PLANAR MAPS: AN INTERACTION PARADIGM FOR GRAPHIC DESIGN

Patrick Baudelaire *Michel Gangnet*

Digital Equipment Corporation
Paris Research Laboratory
85, Avenue Victor Hugo
92563 Rueil-Malmaison France

In a world of changing taste one thing remains as a foundation for decorative design — the geometry of space division.
Talbot F. Hamlin (1932)

ABSTRACT

Compared to traditional media, computer illustration software offers superior editing power at the cost of reduced freedom in the picture construction process. To reduce this discrepancy, we propose an extension to the classical paradigm of 2D layered drawing, the *map paradigm*, that is conducive to a more natural drawing technique. We present the key concepts on which the new paradigm is based: a) graphical objects, called *planar maps*, that describe shapes with multiple colors and contours; b) a drawing technique, called *map sketching*, that allows the iterative construction of arbitrarily complex objects. We also discuss user interface design issues in map based illustration software.

KEY WORDS

Illustration Software, Drawing Paradigm, Planar Map, Map Sketching, User Interface Design.

INTRODUCTION

Consider drawing on a computer screen, using a typical interactive graphics program, two orthogonal pairs of parallel lines as shown in Fig. 1. This is a trivial task on any geometrical drawing software. However, this apparent ease actually hides an intriguing difficulty. To the eye of the designer, this picture can be viewed in many different ways, two possibilities being as a set of four lines or as a rectangular area. If it is viewed as a rectangular area, then a natural option would be to fill the rectangle with color. With traditional graphic arts media (pencil, ink, paint, etc.), the designer would have complete freedom to do so.

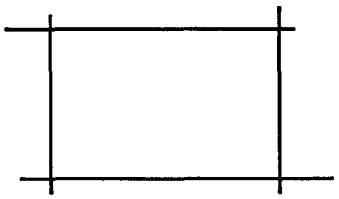


Figure 1: Four lines or a rectangular area ?

Unfortunately, with typical drawing software this dual interpretation is not possible. The picture contains no manipulable objects other than the four original lines. It is impossible for the software to color the rectangle since *no such rectangle exists*. This impossibility is even more striking when the four lines are abutting as in Fig. 2. We feel that such a restriction, counter to the traditional practice of the designer, is a hindrance to productivity and creativity. In this paper we propose a new drawing paradigm that will permit a dual interpretation of Fig. 1 and Fig. 2.

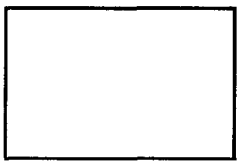


Figure 2: A rectangle or four lines ?

The solution we present here is an extension to the classical drawing paradigm that is the foundation of current graphics software. Our solution allows the designer to create and manipulate 2D graphical objects of arbitrary complexity, which we call *planar maps*. We also propose a new drawing method, called *map sketching*, that seems more natural and more efficient for constructing certain classes of drawings. We begin by describing the classical paradigm. Then we present the map concepts and we discuss several issues that come up when designing a user interface that accomodates both the classical and the planar map drawing paradigms. All figures were produced with map sketching software. The most general graphics primitive in the current implementation is a path made of Bézier arcs [10].

Digital Equipment Corporation is pursuing patent protection on part of the technology described in this paper.

THE CLASSICAL DRAWING PARADIGM

To set the stage, let us first summarize the objects and editing functions that characterize the *classical paradigm* for 2D drawing. We are excluding from our analysis bitmap or pixel painting software. Instead, we are interested in the interactive construction of drawings that are represented as geometric objects and that can be viewed and printed in a resolution independent way.

Objects are typically *open* or *closed paths*, made of line segments or curves, with rendering attributes such as color, texture, width, etc. A closed path (also called a *contour*) may be filled with a single color, texture, pattern, paint, bitmap, etc. A path may be self-intersecting. These objects share a unifying feature: they are drawn in one single stroke, without lifting the pen. Note that in the PostScript language, paths can include "move to" statements, which means that PostScript objects may have holes or comprise distinct pieces. However, to our knowledge, no interactive 2D drawing software takes advantage of this possibility.

Drawing tools provide two ways to create a path:

- From a *template* that produces a stretchable predefined path such as a rectangle, circle, oval, etc.
- As an ordered list of points: for instance a polyline, polygon, or polyspline, defined by its end points, vertices, spline knots, Bézier control points, etc. In this paper, we will cover all these cases by generic terms: *polyarc* and *control points*.

The classifications of a path as either open or closed, and as either a template or polyarc, are independent of each other.

Layering serves two distinct purposes:

- Rendering : The image is produced by overlaying objects from back to front, a process that harmonizes computer graphics technology (e.g., algorithms for painting bitmaps, the concept of a display list, the PostScript imaging model) with traditional graphic arts techniques such as cut and paste or animation cells.
- Logical ordering: Semantic attributes or names are associated with sets of layers, allowing either partial display or selective user action. This feature is typically found in CAD applications.

Here we only consider rendering layers, a concept usually described as 2 1/2D imaging. In the classical paradigm, there is one and only one object per layer.

Global editing functions are used to rearrange and layout the picture: erase, move, copy, transform, group, etc. They apply in an identical manner to both template and polyarc objects.

Shape editing functions are specific to each type of path. Bounding box handles are used for resizing and reshaping templates. Polyarcs are modified by editing control points.

A number of known research and commercial applications that have been developed for electronic publishing follow this drawing paradigm, including: Draw [6], Griffin [16], MacDraw [3], Illustrator [1], Freehand [2].

EXTENDING THE CLASSICAL PARADIGM

Although it is feasible to create intricate pictures with current CAD or illustration software it is not always easy. A good measure of a program's effectiveness is not the maximum image complexity that it can produce but the simplicity of the drawing process it induces. In this respect, the classical paradigm has clear limitations: certain kinds of illustrations are difficult to construct even though they are graphically simple. Too often, to achieve a desired graphical result the designer must resort to an indirect drawing strategy. Let us consider two typical examples.

The "jigsaw puzzle" problem is a common one: in Fig. 3 a shape is to be divided by some arbitrary path. Typically, the designer must carefully construct two pieces whose edges match perfectly. Even with a good replication function, this process is more work than it needs to be.

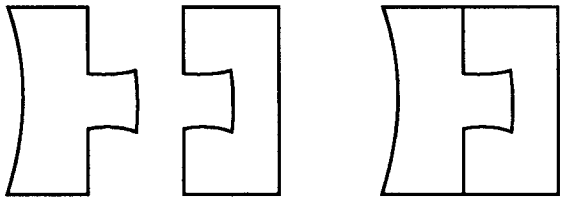


Figure 3: Building a divided shape.

Another case is when a given shape cannot be produced with available drawing tools (often limited to rectangles, ovals, and polylines). To produce a non standard shape, the designer must resort to collage and masking, hiding parts of objects with other overlapping objects (often in the color of the background) to create an illusion that looks like the desired result (Fig. 4—masking shapes are outlined).

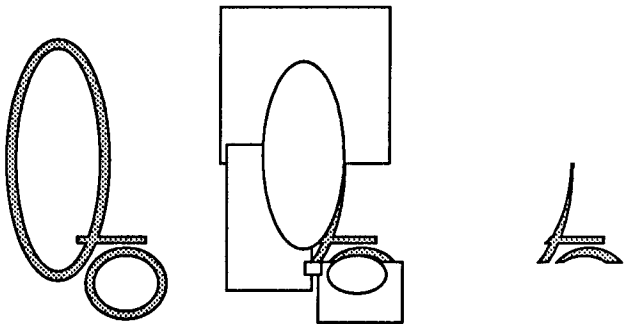


Figure 4: Collage and masking.

Computer drawing certainly provides superior editing power for modifying objects after they have been drawn. But compared to a pencil and eraser, today's computer techniques are still too restrictive.

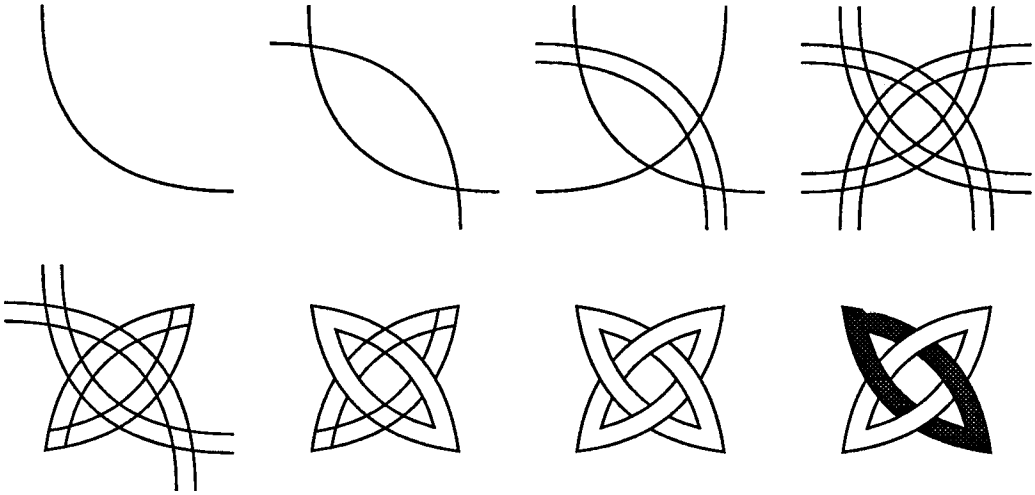


Figure 5: Map sketching.

Several avenues of research towards more effective drawing systems have been explored. In Juno, Nelson offers a constraint language to express spatial relationships between objects [14]. Pier, Bier, and Stone, propose in Gargoyle sophisticated construction tools reminiscent of the ruler and compass [8], [15]. In Tweedle, double view editing gives a choice of direct manipulation or procedural description of pictures [4].

In the two examples of Fig. 3 and 4, the common missing feature is the ability to build shapes from other shapes. It would be nice to regain this freedom, without losing the known advantages of computer drawing. In the work presented here, we propose a new paradigm that induces a drawing process closer to traditional practice. More importantly, we achieve this by extending the classical paradigm, which remains fully adequate in many cases.

The solution can be summarized as follows:

- Extend objects to be multicolor, multicontour shapes, which we call *maps*, an abbreviation for planar maps of graph theory [17]. Thus, a path is a simple map.
- Offer a new construction process for building and editing maps by iteration of three basic steps: drawing, erasing and coloring. We call this process *map sketching*.

We keep everything else the same: 2 1/2D layering, global editing and shape editing functions. Hence, the map paradigm is a superset of the classical drawing paradigm. In the next section, we describe the map sketching process and illustrate with examples that show the properties of maps.

MAP SKETCHING

This technique consists of applying a succession of elementary drawing, erasing, and coloring operations *in a single layer*, as if working with pen, eraser and color on a transparent sheet of paper. Consider doing in sequence several standard drawing operations, either template or polyarc, so that the strokes cross each other (Fig. 5 top). At each step, the resulting graphical object is a multi-contour or multi-path shape that becomes more and more complicated: it is a *map*.

At this stage, let us present some more terminology borrowed from graph theory. End points and corner points of the strokes, along with points at which they intersect, are called the *vertices* of the map. A portion of a stroke delimited by two adjacent vertices is called an *edge*. A region of the plane that is bounded by a set of connected edges is called a *face*.

To modify a map, two more operations are used: *erasing* an edge and *coloring* a face (Fig. 5 bottom). These are simply variants of the erasing and coloring operations of the classical paradigm. Drawing, erasing, and coloring can be iterated *in any order* to build a map. Graphical objects of arbitrary complexity can be constructed in this way. The fundamental property of a map is that it is transformed into another map by simple graphical operations. Map sketching is thus a general process for building shapes from other shapes. In addition, automated compound operations, easy to implement on a map data structure, are sometimes useful. They are best described by the following illustrations (Fig. 6, 7, and 8).

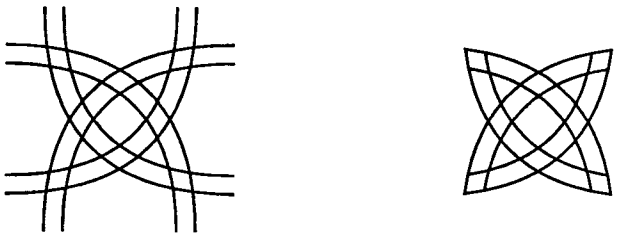


Figure 6: Cleaning a map removes dangling edges.

Figure 7: Outlining a map yields its outer contour.

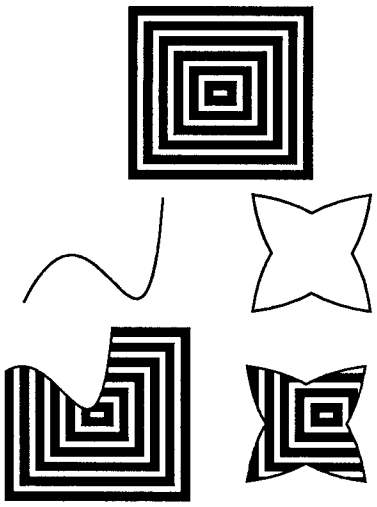


Figure 8: Cutting or punching a map with a contour.

Map sketching is in fact somewhat similar to the traditional pencil and eraser techniques. The following examples illustrate the convenience of the process.

In many designs, contours are traced over elaborate construction lines. In traditional media, ink and color are applied over precisely built pencil strokes. This drawing technique is commonly used in designing logos, monograms and other symbolic artwork [5]. Even with the help of grids and alignment tools, exact positioning and fitting of separate pieces is difficult or cumbersome. In map sketching, one goes directly from intersecting construction lines to final contours by erasing the parts of the construction lines that do not belong to the contours (Fig. 9 and 10, design by Eurosud [5]).

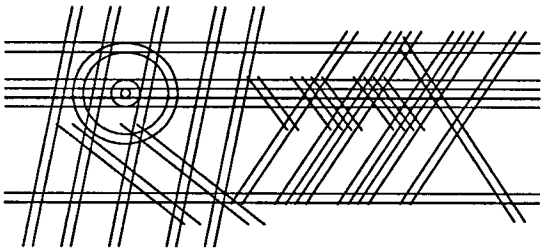


Figure 9: Construction line map for a monogram.

Figure 10: Final map of monogram.

Other designs rely explicitely on the visual interplay of intersecting contours, see [12] and [11]. This contruction style, akin to op-art, is frequently used in logos (Fig. 11). Map sketching seems the most straightforward drawing technique in this case.

Figure 11: Map for CHI'88 logo.

Another category is illustrated on Fig. 12. These are simple pseudo-3D drawings where perspective and hidden line removal are done by the user. Edge erasing on a map is a very simply way of removing hidden lines on a hand-drawn wire-frame perspective.

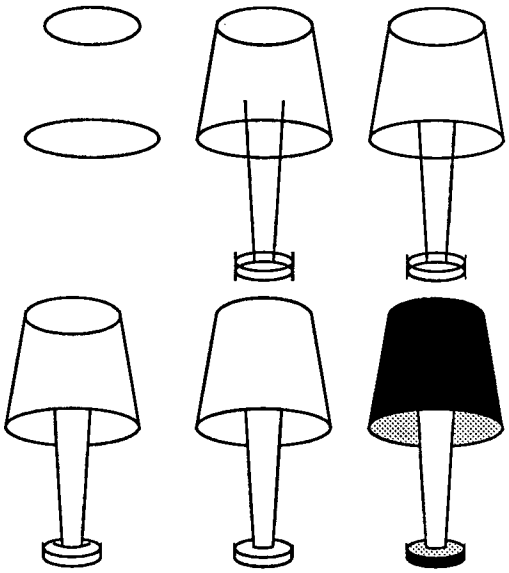


Figure 12: Map sketching of a simple 3D view.

USER INTERFACE DESIGN ISSUES

Maintaining compatibility with the classical paradigm and taking the new vantage point offered by planar maps turn out to be somewhat opposing goals. The main difficulty is that the user interface must deal in a homogeneous manner with simple objects (the single stroke paths of the classical paradigm) and with complex objects (maps) that are themselves built from simpler objects. The need to address both levels of complexity arises for most commands and the resulting asymmetry is a significant interface design obstacle. We have found that the two key design problems were *layering* and *local editing* of maps.

Layering

Reducing ambiguity is a good principle in user interface design but ambiguity cannot be avoided in graphics: there is never a unique way to construct or interpret a drawing. For instance, the rectangular shape in Fig. 2 can be built in at least nine ways from a rectangular template or polyline primitives.

This first aspect of graphical ambiguity results from object typing: four abutting lines forming a rectangle will behave differently than a template rectangle. The map paradigm tends to suggest a unique representation for objects: as a map, the drawing of Fig. 2 allows action both on the edges (i.e., the lines) and on the face (i.e., the rectangle).

Another source of potential ambiguity is layering: how many objects, in how many layers, compose a drawing ?

One way to reduce ambiguity would be a single-layer single-map paradigm. Although drawings of arbitrary complexity could be produced with this paradigm, it is clearly too restrictive: multi-object layering appears to be a fundamental graphics design principle, and we have adopted this requirement from the start for compatibility with the classical paradigm.

In the classical drawing process, any new object (either created from scratch or by duplication) becomes a new layer. In the map sketching process, the designer creates a new shape by working repeatedly on the same layer. Clearly, we want a user interface that allows both.

To achieve this goal, we view a drawing as composed of layers, each of which contains either a classical single stroke path, produced by standard drawing tools, or a map created by sketching. This solution raises two distinct questions: layer control and connectivity.

Layer control addresses three functions:

- Switching between the standard drawing process (single stroke paths) and map sketching.
- Getting a blank sketching layer, to start a new map.
- Getting an existing layer, to modify a map or a single stroke path.

Thus, map sketching and the classical drawing process have to be alternate drawing modes. We believe that this can be done by adding only one new explicit layer control command, for instance to conclude a map sketching sequence.

In addition, we also have found it useful to provide a function that extracts a sequence of edges from a map and turns it into a simple path, in a new layer, that can be edited with the standard tools. This is discussed further below.

The issue of *connectivity* has a bearing on the concrete metaphor used for maps. In the classical paradigm, disconnected shapes are by definition separate objects contained in separate layers, even when there is no way to check this fact visually. To use a graphic arts analogy, objects can be seen as pieces of colored paper cut into different geometrical shapes.

The map paradigm calls for a quite different physical analogy. Theoretically, the concept of a planar map allows disconnected shapes in a single layer to compose a unique map. This may cause ambiguity since there is nothing that distinguishes visually the rendering of one map with two components from that of two separate maps.

Although this multiple component principle apparently conflicts with the classical paradigm, we believe that it is useful to preserve it. Invoking the analogy of geographical maps, it seems natural to consider that an island belongs to the same map as the mainland nearby. To conclude on what appears to be an adequate physical metaphor, a map should be viewed as an infinite transparent sheet on which colored faces are painted, allowing any number of separate components as well as holes, which are just transparent faces.

Editing a Map

Although map sketching is the fundamental creation and local editing paradigm for maps, there are other local editing methods that also deserve attention.

Moving a vertex and its incident edges is simple, but it may cause edges to cross. More generally, this situation will come up if we want to apply to maps the same control point functions that are used with polyarcs. Self-crossing, although allowed in a polyarc, contradicts the very definition of a map: two edges may cross during rubber-banding, but they will necessarily produce a new vertex in the final map. This rule puts constraints on vertex editing, as demonstrated in Fig. 13. A good solution is to *extract* a set of edges from a map, edit it outside the map as a polyarc path and reinsert it into the map.

Figure 13: The effect of edge-crossing on vertex dragging.

To go even further, it is interesting to allow in place editing of original constituent objects, using their intrinsic editing mode (i.e., with the handles of a template or the control points of a polyarc). This operation should also work if some of the edges have been erased and even if the object spans more than one component of the map (Fig. 14).

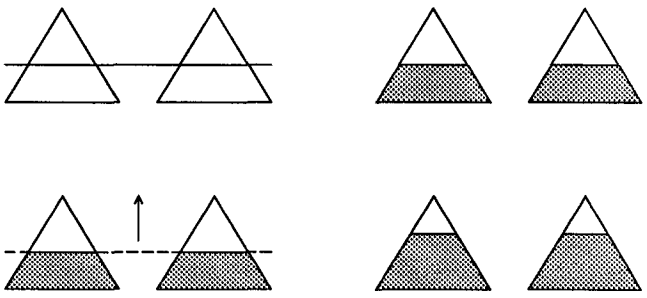
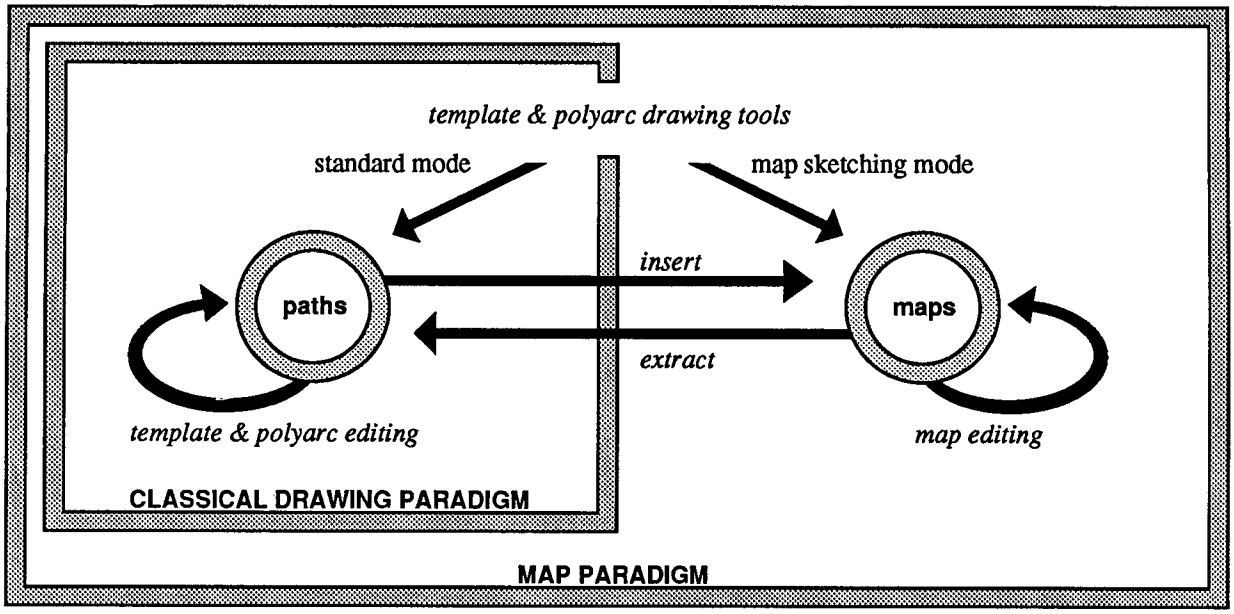


Figure 14: Dragging a constituent path within a map.



SUMMARY OF THE MAP PARADIGM

ACKNOWLEDGEMENTS

The initial work on planar maps was started in 1983 by Michel Gangnet and Dominique Michelucci at Ecole des Mines de Saint-Etienne, as a tool for architectural design [13]. This work was pursued at Tangram Inc. The first illustrator prototype demonstrating the map paradigm, Cadix, was implemented on a Unix workstation [9]. The user interface of a strict extension to MacDraw following the map paradigm was specified in 1986 [7]. The technology was acquired by Digital Equipment Corporation in 1987 and the work is being pursued at the Digital Paris Research Laboratory.

The authors thank Jean-Manuel Van Thong for his assistance in producing the figures, and Brad Chen, Henri Gouraud, and David Salesin for helpful suggestions on the writing of this paper.

REFERENCES

[1] *Adobe Illustrator User's Manual.* Adobe Systems Inc., Palo Alto, 1987.

[2] *Freehand User's Manual.* Aldus Corp., Seattle, 1987.

[3] *MacDraw Manual.* Apple Computer Inc., Cupertino, 1984.

[4] P. J. Asente. *Editing Graphical Objects Using Procedural Representations.* Research Report 87-6, Digital Equipment Western Research Laboratory, Palo Alto, 1987.

[5] D. Baroni. *Art Graphique Design.* Editions du Chêne, Paris, 1987.

[6] P. Baudelaire. *Draw Manual, in Alto User's Handbook.* Technical Report, Xerox Palo Alto Research Center, Palo Alto, 1979.

[7] P. Baudelaire. *MacMap: Functional Specifications and User Interface.* Technical Report, Tangram Inc., Issy-les-Moulineaux, 1986.

[8] E. A. Bier and M. C. Stone. Snap-Dragging. *ACM Computer Graphics*, Vol. 20(4):233–240, 1986.

[9] M. Gangnet and J. C. Hervé. D2: Un éditeur graphique interactif. In *Actes des Journées SM90*, Eyrolles, Paris, 1985.

[10] M. Gangnet, J. C. Hervé, T. Pudet, and J. M. Van Thong. *Incrémental Computation of Planar Maps.* 1989. Submitted for publication.

[11] S. Horemis. *Optical and Geometrical Patterns and Designs.* Dover, New York, 1970.

[12] C. P. Hornung. *Handbook of Designs and Devices.* Dover, New York, 1946.

[13] D. Michelucci and M. Gangnet. Saisie de plans à partir de tracés à main-levée. In *Actes de MICAD 84*, Hermès, Paris, 1984.

[14] G. Nelson. Juno, a constraint-based graphics system. *ACM Computer Graphics*, Vol. 19(3):235–243, 1985.

[15] K. Pier, E. Bier, and M. C. Stone. An Introduction to Gargoyle: An Interactive Illustration Tool. In *Proceedings of EP'88*, CUP, Cambridge, 1988.

[16] M. C. Stone. *How to use Griffin.* Internal Memo, Xerox Palo Alto Research Center, Palo Alto, 1980.

[17] W. T. Tutte. *Graph Theory.* Addison-Wesley, Reading, 1984.