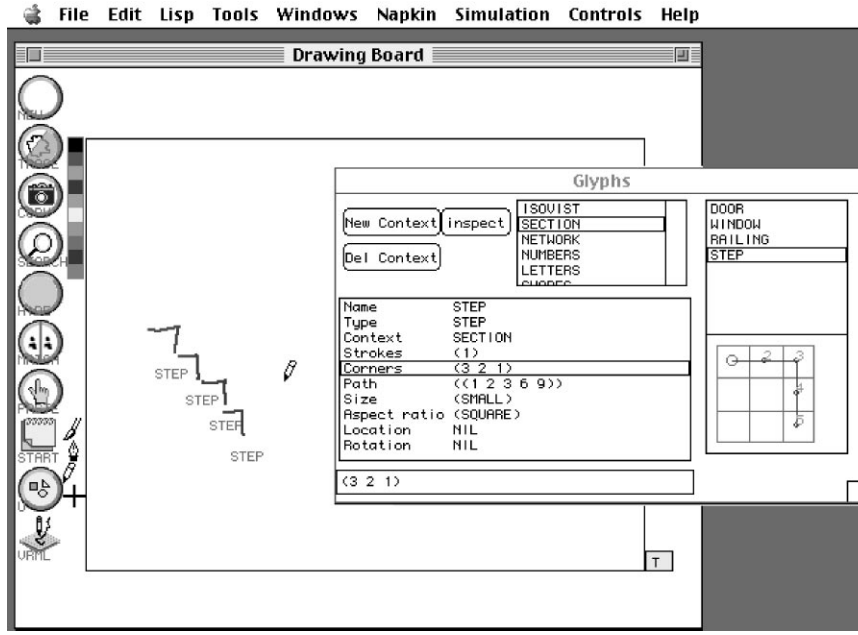

Fig. 3. The symbol template for step in the section context, showing slots and allowable values, and displaying the pen path in a 3×3 grid.

template. Fig. 3 shows a template for a 'step' glyph showing feature slots and allowable values; for example, the step has been trained with one, two, and three corners.

Matching an input glyph against a library of symbol templates proceeds in several steps. An exact match is attempted by using the pen path as a key in a hash table of glyph templates. This retrieves any templates whose pen path feature matches that of the candidate without searching the entire library. If a candidate's pen path exactly matches that of a template and other glyph features match the template then the match succeeds and the new symbol is granted the maximum certainty value of 5.

Otherwise, ninety degree rotations, reflections, and pen path reversals are compared in a similar hash-table match against the templates. The 3×3 grid representation of the pen path makes it simple and fast to check these transformations using a lookup table. The library of templates need store only one rotation or reflection of a symbol yet it can match the various transformations. Of course, not every glyph may be drawn in rotation or reflection: A reflected "N" is not an "N" and a "U" rotated makes a "C". Each template records the allowable transformations for the symbol, so for example, a U may be drawn backward but not upside down. If a glyph's pen path transformed matches a template in the library, and the other glyph features also match, then the symbol is granted a maximum certainty value of 5.

If the candidate's pen path (directly or in transformation) fails the hash-table match, the glyph recognizer begins a more laborious feature-by-feature comparison of the input glyph with each symbol in the library. First, the recognizer attempts a partial match of pen path that tolerates minor variations, also comparing rotations, reflections, and reversals. Next, it compares number of strokes, number of corners, and corner location in the 3×3 grid, as well as overall size and aspect ratio. This enables the recognizer to distinguish, for example, a large box from a small one, and a square from a long flat rectangle. Depending on the number of matching features, the recognizer assigns a certainty value of 0–5 to the match, where 0 indicates that the recognizer failed entirely to find a match, and 5 indicates that the candidate glyph matched the template in all its features.

After the recognizer has done its best to identify the input glyph it sets the glyph's name and type slots. The matching process results in a set of matches each with an associated certainty value. The recognizer sorts matches by certainty and selects the most certain match. If the recognizer cannot identify an input glyph (zero matches) its name and type remain 'unknown' and are left for future identification. On the other hand, several library templates may match the input glyph equally well. Then the ambiguous input glyph is assigned a set of possible identities that may be refined or resolved later.

4.1.3. Parsing visual language

The visual language parser (VLP) identifies configurations of symbols in the drawing. When the designer pauses, the parser attempts to assemble configurations

from elements in the drawing. It applies a grammar of spatial replacement rules that define configurations in terms of their elements and binary spatial relations. Recognized spatial relationships include adjacent, containing, and overlapping shapes, and intersection, parallel, and tee conditions among line segments. The following simple rule, for example, might be used in a grammar for flow charts:

```
labelled_box:: = = (contains box label)
```

where box is a terminal element (a defined graphic symbol) and label is defined by the recursive rule, which recognizes any string of one or more letters as a label:

```
label:: = = (or letter
              (immediately_right_of label letter))
```

The binary spatial relationships `contains` and `immediately_right_of` are each a constraint on the positions and/or dimensions of a pair of drawing elements. Fig. 4 shows a dialog for defining and editing a visual grammar rule in which a set of adjacent steps make up a configuration called 'stairs.'

For each grammar rule the parser tries to find matching instances in the drawing. It first finds a subset of all the drawing elements that match the element types in the rule (in this case, elements of type `label` or `letter`). Then for each set of possible bindings of drawing elements to variables in the rule, the parser checks whether the spatial relation holds. When the parser finds a set of elements that match the types and spatial relations in the rule, it adds the configuration (a `label`) to the drawing in place of the elements, which become the new configuration's parts. It applies the grammar's rules iteratively and exhaustively.

4.1.4. Using configurations to recognize ambiguous and unknown symbols

A glyph's position in a configuration can resolve ambiguity or uncertainty about its identity. When the recognizer fails to uniquely identify a glyph, the glyph is assigned an 'unknown' identity or, when it matches several known symbols equally well, a multiple, 'ambiguous' identity. In attempting to parse a configuration, the parser may find that an unknown or ambiguous glyph is in the expected spatial relationship with other parts of the configuration to be a certain type of element. The parser can use this to resolve or coerce the glyph's identity. The parser resolves an ambiguous glyph if one of its possible identities matches the part the glyph would be in the configuration. For an unidentified glyph, the parser requests the recognizer to determine whether the glyph's features contradict the proposed identity, and if not, it coerces the unidentified glyph.

4.1.5. Contexts

Diagrams in different domains employ the same basic drawing symbols in different ways. As with natural language, context often helps determine the meaning of a visual language expression. We saw above how an element's part in a configuration is used to resolve ambiguous and unknown glyphs. Knowledge about

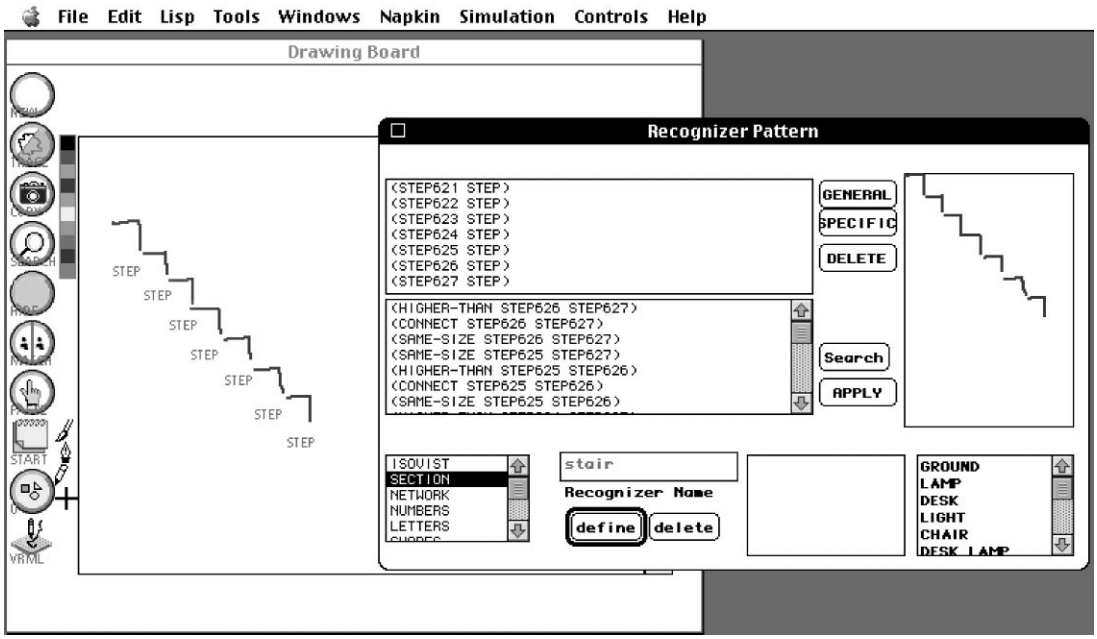

Fig. 4. A stair configuration defined in the section context links a set of steps.

the context of the drawing as a whole can also help recognition.

The library of recognizable symbols and configurations is organized as a hierarchy of drawing contexts, which allows the program to interpret a diagram in specialized domains. The program can use knowledge of the context to assist recognition and interpretation of glyphs and configurations. Conversely, a unique glyph or configuration can reveal the context of a diagram. Each context contains definitions for:

- graphic symbols (glyph templates) that are unique to the context,
- configurations (recognition rules),
- mappings of symbols and configurations from more general contexts.

Contexts are organized in a hierarchy with "the general context" at its root. The default, or general, context defines graphic symbols that are common to most diagrams, such as line, circle, box, and arrow. In addition to defining special symbols and configurations used in a particular domain, a context may map symbols and configurations that are defined in more general contexts to more specific definitions. For example, the section context maps the general symbol `rectangle` to `window`. The same general symbols have different mappings in different specific contexts, so the glyph templates used to recognize the symbols are stored only once, in the general context. For example, the diagram in Fig. 5 has been recognized as a 'section' diagram, and the system has made the appropriate mappings.

The system keeps track of the current context, and the recognizer and the parser first try the current context for recognizing input glyphs and configurations. If this yields no matches or only uncertain ones, then they proceed to the next most general context. If they find a match in a more general context, they apply the current context's mappings, for example changing the recognized `rectangle` symbol to a `window`.

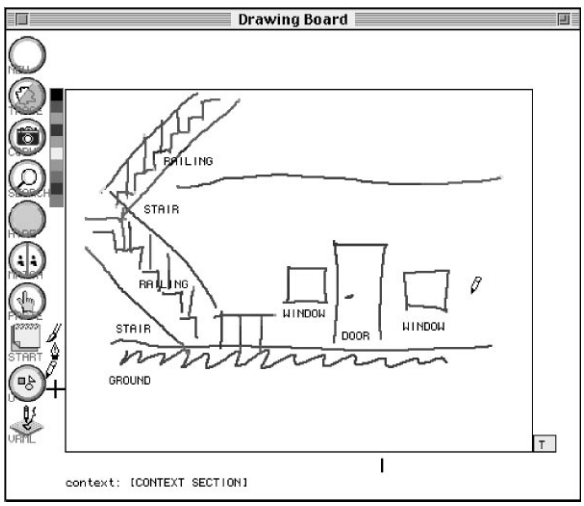

Fig. 5. In the section context, a rectangle maps to a window, and a horizontal line above a wiggly line maps to the ground.

If no context in the current chain yields a match for an input glyph, then the recognizers and parser will search other, more specific contexts. Likewise, the parser will try to recognize configurations defined in more specific contexts. If a symbol or configuration from a more specific context is recognized, this sets the drawing's current context, which guides subsequent recognition. This enables the system to proceed, for example, from the general default context to a more specific one.

4.2. Drawing management and geometric processing

4.2.1. Gesture commands

Any BoE system command may be assigned a glyph or 2D gesture. For example, we have trained the system to recognize the Greek letter alpha as a "clear screen" command. Gesture commands are modeless — they may be executed at any time and screen location, so they must be distinguishable from other drawing symbols. A command may operate on drawing elements that have been previously selected or are indicated by the spatial extent of the gesture. For instance, the "erase" gesture applies to the drawing elements it covers.

4.2.2. Trace layers, underlays, and transparency

Designers use tracing paper to copy and combine parts of drawings, to try alternatives, and to move and rotate design elements. Its not uncommon to find three or four layers of tracing paper overlaid on top of a base drawing. The BoE offers a system of layers that simulates translucent tracing paper: layers on top partially obscure lower layers so that marks on lower layers appear fainter. Each layer may be individually manipulated, and a drawing layer may be pulled out of the drawing and posted on a bulletin board, or pasted in to the sketchbook. Scanned images and drawings made in other applications may be brought in as a base layer for the drawing (Fig. 6), or the entire drawing pad may be rendered transparent and placed on top of another application window.

4.2.3. Overtrace filtering

Multiple overtracing strokes often produce a drawing with many elements that appears quite complex. When overtraced lines are removed, or filtered, the drawing appears much more concise. Filtering reduces overtraced drawings to a simpler form by replacing multiple elements in approximately the same location by a single one. The drawing can also be filtered to eliminate elements smaller than a certain size, or elements that the program cannot recognize. The program notes the number of overtraced elements and stores this number in the one instance that remains in the filtered diagram. Filter-

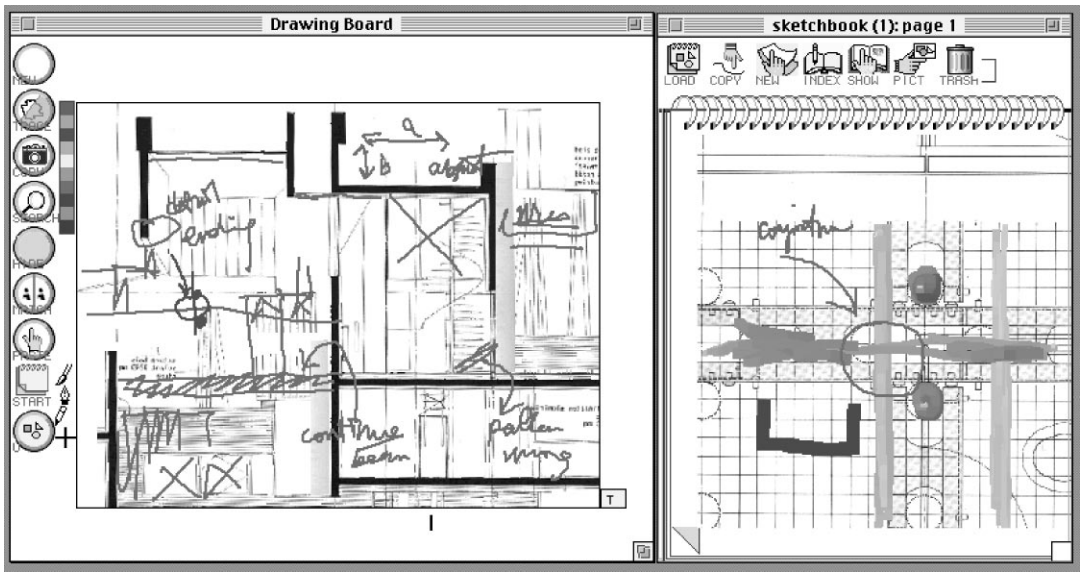

Fig. 6. The designer draws on trace layers, on top of an imported base underlay image.

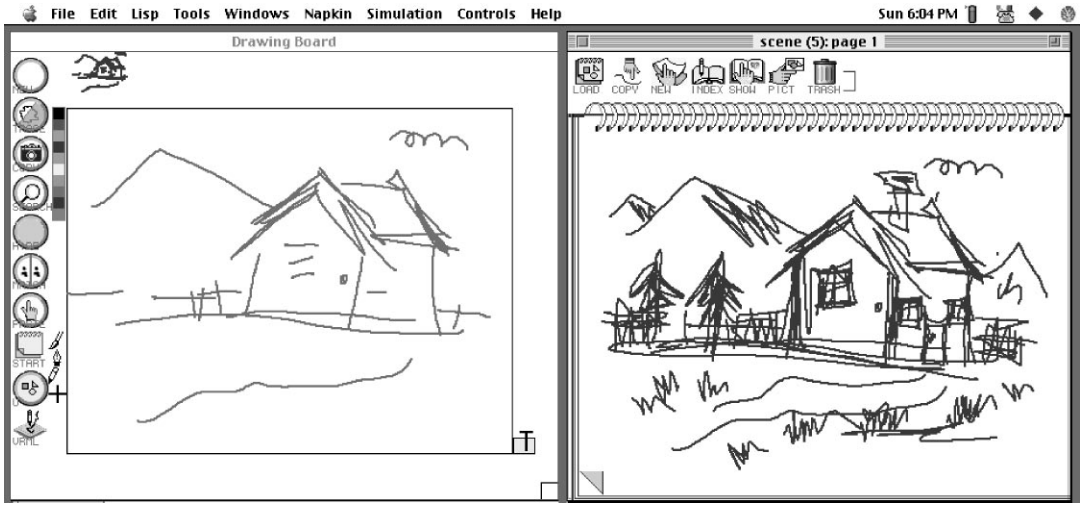

Fig. 7. Filtering can simplify a drawing to essentials.

ing allows the system to treat the basic elements of a drawing without managing large amounts of irrelevant detail (Fig. 7).

4.2.4. Graphical search and inexact configuration matching

The BoE's similarity matcher (SM) identifies drawings that have common features but are not identical. Inexact matching is useful, for example, to find diagrams in a stored database that resemble in one way or another a drawing made on the drawing pad. The Back of an Envelope employs several measures of similarity that can combine to make a single weighted average (see Fig. 8). Two drawings may be compared with respect to their number of elements: If both have the same element count they are equivalent with respect to this measure. Two drawings may be compared with respect to the number of elements they share: if they have all the same elements they are equivalent, otherwise they may differ by some number of elements. They may be compared with respect to spatial relationships: If their elements are arranged in the same spatial relationships, they are equivalent with respect to this measure, regardless of the element identities.

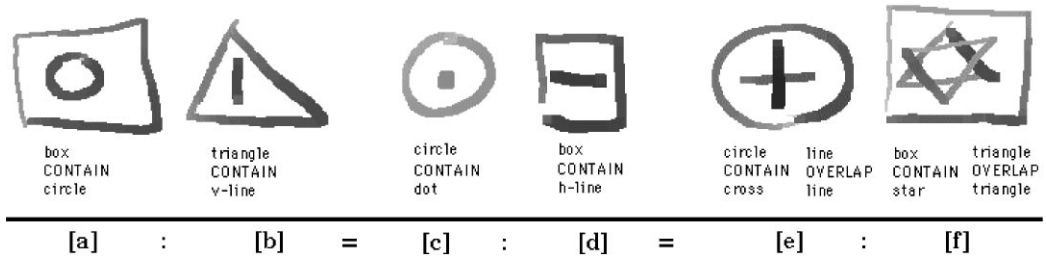

Fig. 8. Each pair of diagrams is equivalent with respect to spatial relations, but different with respect to element type [40].

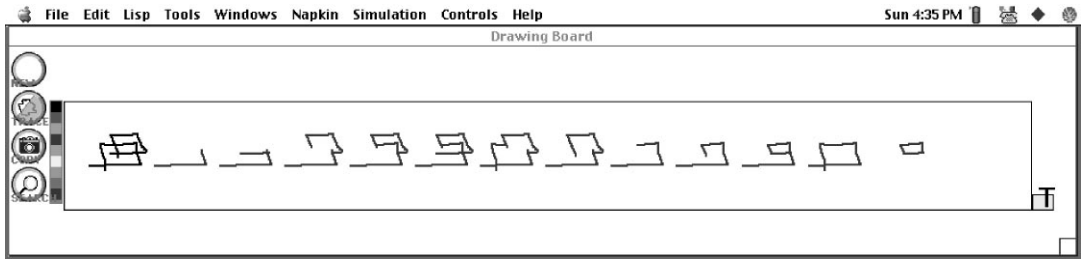

Fig. 9. The emergent shapes recognizer generates alternate readings of a given figure.

4.2.5. Constraints

The spatial relationships among drawing elements can be understood as constraints on their relative positions, dimensions, and orientations. The symbolic recognition routines use these spatial relationships to identify configurations and apply graphical replacement rules, but the same spatial relationships can be treated as enforceable constraints. In one prototype system we implemented a simple constraint propagator that enforced spatial relationships that the visual language parser recognized among drawing elements. For instance, if two circles were drawn linked by a line segment, the system would keep them linked; if a square was drawn inside a circle, it could not be moved out. No special user action was required to apply the constraint: any recognized spatial relationship was applied and enforced automatically. A variation on this constraint enforcement is to apply constraints to drawing elements upon recognition: drawing elements of certain types automatically get certain constraints that govern their layout and interaction behavior with other elements.

4.2.6. Emergent shapes

Draw programs usually record and recognize only the elements that the designer has entered — lines and geometric shapes — and only these elements may be selected and operated on. Some draw programs recognize that crossing lines form new segments and they allow the designer to select these segments, or even the continuous polygons formed by sets of these implicit or 'emergent' line segments. This is a simple example of the more general 'emergent shapes' issue: When two or more drawing elements are combined the human eye often perceives figures that were not explicitly drawn. The machine, however, must be programmed to attend to these 'emergent' shapes.

The BoE's emergent shapes recognizer (ESR) comes into play when two or more distinct glyphs overlap or intersect. When the designer is not drawing, the ESR identifies the glyphs' intersections, generates a set of candidate emergent shapes, and runs the glyph recognizer on the candidates. Beginning at the start point of one of the two glyphs, the generating algorithm walks to the next intersection and proceeds in one of the three possible directions. At each intersection, it again chooses a direction. When it reaches an endpoint, it has generated one possible reading of the figure. It then repeats the process from the starting point, but choosing a different path each time, until it exhausts the possibilities. This results in a set of alternative partial readings of the diagram (Fig. 9). Additional smaller candidates are generated by taking subsets of the each of the candidates. Each candidate is passed to the glyph recognizer. Most of the candidates are nonsense shapes and the ESR will discard them, but occasionally one will match a previously defined graphic symbol in the library, for example, a rectangle. When the ESR recognizes an emergent shape it momentarily highlights the shape and then adds it to the drawing as though the designer had drawn it explicitly.

5. Application programs

We have constructed calligraphic interfaces to a variety of application programs. These range from visual

database retrieval to simulation to Web page layout and three-dimensional modeling. Together these prototypes demonstrate that calligraphic interfaces can be used for diverse application types and in a range of domains.

5.1. *Indexing and retrieval, or query-by-sketch*

We have used the BoE system to index and retrieve items in visual and non-visual databases. In each instance, database items are first indexed with diagrams, and subsequent queries are compared with these index diagrams. A lookup table of diagrams stored in the BoE sketchbook serves as a catalog to the database. For example, we used diagrams to index "The Great Buildings Collection", a CD-ROM database of famous architecture. Diagrams were drawn for database entries on the pages of a BoE sketchbook. To retrieve a database entry, the similarity matcher compares a query diagram drawn on the drawing pad with diagrams on the sketchbook pages, and selects the most similar page. Then the BoE sends an interapplication "retrieve" message to the Great Buildings Collection database for the record that is linked to the sketchbook page. We applied this scheme to several other databases: Archie II case base of design information about libraries and courthouses (written in Lisp), and pages in the World Wide Web (browsed with Netscape or Internet Explorer). A sketchbook page may include pointers to several different databases, so a single diagram query may trigger numerous and diverse responses.

5.2. *Simulation*

We have also applied the BoE as an interface to simulation programs, where the drawing specifies the system to be modeled. Most simulation programs today require fairly formal and complete descriptions in order to run, although crude simulations for conceptual analysis could be driven by sketches and diagrams.

One example is the IsoVist program, which simulates the two-dimensional viewshed of an observer in a room, graphically displaying the floor plan area visible from a given standpoint. (Viewshed simulations are important in designing auditoriums, galleries, courtrooms, and other public places. They give valuable information to the designer about lines of sight and the degree of privacy that a space will provide.) Input to a viewshed simulator would typically be a CAD drawing, but our interface allows the designer to sketch the floor plan and import the sketch into the IsoVist simulator (Fig. 10). Line segments, which represent walls, are extracted from the drawing database, and the position data from their endpoints are used to create wall objects in the IsoVist database. Once the floor plan has been brought into IsoVist the designer can locate standpoints and examine viewsheds. The designer can also move walls in IsoVist program by direct manipulation and later bring the modified floor plan back into the BoE environment.

5.3. *Construction*

A third area of application for calligraphic interfaces is modeling and construction in both two and three dimensions. These applications rely not only on the symbolic content of the drawing, but use the drawing geometry — positions and dimensions of drawing elements — as well.

The first example deals simply with two-dimensional graphic layout: WebStyler is a calligraphic interface for Web page construction (Fig. 11). The designer draws a Web page layout, indicating positions of text, headlines, indents, and graphics using graphic design symbols. Recognition routines identify the symbols and configurations for the various types of layout objects; sizes and positions are also obtained from the drawing. Then, given a collection of graphics and text to lay out WebStyler generates HTML code that formats these materials according to the sketch.

The second example, Digital Clay, generates three-dimensional models from two-dimensional line drawings (Fig. 12). DC illustrates how a designer might use a calligraphic interface to interact with a CAD modeling

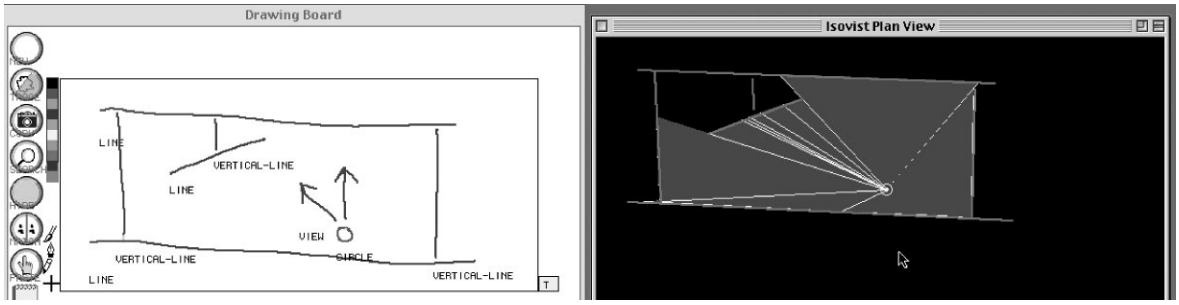

Fig. 10. Floor plans sketched in the BoE environment are brought into the IsoVist simulator (right) to check view shed from various locations.

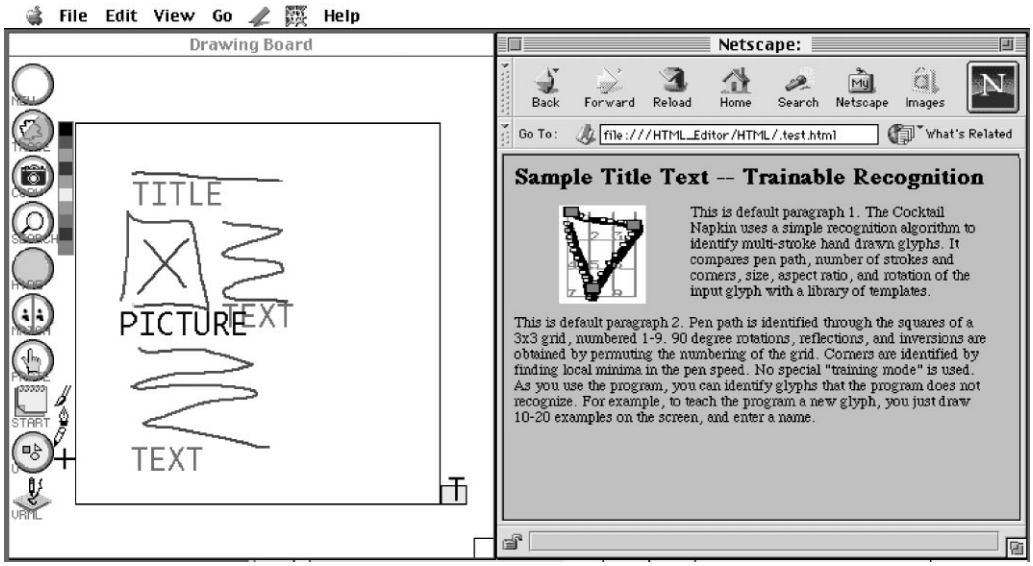

Fig. 11. WebStyler is a two-dimensional calligraphic layout tool.

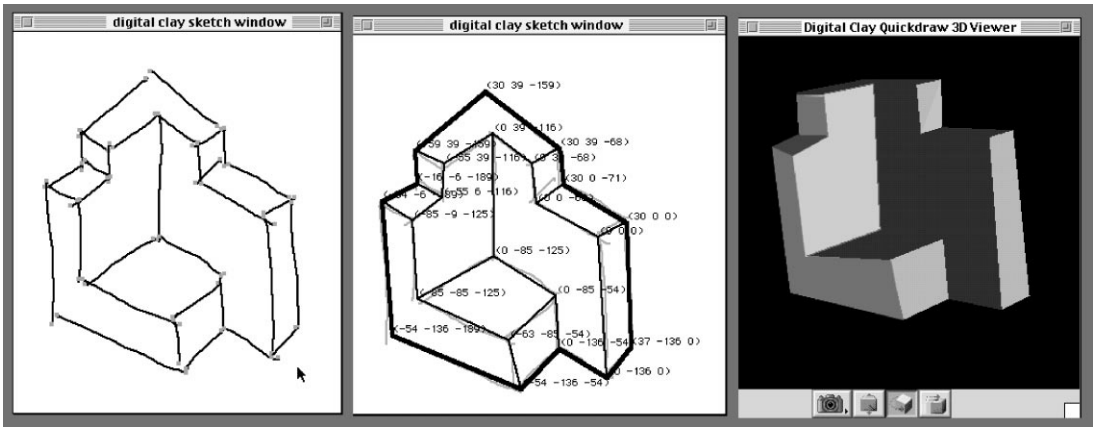

Fig. 12. Digital Clay uses constraint propagation on vertices to produce 3D solid and surface models from sketches.

environment such as Maya, Form·Z, or 3D Studio Max. (Our approach may be contrasted with the Sketch! Program [41], where the approach is to define a gestural language to drive a modeler; our approach is to recognize and interpret line drawings.) The designer makes a line drawing of a 3D rectilinear geometric form. After straightening and latching lines, DC uses constraint propagation [42–44] to determine topology (which vertices protrude toward the viewer and which recede), and then assigns 3D coordinates to the topological representation. The designer can choose between a surface model, which includes only the surfaces that the drawing shows, or a solid model, which requires DC to make some assumptions about hidden vertices and surfaces. In either case, once DC has created and displayed the three dimensional model, the designer can rotate it and specify the previously hidden geometry by drawing additional lines over the model.

Finally, we have begun using a calligraphic interface to directly generate three-dimensional virtual reality worlds. Our SketchVR prototype extrudes a floor plan drawing into three dimensions (Fig. 13). The floor plan may include symbols and configurations for architectural elements such as columns and furniture as well as viewpoint and path indicators to be used in the VR world. SketchVR produces a VRML file that can be viewed in a standard browser, including architectural elements and furniture chosen from a catalog that corresponds to the

symbols and configurations that the designer drew in the plan. The designer can sketch a floor plan and immediately experience what it is like to be inside the space. The VRML browser continually reports the viewer's location in the floor plan, and back in the sketched floor plan the viewer's path is plotted as a series of dots.

5.4. Right tool–right time manager

If calligraphic interfaces can be used in diverse application domains, perhaps the interface itself could be programmed to supervise application management, directing calligraphic input toward intended back end applications. The "Right Tool at the Right Time" manager explores this idea. In a pilot study of 62 architecture students, and a follow up study of four architectural designers doing design, we found that the designer's task could be identified by analyzing the drawing [45]. When designers were asked to solve a lighting problem, for example, they drew certain symbols and configurations; when they were asked to solve a space arrangement problem, they drew others. Based on this observation, we programmed a management routine to launch and direct calligraphic input at different applications, depending on the symbols, configurations, and context detected in the drawing. Fig. 14 shows the steps this RTRT manager

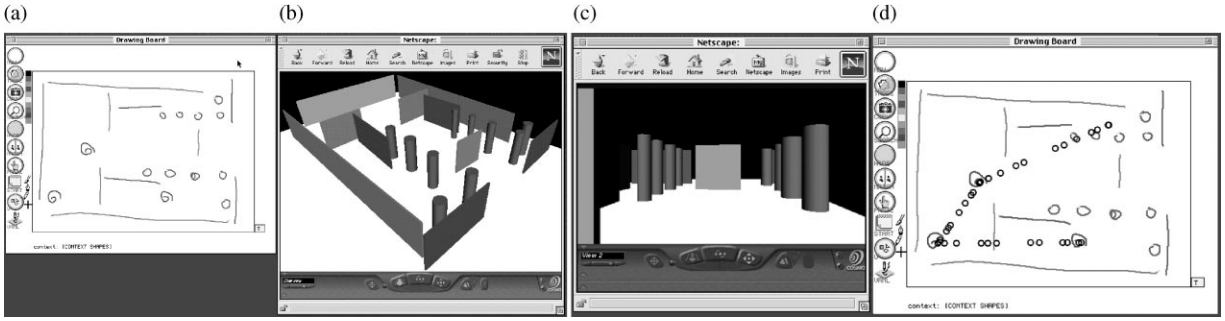

Fig. 13. A sketched floorplan (a) becomes a VRML model (b) that the designer can enter (c) while the designer's path through the model is recorded on the plan (d).

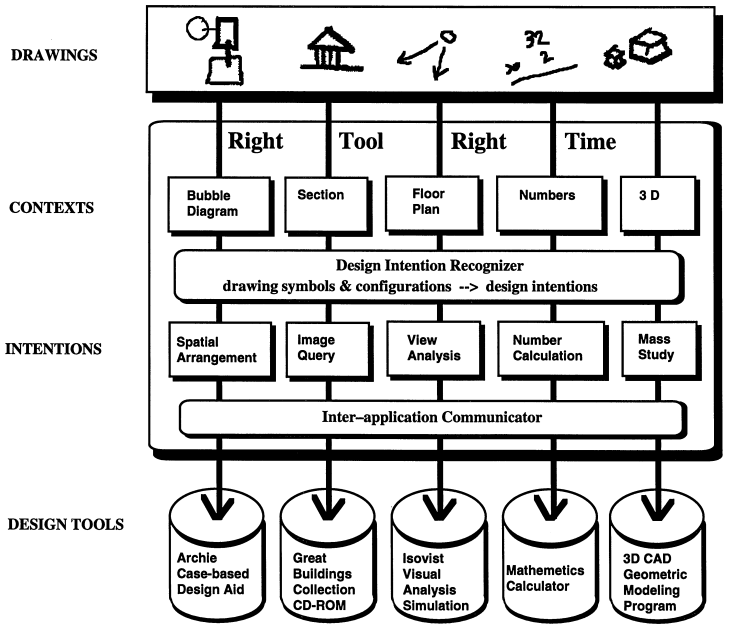

Fig. 14. The right tool/right time manager accepts calligraphic input, identifies the drawing context, from which it infers the designer's intention, and passes input to an appropriate application.

follows to identify an appropriate application based on an input diagram. Although this management may fall short of a general purpose operating system for calligraphically driven applications, it could be useful within the scope of a single multi-application domain such as architectural design.

6. Discussion and future work

The Back of an Envelope project is exploring a broad range of applications for calligraphic interfaces. We have taken several approaches, in some cases looking at drawings as symbolic visual expressions, in others looking at drawings as geometric entities. We have aimed for end-user programmability rather than trying to support drawing in one specific domain. Although we do not make great claims for the performance of our glyph recognizer, it does have some simple virtues. It is trainable and it easily supports transformations of the glyph, and we can specify on a per-glyph basis whether rotations, reflections, and variations in size are isomorphic to the trained template. We tolerate ambiguous glyphs, and identification can be postponed. The configuration parser is also simple; again end users program it by drawing examples. The contextual recognition and the recognition of context allows for an interaction between the low-level recognizer and the parser. An important difference between our effort and many other pen-based drawing editors is that, although we support visual language parsing, we also tolerate drawing expressions that the parser cannot recognize.

The decision to locate glyph templates and visual language rule sets in individual hierarchically structured contexts seems to be a powerful addition to the basic recognition and parsing mechanisms that we had implemented previously. It has made it possible to use the same drawing environment to interact with different applications at different times, and it mitigates recognition errors by partitioning symbols that might otherwise be confused. However, we have not systematically considered variations of the context mechanisms (and the mapping of glyphs from one context to another). This is an area for future work.

Our approach to constructing a calligraphic interface to an application requires controlling the application externally, in effect by redirecting pen input as events that the application understands. One of the biggest challenges has simply been that personal computer applications vary widely in their ability to accept control other than through their own keyboard and mouse interface. Microsoft Word, for example, will respond to scripts written in Visual Basic that can manipulate text on the screen, but it is difficult to get Word to identify the point in the text buffer that corresponds to a given window location in pixels. Some programs, for example the popular 3D modeler Form·Z, simply do not accept external commands. Because the applications do not offer a consistent external interface, we have had to be more application specific than we would prefer, and we have occasionally resorted to interacting with applications by saving and loading files.

Graphically, we have done little work so far with local properties of lines and curves, for example, finding and working with points of inflections, concave and convex sections, discontinuities. Many drawings, especially non-diagrammatic ones, depend on these properties to convey their meaning. We intend to add routines for recognizing features of lines and curves, and add these to the kinds of constraints and relationships that our system supports.

We have explored using the BoE systems in several groupware drawing explorations. In one version, we linked several designers working on different machines with a central drawing server, a Java application we built called NetDraw that handles synchronization among multiple clients. Building on this work, we are presently exploring a system to support whiteboard discussions among several designers, both co-located and in different locations. Similar systems have been explored [46–49]. Ours, focusing on design discussions, will link the symbolic recognition and geometric manipulation facilities of the current system with off-the-shelf speech recognition software and a history-keeping mechanism. It will record and interpret the graphical and audio track of a design discussion, linking them, and allow for graphical and textual queries. For example:

> Show drawings for all discussions about "lighting."
> Show the transcript for this decision (pointing),
> What were we saying about the corner condition here?

We are excited about using drawing to generate, modify, and annotate in three-dimensional models and virtual worlds. We plan to extend the capabilities of the Digital Clay routines that interpret sketches of 3D objects, to support non-rectilinear forms, incomplete (clipped) drawings, and use hatching and shading to guide three-dimensional interpretation. Building on the SketchVR project, we plan to support drawing "into" three-dimensional models, enabling architects to enter a virtual environment, adding to and editing the model as they move about. Our current prototype supports only extruding a floor plan drawing, but we intend to support drawing in section and elevation projections as well as incorporating three-dimensional sketching into the VR model.

In a more speculative vein, we are also interested in developing three-dimensional interfaces for drawing, in which gestures in three dimensions are used to generate shape and form. Our initial approach would extend the systems that we have built that work with two-dimensional drawing to work with three-dimensional sensors.

Acknowledgements

We thank Dongqiu Qian, Eric Schweikardt, Jenniffer Lewin, and Adrienne Warmack, former students at the University of Colorado, College of Architecture and Planning who worked on various parts of the Back of an Envelope project. This research was supported in part by the National Science Foundation under Grant No. IIS-96-19856/IIS-00-96138. The views contained in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] Glasgow J, Narayanan NH, Chandrasekaran B, editors. Diagrammatic reasoning: cognitive and computational perspectives. Menlo Park, CA: AAAI Press/MIT Press, 1995.

[2] Frankish C, Hull R, Morgan P. Recognition accuracy and user acceptance of pen interfaces. CHI '95 Conference Proceedings. ACM SIGCHI, Denver, 1995. p. 503–10.

[3] Oviatt S, Cohen P. Multimodal interfaces that process what comes naturally. Communications of the ACM 2000;43(3):45–53.

[4] Sutherland I. Sketchpad: a man-machine graphical communication system. Proceedings of the 1963 Spring Joint Computer Conference. Baltimore, MD: Spartan Books, 1963. p. 329–46.

[5] Herot CF. Graphical input through machine recognition of sketches. Proceedings. SIGGRAPH '76. 1976. p. 97–102.

[6] Negroponte N. Recent advances in sketch recognition. AFIPS (American Federation of Information Processing) National Computer Conference, vol. 42, Boston, MA, 1973. p. 663–75.

[7] Goel V. Sketches of thought. Cambridge, MA: MIT Press, 1995.

[8] Funt B. Problem solving with diagrammatic representations. Artificial Intelligence 1980, 13(3).

[9] Larkin J, Simon H. Why a diagram is (sometimes) worth 10,000 words. Cognitive Science 1987;11:65–99.

[10] Robbins E. Why architects draw. Cambridge, MA: MIT Press, 1994.

[11] Fish J, Scrivener S. Amplifying the mind's eye: sketching and visual cognition. Leonardo 1990;23(1):117–26.

[12] van Sommers P. Drawing and cognition — Descriptive and experimental studies of graphic production processes. Cambridge, England: Cambridge University Press, 1984.

[13] Ullman D, Wood S, Craig D. The importance of drawing in the mechanical design process. Computer Graphics 1990;14(2):263–74.

[14] Wong YY. Rough and ready prototypes: lessons from graphic design. Proceedings ACM Human Factors in Computing (CHI) 92.1992. p. 83–84.

[15] Novak G. Diagrams for solving physical problems. In: Glasgow J, Narayanan NH, Chandrasekaran B, editors. Diagrammatic reasoning. Menlo Park, CA: AAAI Press/MIT Press, 1995. p. 753–4.

[16] Glinert E, Tanimoto S. PICT: an interactive graphical programming environment. Transactions IEEE 1984;17(11):7–25.

[17] Lakin F. Visual grammars for visual languages. National Conference on Artificial Intelligence (AAAI). 1987. p. 683–8.

[18] Kirsch R. Experiments in processing pictorial information with a digital computer. Proceedings. Eastern Joint Computer Conference, December 9–13. Inst. Radio Eng. and Assn. Computing Mach., 1957.

[19] Futrelle R, Kakadiaris IA, Alexander J, Carriero CM, Nikolakis N, Futrelle JM. Understanding diagrams in technical documents. IEEE Computer 1992;25(7):75–8.

[20] Golin E. A method for the specification and parsing of visual languages. PhD, Brown University, 1991.

[21] Golin E, Reiss SP. The specification of visual language syntax. Proceedings. IEEE Workshop on Visual Languages. New York: IEEE Press, 1989. p. 105–10.

[22] Helm R, Marriott K, Odersky M. Building visual language parsers. In: Human factors in computing systems (CHI '91). New Orleans, LA: ACM Press/Addison Wesley, 1991. p. 105–12.

[23] Marriott K. Constraint multiset grammars. In: IEEE Symposium on Visual Languages. St. Louis: IEEE, 1994. p. 118–25.

[24] Meyer B. Pictures depicting pictures: on the specification of visual languages by visual grammars. Proceedings of the IEEE Workshop on Visual Languages 1992. Seattle, WA: IEEE, 1992. p. 41–7.

[25] Wittenburg K, Weitzman L. Visual grammars and incremental parsing. IEEE Workshop on Visual Languages, Skokie, IL, 1990. p. 235–43.

[26] Apte A, Kimura TD. A comparison study of the pen and the mouse in editing graphic Diagrams. Proceedings. 1993 IEEE Symposium on Visual Languages. Los Alamitos: IEEE Computer Society Press, 1993. p. 352–7.

[27] Rubine D. Specifying gestures by example. Computer Graphics 1991;25(4):329–37.

[28] Zhao R. Incremental recognition in gesture-based and syntax-directed diagram editors. Proceedings. INTERCHI '93. Amsterdam: ACM/Addison-Wesley, 1993. p. 95–100.

[29] Landay JA, Myers BA. Interactive sketching for the early stages of interface design. CHI '95 — Human Factors in Computing Systems. Denver, Colorado: ACM Press, 1995. p. 43–50.

[30] Egenhofer M. Spatial-query-by-sketchIn. In: IEEE Symposium on Visual Languages. Boulder, CO: IEEE, 1996. p. 60–7.

[31] Moran T, van Melle W, Chiu P. Spatial interpretation of domain objects integrated into a free form whiteboard. Proceedings of UIST '98. San Francisco: ACM, 1998. p. 174–84.

[32] Moran TP, Chiu P, Melle Wv, Kurtenbach G. Implicit structures for pen-based systems within a freeform interaction paradigm. CHI '95 — Human Factors in Computing Systems. Denver, Colorado: ACM Press, 1995. p. 487–94.

[33] Stahovich TH, Davis R, Shrobe H. Generating multiple new designs from a sketch. Proceedings. Thirteenth National Conference on Artificial Intelligence, AAAI-96. 1996. p. 1022–9.

[34] Schilit BN, Golovchinsky G, Proce M. Beyond paper: supporting active reading with free form digital ink annotations. In ACM SIGCHI, 1998. p. 249–56.

[35] Ishii H, Kobayashi M. Clearboard: a seamless medium for shared drawing and conversation with eye contact. CHI '91 — Human Factors in Computing Systems, Monterrey, CA, 1991. p. 525–32.
[36] MacKay W, Verlay G, Carter K, Ma C, Pagani D. Augmenting reality: adding computational dimensions to paper. Communications of the ACM 1993;36(7):96–7.
[37] Mackay WE, Pagani DS, Faber L, et al. Ariel: augmenting paper engineering drawings. In: Conference Companion CHI95. Denver: ACM, 1995. p. 421–2.
[38] Newman W, Wellner P. A desk supporting computer based interaction with paper documents. ACM Human Factors in Computing (CHI) 92. New York: ACM, 1992. p. 587–92.
[39] Wellner P. Interacting with Paper on the Digitaldesk. CACM 1993;36(7):87–96.
[40] Evans TG. A program for the solution of a class of geometric analogy intelligence — test questions. In: Minsky M, editor. Semantic information processing. Cambridge MA: MIT Press, 1968. p. 271–353.
[41] Zeleznik R, Herndon KP, Hughes JF. SKETCH: an interface for sketching 3D scenes. SIGGraph '96 Conference Proceedings. 1996. p. 163–70.
[42] Clowes MB. On seeing things. Artificial Intelligence, 1971;2:79–116.
[43] Grimstead IJ, Martin RR. Creating solid models from single 2D sketches. In: Rossignac J, Hoffmann C, editors. Third Symposium on Solid Modeling and Applications. Salt Lake City: ACM, 1995. p. 323–37.
[44] Huffman DA. Impossible objects as nonsense sentences. In: Meltzer B, Michie D, editors. Machine intelligence. Edinburgh University Press, 1971. p. 295–323.
[45] Do EY-L. The right tool at the right time: investigation of freehand drawing as an interface to knowledge based design tools. PhD, Georgia Institute of Technology, 1998.
[46] Abowd GD, Atkeson CG, Feinstein A. Teaching and learning as multimedia authoring: the classroom 2000 project. In ACM Multimedia '96. ACM, Boston, 1996. p. 187-198.
[47] Chiu P, Wilcox L. A dynamic grouping technique for ink and audio notes. Proceedings. UIST 98. ACM, 1998. p. 195–202.
[48] Moran TP, Palen L, Harrison S et al. I'll get that off the audio: a case study of salvaging multimedia meeting records. Proceedings. CHI'97 Conference on Human Factors in Computer Systems, Atlanta, GA, 1997
[49] Mynatt E, Igarashi T, Edwards WK, LaMarca A. Flatland: new dimensions in office whiteboards. Proceedings ACM Human Factors in Computing (CHI 99). New York: ACM, 1999. p. 346–53.

VR Sketchpad

Create Instant 3D Worlds by Sketching on a Transparent Window

Ellen Yi-Luen Do
Design Machine Group, Department of Architecture, University of Washington, Seattle, WA 98195-5720, USA

Key words: pen-based interface, freehand sketches, diagramming, transparent window, Virtual Reality Modelling Language (VRML)

Abstract: This paper describes VR Sketchpad, a pen-based computing environment for inputting and locating 3D objects in a virtual world. Designer can use the transparency layers to quickly trace and extract any image underlay from other application software. The 3D scene generation has three levels of complexity: simple extrusion of any drawn lines of shapes (i.e., straight or curved wall and column extrusion), solid modelling from a given geometric object representation (spheres, cones and boxes), and complex configuration with objects from graphics library (furniture layout).

1. INTRODUCTION

As on-line communication and entertainment enterprises increasingly use three-dimensional geometry, the ability to quickly create 3D content becomes important. Building three-dimensional worlds for the World Wide Web usually involves complicated CAD and modelling software that uses WIMP (windows, icons, menus, and pointers) interfaces. VR Sketchpad gives creators of three-dimensional content a quick and easy way to author 3D virtual worlds by sketching without coding VRML (Virtual Reality Modelling Language) or learning to model using CAD software.

With experience and training, designer can use CAD systems to create accurate models of 3D worlds. CAD systems are good at producing precise 3D models and supporting detailed editing and revision. However, pen-based systems enable designers to communicate ideas rapidly through approximate

sketches with low overhead (direct manipulation by freehand drawing). Sketching requires no precision or specialised knowledge, and makes it easy for designers to draw, evaluate, and explore as well as to revise and correct. Sketching is useful in settings that require a fast turn-around creation-feedback cycle, for example, during conceptual design. These settings include building and product design, as well as designing computer games and virtual worlds for VR applications, on-line exhibits, and cyber communities on the Web.

Providers of 3D content often start conceptual design by sketching with pencil or markers on paper or by building a physical model to simulate the proposed design. Later they use CAD software or game level editors to create these imagined 3D worlds. The process of making a 3D model with structured commands and operations is far more elaborate than using a pen and paper to sketch. A freehand drawing user interface for 3D modelling—as VR Sketchpad illustrates—provides designers more freedom to explore than a WIMP interface.

This paper presents VR Sketchpad, a freehand drawing interface for rapidly generating three-dimensional geometry in VRML format by tracing diagrams on a transparent window over an image source. VR Sketchpad employs a novel approach for creating 3D worlds. It enables designers to simply make diagrams for spatial partitions such as walls, columns and furniture on a 2D floor plan to quickly generate 3D worlds on the Web. For example, a quick sketch of a floor plan of an exhibition hall generates a virtual scene with walls and columns. Immediately, the designer can experience a virtual walkthrough from a Web browser. Preferred viewer positions marked on the floor plan are translated and embedded into the VRML viewpoint control panel, providing a guided tour path. As the visitor browses the virtual exhibit hall, VR Sketchpad displays the visitor's position on the floor plan sketch. This instant feedback from 3D VR world to 2D floor plan supports navigation and understanding of the created space for both visitors and designers.

2. RELATED WORK

Many projects investigate the use of transparent windows as an interface to facilitate information display or object generation. For example, PAD (Perlin and Fox 1993) uses transparent windows to support "zooming" from abstract information to more details. A reader can navigate through the document space by applying transparent pads on top of the area of interest. Toolglass and Magic Lenses (Bier, Stone et al. 1993) adopt a similar approach of having transparent objects carry with them functionality for user

interaction. They employ transparency to allow users to construct new artefacts with overlapping elements. Translucent Patches (Kramer 1994) use freeform shapes for copying and grouping information. The different patches can overlap and interact with each other (e.g., performing mathematical calculation of figures from overlapping patches).

Electronic Cocktail Napkin's trace layer allows selecting and copying of diagram sketches between different layers, it also supports re-arrangement of the layers through a post-up action with thumbnail representation of the drawings (Gross 1994; Gross 1996; Gross and Do 1996). The Napkin program also supports different pen types. Users of the system can leave marks in different thickness and darkness with the pressure and velocity data derived from the pen stylus. Trinder argues that architects use transparent media such as tracing paper and drafting film not only to bring images together, but also to compose configuration with hierarchy (Trinder 1999). His 'transparent medium' investigates using simple tools such as "push and pull" to translate pixel maps between two layers and a "sketching tool" that translate stroke information as different thickness. His empirical studies on twelve test subjects reveal that using transparent layer and sketching are a good way for design professionals to interact with computer software.

Several researchers also conducted usability studies of transparent user interfaces (Harrison 1996; Cox, Chugh et al. 1998). Their findings are encouraging. They found people could use transparent layers easily to shift their focus rapidly between the different views (e.g., a layer containing an instance of an element detail and an overview layer). This suggests that transparency user interfaces are useful.

Three-dimensional scene creation is of interest for many researchers at the human computer interaction paradigm and design research. For example, SKETCH is a system (Zeleznik, Herndon et al. 1996) that enables users to use gesture commands to specify the creation of 3D objects. Sketching three axial arrow lines will generate a box with corresponding dimensions. The efforts from Grimstead and Martin describe the method of constructing a solid model from a 2D line drawing. The process includes hidden line removal, line labelling, region recognition and alignment, and adjustment of vertices (Grimstead and Martin 1995). Digital Clay has a similar approach, however, unlike Grimstead's and Martin's approach, it uses freehand drawing as a front end interface instead of line drawing. It is a 3D isometric sketching environment that uses constraint propagation method of Huffman and Clowes (Schweikardt and Gross 1998) to infer the occluding, convex and concave edges to build 3D model.

Quick-sketch (Eggli, Bruderlin et al. 1995) automatically adjusts angles and connects freehand sketches to infer construction of geometric shapes and arcs in a 2D view. It also extrudes the plan profile to generate 3D shapes

using object library from a graphical user interface toolkit. The project DDDoolz allows user to add or delete connected voxel boxes in six directions (of the cube) to create 3D scenes (Achten, Vires et al. 2000). Teddy (Igarashi, Matsuoka et al. 1999) enables user to create spherical object models by drawing a 2D profile (a closed oval) then rotate the drawing sideways to draw the section for extrusion.

Our project, VR Sketchpad is well located in this series of research. It uses freehand sketching as a way to create 3D objects. On the most basic level, similar to Quick-sketch, VR Sketchpad uses extrusions to generate shapes. Beyond simple extrusion, VR Sketchpad also supports generations of solid objects such as boxes and spheres, as well as object placements such as furniture layout creation from 2D symbol configurations.

3. VR SKETCHPAD IN ACTION

VR Sketchpad employed Electronic Cocktail Napkin's various capabilities of diagram parsing and customised user training (Gross 1994; Gross 1996; Gross and Do 1996) as a base graphic engine. The symbolic processor in the Napkin program enables designers to train and define their own graphic symbols and gestures (circle, line, arrow, etc) by drawing examples into the system (with a pen and tablet). Contrary to the approach of PDA (Personal Digital Assistant such as Palm Pilot) that can recognise only a fixed set of shapes, our system lets designers create their symbol library with personalised definitions and drawing features (sequence, pressure, angle, etc). VR Sketchpad then provides meaning association (circle to column, arrow to viewpoint) and performs geometry translations. Figure 1 below illustrates the system architecture of VR Sketchpad.

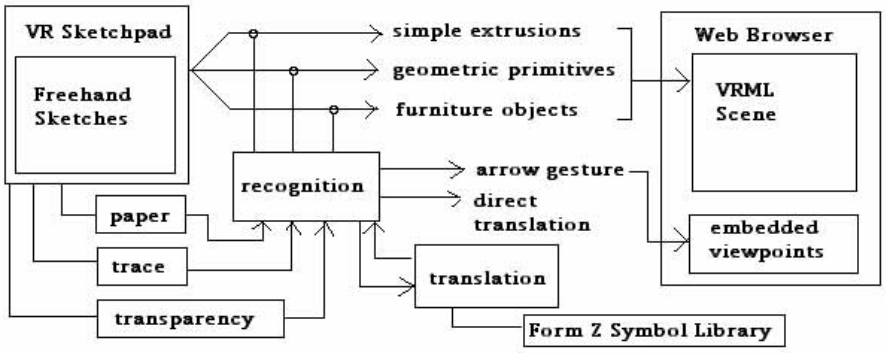

Figure 1. System architecture of VR Sketchpad

Three types of interfaces are available for the sketching input. First, the regular drawing board, paper like interface that allows designers to sketch onto a digitising tablet using a pen stylus. The drawing surface is opaque and displays sketches as drawn. The second type of interaction allows bringing in a picture underlay (raster image) and multiple translucent layers on top. Figure 2 shows a construction drawing brought in to the paper drawing board environment. Several trace layers are overlaid on top to add modifications and annotation diagrams. The thumbnail images on the top portion of the drawing board ("post-it" layer) allows easy management (hide, overlay) of the different trace layers. Each trace layer adds opaqueness to the drawing board while underlay drawings remain visible through the layers. Figure 3 shows a hand drawn diagram on the transparent window overlays on top of other applications (e.g., Form•Z model, screen capture of a drafting document, and a PDF file). The transparent window maintains the drawing functionality of the previous two types of interface (paper, trace layer).

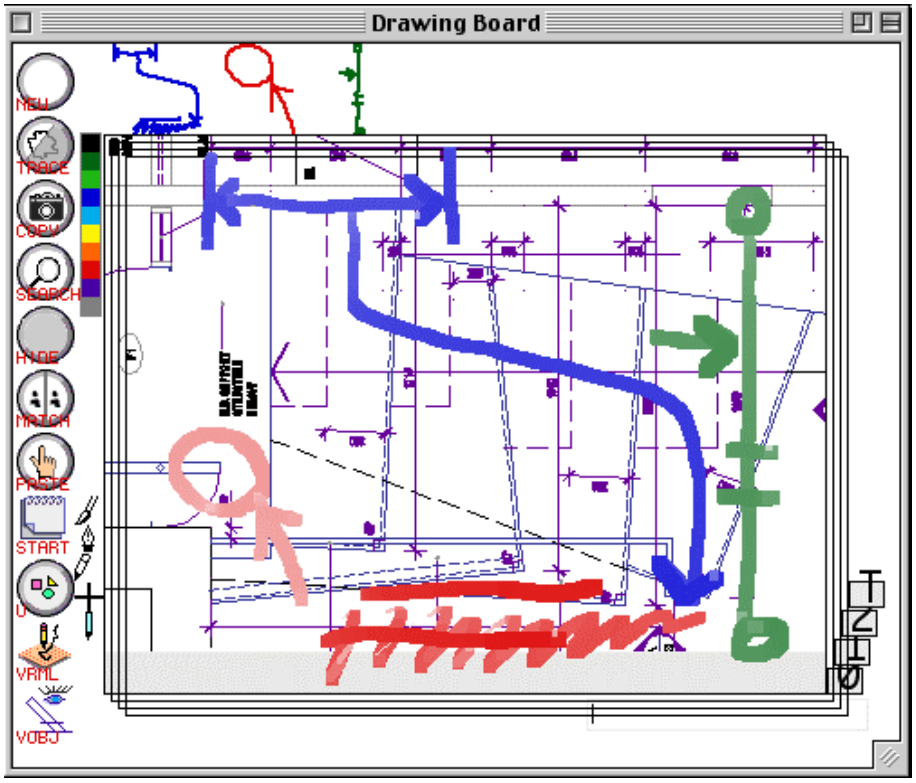

Figure 2. A construction drawing brought in as picture underlay with annotations and modifications on different trace layers. (lower right corner shows tabs for each trace layer for easy selection).

A transparent window is a window (drawing surface) that displays the screen's pixel map as a background for its contents. It appears as though you are drawing on top of the windows belonging to other software applications, or the operating system. With inter-application communication protocols, the Sketchpad would be able to directly interact with the underlay applications. Currently, when user resizes or moves the window, the transparent window instance captures the screen pixel map underneath, and builds the bitmap information into a memory cache. The content drawing method of the window draws the background images first then displays the current user sketch objects.

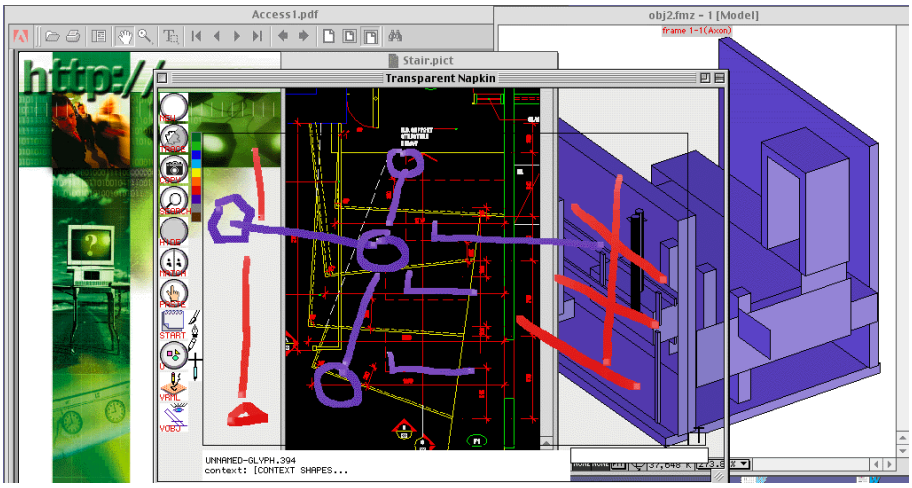

Figure 3. Transparent window overlays on top of three different applications (left to right: PDF document, drafting document, and a Form•Z model). User can sketch and trace images at the transparent window.

With initial input of sketches, VR Sketchpad's processor recognises the drawings (Figure 4 left), it translates the drawing into VRML objects, and then launches a Web browser to display the result (Figure 4 right, VRML enabled Netscape browser).

VR Sketchpad has three levels of diagram recognition and translation processing. First, simple shapes such as lines and circles are extruded to make walls and columns as shown in Figure 4.

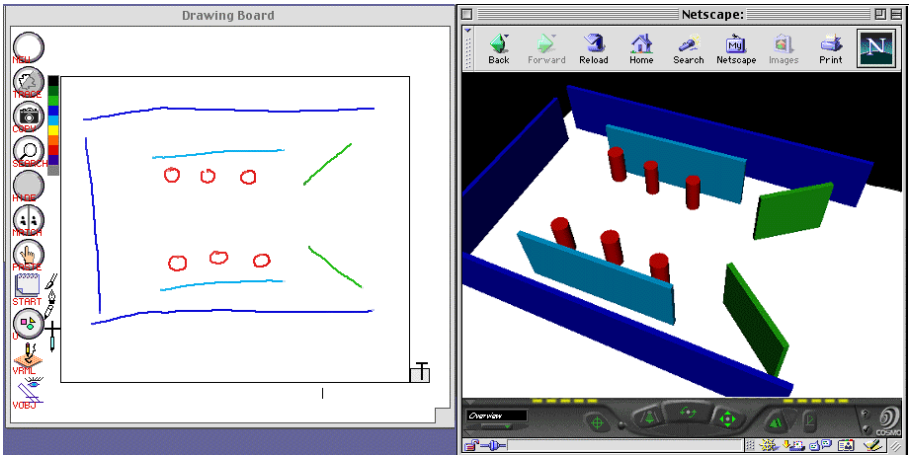

Figure 4. Lines and circles are extruded to make VRML model of walls and columns.

The basic level of translation is simple extrusion. VR Sketchpad also supports extrusions of free form, irregular shapes as desk height (wall height is to the ceiling). The translation processor of VR Sketchpad takes all the user input points in the stroke and converts them into VRML surfaces.

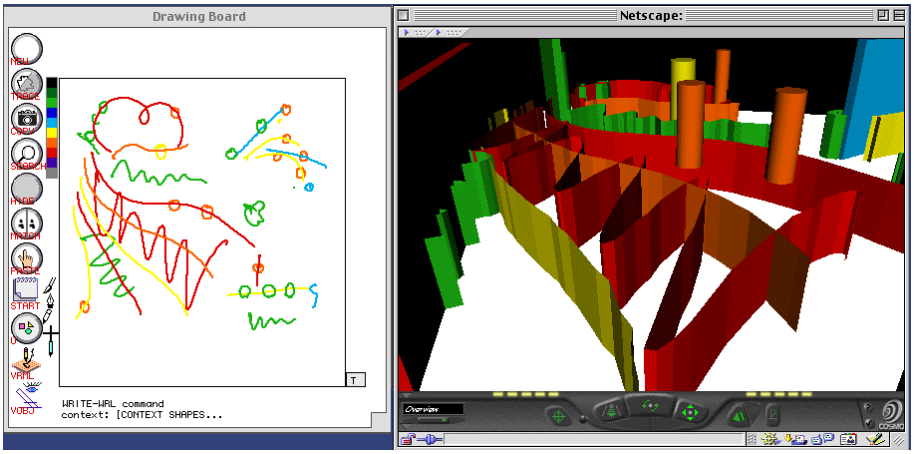

Figure 5. All curve shapes and lines can also be extruded to make curvy partitions. (Note: in this illustration, cylinders are inserted over the partition walls).

The second level of diagram processing deals with solid object mapping and placements. Instead of extrusion, it recognises geometric primitives through symbol conventions or configurations. For example, a rectangle is identified as a box object and translated as a solid cube. A circle that is concentric with a dot indicates a sphere object with desired radius. Figure 6 shows a VRML scene constructed from sketches of geometric primitives.

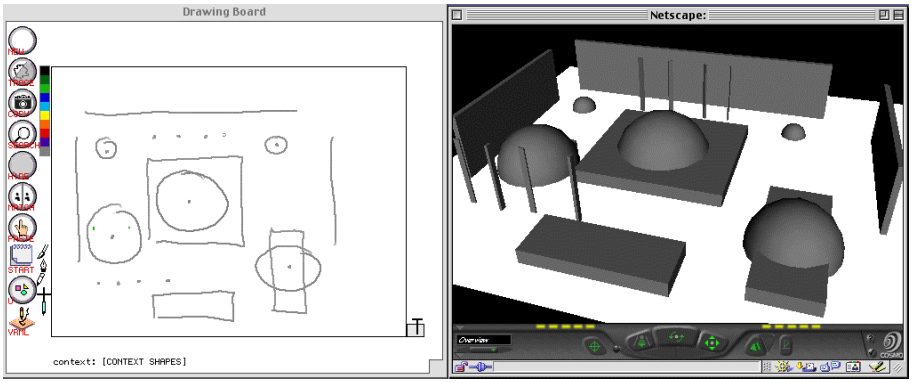

Figure 6. Graphic symbols can be translated into solid VRML object models.

The third level of processing involves recognition of the diagram configurations and the translation with calling of external 3D pre-made objects from a symbol library (e.g., furniture library from Form•Z). Figure 7 shows some shorthand symbols of architecture elements for kitchen and bathroom. Designers can train the configuration processor to recognise graphic standards or personal symbols by drawing freehand diagrams. For example, a toilet can be defined as a rectangle directly above two concentric circles; a dining table set can consist of four chairs (rectangles) surrounding a round table (circle) as shown in Figure 7-9.

Figure 7. Graphic symbols for stove top, sinks, toilet and bath rub.

With the combination of "level one's" partitions (walls and columns) and the translations of higher level complex figures (furniture elements from 3D symbol library), VR Sketchpad user can quickly sketch out an interior layout configuration and see the design in 3D VRML format by pressing a button. Figure 8 shows a furniture layout diagram and the corresponding 3D scene. Notice that all the furniture objects are in real human scale because we use real life object symbols from the 3D graphic library. Diagrams in this instance serve as placement reference instead of scale.

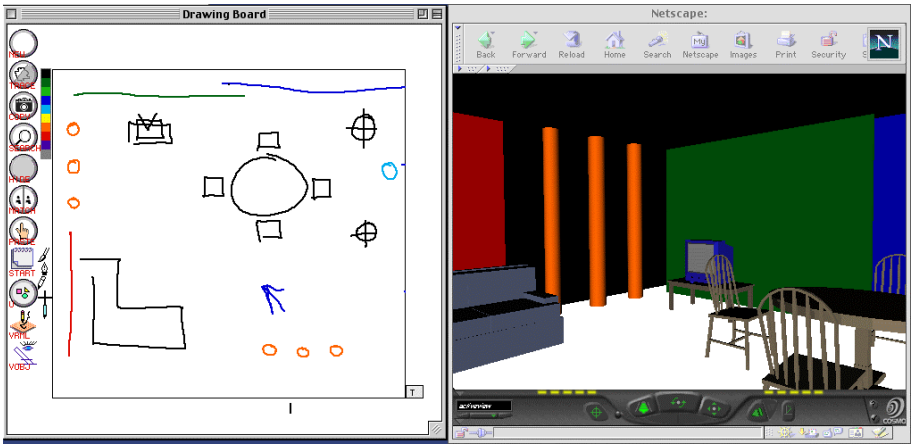

Figure 8. Furniture layout sketch (TV, couch, dining table set, columns, and walls) creates a 3D VRML world.

One important feature of VR Sketchpad is that it also supports gesture recognition for user to specify preferred viewpoints in the 3D VRML scene. The arrow in Figure 8 defines a standpoint and viewing direction toward the scene. The Web browser window on the right shows a particular view angle as indicated on the drawing on the left.

Figure 9 shows that user can draw a sequence of arrows to indicate location of interests and therefore define a viewing path into the 3D world. There is no limit on how many kinds of object configurations or how many objects can be placed in a 3D VRML scene (memory permitting). Users can define their own shorthand symbol sketches and use them to indicate desired element placements. The 3D scene in Figure 9 shows a particular view (lower left arrow on the floor plan) behind the columns looking toward the dining table and chairs, with walls and columns in the background.

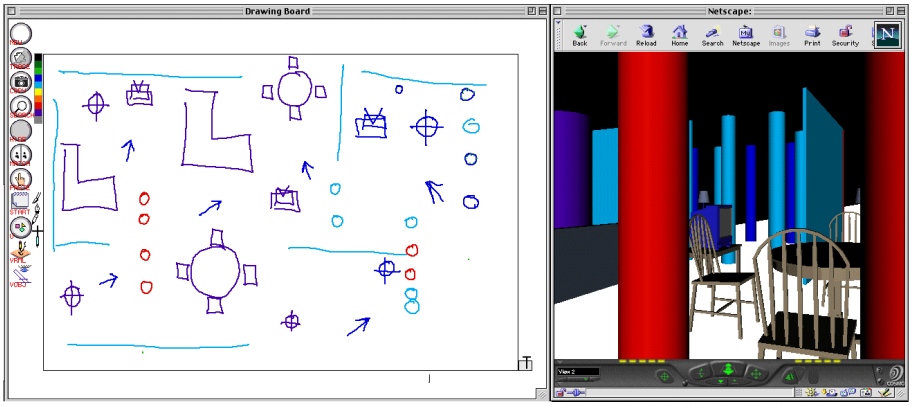

Figure 9. A sequence of arrows in the floor plan sketch indicates places of interest and provides a guided tour to the 3D VRML scene. The bottom left arrow on the drawing (left) shows a location and view into VRML scene on the Web browser (on the right).

4. DISCUSSION AND FUTURE WORK

VR Sketchpad is an application that allows user to sketch in two-dimensional freehand diagrams to quickly generate three-dimensional VRML scenes on the web. The idea of the project came from the teaching of beginning architecture students. The novice architecture students have a hard time to see the relations between the two-dimensional floor plan as a three-dimensional space. The obvious tool for the students to understand the interplay between plan view and its corresponding space is to extrude their 2D drawings into 3D form. Therefore, VR Sketchpad was created to illustrate the relationship between 2D and 3D. VR Sketchpad has then added many features and now supports not just direct one-to-one mapping of object translations but also the higher level complex figure configurations and substitutions (diagram meaning, as well as replacement object symbols). It also shows great potential for on-line gaming and artefact generation for the Web.

The system implementation is straightforward. The interface is simple, yet powerful. Many designers requested to test the system once they saw it in action. We plan to release the prototype VR Sketchpad to architecture students to use in their design studio for conceptual design. We would also like to conduct formal usability studies of the system to find out the problems and desirable functionality.

We are currently adding a slider function in the drawing palette to allow user to specify the extrusion heights. We are exploring adding a 2D sectional view with the 2D floor plan view. We are extending VR Sketchpad to support creating virtual terrain with freehand sketches. Another direction of

the project is to provide an interface to support the display and selection of product catalogue (e.g., Sweets and Steelcase) items with hand drawn symbols (wall, furniture, etc.). We would like to add VR Sketchpad as an interface to a modelling program (e.g., autodessys inc. recently announced its plan for providing interface into their Form•Z software). We have chosen to implement in VRML format because of its ISO (International Standards Organisation) status and because all modelling programs have translation modules to read it as input.

There are many questions worth investigating. Should we extend 2D sketch with isometric view extractions? A previous project Digital Clay (Schweikardt and Gross 1998) from our research group addresses exactly this concern. What does it mean to be able to sketch in 3D? Shall we sketch into the VRML scene to modify the design "right on the spot"? A project in our research group called Space Pen (Jung 2001) is currently exploring this aspect of sketching interactions in 3D.

VR Sketchpad works, but it also has flaws. For example, the furniture placement function uses only the sketch locations as reference for placement and discards the dimensional info (because the furniture models are constrained to real dimension). We have explored placing element dimensions according to user's sketches. However, this would generate scaled down (or up) furniture (that's not usable in real life). It seems to be a logical next step to embed constraints into the sketches such as Stretch-a-Sketch (Gross 1994) so that user, while drawing, will be aware of the human scale concerns and ergonomic dimensions. Or one could display scale and a grid underlay in the drawing surface. We also plan to extend the transparent windows as an interface to other applications. For example, sketching over a VRML scene on a Web browser could potentially add new elements to the 3D scene.

5. ACKNOWLEDGEMENTS

This research was supported in part by the National Science Foundation under Grant numbers IIS-96-19856 and IIS-0096138. The views contained in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

6. REFERENCES

Achten, H., B. D. Vires, et al., 2000. *DDDOOLZ*. CAADRIA 2000. B.-K. Tan, M. Tan and Y.-C. Wong. Singapore, National University of Singapore: 451-460.

Bier, E. A., M. C. Stone, et al., 1993, "Toolglass and Magic Lenses: The See-Through Interface", *Computer Graphics* (SIGGRAPH '93 conference proceedings), 73-80.

Cox, D. A., J. S. Chugh, et al., 1998. *The Usability of Transparent Overview Layers*. CHI 98: 301-302.

Eggli, L., B. D. Bruderlin, et al., 1995. *Sketching as a Solid Modeling Tool*. Third Symposium on Solid Modeling and Applications. C. Hoffmann and J. Rossignac. Salt Lake City, ACM: 313-321.

Grimstead, I. J. and R. R. Martin, 1995. *Creating Solid Models from Single 2D Sketches*. Third Symposium on Solid Modeling and Applications. C. Hoffmann and J. Rossignac. Salt Lake City, ACM: 323-337.

Gross, M. D., 1994. *The Fat Pencil, the Cocktail Napkin, and the Slide Library*. ACADIA '94. M. Fraser and A. Harfmann. St. Louis, MO: 103-113.

Gross, M. D., 1994. *Stretch-A-Sketch, a Dynamic Diagrammer*. Proceedings of the IEEE Symposium on Visual Languages '94. A. Ambler, IEEE Press: 232-238.

Gross, M. D., 1996, "The Electronic Cocktail Napkin - working with diagrams", *Design Studies* 17 (1), 53-69.

Gross, M. D. and E. Y.-L. Do, 1996. *Demonstrating the Electronic Cocktail Napkin: a paper-like interface for early design*. CHI 96, Conference on Human Factors in Computing Systems. Vancouver, British Columbia, Canada, ACM. Conference Companion: 5-6.

Harrison, B., 1996, *Design and Evaluation of Transparent User Interfaces*. PhD: Toronto, University of Toronto.

Igarashi, T., S. Matsuoka, et al., 1999, "Teddy: a sketching interface for 3D freeform design", *Proceedings of the SIGGRAPH 1999 annual conference on Computer graphics* , 409-416.

Jung, T., 2001. *Space Pen: annotation and sketching on 3D models on the Internet*. CAAD Futures 2001. Eindhoven, Kluwer Academic Publishers.

Kramer, A., 1994. *Translucent Patches - dissolving windows.* ACM Symposium on User Interface Software and Technology, Marina del Rey, CA, ACM Press.

Perlin, k. and D. Fox, 1993. *PAD: an alternative approach to the computer interface*. SIGGRAPH. Anahelm: 57-62.

Schweikardt, E. and M. D. Gross, 1998. *Digital Clay: Deriving Digital Models from Freehand Sketches*. Digital Design Studios: Do Computers Make A Difference? ACADIA 98. T. Seebohm and S. V. Wyk. Quebec City, Canada, ACADIA: Association for Computer-Aided Design in Architecture: 202-211.

Trinder, M., 1999. *The Computer's Role in Sketch Design: A Transparent Sketching Medium*. Computers in Building; Proceedings of the CAAD Futures '99 Conference. G. Augenbroe and C. Eastman, Kluwer Academic Publishers: 227-244.

Zeleznik, R. C., K. P. Herndon, et al., 1996, "Sketch: An Interface for Sketching 3D Scenes", *SIGGRAPH '96* , 163-170.