EMERGENCE IN A RECOGNITION BASED DRAWING INTERFACE

MARK D. GROSS
Design Machine Group, Department of Architecture
University of Washington
+1.206.616.2817
mdgross@u.washington.edu

Abstract People perceive patterns in representations, patterns that may not have been initially intended. This phenomenon of emergence is deemed to play an important role in design. Computer based design assistants can and should support this human perceptual ability, using pattern recognition to anticipate human designers' perception of emergent shapes and supporting the subsequent manipulation of and reasoning with these shapes as part of the design. Freehand drawing programs with gesture recognition are well positioned to implement shape emergence. Support for emergent shapes in the Back of an Envelope system is described.

1. Emergence in Visual Representations and Design

Textbook examples of visually ambiguous figures such as the "two faces or a vase," "young or old woman", or the Necker cube (Solso 1994) remind us that visual representations need not dictate a single reading. Yet these figures are more than curiosities: They illustrate a powerful visual-cognitive effect that artists and designers take advantage of in everyday work. Multiple readings in visual representations play a special role in designing, especially in its conceptual, creative phases. They provide a designer opportunities to consider alternative interpretations of the representation. These readings serve several functions: they recall relevant (but previously unconsidered) design cases, they suggest specific operations to apply to the representation, and they can suggest ways to restructure or recast a design problem.

Multiple readings and emergent forms are not always helpful in visual communication. Sometimes a figure should mean just what its maker intended and no more. Visual languages developed to program computers assume a one-to-one mapping between a visual representation and a computational structure. When a diagram is intended to produce a specific outcome, as in a specific data structure or sequence of instructions, multiple interpretations are undesirable. In a programming language syntactic ambiguity is a flaw, not a feature: Therefore visual programming languages (and most visual human-computer interfaces) appropriately avoid ambiguity and multiple readings.

This is not to say that the benefits of emergent form are limited to the artistic domains of design. For example, an analog circuit designer might specify parts of a circuit independently. Upon drawing the circuit layout the designer notices components from different parts of the circuit are located so

as to interfere with one another electrically, or generate too much heat. The problem only becomes apparent when the designer makes and examines the drawing, noticing the unintended and incidental proximity of two otherwise unrelated parts of the circuit.

1.1 EMERGENCE IN DESIGN

In his studies of designers in action, Don Schön noted the importance of drawing as a part of the reflective process. Schön and Wiggins (Schön 1992) argued that making a drawing is a necessary step that enables the designer to step back momentarily from the making process and reflect on the drawing, and that in this reflection, or re-examination, the designer sees patterns that stimulate the next cycle of drawing. Goldschmidt (Goldschmidt 1991, 1999) explores the way drawings 'talk back' to their makers in a series of empirical experiments with both novice and experienced designers. Mitchell points out, "Drawings are valuable precisely because they are rich in suggestions of what might be" (Mitchell 1990). Stiny elaborates, "An evolving design may thus have alternative descriptions that may change from time to time in unanticipated ways; it may be decomposed and manipulated in this way and in another way later without difficulty. The structure of the computer drawing, however, makes all this impossible. The structure is fixed in definite drawing operations" (Stiny 1990). Edmonds et al (Edmonds, Candy, Jones et al. 1994) develop this argument and outline a pixel based, machine vision approach for a computational implementation, described in detail by Soufi and Scrivener (Soufi and Scrivener 1992). These design researchers make the case that emergent forms in drawing is an important phenomenon and a worthwhile goal for a computer supported design system.

Proposals for how shape emergence might be implemented computationally have been made. Stiny shows how replacement rules can be used to recognize emergent shapes (Stiny 1993). Gero and Yan (Gero and Yan 1993) show how data driven search—extending the line segments in a figure to generate new intersections—can generate a set of emergent shapes, similar to the more primitive system built by Tan (Tan 1990).

Despite the widespread interest in emergent shapes, working implementations remain scarce. For the most part they have been limited to demonstrations of techniques for identifying emergent shapes. For example, Tan's Emergence II project (Tan 1990) showed how extending segments in a drawing to make construction lines and finding intersection points with other similarly constructed lines could generate a set of candidate emergent

shapes. This method works with hard line drawings rather than freehand sketches, although their techniques might be adapted to a freehand system. Nagakura's system (Nagakura 1990) for shape recognition and transformation used a directed search to identify specific subshapes in a drawing, and Nagakura proposed to use this more generally to support shape emergence. Liu (Liu 1993; Liu 1995) explored using neural networks to recognize emergent shapes in simple drawings.

PerSketch (Saund and Moran 1994) is one of the few systems to support shape emergence in a drawing environment: Computer vision image processing techniques were used to build what the authors termed a WYPIWIG (What You Perceive is What You Get) system. The system decomposed drawing elements into arcs and segments defined by intersections and corners, and the user could edit the drawing at various levels. The user could select and operate on the lowest level of segments and arcs, or could select any shape composed of connected segments and arcs, identifying the shape by tracing over its outline. We have adopted a similar means of selection, although the system identifies candidates only through a recognition based process.

Much of the work on emergent shapes emphasizes the salience of specific geometry: for example, closed figures created as a by-product of intentional drawing will emerge. Viewing emergence this way overlooks the important role of drawing context and the trained eye. The trained eye may see more, or different, shapes in the same drawing than the eye of the casual observer. Painters, photographers, and architects speak of "learning to see" as an important part of their education. It isn't so much that emergence is in the eye of the beholder, as one report concludes, "emergence ***is*** the eye of the beholder" (Edmonds, Moran and Do 1998). The recognition driven approach developed here emphasizes the role that experience has on shape emergence.

1.2 TYPES OF EMERGENCE

Emergence is the term used for perception of (unintended) patterns in a representation, and specifically "emergent shapes" describes the phenomenon of multiple readings of a diagram. Three sources of these multiple readings are:

• intersecting figures: two or more figures intersect to create recognizable subfigures.

• alternative configurations: a configuration's parts can be grouped in several ways.

• figure-ground reversals: a new figure is formed by the edges of several drawn figures.

Figure 1. Three forms of visual emergence:(a) intersecting figures,(b) alternative configurations, and (c) figure-ground reversal.

Intersecting figures (figure 1a) occur when one or more marks intersect to make figures that were not initially intended. This canonical example shows three of the figures created as a side effect of drawing two intersecting rectangles, one horizontal and one vertical.

Alternative configurations (figure 1b) occur when the designer draws two (or more) configurations made up of components that once drawn, can be regrouped or read as belonging to a new configuration. The designer has drawn a plan diagram with two tee-configurations (left); when juxtaposed this way the stems of the tees form a doorway configuration (right).

Figure-ground reversal (figure 1c) occurs when a mark or set of marks bound or define a recognizable shape. For example, in drawing the ground plan or 'footprint' of city block-sized buildings, the designer might create a plaza or street between the blocks. While drawing the blocks the designer is actually (also) thinking about the space between. This emergent shape is not inadvertent but intended, although its boundaries comprise the edges of other shapes.

2. Computational Drawing Environments

2.1 MENU DRIVEN COMPUTER AIDED DESIGN PROGRAMS

Most computational drawing and design environments are media: they support making—not interpreting—graphical representations. Paint

programs, for example, simulate traditional paper and ink instruments but they maintain no representation of the figures drawn beyond the pixels left on the screen. Drawing and CAD programs require a designer to assemble a drawing out of primitive geometry elements, and they maintain a representation of elements assembled. A CAD program interprets a figure in only one way—just as the designer constructed it. On the other hand, once a figure is drawn the designer may read it in several different ways. In short, the machine does not share the user's perceptual ability to recognize emergent forms.

Some drawing and CAD programs now recognize intersection points of figures and use these points to support 'smart snap' and 'smart trim' operations. A more comprehensive recognition based approach, however is lacking. Recognizing and retaining multiple readings of a drawing— just as designer might do—could better support designing, as the designer switches between alternative readings. This, however, entails significant changes in the underlying software architecture of these programs as well as advances in the design of their user interface. Specifically, the data structures for describing drawings would have to be extended to support ambiguity and indeterminacy as well as multiple and alternative configurations of the drawing components, and the human-computer interface would have to support these extensions to the data structures.

The architecture of drawing and CAD programs reveals a popular prejudice about drawing. Drawing is seen as making a record of a picture that is already in the designer's head. The role of the pencil, or the CAD program, is simply to facilitate this recording process. Accordingly, CAD programs enable the designer to select from libraries of graphical elements and add them to a drawing. The programs allow the designer to edit the drawing, moving and replacing elements, but the programs do not allow for reinterpreting the drawing once it is made. The question of interpreting does not come up, because the program relies on the designer's unambiguous articulation (selection, placement) of drawing elements in the first place. Thus, for a conventional CAD or drawing program to support emergence and multiple readings, a new pattern recognition module must be added. So far, menu driven CAD systems have not taken this approach.

2.2 RECOGNITION BASED DRAW PROGRAMS

Freehand drawing programs, as distinct from menu-driven draw and paint programs, depend on gesture or character recognition to identify figures that

the designer draws. In a menu driven draw program the intended elements of a drawing are determined a-priori by the interface, but a recognition based program must identify the elements as the designer draws them. The simplest recognition based programs resolve every drawing mark immediately, but more sophisticated architectures allow for ambiguity and indeterminacy (Mankoff, Hudson and Abowd 2000). One reason is that even when the designer has a clear intent, recognition may fail; supporting ambiguity and indeterminacy allows for the program's graceful degradation of performance, and recovery from recognition errors. A second reason is that the designer may lack clear intent, and it is then valuable for the program to allow for this and not to force a more determined and precise representation than the designer had in mind.

Support for emergent forms is easier to implement in a recognition based freehand drawing program. It is a more natural component than in conventional menu-driven CAD because in a recognition based draw program machine interpretation is already an inherent part of the system.

The following sections describe the support for emergent form in a freehand drawing program, the Back of an Envelope system (Gross and Do 2000), an enhanced version of the earlier Electronic Cocktail Napkin (Gross 1996; Gross and Do 1996). The focus here is on the program's architecture and the mechanisms for supporting emergent form, rather than on the application of these techniques in design. The program supports two of the three types of emergent form identified above: intersecting figures and alternative configurations. The remainder of this section (2) briefly outlines this program's architecture. Section 3 then describes how the Back of an Envelope constructs candidates for emergent form by intersecting pairs of individual figures, and Section 4 outlines how the program selects emergent forms from this set of candidates. Section 5 discusses the program's ability to parse alternative configurations to support this type of emergence. Section 6 describes how candidates for emergent shapes could also be produced by figure-ground reversal. Finally Section 7 concludes with a discussion of user interface issues posed by emergence in a freehand drawing system.

Figure 2 shows the basic structure of the Back of an Envelope draw program. The designer draws freehand on a digitizing tablet. Two recognizers process this input. A glyph recognizer (R1) compares drawing elements against a library of previously trained templates, and tags these drawing elements if it can identify them. A second recognizer, the visual language parser (R2) continually examines the drawing for patterns of elements

arranged in previously trained specific spatial relationships. (The two libraries of previously trained templates and configurations are not shown in the diagram). These two recognizers, normally used to process the designer's input, are also used to process emergent form in the drawing.

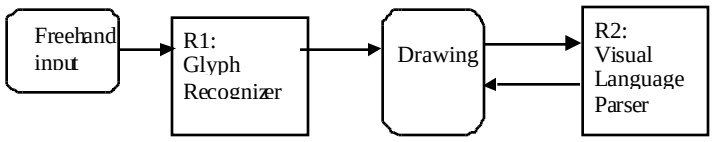

Figure 2. Two recognizers process freehand drawing input in the Back of an Envelope program.

It may be worth pointing out that if the purpose of the draw program is simply to construct a visual representation of a design, a menu-based structured draw program may suffice. Emergence is useful for supporting more sophisticated editing operations, as mentioned above for example, selecting segments of a line formed by line intersections. The real payoff, though, comes when the drawing is the grounds for further inference, for example by knowledge based systems that reason about the design based on the drawing. The Back of an Envelope is intended not simply as a draw program *per se* but as a 'front end' for knowledge based design systems. For this purpose, emergence is more than an editing convenience. By providing a more multivalent representation, emergence in the drawing interface allows the 'back end' applications to reason more effectively about the design.

3. Intersecting Figures

One of the canonical forms of emergent visual form derives from intersecting figures (figure 1a). In the Back of an Envelope system it is straightforward to generate a set of candidate subfigures formed by the intersection of any pair of figures, because the program stores each figure as a sequence of strokes, each consisting of a sequence of points. This section outlines this computation.

A set of intersecting shapes is constructed from two figures A and B as follows (see figure 4):

Identify the intersecting points of the two figures i(1, ..., n).

For each figure A and B, add the endpoints i0 and i n+1.

Each segment of figures A and B can then be described as a triple:

s =(**f j k**)

where f is A or B and j and k are contiguous integers (j = k ± 1) and the segment s describes the segment along figure f from intersection point ij to ik.

Subfigures SAB of the intersection of shapes A and B are the sequences of segments s0, s1, s2, ... such that the points p2 and p1 of successive segments sn and sn+1 link, that is sn(p2) = sn+1(p1) The set of maximal subfigures SAB are the set of paths from between the start and end points of figures A and B that lead through intersection points I, traversing each segment no more than once.

For example, the maximal subfigures for two figures A and B with two intersections are given by the paths:

```
((A 0 1) (B 1 2) (A 2 3))
((A 0 1) (B 1 2) (B 2 3))
((A 0 1) (A 1 2) (B 2 3))

((B 0 1) (A 1 2) (B 2 3))
((B 0 1) (A 1 2) (A 2 3))
((B 0 1) (B 1 2) (A 2 3))
```

and the following two subfigures, which visit the two intersection points twice:

```
((A 0 1) (B 1 2) (A 2 1) (B 1 0))
((A 0 1) (A 1 2) (B 2 1) (B 1 0))
```

This list is generated by a tree-generating program that starts at (A 0 1) and at each intersection point generates three alternative paths (forward along A, forward along B, and backward along B), proceeding until it reaches an endpoint or an intersection with no untraversed paths.

For example Figure 4 (left) shows two simple intersecting figures A and B with labeled points of intersection. Figure 4 (right) shows the subfigure made up of segments:

```
( (A 0 1) (A 1 2) (B 2 1) (B 1 0) )
```

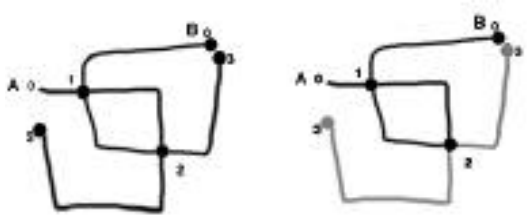

Figure 3. Left: two intersecting figures A and B, with labeled endpoints and intersections. Right: A subfigure.

A subfigure description is not unique; several names (corresponding to different traversal sequences of the figure) can describe the same figure. For example, the traversal sequences

```
(A 0 1) (B 1 2) (A 2 1) (B 1 0)
```

and

```
(B 0 1) (A 1 2) (B 2 1) (A 1 0)
```

both describe the subfigure on the right of Figure 3: one beginning at the A end and the other beginning at the B end.

The set of all subfigures may be constructed by trimming segments from the front and back of the maximal subfigures. The set of subfigure descriptions depends only on the number of points of intersection of the two figures A and B, and contains no geometric information. Thus one set of subfigure descriptions covers all pairs of figures with two intersections, another for those with three intersections, etc. It is however easy to construct a figure given the subfigure description and the two figures A and B, just by sequencing the appropriate segments. The program can generate not only the subfigure descriptors, but also map each descriptor to the geometry provided by the figures A and B.

4. Interesting Figures

Clearly not every member of the set of subfigures is salient or interesting. The program cannot determine which subfigures will be interesting simply from the labeled segment descriptors: interest depends on geometry as well as on context. Many of the resulting subfigures will be uninteresting. After computing the set of subfigures the program's next task is to filter out the

uninteresting shapes and retain those that are interesting. How then to select from among the candidates?

Simply, a shape is interesting if the program 'knows' it. In the example below, the Back of an Envelope program has determined that the four subfigures shown in figure 4 are 'interesting'. (Examples of 'uninteresting' figures are shown in figure 5.) It runs all the generated subfigures through its glyph recognizer and retains only those that match shapes that it finds in its library of templates—shapes that the designer has previously trained. In the example above, if we have trained the el-shape and a rectangle, then the program will recognize them. The program discards all generated subfigures that do not match a previously trained shape. It considers interesting only those shapes it can recognize.

For example, the rightmost subfigure (Figure 5d), which is deemed "uninteresting," might in some contexts be interesting: It resembles a diagram of the Big Dipper constellation in the Northern Hemisphere.

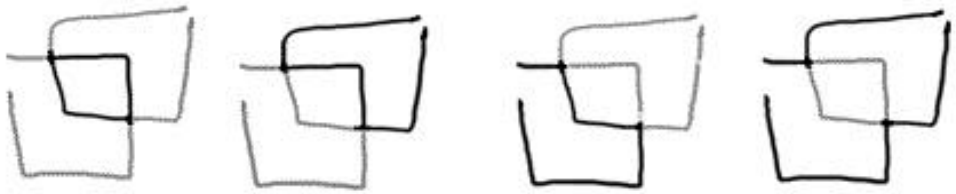

Figure 4(a-d): Only four of the subfigures of the intersecting shapes are 'interesting'. The program has been trained to recognize them.

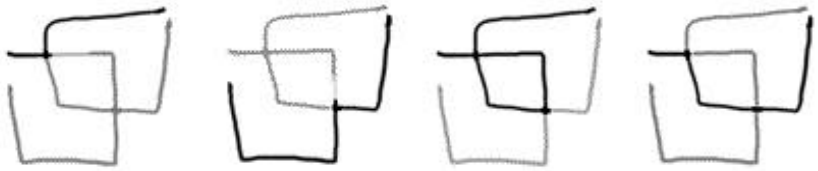

Figure 5(a-d): The set of all subfigures includes these 'uninteresting' ones. The program has not been trained to recognize them.

There are some minor additions to the algorithm. Figure 4d cannot be extracted using only the method outlined above. The program must

recognize that the two endpoints of B are close enough to latch; a modified version of the algorithm takes this into account. It is also possible to generate emergent shapes with a single glyph that crosses over itself. Another version of the algorithm handles this case. Finally, the example here shows two single-stroke glyphs, although the program also supports glyphs drawn with multiple strokes (e.g., a rectangle drawn with four single strokes may be considered a single glyph if the strokes are made in rapid succession). The emergent form generator can consider the individual strokes in a multi-stroke glyph as segments to be combined, just as the segments made by intersecting two glyphs.

5. Alternative configurations

A second type of emergence in visual representation, one quite different to finding forms in intersecting figures, is recognizing alternative configurations (figure 1b). The principle is simple: a designer may draw a set of elements as part of a configuration, and subsequently reinterpret (mentally regroup) the elements to form an alternate configuration. This is illustrated abstractly in the structural hierarchy in figure 6(a-d) below. The four circles in figure 6a represent individual elements; the two squares (figure 6b) represent two configurations that the two elements are deemed to belong to; and the square (figure 6c) represents an alternative grouping of two of the drawing elements. Figure 6d illustrates the composite view, in which two of the drawing elements are now understood as belonging simultaneously to two different configurations.

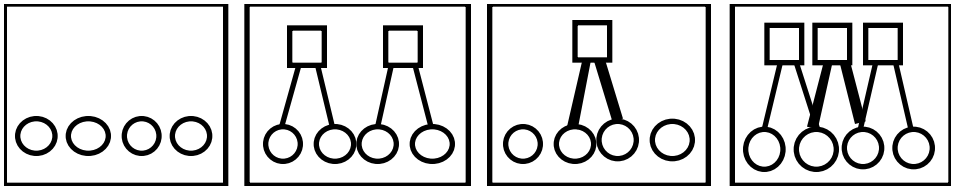

Figure 6. Part-whole hierarchy of a simple four-element drawing, showing alternate groupings of elements. (a) four drawing elements; (b) parsed into two configurations; (c) alternate configuration for two elements; (d) multiple representations for the two configurations.

From the designer's perspective, figure 6a represents the four elements that are actually drawn; figure 6b represents what he or she had in mind when

drawing the elements; figure 6c represents a new grouping that becomes apparent upon examining the configuration; and figure 6d represents the designer's expanded understanding of the drawing after observing the emergent grouping.

From the machine's perspective, figures 6b and 6c simply represent alternative parse trees of the terminal drawing elements in figure 6a. That is, the machine has no representation of what the designer intended to construct, so both 6b and 6c are plausible groupings of the drawing elements. There is no reason to prefer one over the other, so the program admits both parse trees as alternative interpretations of the drawing.

As with the intersecting subfigures, support for this type of emergence is also driven by previously trained structures. That is, only configurations can emerge that the machine has previously been shown. Emergence depends on the library of configurations that the program's parser has been trained to recognize.

The Back of an Envelope's visual language parser examines the drawing elements seeking combinations of elements (e1, e2, ... en) arranged in certain spatial relations (r1, r2, ... rm), An initial set of simple glyph elements (line, box, circle etc.) are built into the program and additional ones may be trained by the end-user. The spatial relations (adjacent, contains, connects) are coded into the program and supported by its recognizer.

The combinations of elements and spatial relationships are organized as a set of production rules of the form ei <— (rj, ek, el) . This production rule indicates that when found in spatial relationship rj, elements ek, and el form a configuration element ei. (We might instead call this ci, to distinguish the new configuration from a simple element. However, the parser does not distinguish between simple elements and configurations when it searches the drawing for applicable rules). When the parser identifies such a pattern of elements, it constructs a configuration (group) ei with the elements ek, and el as its parts, asserting the part-of relation P(ei, ek, el) (read: element ei is a configuration with parts ek and el). There is no restriction on the number of groups that a part can belong to; therefore the parser can identify multiple groupings of the same set of parts.

6. Figure-Ground reversal

Figure ground reversal (figure 1c) illustrates yet another type of form emergence that can be computationally supported. In the current

implementation it is not but it is a straightforward extension to the program. It is mentioned here as a reminder that other types of shape emergence can be integrated into this framework, which is not dependent on a particular geometry algorithm.

Figure ground reversal would require writing an additional generator for emergent shape candidates, similar in role to the intersecting shapes generator described above in Section 3. The **fgGen** generator for figure-ground reversal would first identify all segments in existing figures, using corners and intersection points to break each glyph into minimal segments (figure 7).

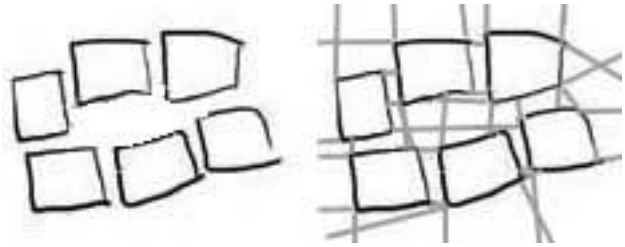

Figure 7. After finding the minimal segments in each shape, fgGen will extend each segment to identify possible figure-ground reversal shapes.

Then **fgGen** would seek to join colinear segments and nearby intersections of extended segments, much as in the earlier schemes of Tan and Gero described above. It would then walk the existing shape boundary segments and the constructed line segment extensions to construct candidate shapes. Finally, as with the intersecting subshapes it would apply the recognizer to the candidates and select only those shapes that match a previously trained figure.

Like the intersecting shapes generator, **fgGen** would run whenever the designer was not busy drawing.

7. Summary and Discussion

The goal of this research is to develop computer aided design tools that can 'see' the same emergent shapes in a drawing that a designer does. Most CAD programs today can't do this. They store only the structured representation that the designer makes when constructing the drawing.

Emergence is pattern recognition. Programs that employ freehand input employ, perforce, a pattern recognizer to identify drawing elements as the designer enters them. A draw program that already employs pattern recognition to identify elements and configurations can employ the same mechanisms to support emergent form. The Back of an Envelope program is an example of this type of program. It employs a two step recognition driven process to support shape emergence. First an emergent shape generator produces a set of candidate forms. This set is then filtered through the program's recognizer to produce a set of emergent forms. As the Back of an Envelope program illustrates, recognizing emergent form is a small and natural extension to this kind of program. However, most draw programs are not recognition based, and for these programs to support emergence, a recognition module would need to be added. Thus, to add support for emergent form to a conventional menu based CAD program requires serious extension to the software's architecture.

The approach developed here differs from other emergent form efforts mentioned earlier in various ways. As a freehand drawing system, it does not depend particularly on maximal line constructions; methods that depend on stright line geometry work for hard line drawings but it is not obvious how they support emergence from curves and irregular figures. Support for emergent form would seem to be more appropriate for early design, in which freehand sketching is traditional, than for the later phases of design that employ hard line drawing. Unlike PerSketch, though, which provides the designer complete freedom to specify any perceived emergent form, the approach described here limits emergent forms to those the program recognizes. PerSketch does not—on its own—recognize any emergent shapes, so it is dependent on the designer to point these out. Whereas the Back of an Envelope can make inferences based on forms latent in the designer's drawing, without waiting for the designer to point them out, PerSketch cannot do this.

Given the ability to recognize emergent forms of various types (intersecting figures, alternative groupings, figure-ground reversals), a draw program must still make this functionality useful to designers. For example, Suwa and Tversky (Suwa and Tversky 1996) suggest that computer aided design programs might, upon recognizing patterns in the designer's drawing, bring these patterns to the designer's attention. This might be done by showing a figure-ground reversal, or by animating the drawing in ways to make alternative readings more salient. Based on informal observation, we found that designers have an excellent ability to perceive emergent forms in

a drawing, and hardly need a drawing program to point these out. (In one mode, the program's glyph recognizer identifies glyphs as the designer draws them, printing "circle", "box", etc.; designers who see this wonder why the program is telling them what they already know). We assume that designers will readily recognize emergent forms. (Revealing emergent shapes visually might help novice designers learn to see them). It is important for the drawing program to recognize the emergent shapes as well, and to make them available for manipulation.

In 'diagnostic' mode, the Back of an Envelope does exactly what Suwa and Tversky suggest—it displays the emergence process visually—as the program recognizes emergent forms it highlights them momentarily, bringing them to the attention of the designer. Normally, though, the forms simply remain latent in the drawing until they are needed. The designer can select an emergent form by tracing over it. As in the PerSketch program (Saund and Moran 1994), if the overtraced lines are close enough to the latent shape, it is selected.

The Back of an Envelope is designed not 'merely' as a draw program, but as an interface to other application programs. We argue that freehand drawing is an appropriate way to enter information about designs to knowledge based design aids. We have used the Back of an Envelope to build sketch based interfaces to various knowledge based programs, e.g., simulation programs and databases. In these interfaces, the drawings that a designer makes serve as input to the application program. For example, the forms in a drawing serve as a sketch-based query to a database of designs, or as the input to a visual access simulation. Normally (that is, without emergent forms) the drawings presented to the application are simply those the designer makes. With emergent form processing the emergent forms are also available to these applications. Emergent forms in the drawing can trigger searches in databases or serve as input to simulation programs. This can happen automatically: the designer need not explicitly point them out to the program.

By no means does the Back of an Envelope system fully exploit the opportunities that emergent form recognition provides, nor does it solve all the representational issues that emergent form raises for computer aided design. Yet the system provides in a demonstration prototype form, an example of how emergent form recognition can be embedded into a freehand drawing program. It thus offers a computational laboratory for exploring these issues and opportunities.

Acknowledgements

This research was supported in part by the National Science Foundation under Grant No. IIS-96-19856 and 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. My colleague Ellen Yi-Luen Do worked on other parts of the Back of an Envelope system and participated in many discussions of shape emergence issues. A conversation with Ernest Edmonds first suggested this extension of the program; Masaki Suwa contributed valuable suggestions about forms of emergence and suggested the figure-ground reversal functionality.

References

Edmonds, E., L. Candy, R. Jones, et al. (1994). "Support for Collaborative Design: Agents and Emergence." Communications of the ACM **37**(7): 41-47.

Edmonds, E., T. Moran and E. Do (1998). "Interactive Systems for Supporting the Emergence of Concepts and Idea, a CHI 97 workshop." SIGCHI bulletin, a quarterly publication of the ACM Special Interest Group on Computer-Human Interaction **30**(1).

Gero, J. S. and M. Yan (1993). Discovering Emergent Shapes Using a Data-Driven Symbolic Model. CAAD Futures '93. S. van Wyk and U. Flemming. Pittsburgh / USA, Springer**:** 3-17.

Goldschmidt, G. (1991). "The Dialectics of Sketching." *Creativity Research Journal* **v.4**(no.2): 123-143.

Goldschmidt, G. (1999). The Backtalk of Self-Generated Sketches. Spatial and Visual Reasoning in Design. Sydney, Australia, Key Centre of Design Computing.

Gross, M. D. (1996). "The Electronic Cocktail Napkin - working with diagrams." Design Studies **17**(1): 53-70.

Gross, M. D. and E. Y.-L. Do (1996). Ambiguous Intentions. Proceedings, ACM Symposium on User Interface Software and Technology (UIST '96). Seattle, WA, ACM SIGGRAPH and SIGCHI**:** 183-192.

Gross, M. D. and E. Y.-L. Do (2000). "Drawing on the Back of an Envelope: a framework for interacting with application programs by freehand drawing." Computers and Graphics **24**(6): 835-849.

Liu, Y.-T. (1993). Recognizing Emergent Subshapes in Design Problem Solving: A Connectionist Investigation. Education and Practice: The Critical Interface [ACADIA Conference Proceedings]**:** 131-139.

Liu, Y.-T. (1995). Problem Decomposition on Restructuring Shapes in Terms of Emergent Subshapes. Sixth International Conference on Computer-Aided Architectural Design Futures. M. Tan and R. Teh. Singapore**:** 439-451.

Mankoff, J., S. E. Hudson and G. D. Abowd (2000). Providing integrated toolkit-level support for ambiguity in recognition-based interfaces. Proceedings of the CHI 2000 conference on Human factors in computing systems**:** 368 - 375.

Mitchell, W. J. (1990). Introduction: A New Agenda for Computer Aided Design. The Electronic Design Studio. M. McCullough, W. Mitchell and P. Purcell. Cambridge, MA, MIT Press: 1-16.

Nagakura, T. (1990). Shape Recognition and Transformation: A Script-Based Approach. The Electronic Design Studio:Architectural Knowledge and Media in the Computer Era. M. McCullough, W. Mitchell and P. Purcell. Cambridge MA, MIT Press: 149-170.

Saund, E. and T. P. Moran (1994). A Perceptually-Supported Sketch Editor. ACM Symposium on User Interface Software and Technology, Marina del Rey, CA, ACM Press.

Schön, D. (1992). "Designing as Reflective Conversation with the Materials of a Design Situation." Knowledge Based Systems **5**(3).

Solso, R. L. (1994). Cognition and the Visual Arts. Cambridge, MA, MIT Press.

Soufi, B. and S. A. R. Scrivener (1992). Perceptual grouping algorithms and object identification. Proc. Third International Conference on Visual Search. Nottingham.

Stiny, G. (1990). What Designers Do that Computers Should. The Electronic Design Studio. M. McCullough, W. Mitchell and P. Purcell. Cambridge MA, MIT Press: 17-30.

Stiny, G. (1993). Emergence and Continuity in Shape Grammars,. CAAD Futures '93. S. van Wyk and U. Flemming, Springer: 37-54.

Suwa, M. and B. Tversky (1996). What Architects See in Their Sketches: Implications for Design Tools. ACM Human Factors in Design '96 (CHI '96) Conference Companion. Vancouver, Addison Wesley: 191-192.

Tan, M. (1990). Saying What It Is by What It Is Like - Describing Shapes Using Line Relationships. The Electronic Design Studio:Architectural Knowledge and Media in the Computer Era. M. McCullough, W. Mitchell and P. Purcell. Cambridge MA, MIT Press: 201-213.