The Immediacy of the Artist's Mark In Shape Computation: From Visualization to Representation

by
Jacquelyn A. Martino

Bachelor of Arts
Mount Holyoke College, 1985

Master of Arts in Instructional Technology and Media
Columbia University, Teachers College, 1990

Master of Fine Arts in Computer Graphics
Pratt Institute, 1995

Submitted to the Department of Architecture in Partial Fulfillment
of the Requirements for the Degree of

Doctorate of Philosophy in Architecture: Design and Computation
at the
Massachusetts Institute of Technology

September 2006

Signature of Author: [illegible]
Department of Architecture
31 May 2006

Certified by: [illegible]
George Stiny
Professor of Design and Computation
Thesis Supervisor

Certified by: [illegible]
Terry Knight
Professor of Design and Computation
Thesis Supervisor

Accepted by:
Yung Ho Chang
Professor of Architecture
Chair, Committee on Graduate Students

DISSERTATION COMMITTEE

Terry Knight
Professor of Design and Computation
Massachusetts Institute of Technology

Joe Marks
Vice President and Research Lab Director
MERL, Mitsubishi Electric Research Lab

George Stiny
Professor of Design and Computation
Massachusetts Institute of Technology

The Immediacy of the Artist's Mark In Shape Computation: From Visualization to Representation

by

Jacquelyn A. Martino

Submitted to the Department of Architecture on 31 May, 2006
In Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy
In Architecture: Design and Computation

ABSTRACT

Approaches to shape computation and algorithmic art–making within the fields of shape grammars and computer graphics still do not consider the immediacy of the artist's mark in drawing and painting. This research examines the canvas, or 2D picture plane, as the expressive and dynamic problem space of the artist who naturally reframes both the problem and the solution with each successive mark. The fluidity of the artist's mark is the most important element in transforming the blank canvas into an image. In this research, I use my own traditionally drawn and painted artwork as the baseline corpus for analysis. From my analysis, I define a nonsymbolic, formal grammar for the synthesis of images in the style of the baseline corpus and develop software prototype "sketches" to support the geometric representation of freehand sketching consistent with mark–making.

Curve generation is critical to the expressive marks of the artist. The result of this research is a curvilinear shape grammar that supports both explicit and implicit shape recognition while affording the artist the ability to draw (shape union) and erase (shape difference) computationally. I compare the results of the synthesis phase with my traditional sketches showing that it is possible to compute imagery consistent with the evolving style of the artist's own hand. Additionally, the analysis phase of the research supports the supposition that formal algorithmic understanding of one's artistic process has directive and positive influences on the evolution and refinement of the style.

Thesis Supervisor: George Stiny
Title: Professor of Design and Computation

Thesis Supervisor: Terry Knight
Title: Professor of Design and Computation

ACKNOWLEDGEMENTS

It is impossible to complete a work the magnitude of the Ph.D. without the assistance of many. The process is long and challenging with the dissertation itself being only one component of the journey. I thank those who have given the benefit of their guidance, humor and goodwill throughout the time of my studies. In particular I extend my most heartfelt gratitude to those whose names follow.

For their encouragement, insights and expertise, I thank my committee: Terry Knight, Joe Marks and George Stiny.

For their friendship, conversations and moral support, I thank Lira Nikolovska and Mine Özkar.

For their understanding and generosity, I thank Annette and Philip Martino, Kristen Anderson, Jennifer Young and Cathy Polera.

And, for her partnership and unwavering belief throughout this and all our adventures, I thank Joanne Murphy.

The Immediacy of the Artist's Mark In Shape Computation: From Visualization to Representation

by

Jacquelyn Martino

TABLE OF CONTENTS

Chapter 1 Background and Context

This dissertation examines some of the key issues in shape computation as they relate to artistic practice from process to product. It employs the extensive body of literature in shape grammars and applies time–tested techniques from the field of computer graphics. While both the disciplines of shape grammars and computer graphics have a wealth of research on visual design process and product, relatively little research exists with the specific focus on a living artist's body of work and its analysis and synthesis from a computational perspective. The main contribution of this work, then, is to produce further insights in computational analysis and synthesis of a specific artistic body of work that is still evolving stylistically. The evolving style in question is my own as this research examines a sub–set of my drawing and painting style as I describe in Chapter 2. Further, this research results in a roadmap for continued work in the integration of shape grammar theory with computer graphics practice.

The Problem

Visual artists and designers who work in the area of digital media require tools and methods that allow for both flexibility and repeatability of process.

Digital computers are perfectly suited for enumeration and search of large data spaces based on rules, or algorithms. However, for the digital artist who wants to work algorithmically, employing a process that is rooted in enumeration and search techniques will eventually pose a major creative difficulty. The digitally algorithmic approach casts the artist as an enumerator who passively examines the output of a vast combinatorial production space and evaluates results as visually interesting or compelling. At the point that the visual results of such an implementation do not correspond to the visual concept, the artist must refine the model and begin the output process anew. But during each iteration, the initial concept limits the system. While the artist can alter the limiting boundary between iterations to accommodate new vision, the design space at any given moment remains static. This condition gives rise to two separate modalities. The first is the conceptual state where the artist is seeing and considering subsequent marks on the canvas. The second is the action state, or implementation, where the actual mark–making occurs.

Since the practice of visual creation is one of shifting priorities, the more significant role for the artist is one of creator who dynamically re–frames the problem space with every new visual realization. The realization can be at the conceptual level—the result of simply looking at the existing design with new insights—or at the implementation level––the result of physically adding or subtracting from the design, for example, via new marks or erasures. Such a system implies that the designer is able to make inputs dynamically and move beyond the constraints of a static model, or boundary system. Specifically, this approach implies a supple system that accommodates process at both the conceptual and implementation levels. Present day design tools provide a brittle environment for the fluid demands of this visual process and the result is frequently an obvious product of a given tool suite.

Flexibility is not the only requirement, however. Digital computation tools should also allow for a formal and repeatable process. The environment must provide both tools and methods that allow for an emerging schema of visual problem solving. To achieve this, the formal description of the mark and its physicality on the canvas are ideally one and the same and without syntactic constraints. In the ideal situation, the methodology does not depend either on canonical primitive forms or hierarchies that impose a structure on form. In this constrained state, the combinations allowable by the canonical forms in their hierarchies limit the artistic vision, as there is no accommodation for ambiguity. In the role of creator, the artist necessarily translates ambiguities into non–ambiguities, if only on a temporary basis.

Flexibility and repeatability, however, do not imply an *adaptive* system. Rather, the requirement is an *elastic* schema that allows the designer to identify and use emergent shape, color and texture. In such a design environment, the process itself is visibly apparent and affords the designer the tools and methods to frame and solve the problem simultaneously. Specifically, the artist's description of the design and her implementation, or composition, is one and the same. This requirement becomes clear when you consider geometry where the visualization—seeing the form on the page—and representation—symbolically defining the form for rendering on the page—are two separate entities as contrasted with art where the visualization and the representation are the same entity. Sutherland [55, 56], the grandfather of computer graphics, is very explicit on this exact point articulating that visualization is different than the geometric computer representation.

The division between visualization and representation typically leads the digital algorithmic artist to a product that ignores the dynamic role of the hand and the eye in visual process. In particular, the immediacy of the mark as a form that restructures the picture plane with every new placement has historically been lost in computation. Through self–analysis on a body of my own work, this research pursues a computational apparatus that considers the immediacy of interaction between the artist and the canvas while also employing algorithmic devices.

The Role of the Artist's Mark

Prior research has not addressed the immediacy of drawing and painting as a device in computational and/or algorithmic art. Mark–making— anytime the marking tool comes in contact with the picture plane for the purposes of adding or subtracting—is a critical component of immediacy in artistic process because it is the device through which the artist transforms the canvas into an image. This transformation occurs in the tight loop between the hand and the eye where every mark influences every other mark in a re–framing of the picture plane.

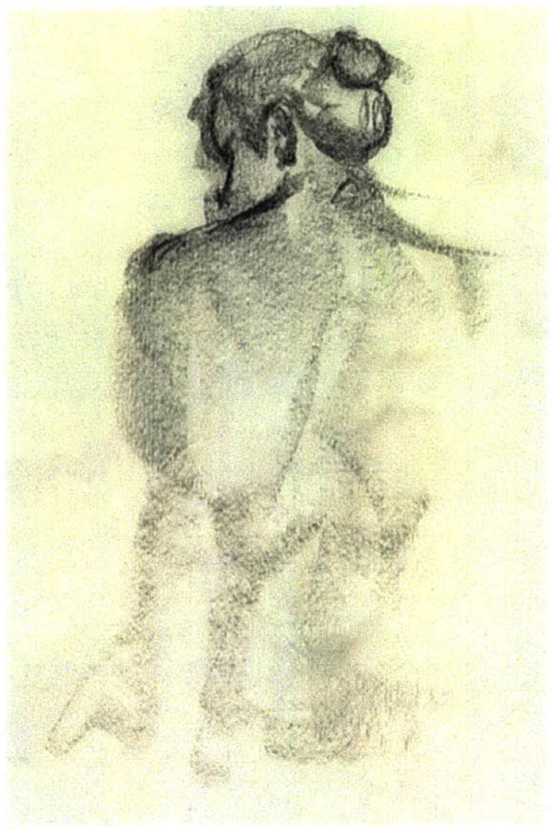
Given this relation, the artist cannot reasonably separate process and product during visual explorations. The interaction requires the artist to conceive the picture plane as a dynamic surface. The artist regards the mark as a form that influences what comes next rather than as a descriptor in analysis. This supports both conceptual input and representational output as one and the same. As such, models of interaction that require a hierarchical structuring of canonical forms can generate interesting imagery, but they will not permit immediacy.

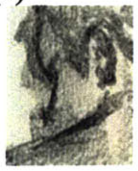
The mark can be a means or an end. To put this in terms of the shape–grammar literature, the mark can exist on the page as an individual element [50] free of syntactic strictures either ready to exist on its own, or in combination with other marks to construct a greater whole. The types of marks that exist may be as varied as the artists who make them. Characteristics of the mark include its direction, weight, duration, pressure and angle. Some examples that are important to my work are present in Figures 1–1 and 1–2.

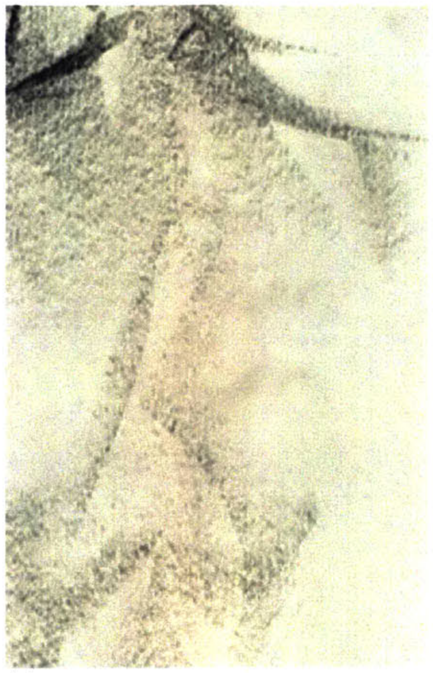
The Algorithm as a Method for Understanding and Producing Art

The algorithmic artist bases rules on a formalized visual understanding. The coupling of rules with an appropriate production tool is commonly known as analysis and synthesis. Algorithmic approaches facilitate formal understanding of visual productions as they de–mystify the art making process. De–mystification results in a formalized process that is repeatable and affords the artist the ability to explore a direction that is not dictated by off–the–shelf software [Mitchell in [1]].

The degree of formal understanding varies, however, depending on the methods artists use. For example, shape grammars are highly explanatory spatial algorithms that inherently provide mechanisms for visual insight [26] while other symbolic algorithms leave aspects of the production opaque to the artist.

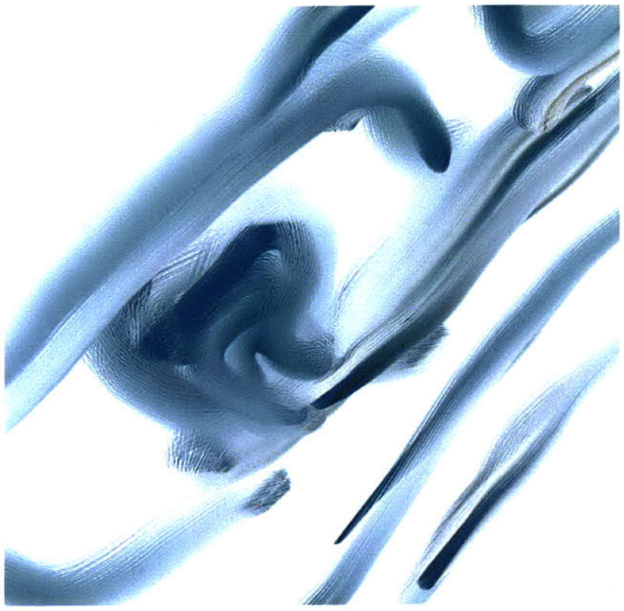

(a)





(b) (c)

Figure 1–1. Pencil drawing from life. (a) Note the use of different pressure strokes and the angle of usage of the tool to bring attention to different areas of the model. (b) Detail of profile. Structural marks emphasize minimal facial features. (c) Detail of shoulder and upper back. Ambiguous marks suggest form. Artist: J. Martino


(a)

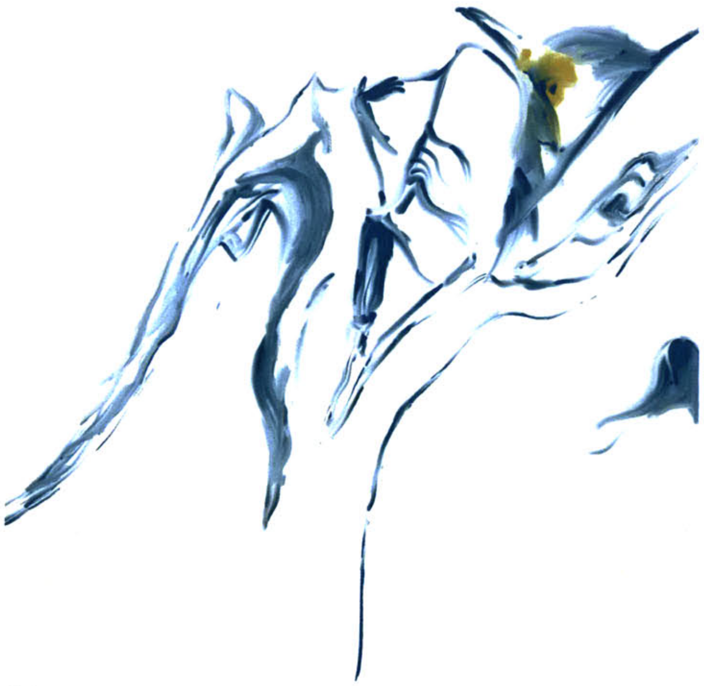

(b)

Figure 1–2. Digital painting of gestural marks. (a) Painting detail. Note how direction, weight, duration, pressure and angle affect the quality of the marks. (b) This painting is based on an initial small vector graphic scribble that was enlarged many times and colorized. The colorized scribble became the basis for this larger work. 20" x 20". Artist: J. Martino

Some such examples are those that rely on artificial intelligence [8, 35] genetic algorithms [43], cellular automata [Brown in [3]] or evolutionary techniques [59]. The majority of this work requires an a priori modeling of an environment, non–visual mathematical relationships, non–visual primitives and/or (pseudo) randomization.

Bentley in [3] summarizes, in part, the difficulty with these symbolic methods.

> Often the style of form generated using a particular representation is more identifiable than the style of the artist used to guide the evolution. [...] The cause of this 'style problem' is perhaps due to the initial preconceptions and assumptions of the designer of the representation. By limiting the computer to a specific type of structure, or a specific set of primitive shapes and constructive rules, it will inevitably always generate forms with many common and identifiable elements.

Bentley implies that attempts to conform to an underlying structure and badly selected primitives are at the root of the problem. Yet, symbolic techniques require that the artist make these choices upfront and before seeing any visual production. But another potential benefit of the algorithm is that of seeing what may not have already been seen and having a mechanism, i.e., the algorithm, to explore the new insight in a formal way. In the same vein, algorithmic implementations also allow the artist to be highly productive.

The longest–standing example of algorithmic drawing and coloring may be Harold Cohen's program, Aaron. In the past thirty years, Cohen and Aaron have produced a large body of images ranging from cave–style paintings to figurative work [35].

Cohen approaches the problem of drawing as temporally linear and spatially based on a grid. Working from an a priori strategy of moving from the image foreground to the background, each new input adds cumulatively to the output. Once the line is placed, there is no turning back to consider ambiguities or other directions for the image. Aaron has no access to visual information and draws forms from a database of 3D anatomical and plant–form information. Even though Aaron accesses geometric information, the program's strategy is formed from three "cognitive" primitives:

a) Is the form open or closed?

b) How many forms are on the page?

c) How much space is remaining in the page that is divided into a matrix of cells?

While Aaron can convincingly draw and color an unending and impressive stream of images, the immediacy of the mark in the drawing and painting process is noticeably absent. And, although Cohen has generated a large body of work, there is no expanding beyond the combinations allowed for by the fact that he works from a fixed database of spatial definitions.

Cohen's approach is more about the quantitative use of geometry than it is about the qualitative use of shape. Just because shape is qualitative, however, does not imply that the artist cannot use it as a computational device. The shape–grammar literature shows that shape computation is not only possible, but it is in fact a desirable method for computing what we can see.

Specific uses of shape grammars in the visual 2D arts include [52], [47, 48], [53] who defined the first shape–grammar design language for original painting. Their work starts at the stage of rule development without first a formal analysis of an existing style. Since that time, [30] have used shape grammars to analyze and synthesize the art of Kandinsky, [22], [23] have analyzed and synthesized the art of Diebenkorn and Miró , and [24], [26] have analyzed the changes in styles of the deStijl artists, Vantongerloo and Glarner. Knight's work on style shows that design can build upon design creating a family of designs and more critically, that a flexible rule–based design language can define style formally. Despite this rich history, there have been no artists who have used the formal devices afforded by shape grammars to analyze and synthesize work within their own corpus. However, issues of design practice and process are recent concerns in the field of shape grammars as in [32].

In computer graphics, numerous artist/scientists have used algorithmic approaches to generate images. Most of the early computer graphic works serve as historical interest and were products of early research collaborations between artists and scientists. A good historical overview is in [14]. Many of these early endeavors tend toward mathematical visualization rather than art. Additionally, there is a body of literature underscoring the relation between art and mathematics with a comprehensive overview in [12]. Escher may be the most renowned artist in this category. His tiling techniques reflect images possible using cellular automata. Finally, for a recent overview of computer generative art see [5].

In computer graphics, much of algorithmic art is abstract with one major exception being the figurative work of Cohen. Above, I have discussed salient specifics of Cohen's work.

Of the abstract artists, two who remain prolific are Verotsko and Mohr. Here, it bears briefly commenting on their work.

Verotsko classifies his work as instruction compositions for art form generators. He has customized pen–plotters to hold sumi–brushes and writes programs to plot his images. He describes his work as

> [A concentration] on developing my program of procedures, the 'score', for visualizing ... forms. By joining these procedures with fine arts practice I create aesthetic objects to be contemplated much as we contemplate the wondrous forms of nature. [5]

Mohr, whose work focuses on fracturing of the symmetry of the 6–dimensional hypercube, says

> [My work is] always the result of a calculation. At the same time, however, it is not a mathematical art, but rather an expression of my artistic experience. The rules I invent reflect my thinking and feelings. It is not necessarily the system of logic of my work I want to present, but the visual invention which results from it [5].

As symbolic productions, we can consider these implementations as special cases of shape–grammar formalisms occurring in the atomic–dimension of space. While such approaches make it difficult, if not impossible, to consider the impact of a new mark on the page during the dynamic process of creation, it is interesting to note that the procedural differences of the algorithmic artists surveyed do not preclude their interest in understanding their product and their process through a formal expression of their visual selves.

Overview of the Thesis Content

This dissertation is part of my ongoing work to analyze a sub–set of my own drawing and painting style and to progress to its computational synthesis. During the course of this research, I have aimed for a flexible rule–based design language to generate 2D images with attention to form. The primary research objective in formalizing the design language has been to understand my personal artistic process to the extent that it informs my results and to suggest new directions. This understanding has founded the secondary research objective: the development of software "sketches" to eventually synthesize images in the style of the already existent corpus. I document these topics in the following five chapters.

Chapter 2, The Corpus, shows visual examples from the pre–computational phase. Beginning with a sampler of sketchbook content and progressing to time–based analysis tools, I provide an explanation of the techniques and methods I used to develop the initial corpus with an eye toward defining its synthesizing rule base.

Chapter 3, Analysis and Development of the Design Language, shows the development of the shape rules that define my corpus. I define the rules of the design language and show how I refined them as my progressive understanding of the design space increased. Additionally, I discuss the similarities and differences among images from the foundation corpus and the images manually generated once the design language was in place.

Finally, I touch upon the initial steps in analysis of my use of color for eventual development into formal rules for that aspect of my image making.

Chapter 4, Description of Computer Implementation Methods, discusses why some aspects of the manual methods are straightforward and others are difficult when translating a shape grammar design language to a computer implementation. Motivated by the curvilinear grammar of Chapter 3, this chapter contrasts the use of lines with curves and addresses why a computer implementation forces the use of a symbolic model particularly when looking for curvilinear sub–shapes in a canvas.

Chapter 5, Conclusions and Future Work, states the contributions to the field made by this research and proposes a direction for its continuation.

Chapter 2 The Corpus

I began the corpus that would become the foundation for this work as a series of small digital paintings, Devotions, in 1997. At that time, I became interested in the relationship between color and natural elements and phenomena. I reflected on issues such as visualizing the un–seeable. For example, what does thunder look like? What feelings does the moon motivate?

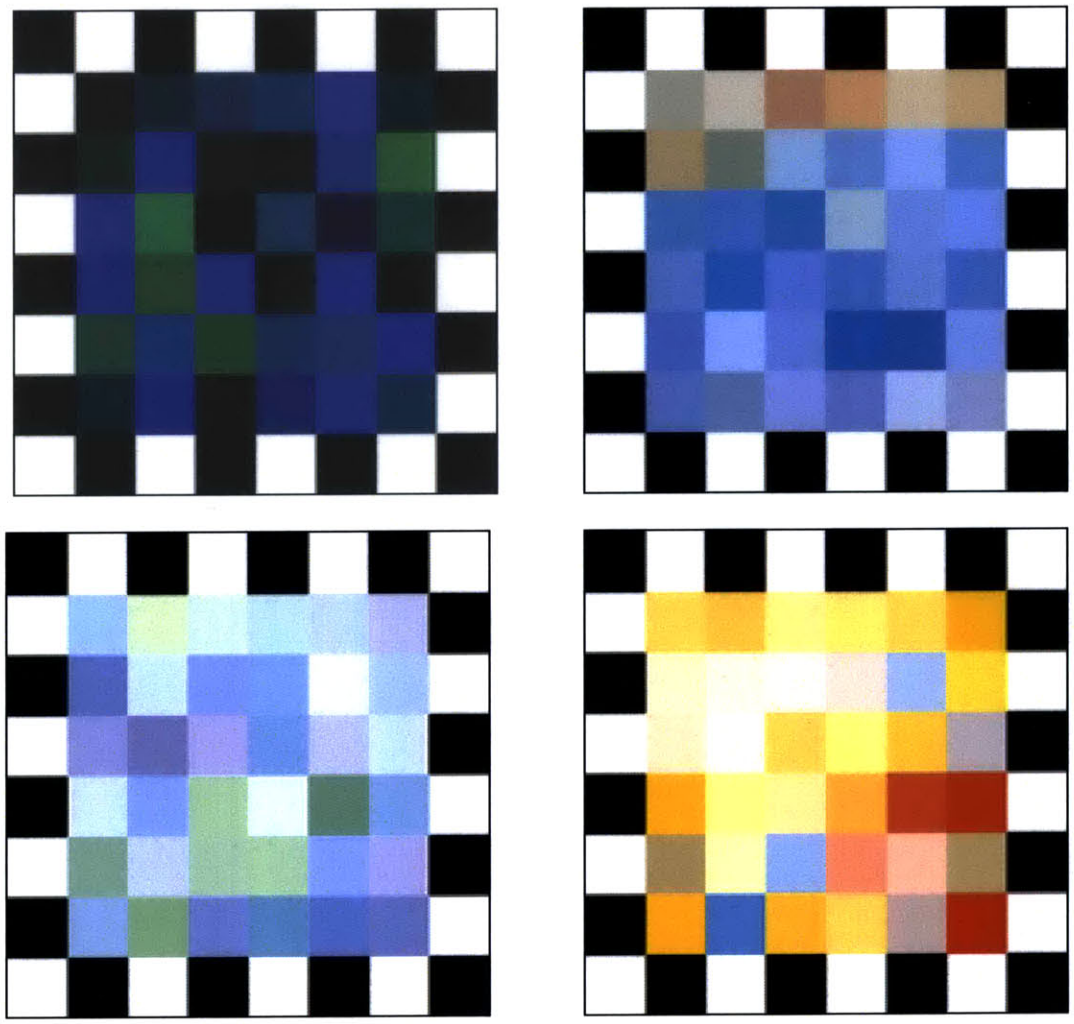


Figure 2–1. Color palettes. Color palette selection for each of Thunder (top left), Moon (top right), Air (bottom left), Lightning (bottom right).). Each palette represents the colors associated with the natural element or phenomenon. In addition to the black and white squares, there are thirty–six (36) colors with the possibility of repetition in each palette. The color placement within the palette is a compositional element.

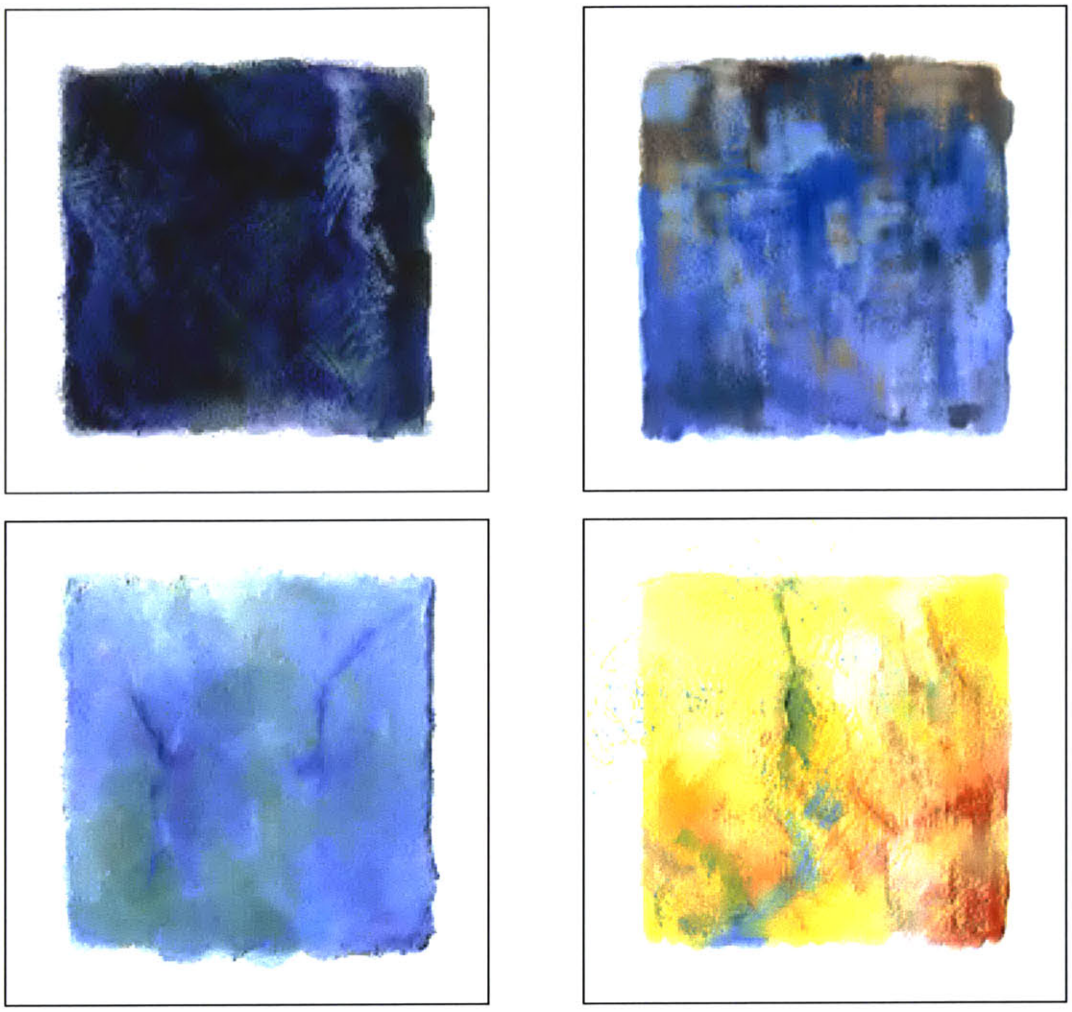


Figure 2–2. Color mixing. These small canvases introduce mixed color and texture for each of Thunder (top left), Moon (top right), Air (bottom left) and Lightning (bottom right). The result becomes the starting point for a larger painting. Digital painting tools such as watercolor, chalk and oil create the mixture and textures based on the palettes of Figure 2–1. Note that in general, the position of color in these paintings aligns with their position from the palette.

As a result, I concerned myself largely with color selection and manipulation in the early phase of the series. Figures 2–1 and 2–2 illustrate the results of color selecting and mixing and serve as the basis for a preliminary discussion of color analysis that follows in Chapter 3.

Early work

My consideration of shape during this period grew from an interest in Mayan glyphs.

Sketchbook entries in Figure 2–3 document this influence.

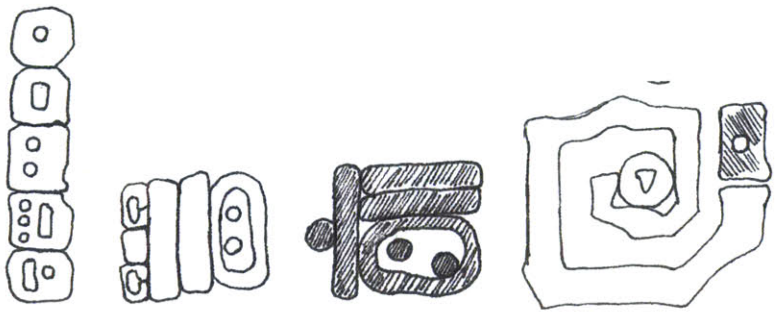


Figure 2–3. Sketchbook drawings. Four instances of sketchbook drawings depict the Mayan glyph influence. All sketches are at 1:1 as they appear in the original sketchbook.

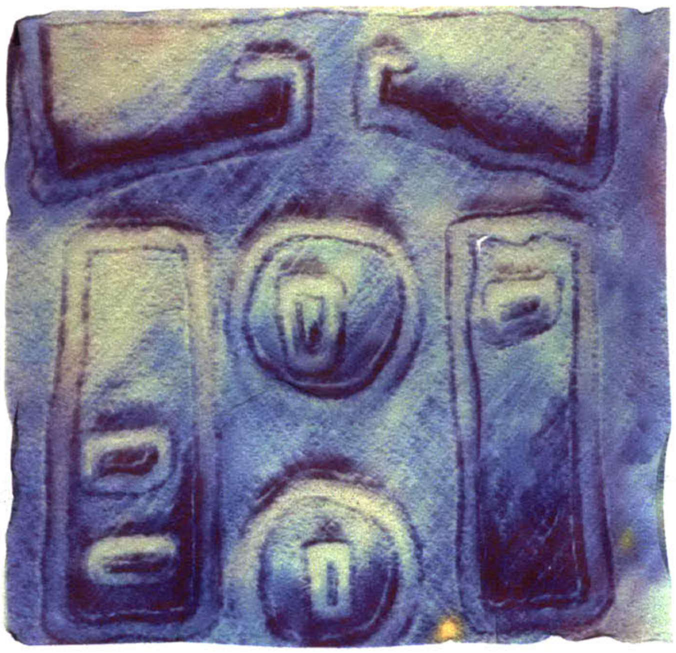


Figure 2–4. Early digital painting depicts Mayan glyph influence. Final output of this painting was as a Polaroid emulsion transfer onto Somerset Velvet. The original is approximately 4" x 4".

Small compositions held some aesthetic interest, as in Figures 2–4 and 2–5. Larger works, however, lacked strong visual organization and stylistic clarity. The compositions were merely a collection of the parts that would eventually become the signatures of the resultant body of work, but made no significant aesthetic statement at the time.

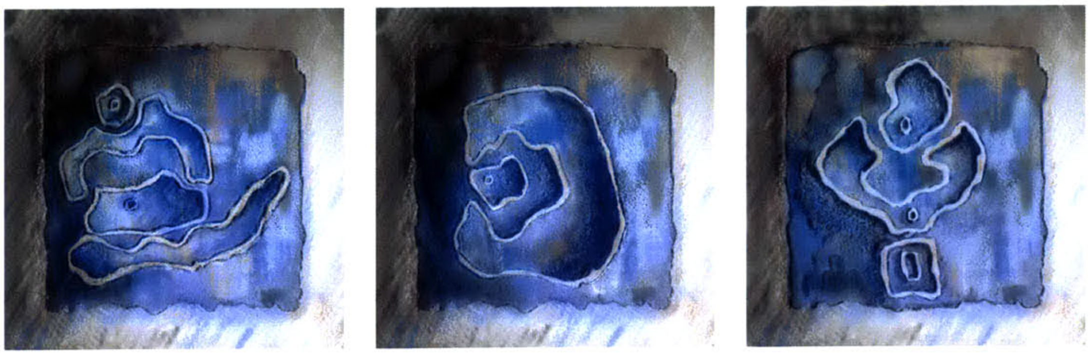


Figure 2–5. Three early digital paintings in the Devotion series. These images use the color palette and mixing baseline from Moon. The shapes depict Mayan glyph influence, but hint at an evolution toward more artist specific forms. The original paintings are approximately 6" x 6".

Content and style

It was not until 2000, then, that I began to dedicate myself in earnest to the aesthetic and stylistic development of these images. My motivation to focus on stylistic concerns was likely due to the fact that many of my early paintings were not compositionally satisfying. From the period 2000 – 2002 I worked almost exclusively on developing the shapes and pursued almost nothing new with respect to color or texture. Essentially, I engaged in a basic sketching process that would inform eventual paintings and small-scale sculpture. Also during this period, I considered the possibility of computational approaches to generating drawings and paintings in the style that I was developing.

I completed the early drawings and paintings outside of any computational framework such as evolutionary graphics, artificial intelligence, machine learning or shape grammars, however. This approach allowed for two important developments. One, I now had a sizable corpus to use as a basis for considering computational devices. Two, I was able to develop the foundation corpus without considering the potential technical constraints or merits of a particular computational approach.

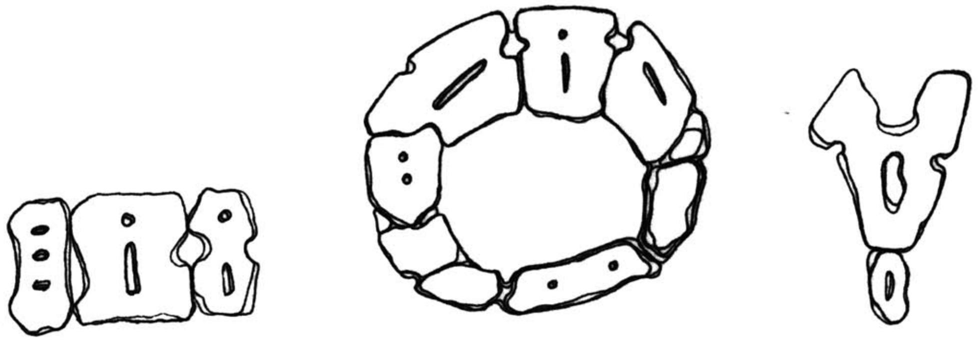


Figure 2–6. Sketchbook drawings. Three instances of sketchbook drawings from mid 2002 that demonstrate a clear stylistic evolution. All sketches are at 1:1 as they appear in the original sketchbook.

Only after achieving a level of stylistic consistency and clarity (Figure 2–6) in mid 2002, did I then turn to the issues of computational devices that I might use to synthesize works similar to those I had drawn in my sketchbooks over the preceding years. It is important to note, however, that stylistic clarity does not mean the end of one's stylistic evolution. An important criterion in deciding on a computational approach was the ability to incorporate the inevitable and dynamic maturation in style without needing to remodel the computational framework. Indeed, it is difficult to find algorithmic approaches to style that do not require a static method.

One example of such a static approach from the computer graphics literature is Freeman [15]. Their method does not consider stylistic maturation, but rather distinct changes in style. They focus on content preservation across styles and opt to develop a training set of lines representing several styles. Then in turn, they apply the various styles to a given drawing. In a process external to the intended drawing, an artist draws the style training set.

By definition, the pre–processing that their method requires considers artistic process as separate from product. Any expressiveness, or immediacy, in drawing style is captured in an action distinct from the subject matter of the drawing. The design space as pre–modeled is not the working methodology of the artist for this type of approach reveals:

a) the impossibility for any mark to act as a visual influencer on any other mark

b) the impossibility to introduce anything in the drawing that is a result of seeing the interrelation among marks

c) that without real–time, dynamic influencing, style cannot develop, it can only be imposed

d) that any style captured by the line sets is likely emulative of some existing style.

The net result is that the style of the drawing changes, but stylistic change, as integral to process does not Figure in their model. This method is radically different from that of Knight's shape grammar approach to analyzing style changes of deStijl artists as demonstrated in [24], [26]. Her work shows that not only can a single style can lead to a range of new ones, but that one can capture this process computationally.

The Tools

During the analysis phase on my body of work, I used two basic sets of tools to document the shape aspect of my image making. The first was quite traditional. I continued my regular practice of sketchbook drawing using either a pen: black ink pigma micron 01, or a mechanical pencil: black polymer lead of 0.5mm width and 4B softness. The second augmented the traditional method. I registered a traditional sketchbook on a Wacom pressure sensitive tablet. Then, I employed a pressure sensitive stylus equipped with a mechanical pencil lead of the same specification that I used for the traditional sketchbook method. As I drew on the sketchbook-augmented tablet, I recorded my drawings as QuickTime movies using Corel Painter. This setup created in essence two original substrates: the sketchbook and the time–based digital canvas.

Although I could have easily completed the time–based drawing process without the addition of a sketchbook and used a typical, non–leaded stylus, the augmented approach was valuable for a variety of reasons. The presence of the sketchbook and the leaded drawing instrument closely emulate the traditional drawing process. Although it is not uncommon for digital artists to place a piece of paper on their tablet to get the physical interplay of paper tooth, the leaded stylus meant the ability to collect the exact same data in two equally useful forms: analog and digital. Drawing with paper and pencil, specifically the feel of the instruments, decidedly influences the quality of the mark and affects the drawing experience.

The activity of traditional sketchbook drawing permitted the execution and collection of hundreds of shape drawings and served as an "unbiased" check for the "correctness" of a synthesized drawing. It was very difficult, however, to capture how the shapes were drawn and to use that kind of information in developing the rule base. The activity of recording the drawing process allowed my repeated viewing of a drawing's progress and permitted a non–automated analysis.

The primary objective in collecting the time–based drawing, then, was to have a method for discerning the temporal order of the marks. Technically speaking, the order of mark making is a non–issue when drawing. Indeed, a drawing—whether finished or in progress—demands no particular structural relationship among the marks at any particular moment in time. In order to compute using the marks as a device, however, shape grammar techniques require the definition of a spatial relation among the marks. It is worth noting that the modeling of a spatial relation is in no way akin to the modeling of a hierarchical relation as no mark is subordinate to any other mark. My basic approach in defining the spatial relations, then, was to look to the actual process that I used to create them. In this way, the resultant design language conforms rather closely to actual process and serves double duty in that it not only synthesizes artistic product but also illuminates artistic process. Figure 2–7 illustrates the creation of a drawing using the augmented sketchbook and shows frames from the resultant time–based data.

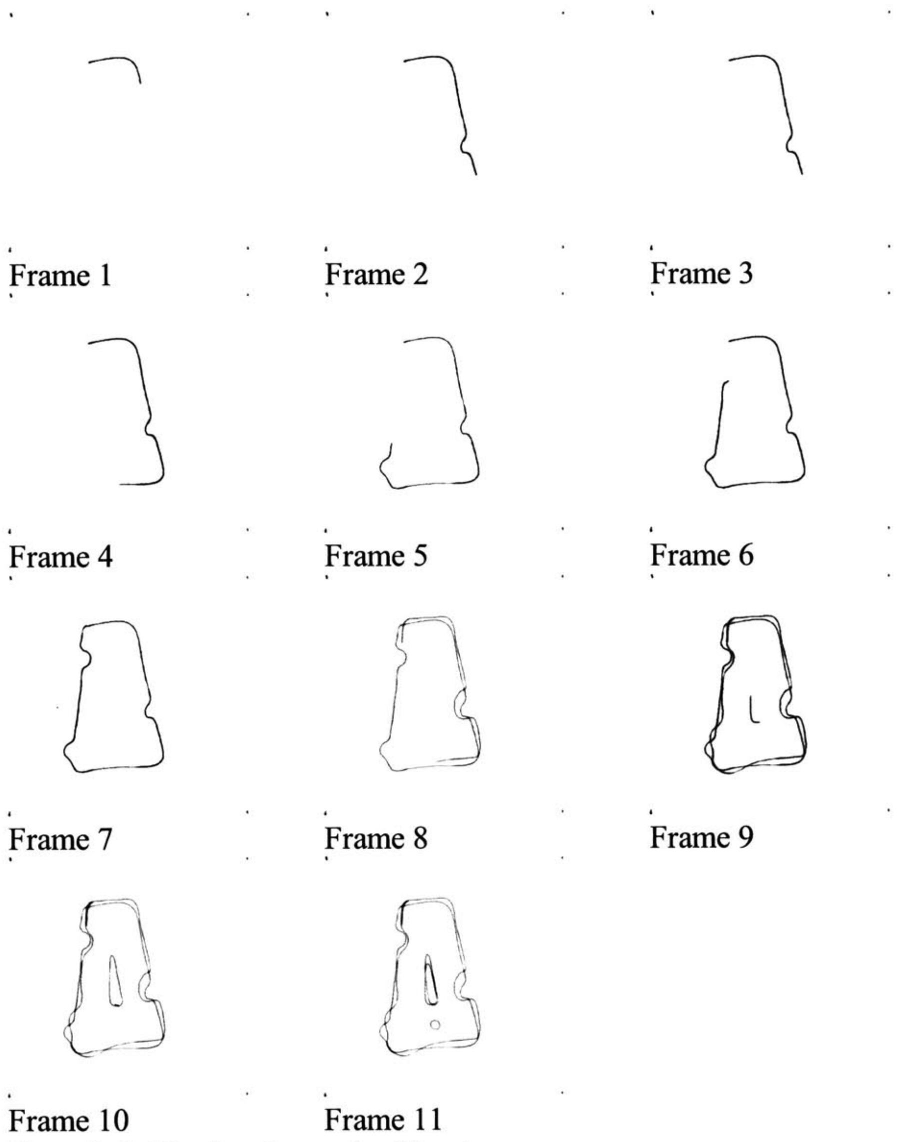


Figure 2–7. Time based example of drawing.

An added benefit of the time–based exercise was that I gained insight into what I was doing as I drew. Even without a formal computational mechanism in place, such self–analysis allowed me to fine tune the drawing style in a manner that may not have been possible otherwise. For example, many sketchbook drawings became overworked and this characteristic manifested itself equally during the time–based drawing exercises. Having the time–based tool allowed me to examine more clearly where the sketches were going off target into the realm of the overworked. In this way, the ability to repeat a drawing by watching served to inform the development of style. The essential aspects began to reveal themselves and the sketchbook drawings became fine–tuned as Figures 2–8 and 2–9 show.

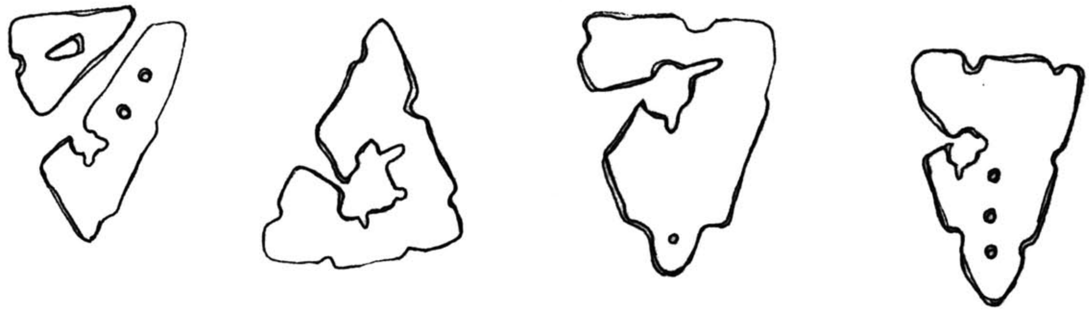


Figure 2–8. Sketchbook drawings circa 2004 based on the triangle as the inspirational shape. Shown 1:1.

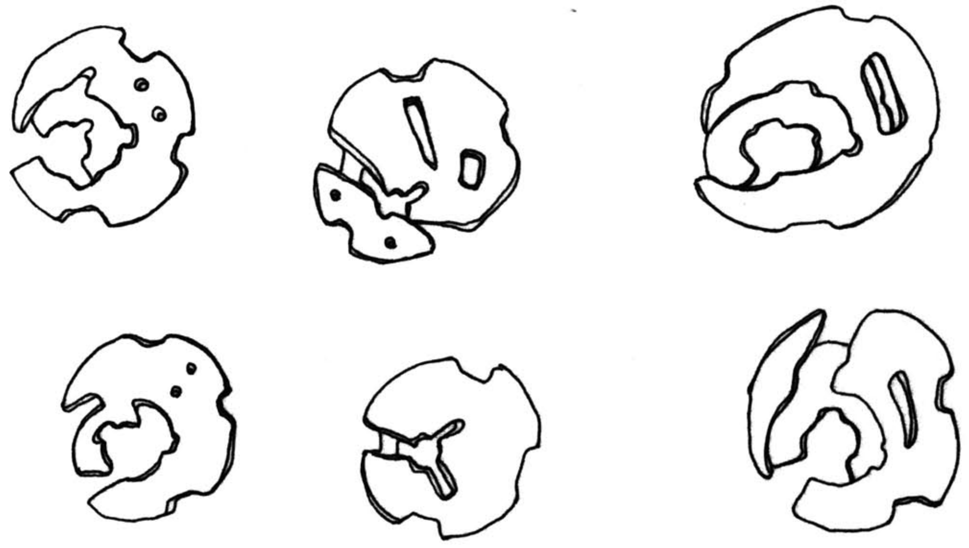


Figure 2–9. Sketchbook drawings circa 2003 based on the spiral as the inspirational shape. Shown 1:1.

Chapter 3 Analysis and Development of the Design Language

Chapter 2 explained the techniques and methods that I used for developing the foundation corpus of my work. One of the activities during that part of the work was to record my drawing process in a time–based medium to afford the benefit of repeated viewing. The resultant collection of short movies served as an important resource in developing the shape rules. This chapter illustrates some of the key movies to influence the initial design language, details the grammar, goes on to show refinements to the grammar and compares and contrasts the results of computed and non–computed product. Finally, I discuss initial work on the analysis of color during painting exercises.

Observation and Analysis

The approximately 70 time–based drawings and continuously growing sketchbook source—approximately 3000 drawings at last count—are the basis for my observations and analysis. While formulating the rules, I gave specific attention to the relation among the marks and their resultant shapes, the order and characteristic of the marks, styling of the shapes and the spatial relations of closed and open shape outlines. A critical observation was that in many cases I was using a non–drawn, invisible shape to reserve and define space on the canvas. Specifically, the visible marks are perturbations, or parametric variations, of simple implied shapes such as squares, circles, triangles, and spirals. Frequently, the final drawing does not bear great resemblance to the initial, non–drawn shape and gives the appearance of much more complexity than the shape rules imply.

Sub–dividing the Canvas

Early in my analysis of the existing corpus, it became clear that my marks were intended to bring attention to small areas of the overall canvas. I defined these areas by a series of outline–like marks that represented both closed and perforated—or segmented—shapes. Figure 3–1 shows an example of a circle as the non–drawn shape. Using the implied circle, I draw a loosely conforming segmented circle. Figure 3–2 continues the same drawing with a second segmented circle–like form at the interior of the first.

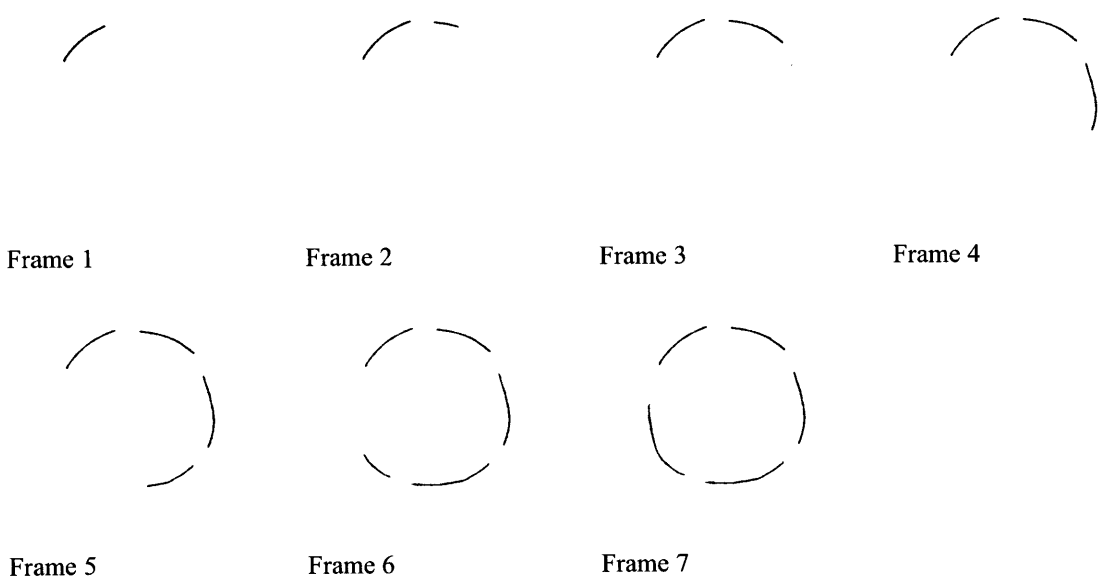


Figure 3–1. Frames 1 – 7 illustrate the use of a circle as the motivating shape and the use of segmented marks to outline an area of the canvas.

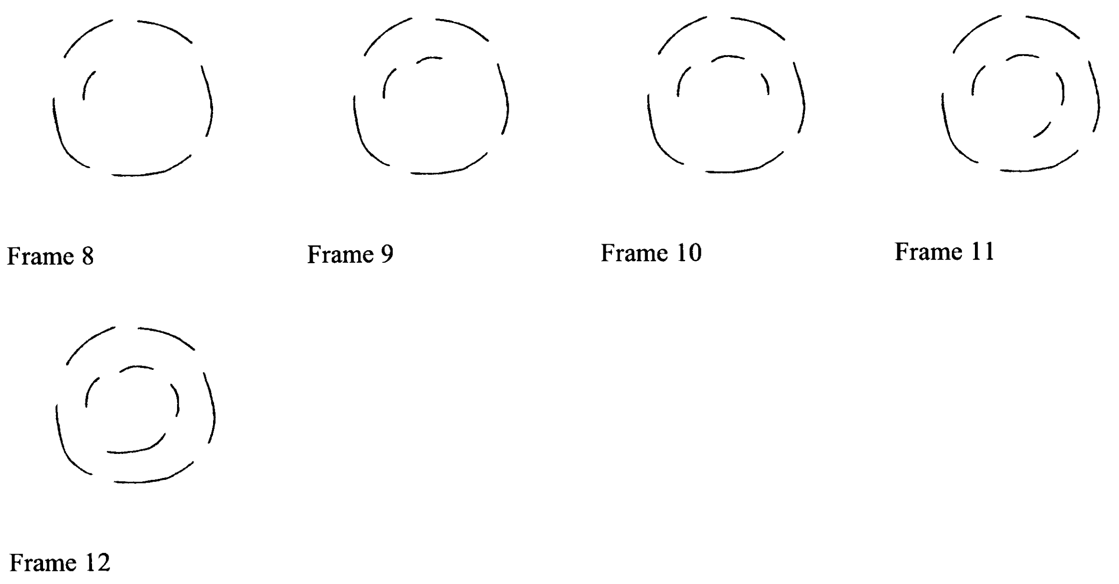


Figure 3–2. Frames 8 – 12 continue the drawing from Figure 3–1 and illustrate a set of interior marks loosely conforming to a second circle.

In addition to single shapes within a shape, other groupings of outlines occur in the foundation corpus. For example, there are numerous cases of a primary outline shape supported by multiple fenestrations, or punctures, as in Figure 3–3a.

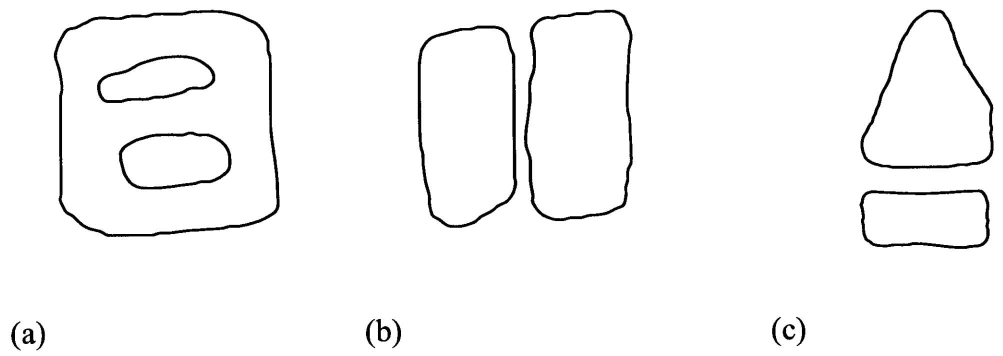


Figure 3–3. Three examples of the spatial relation among shapes in the foundation corpus. Example (a) demonstrates a major shape with two fenestrations while examples (b) and (c) show two closed shapes side–by–side.

Groupings of closed shapes in close side–by–side proximity as in Figure 3–4 and Figure 3–3b and c are also common. Also present, but less common, are cases of open shape outlines that support intersections as in Figure 3–5. Interestingly, however, shapes on top of each other, or layered shapes, are common in my painting examples, but do not occur in drawing. The intersecting shape examples are the closest occurrences of this phenomenon. One supposition for why this is the case is that the drawing phase does not explicitly include erasing. Indeed, erasing is anticipated during drawing by leaving gaps among the marks. As we will see later in this chapter, erasing is a key aspect of my layered painting.

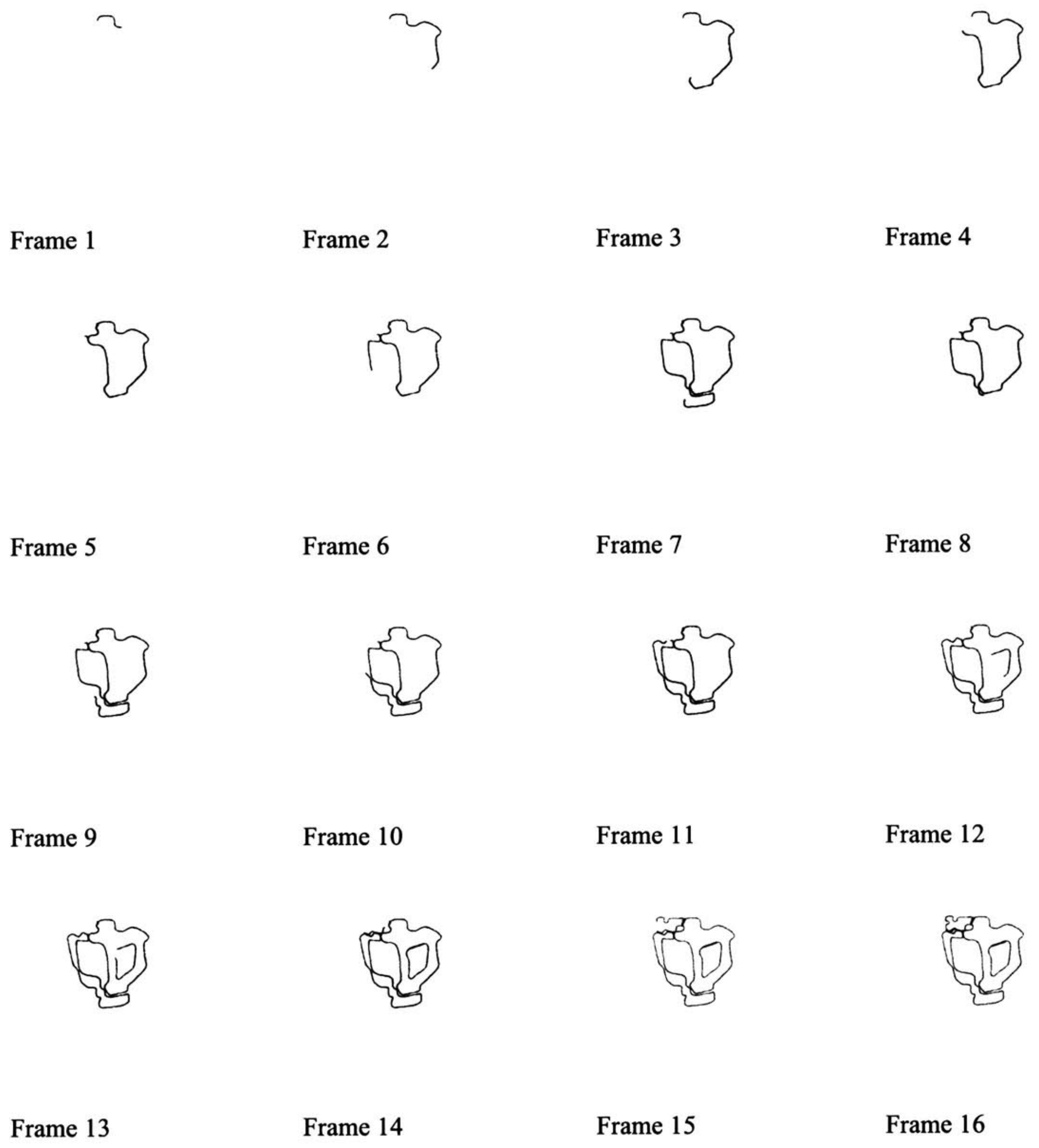


Figure 3–4. An example from the time–based drawings illustrating multiple side–by–side shapes as well as punctured shape.

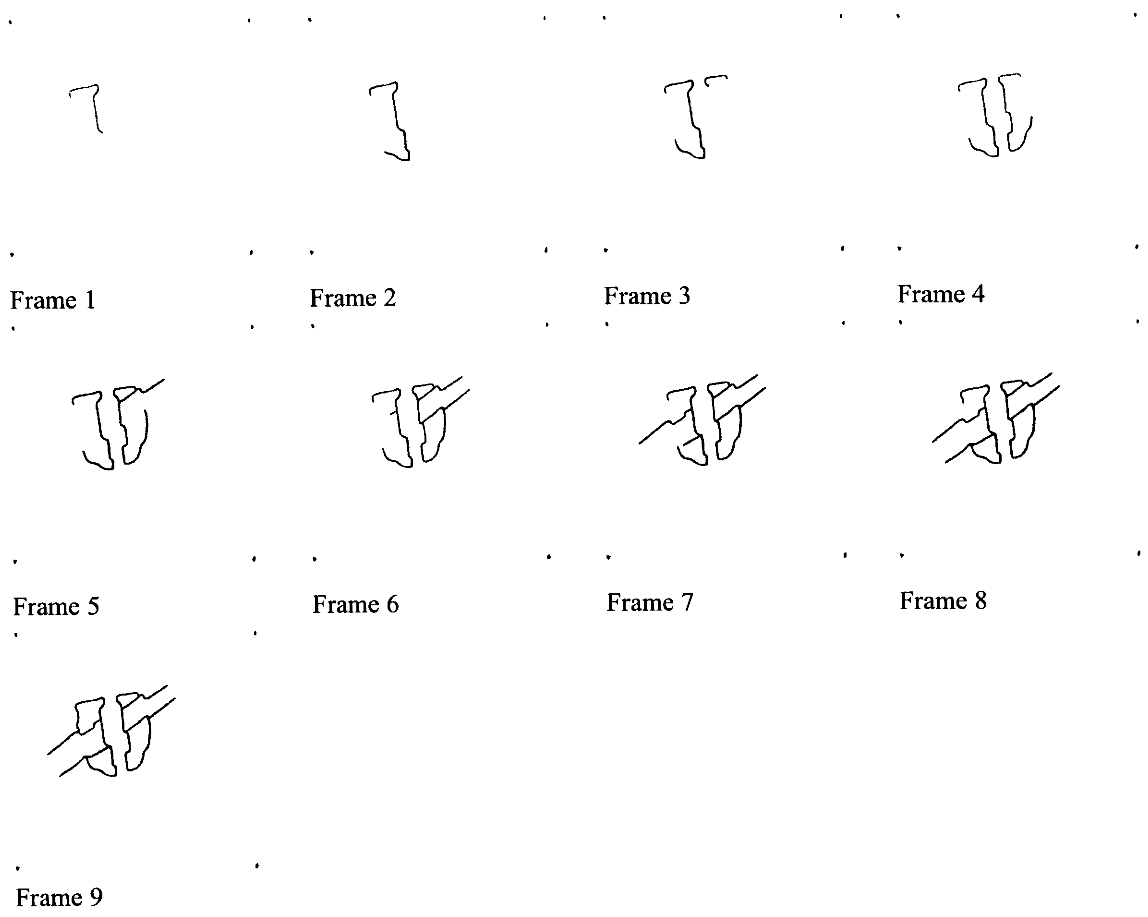


Figure 3–5. An example of shape intersection from the time–based drawings.

Observations on Shape Outlines

It is valuable to note that closed shapes in my drawing style may be inscribed with additional shapes in much the same way as the Ice Ray grammar [49] divides a polygon. In the spirit of this grammar, also known as the Chinese lattice, it is possible to achieve

a) the relation for positioning of the shape within a shape

b) trimming the implied shape's "corner" in order to make a new shape that is proximal to the remaining implied shape.

Specifically, an implied shape may sub–divide as in Figure 3–6.

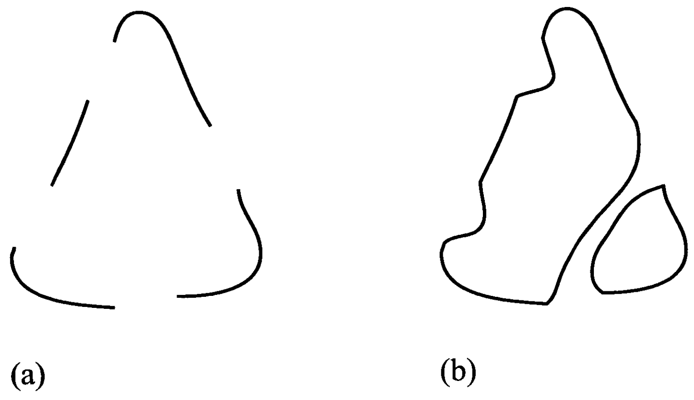


Figure 3–6. Sub–dividing an implied shape. (a) shows the implied triangle that becomes a sub–divided shape as in (b).

Tapia presents a very rigorous algorithm for achieving the above goals in Appendix 2 of [57].

Essentially, the sets of closed outlines fall into two categories as in Figures 3–7, 3–8 and 3–9.

a) internal closures, or shapes punctuated by holes
b) external closures, or clusters of closed shapes

Further in Figure 3–8, where the third outline is a loose reflection of the first outline, and Figure 3–9, where the second outline is a loose reflection of the first outline, we see that clusters of closed shapes serve to create balance in the overall composition.

Additionally, nothing prohibits the puncturing of a closed shape that is part of a cluster of shapes. Indeed, at times outer shape lines function as inner shape lines as in Figure 3–10. Other examples from the time–based analysis suggest a number of embellishments and combinations of inner and outer lines as in Figure 3–11a and b.

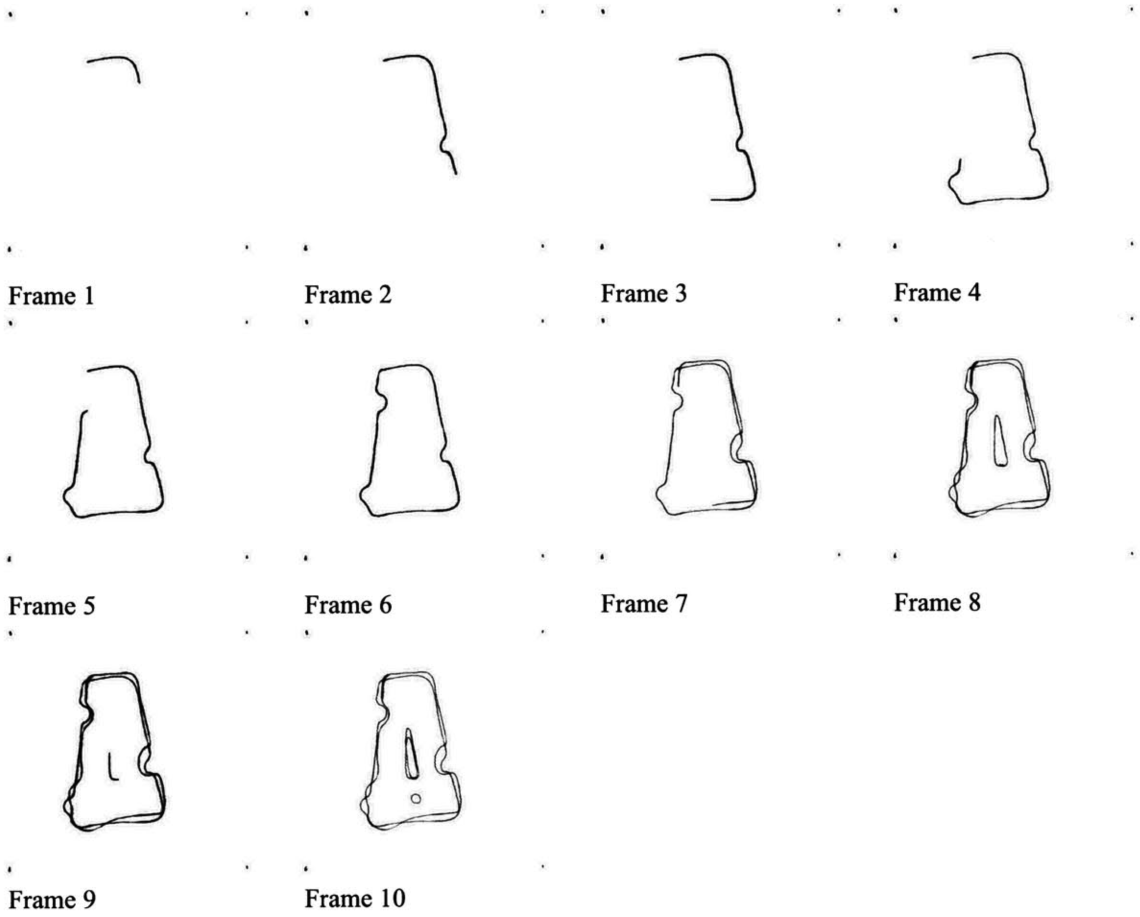


Figure 3–7. Shape punctuated by hole and with embellishments around major outline.

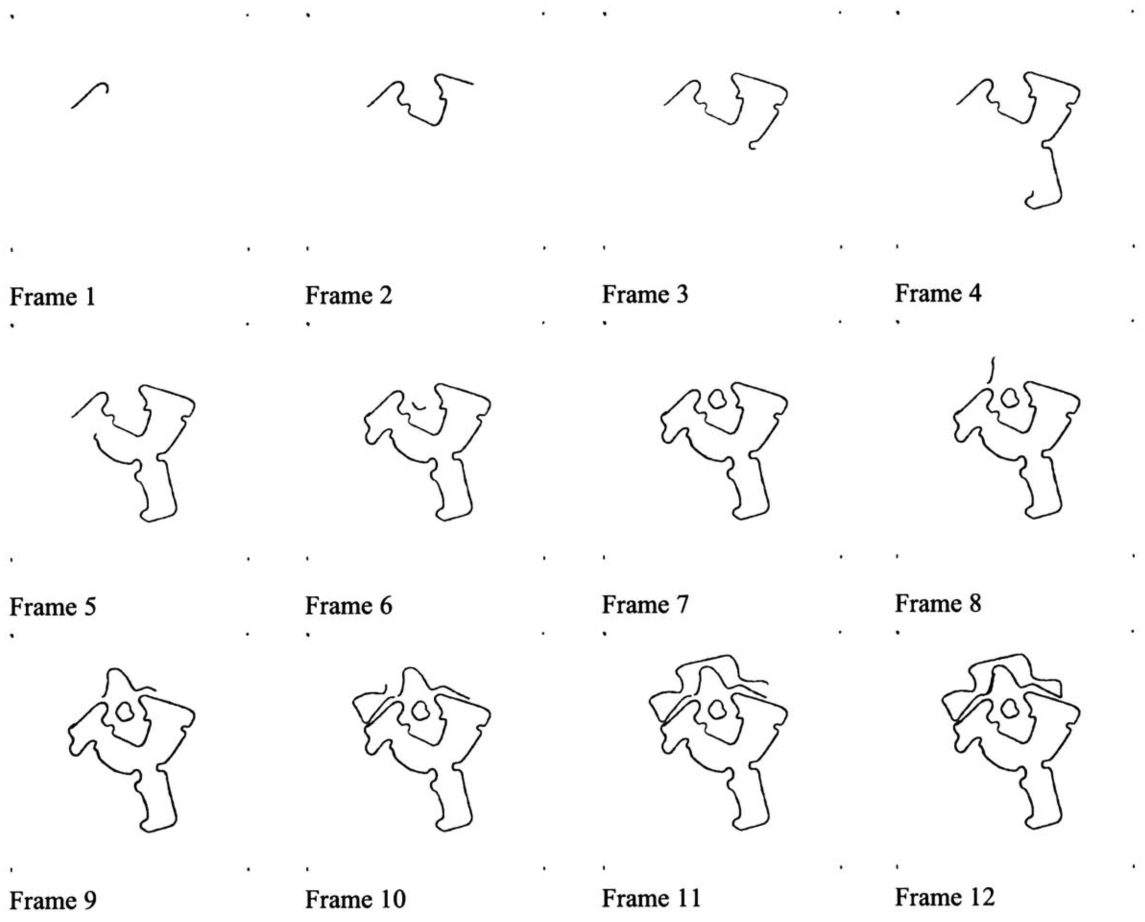


Figure 3–8. Clusters of closed shapes. Positioning creates balance.

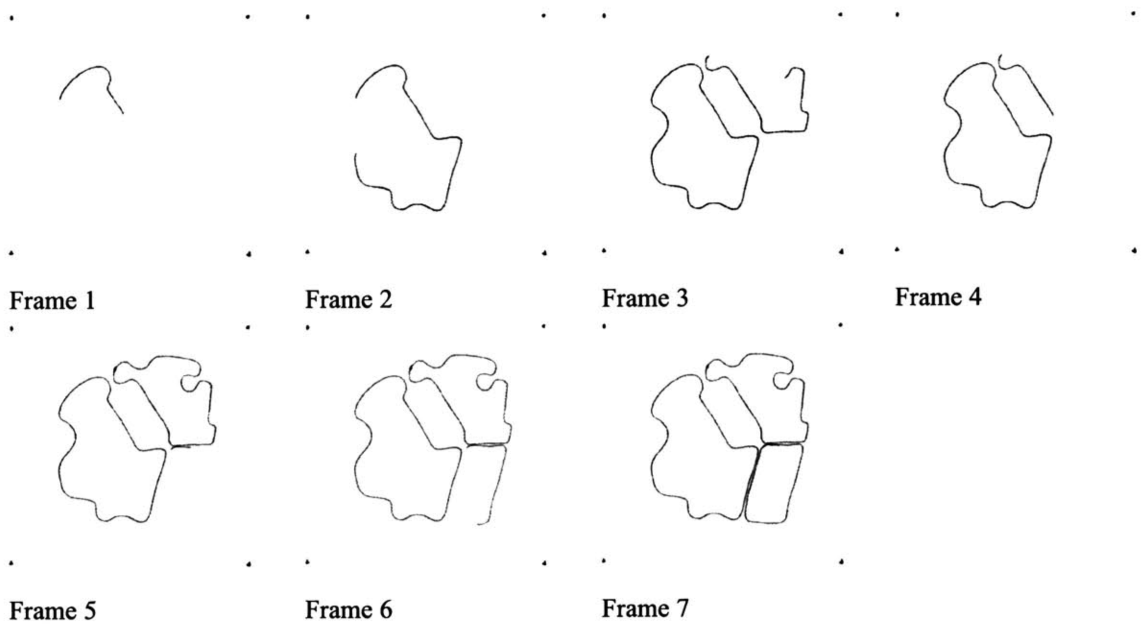


Figure 3–9. Clustering of closed shapes.

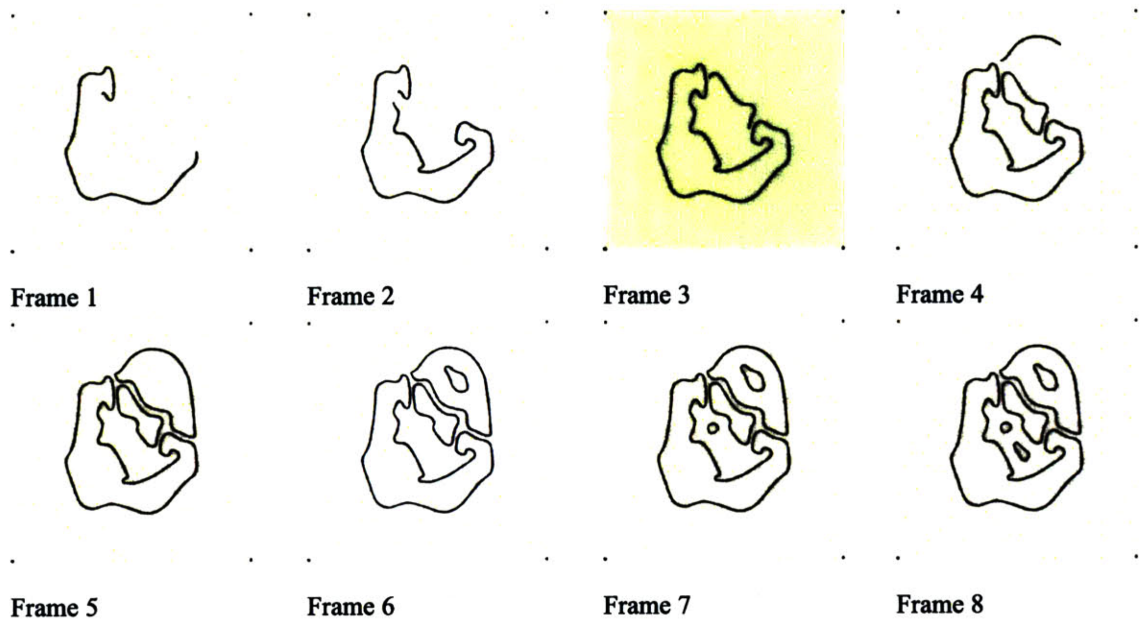


Figure 3–10. Clustering of closed shapes with punctuation.

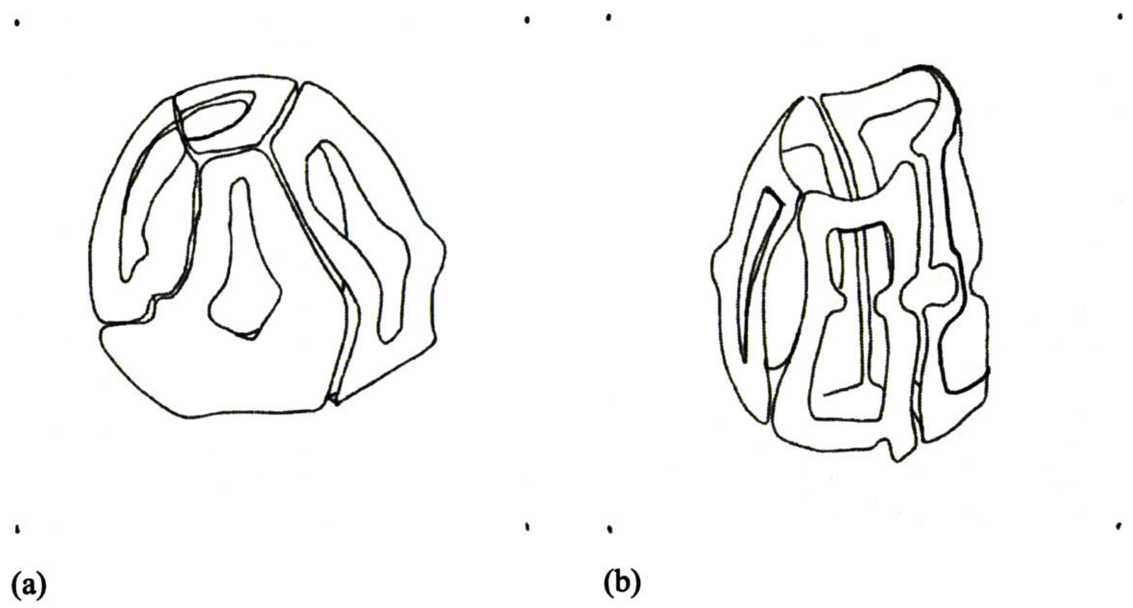


Figure 3–11. Embellished shapes.

Figure 3–12 illustrates the informal grammar that resulted from these observations. The informal grammar does not, however, discern the difference between inscribed, or punctuated, shapes and proximal shapes.

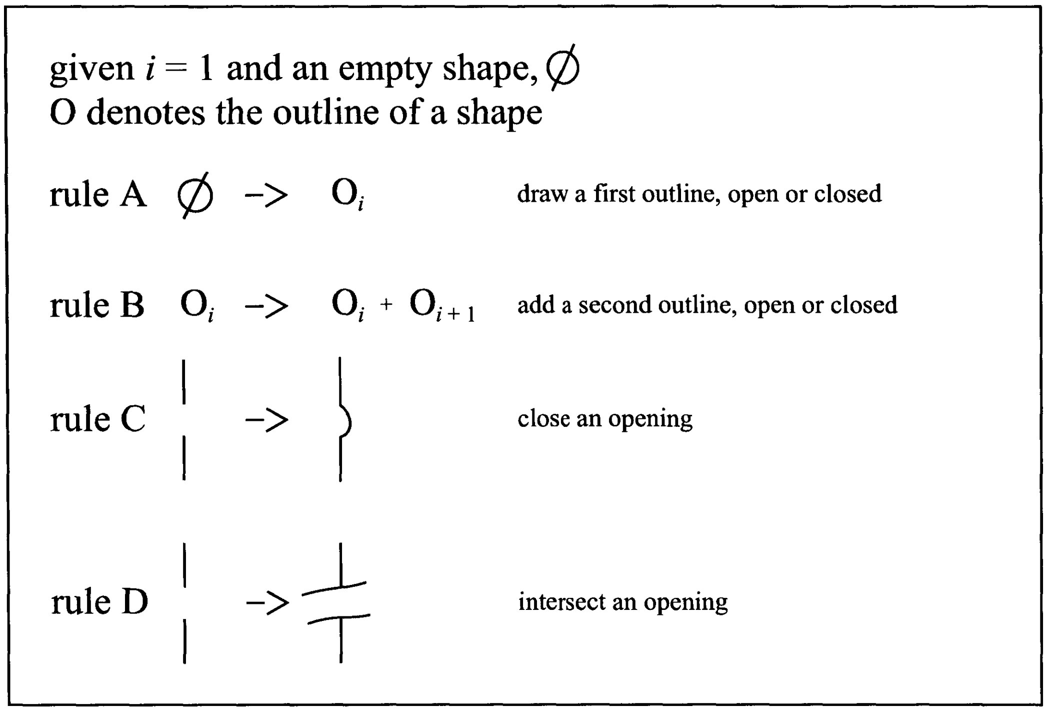
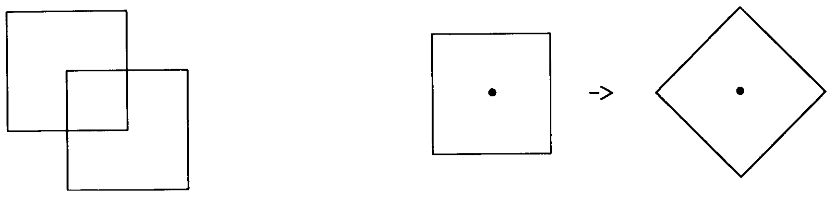

given $i = 1$ and an empty shape, $\emptyset$
O denotes the outline of a shape
rule A $\emptyset$ -> O_i
draw a first outline, open or closed
rule B O_i -> $O_i + O_{i+1}$
add a second outline, open or closed
rule C ->
close an opening
rule D ->
intersect an opening

Figure 3–12. Informal rule set based on generalization of observations and analysis of time–based and sketchbook drawings.

Formal Grammar Based on Observations and Analysis

In this section, I detail the formal grammar for my drawing style. But, before presenting the rules in the grammar for my designs, it is worth stopping a moment to give a definition of a shape grammar and to provide a basic visual example of a formal grammar to illustrate the concepts. Shape grammars, as defined by Stiny [52], provide computational methods for generating designs. Computations start with an initial design state. This state may be the blank canvas, or the empty shape. A computation or rule application is in two parts. The first part is to recognize a shape in the current design state. The second part is to replace the recognized shape with another shape. The replacement can be a transformation, a different shape or an additional shape. The computational rule is defined by a left side, A, and a right side, B. The generalization of a rule, then, is A -> B, and read "erase A and draw B in its place." The rules in the grammar that I will discuss are all visual, but there is nothing that precludes the designer's use of rules to define any aspect of the design such as stroke characteristics, color, materials, etc. [51], [25]. Additionally, the designer may add rules at any point in time without the requirement to re–model the design space. For this reason, the computational approach of shape grammars affords the designer great flexibility and an immediacy of working style that is critical in visual productions.

For our basic example, the initial state is not the blank canvas, but rather a design in progress (Figure 3–13a). A shape rule favored by Stiny that rotates a square 45 degrees (Figure 3–13b) serves as our example. Note that the initial design presents three opportunities for application of the rule as Figure 3–14 illustrates. And Figure 3–15 demonstrates the three possible resultant designs after applying Simple–Rule 1 once. Adding a new rule, Figure 3–16, illustrates the ease of augmenting the design space. Again beginning with the same initial design (Figure 3–13a), we can create some very unexpected new designs as in Figure 3–17.



(a) Initial Design (b) Simple–Rule 1

Figure 3–13. The initial design state begins with an image in progress. Simple–Rule 1 rotates a square 45 degrees about its center of mass.

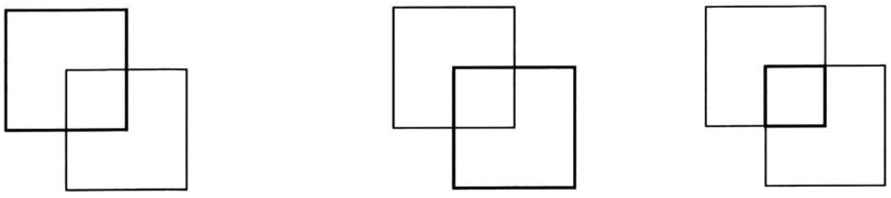

(a) (b) (c)

Figure 3–14. Possible locations for rule application. Each of a, b and c highlight a different square from the initial design where Simple–Rule 1 could be applied

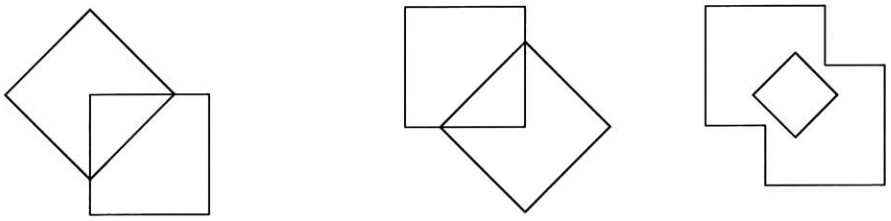
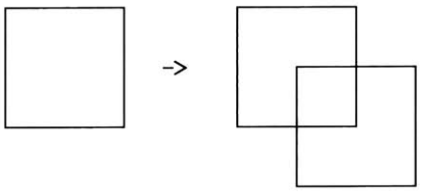

(a) (b) (c)

Figure 3–15. Possible design states. Each of a, b and c correspond to the found squares in Figure 3–15a, b and c above. Here, we see the three possible results of applying Simple–Rule 1 once to each of those cases.



Simple–Rule 2

Figure 3–16. Introduction of a new rule. Simple–Rule 2 allows the addition of an offset square to an existing square. The resultant inner square is exactly one quarter the size of the original square.

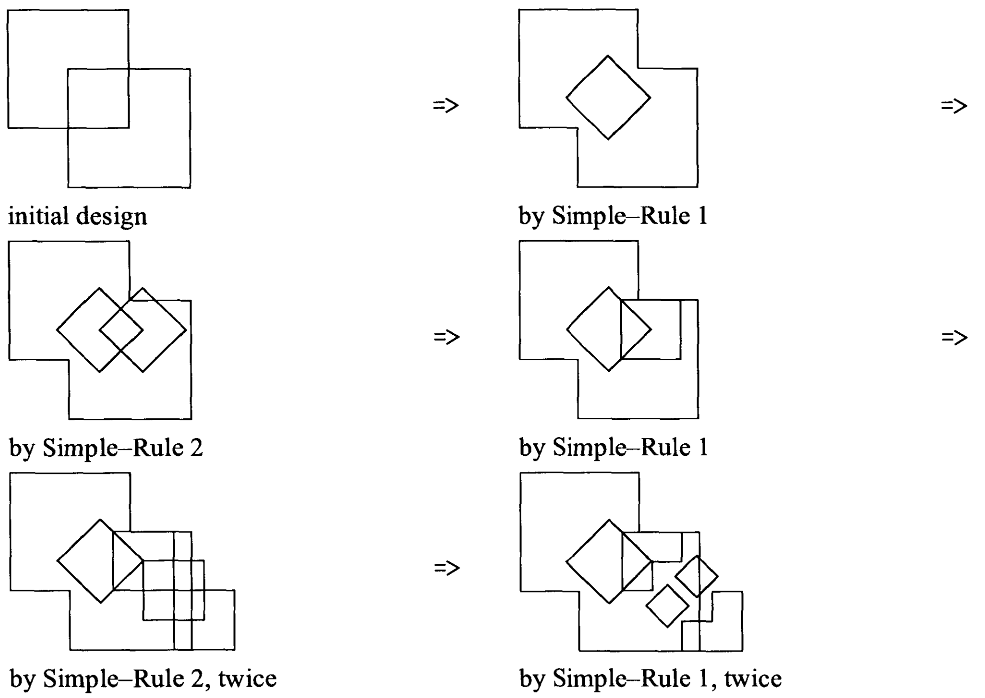


Figure 3–17. A derivation using Simple–Rule 1 and Simple–Rule 2. When considering the simplicity of the rules, the resultant design is somewhat unexpected.

With this simple example, it becomes clear that the designer never needs to remodel the computational framework to accommodate new rules. In theory, one could develop a set of rules over time that represents an entire stylistic evolution. No requirement to supplant earlier rules exists such that information is never lost. New rules, however, can augment existing rules. Such an approach is very freeing as it affords the artist the ability to focus on the different aspects of image making either collectively or separately. For example, my rules for applying color are still in progress. But, I can create the color portion of an image using non–formal, non–computational methods and easily join that product with the formalisms for shape. Once the color rules are in place, it is easy to add them to the formal design language. Nothing that I have created previously with the formal rule base needs to be recalibrated, taught or learned. This is in stark contrast to the processes required in other, fixed algorithmic methods mentioned in Chapter 1.

I will return to the example of Figure 3–14 in Chapter 4 as it serves as a basis for explorations in the computer implementation of shape grammar rules. I dedicate the remainder of this section to the current formalization of the grammar that defines my drawing style. For an earlier rule set that led to this grammar, see Appendix A.

A salient observation on open and closed outlines is that they represent essentially the same forms. An open outline is simply a closed outline with "erasures" loosely fitted along the curve of the implied primitive shape. Further, the primitive shape may be distorted in any number of ways such as stretch, squash and skew. Since ultimately the erased portions serve as space holders for new marks loosely conforming to the same implied shape, the spatial relations of the marked and unmarked portions pose an interesting problem in formalizing the rules related to these segmented shapes. The problem has its roots in the ability to keep an overall balance to the completed shape. Too few segmentations and the result is visibly too similar to its implied primitive. Too many segmentations and the shape does not conform enough to examples from the baseline, hand–drawn corpus.

The first step for the designer is to create a space holder based upon an implied shape. Rule 1 addresses this requirement and Figure 3–18 shows some possibilities. Once the designer determines the implied shape, the next task is to find its center of mass and scale the shape based on the center, as in rule 2. The basic concept behind rule 3 is to replace the implied shape by a loosely conforming piecewise Bézier curve. To define the parametric constraints for manipulating the Bézier, I use minor scaling of the implied shape to generate guide shapes. The current experimental scale factor is +/– 10%. Next, a set of intersecting lines at the interior of the smaller guide shape creates intersection points on the scaled guide shapes and the original implied shape. These intersections serve two functions. The first is to define the sub–shapes of the implied shape that the Bézier curves will ultimately replace. The second is to define the parameters within which the Bézier pieces can be manipulated.

The following rules will use the triangle as an example. I show a specific derivation later in this chapter and leave the specific details of calculating the Bézier curves for Chapter 4.

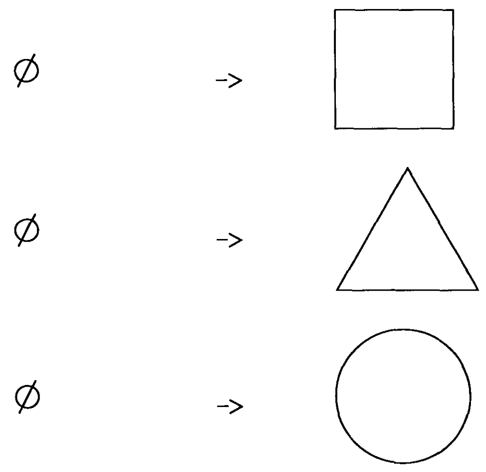


Figure 3–18. Rule 1. Replace the empty shape, ∅, with a closed shape which will act as a space holder, or implied shape, on the canvas. The implied shape need not be limited by the examples shown here. Additional rules are easy to create. For example, a closed shape could be replaced by a rectangle, pentagon, etc.

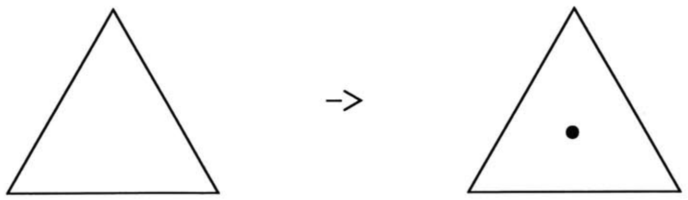


Figure 3–19. Rule 2a. Calculate the center of mass of the implied shape.

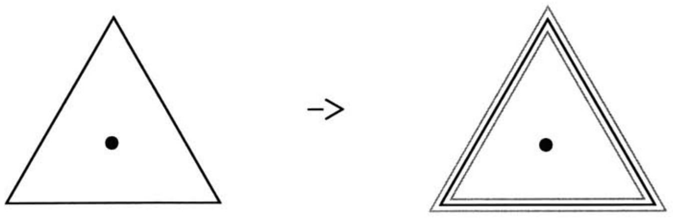


Figure 3–20. Rule 2b. Scale the implied shape up 110% and down 90% from the center to create the parameter space guide shapes for perturbing the implied shape. Here the guide shapes are in light grey.

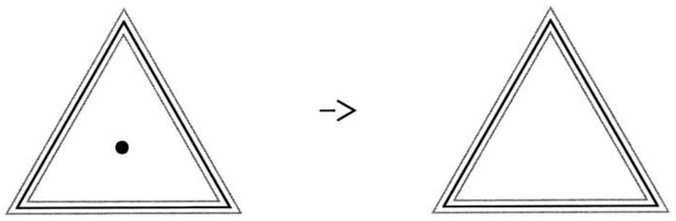


Figure 3–21. Rule 2c. Once the guide shapes are in place, erase the center of mass marker.

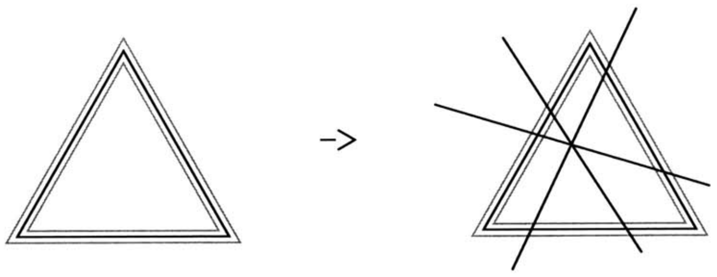


Figure 3–22. Rule 3a. Draw intersecting lines equal or greater in number to the number of sides of the implied shape. Their intersection must be at the interior of the smaller guide shape and they may not coincide.

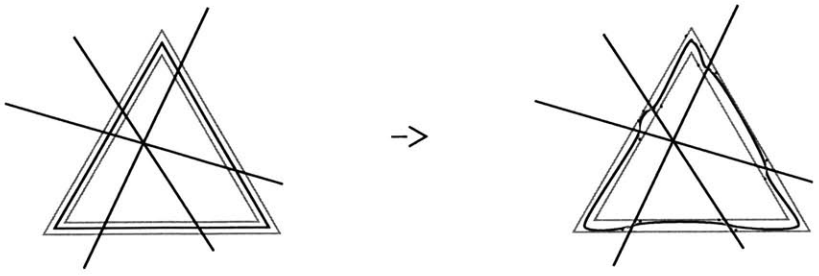


Figure 3–23. Rule 3b. Once the intersecting lines are in place, the points where they intersect the guide shapes and original implied shape become the parameter space. Sub–shapes in the form of Bézier curves replace the original implied shape. The guide shapes remain to serve as the constraints for moving the control points of each individual Bézier.

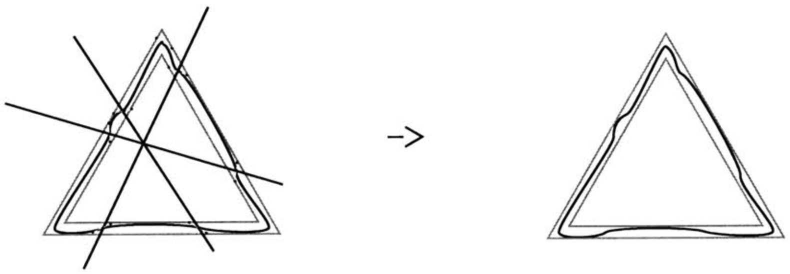


Figure 3–24. Rule 3c. Remove the intersecting lines and hide the Bézier handles.

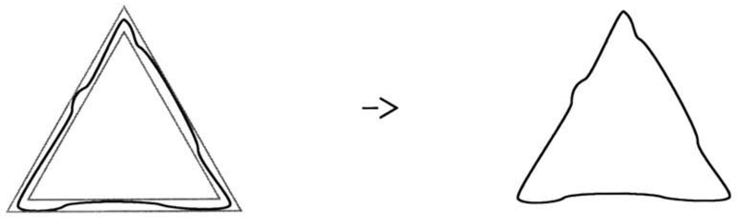


Figure 3–25. Rule 3d. Remove the guide shape to leave the final parameterized shape.

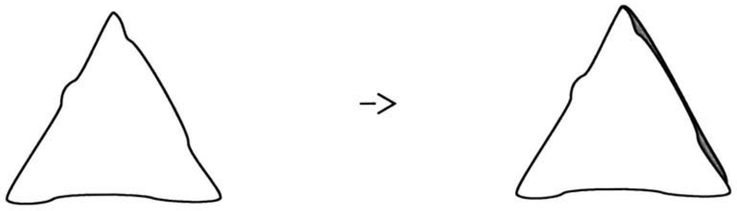


Figure 3–26. Rule 4 shows the ability to add a decoration to the exterior of a shape much like a drop shadow effect to create depth to the form.

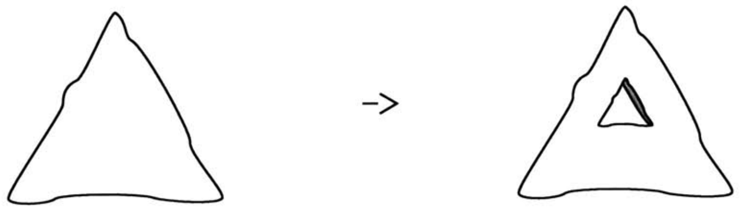


Figure 3–27. Rule 5 shows the ability to add a decoration to an interior shape much like a drop shadow effect to create depth to the form.

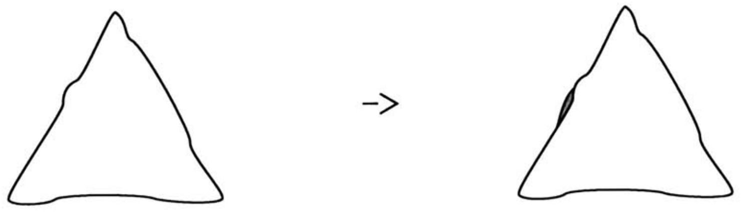


Figure 3–28. Rule 6 is a variation of Rule 4 that allows further decoration to the exterior of a closed shape.

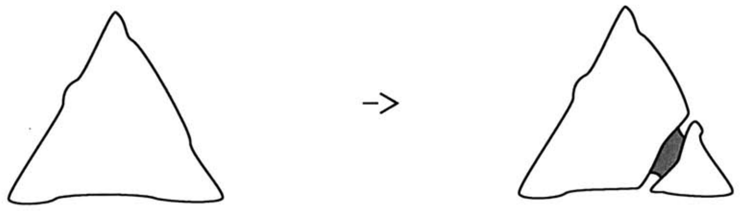


Figure 3–29. Rule 7 allows a decoration to be added between two closed shapes.

Revision of Segmentation Parameters

While the rules from the previous section capture many of the subtleties of perturbing the implied shape, they can produce shapes with unpleasant cusps, or crinkles, that are not consistent with the foundation corpus. This section demonstrates a series of revisions that brings the synthesized design in closer alignment with the hand–drawn shapes that they aim to emulate. Two main changes facilitate the result. The first relates to the adjustment of the corners on the implied shapes. The second is to parameterize further the allowable positions of the points on the Béziers. Essentially, we delete all corners of implied shapes and the set of intersecting lines alone defines the points on the perturbed shape. The result is softer corners defined by the two nearest Bézier control points and gentler undulation along the edges of the shape. The cross–shaped markers, +, constrain the control handles of the piecewise Bézier that "turns", or rounds, the corner. The markers that intersect the implied shape and connect the two guide shapes, _ and |, constrain the remaining points. In essence, the markers serve as sliders for the appropriate points, and when rounding a corner, their control handles. Figure 3–31 shows a close up of the setup and Figure 3–30 illustrates the actual rule. Figure 3–32 shows the perturbed shape less the parametric guides, but with the Bézier points and control handles as well as the final shape.

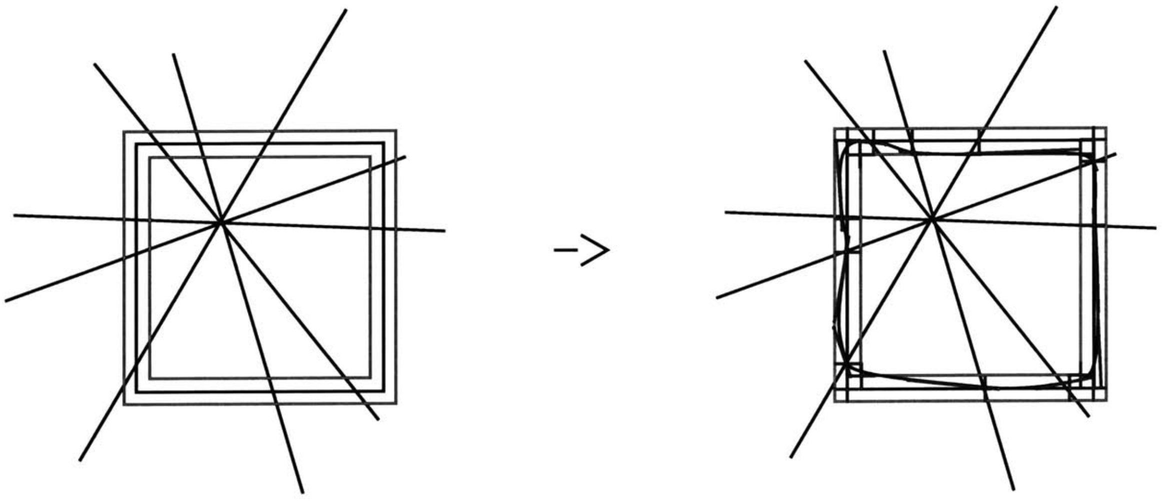


Figure 3–30. The revised rule 3b shown above in Figure 3–23. The major difference is in the handling of the corners and in the parameter space of the individual points of the Bézier curves.

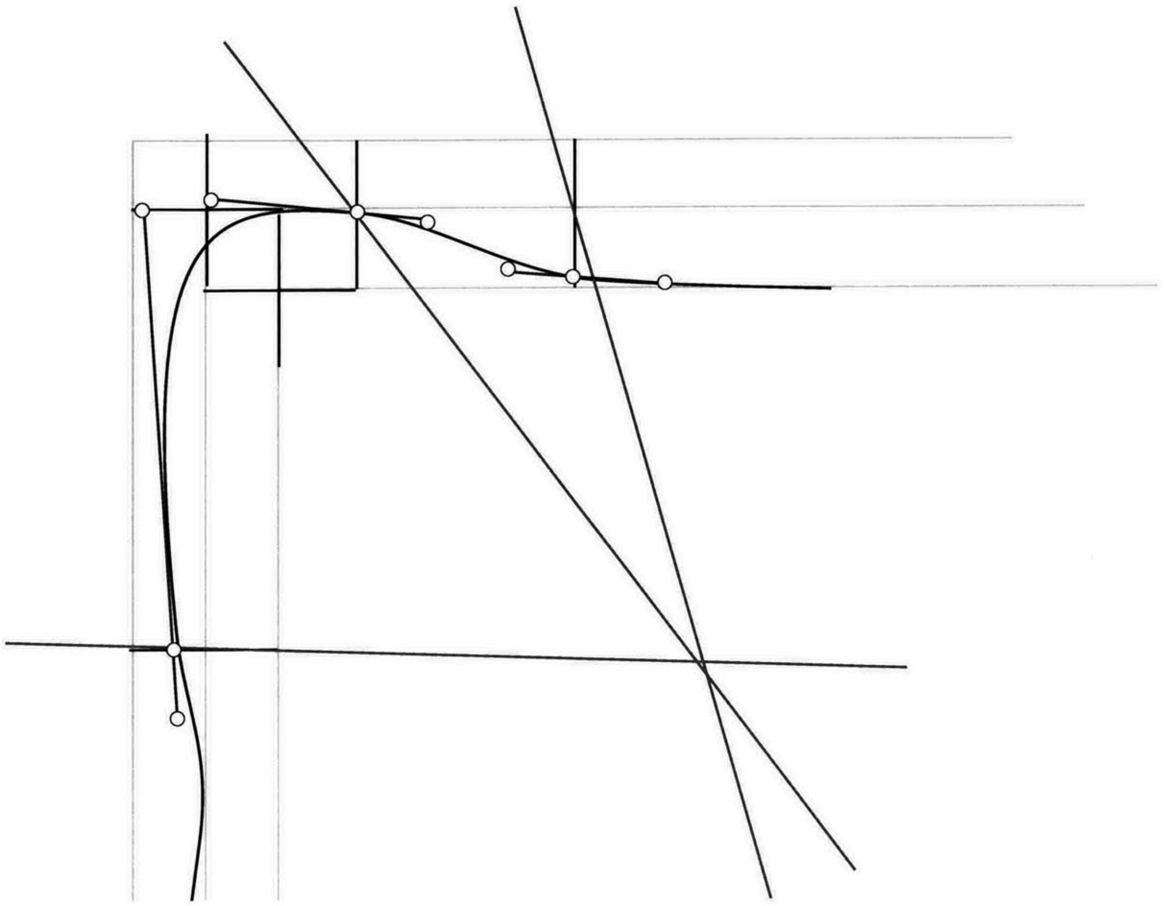


Figure 3–31. Close up of the top left corner of the shape rule. The crossed markers regulate the position of the control points on the corner Bézier. The horizontal and vertical markers regulate the position of the remaining points on the piecewise Bézier. The control points not associated with the corner point must remain within the two guide shapes and may not extend beyond the position of their nearest control point.

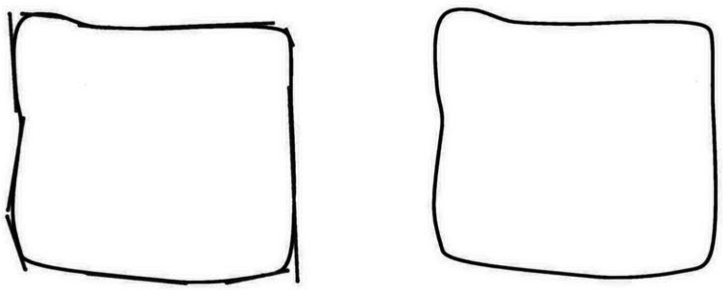
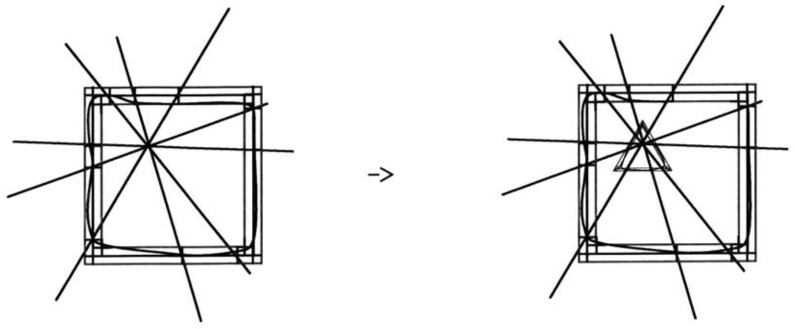
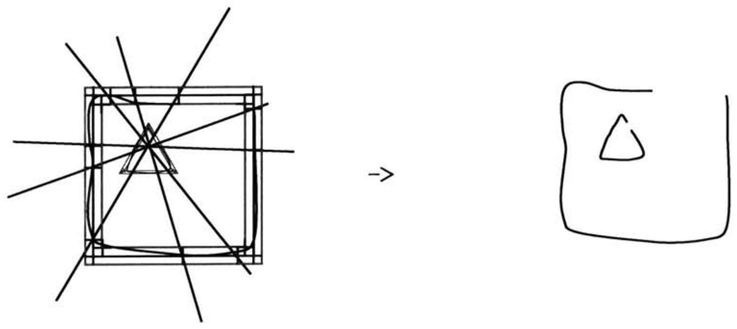


Figure 3–32. (a) perturbed shape without the parametric guides (a) shape with Bézier controls hidden (b).

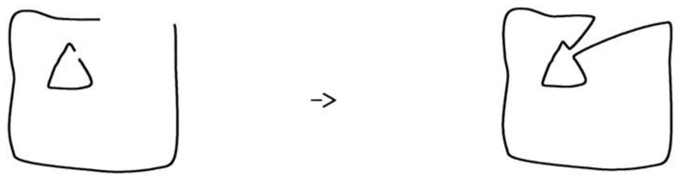
Clearly, this revised rule applies equally to exterior shapes and interior shapes. However, the rule only applies in the cases where these shapes types are not joined. The next step is to build upon these refinements to join interior and exterior shapes. Given the mechanisms now in place for perturbing the interior and exterior shapes, this step becomes rather straightforward. Beginning with the segmented exterior shape as in Figure 3–33a, we draw an interior shape such that the intersection of the parameter lines is at its interior. We follow all the rules for segmenting an implied shape to perturb the interior shape, a triangle in our example, according to the correct parameters. Then, we delete a segment on each of the interior and exterior shapes as the common intersecting lines indicate (Figure 3–33b). We may repeat this step multiple times. Finally, we join the open ends of the interior and exterior shapes. An example of a final drawing follows in Figure 3–33c.



(a)



(b)



(c)

Figure 3–33. Rule 8 a, b, c joining interior and exterior shapes.

Manual Synthesis and Testing of the Design Language

In the previous sections, I have detailed a rule–based design language using a shape grammar. The purpose of this language is to illustrate the ability to emulate the hand productions of my foundation corpus using visual algorithms. In this section, I address the results of the shape rules—the manual synthesis— by comparing their production output with the results of my freehand drawings in the same style. In the first iteration of the test, although I do not use a computer implementation, I strictly adhere to the grammar as I manually construct the shapes. In the second iteration, I draw shapes freehand with an intuitive understanding of the rules. In both cases, I use digital drawing tools. Finally, I compare and contrast the two results in an effort to understand the success and room for improvement with respect to the rules. A caveat remains, however. Although the current–day freeform drawings bear clear resemblance to the drawings of the foundation corpus as in Figure 3–34, my knowledge of the formal grammar significantly informs any drawings that I now do in this style.

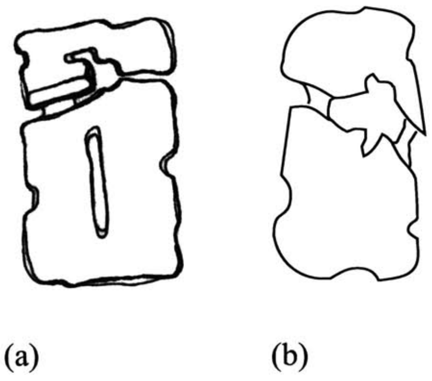


Figure 3–34. Comparison of shapes pre and post computational framework. I have drawn both shapes by hand where (a) illustrates a sketchbook drawing and (b) is a freeform drawing example using a vector graphics software package. Both shapes support the notion of implied shapes, specifically an implied exterior rectangle and an implied interior triangle.

Each iteration of the test begins with the same set of implied shapes as in Figure 3–35a. Figure 3–35b illustrates a derivation based on strict adherence to the rules. Figure 3–35c illustrates the results of drawing freeform, specifically without strict application of the rules.

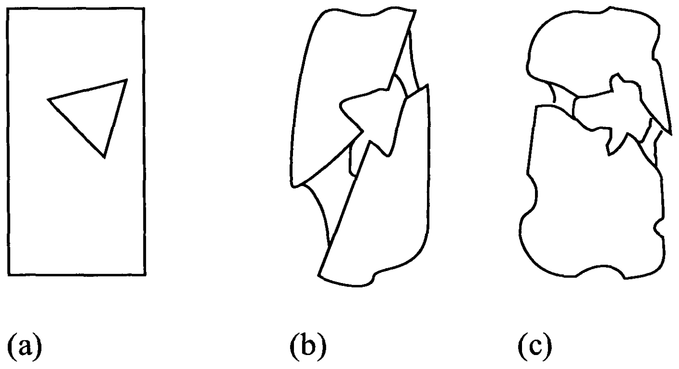
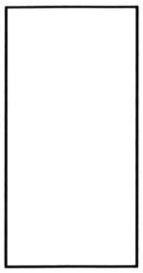


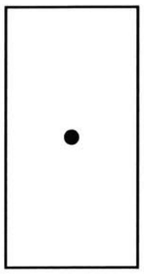
Figure 3–35. Testing the design language. (a) beginning implied shapes (b) rule derived shape (c) freeform shape

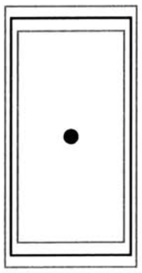
In comparing the results in Figure 3–35b with 3–35c above, we observe that both shapes follow the general spirit of the implied shapes and one may generally conclude that they've been drawn in the same style and perhaps by the same artist. Below, Figure 3–36 illustrates the derivation for the rule–generated shape while Figure 3–37 shows significant stages in the freeform drawing. The rules governing segmentation of the implied shape capture the undulation of the implied form moderately well although we can see that perturbations in the freeform shape are more radical than their rule–based counterparts. The rules for joining exterior and interior shapes handle that relationship reasonably well although the joins in the freeform shape go further toward obscuring the implied interior shape than the formal rules currently support.

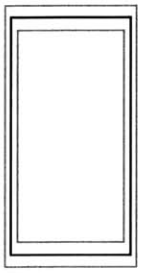
Contrasting the results, we can see that the shape I constructed by strict adherence to the rules is much more economical than its freeform counterpart as it literally contains less data. The current iteration of the shape rules knowingly restricts the number of points and hence the number of Bézier curves that form each shape. When drawing the freeform shape, the curves are far less simplified as the digital drawing tool is set to a high level of precision. Specifically, points are drawn at a higher temporal frequency for the duration of the stroking gesture. This higher level of precision more closely emulates the hand–drawn feeling of the shape. To represent this aspect of the hand, then, future iterations of the shape rules need to capture information about precision and number of data points collected over time. We can liken this phenomenon somewhat to the use of a pencil when the drawing gesture is slower or faster. Clearly, a fast gesture will be more economical even in the continuous, analog domain of the pencil. While conversely, a slow gesture will contain more information and have a less than clean, even jittery, appearance on the page.

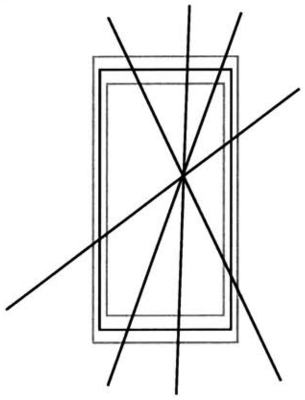
A second observation is that the parameter space defining the "erased" portion of the implied shapes is too predictable and symmetrical in comparison to the freeform equivalent. The experimental 10% value of uniformly scaling the implied shape up and down is too restrictive. A first step in its refinement would be to make the scaling factor non–uniform. Despite the need for further rule refinement, however, the overall impact of the rules is that they generate shapes that are clearly in the style of the non-rule–based forms that they seek to emulate.

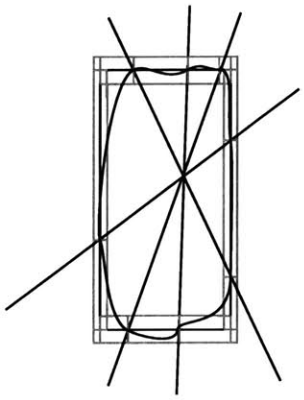

by rule 1a

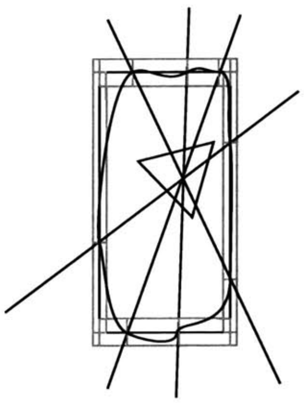

by rule 2a

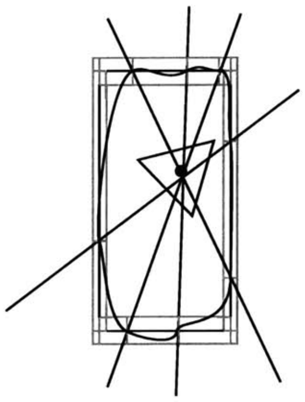

by rule 2b

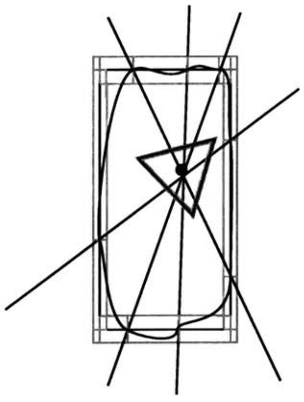

by rule 2c

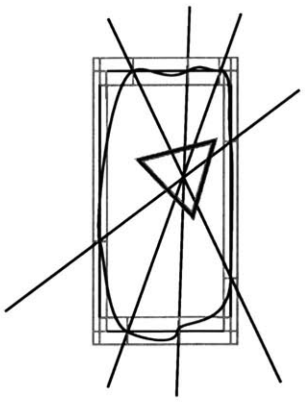

by rule 3a

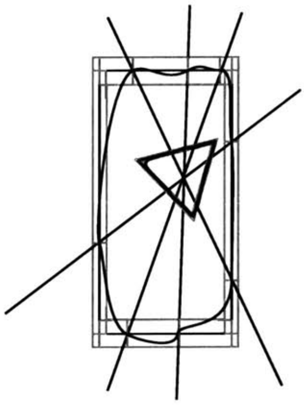

by rule 3b

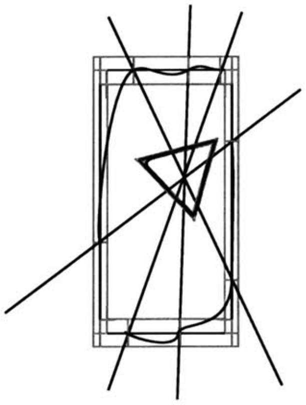

by rule x1a


by rule 2a


by rule 2b


by rule 2c


by rule 3b


by rule 8b twice

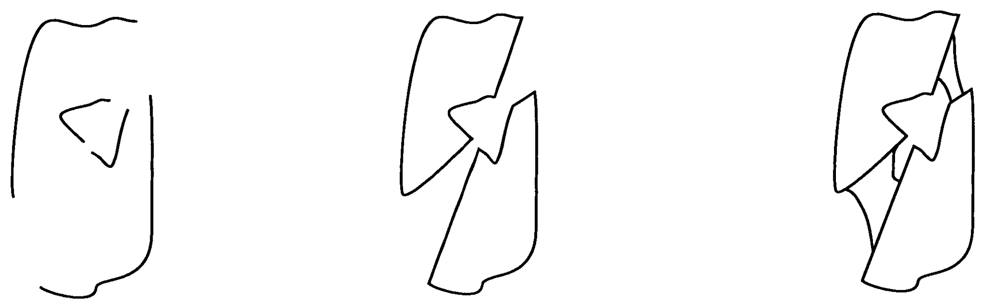


Figure 3–36. A step–by–step derivation of a shape using the rule–based design language.

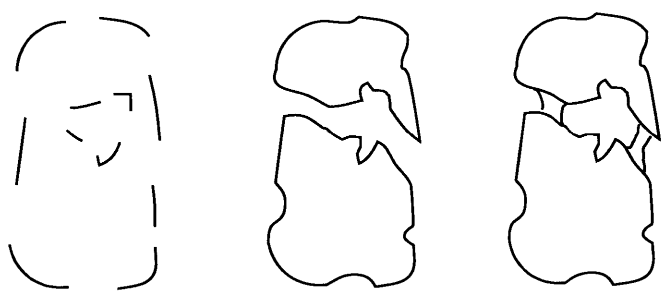


Figure 3–37. A freeform drawing at three different stages. (a) illustrates the use of implied shape to hold space during the drawing of two segmented shapes. (b) shows the connection of the segments and the joining of the interior and exterior shapes. (c) is the finished shape with the addition of depth creating connecting marks between the two shape parts.

Observations about Color Usage

This section discusses my use of color during painting exercises. Although I have not yet formally defined the rules for this usage, this section shows how formalized rules on drawing can easily join with non–formal approaches on the way to creating a finished image. And, at some point in the future it will be useful to join the formal painting rules with the formal drawing rules.

My painting process divides itself into four different stages, or progressions. As Chapter 2 describes, I first define the color palette. On the selection of the actual colors in the palette, I will note that my preference is to maintain the hue while manipulating saturation and brightness. The result is a fairly monochromatic palette. At times, however, I do vary hue and in those cases saturation and brightness remain relatively constant. With these two remarkably simple rules, the colors in the palette are always very pleasing. Additionally, I use these palettes repeatedly for any number of new paintings.

The balance of this section is an overview of the three remaining progressions. We can think of each of these progressions as "ground layers" for the phases and progressions to follow. Progression One is the loosest and quickest of the set, the results of which are in Figure 2–2. Progression Two introduces shape derived from the formal rules for drawing. Progression Three adds a rule–derived semi–transparent shape layer to the painting. I present the visual results of these techniques in Figure 3–39.

Progression One

Phase One

With the original color palette as a clone source, I begin Phase One on a blank canvas and use digital watercolor tools to soften existing colors and mix new ones. I incorporate some of the emergent colors as I elaborate the canvas. The end of Phase One, Figure 3–38a, is a watery, dreamlike representation of color that largely represents the original color palette.

Phase Two

This phase builds on the image of Phase One by introducing additional digital media such as chalk and oil. These tools serve to create additional mark types and texture as well as to introduce newly mixed color to the canvas.

Progression Two

This progression also typically starts with a blank canvas. In terms of rules, we can think of this as subtracting the entirety of the previous progression that has been saved. Eventually, most of its content will be added back to the canvas, but not before adding a shape drawing. I derive the drawing portion of the painting from the shape rules that I have defined earlier in this chapter. Once the shape is in place on the canvas, as in Figure 3–38b, I slowly add color from the end of Progression One back onto the canvas. I use a variety of digital tools, most frequently watercolor, chalk and oil, in this phase of Progression Two. The chalk defines the edges of the shape and the watercolor implies its volume. Although I may dip back into the original color palette, I typically restrict my usage to the colors of the current canvas and those present in the final Progression One image.

Progression Three

This progression is a mixture of formal rules and as yet unformulated color manipulation. Its main purpose is to add a semi–transparent layer to the painting. The visual result is that of looking through a veil or fogged glass to the rest of the painting. This progression also starts with subtraction as I use the shape rules to erase a portion of the painting.

These erasures create highlights on the canvas. I then use the watercolor tool to bring back color and form from the final image of Progression Two thus adding depth to the image. The darkest colors of Progression One create drop shadows and recessed areas of the painting (Figure 3–38c).

Examples of final paintings resulting from these progressions are in Figure 3–39.

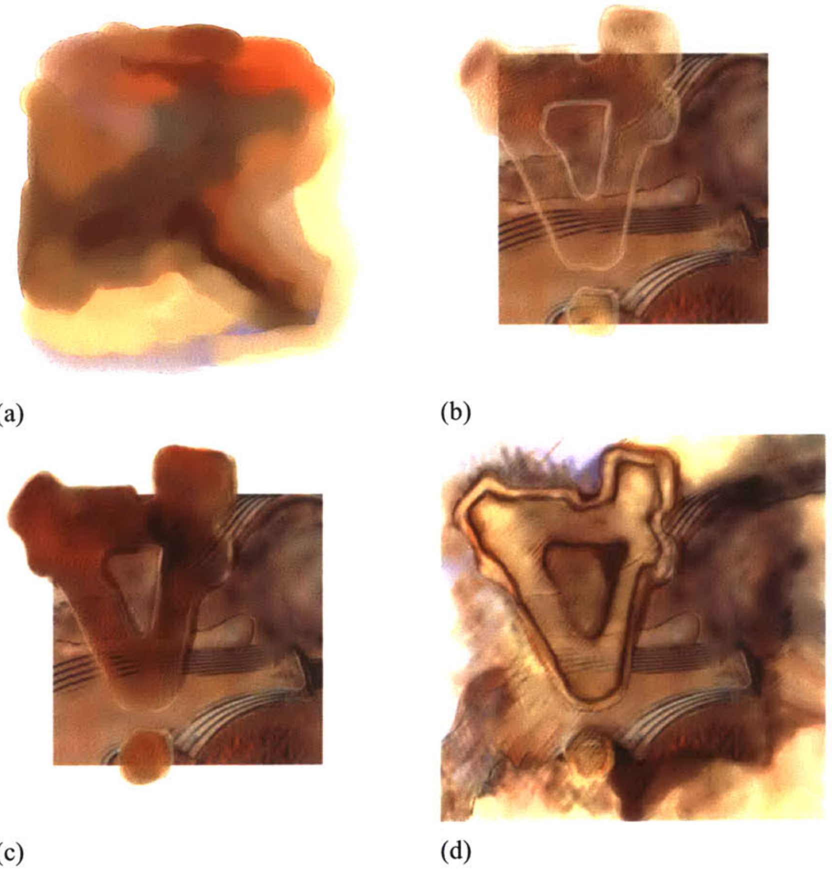
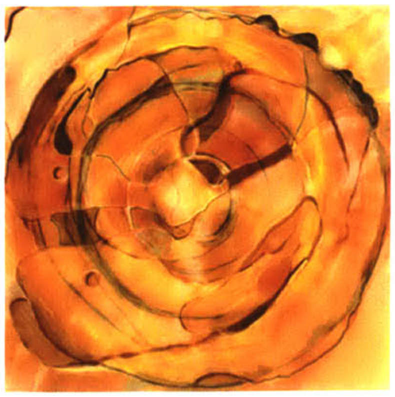
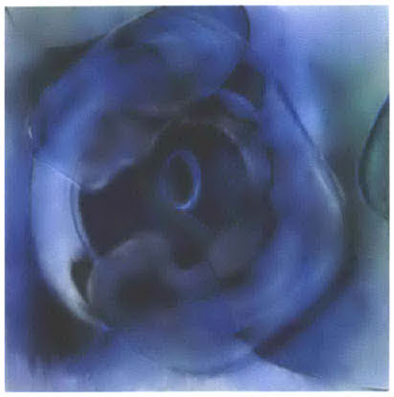
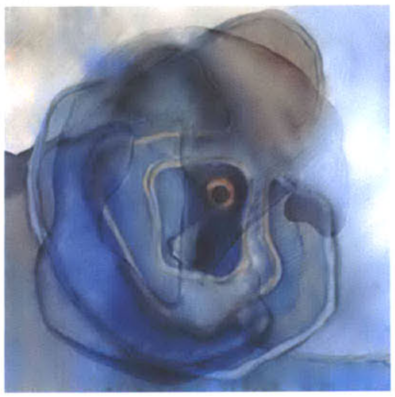

(a) (b) (c) (d)

Figure 3–38. Painting in progress, Fire. The final painting is shown in Figure 3–39. (a) digital watercolor image based on original color palette (b) adding a rule–based shape to the painted canvas (c) semi–transparent layer added to the painting (d) further elaboration using digital chalk and oil tools.


(a) Sun


(b) Thunder


(c) Moon

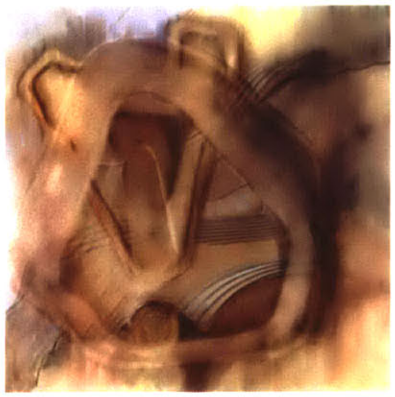

(d) Fire

Figure 3–39. Final digital paintings. Each of these paintings is a combination of formal shape rules and informal color usage.

Chapter 4 Design of Computer Implementation Methods

Chapter 3 dealt with the visualization aspects of this research. Specifically, I described the analysis of my drawing style and the subsequent development of a rule–based design language using shape grammars to synthesize the style. The synthetic method of Chapter 3, however, is a manual computation that does not rely upon computer implementation techniques. With the curvilinear design language of the previous chapter as a motivator, this chapter discusses the representation issues associated with computer implementation methods for a general shape grammar. Here, I consider both rectilinear and curvilinear shapes. Most specifically, I will focus on a novel use of cubic Bézier curves as representational forms when programming a computer implementation of a shape grammar.

The shape–grammar literature is very explicit on the techniques required to compute with rectilinear shapes using both manual computational [18] and computer implementation methods [6], [57, 58]. Relatively little exists, however, on the topic of computation with curvilinear shapes [52],[20]. Given the objective of this research, namely to consider the immediacy of the mark in computation, the need to include representations of curvilinear forms is great. Conceptually, the formal devices used in shape–grammar computations support lines and curves equally. Practically, however, choices of representation figure significantly in the computer-programming phase.

Computer Implementation for Rectilinear Shapes

Explicit shapes are those drawn directly by the artist on the canvas while implicit shapes are those that emerge as a result of explicitly placed shapes. Explicit and implicit shapes may by rectilinear or curvilinear. The search and replacement of explicit shapes is easily accomplished in today's off–the–shelf vector graphics software packages. The search and replacement of implicit shapes, however, is still not available. Figure 3–14c is an example of rectilinear explicit and implicit shape types as shape–grammar techniques afford the ability to locate and transform both these forms. The fundamental devices for achieving are the maximal element and the embedding relation. Specifically, in the case of rectilinear forms, the maximal–element is the line. A line is maximal if it has no further decomposition. Meaning, it is not part of any other line on the canvas and is uniquely describable. Any lines that are collinear and non-disjoint can be reduced to a maximal line. Figure 4–1 shows the cases where two lines are reducible to a single, maximal line [47] while Figure 4–2 shows several examples of maximal lines.

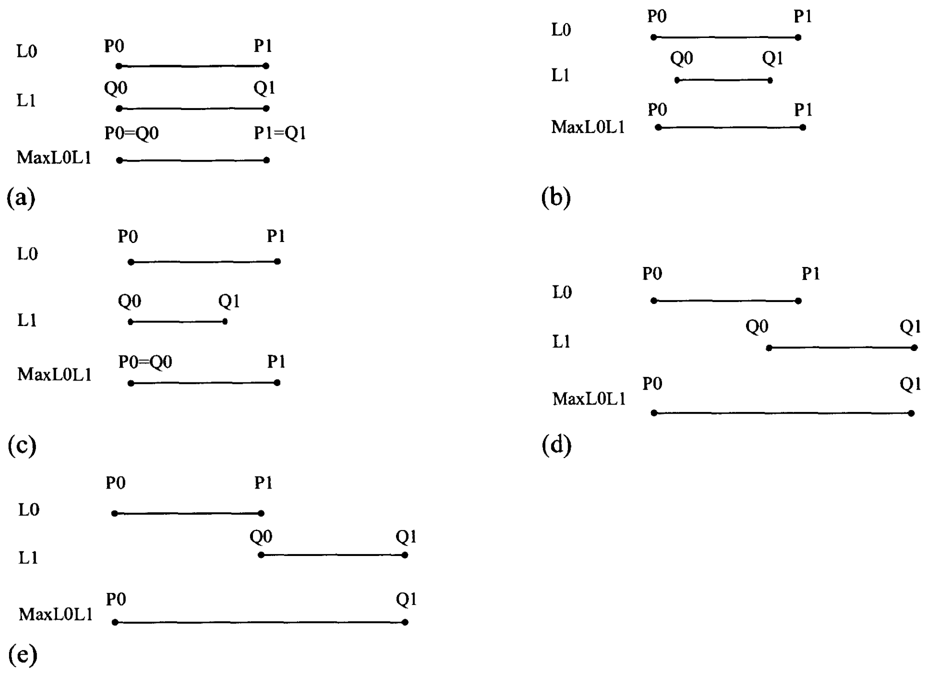


Figure 4–1. Reduction rules. For clarity, the lines are not drawn collinearly as they would appear on the canvas. The lines L0 and L1 are collinear and defined respectively by the points (P0, P1) and (Q0, Q1. MaxL0L1 is the maximal lie representation. (a) Both boundaries shared. L0 and L1 represent the same line where P0 = Q0 and P1 = Q1. The points (P0, P1) define the resultant maximal line. (b) No shared boundaries and complete overlap. P0 < Q0 < Q1 < P1, the resultant maximal line is (P0, P1). (c) One shared boundary and overlap where P0=Q0, Q1 < P1. The resultant maximal line is defined by (P0, P1) (d) No shared boundary and partial overlap of L0 and L1 where P0 < Q0 < P1 < Q1. The resultant maximal line is the reduction defined by (P0, Q1) (e) One shared boundary and no overlap where P1 = Q0, P1 < Q1. The reduction is defined by (P0, Q1).

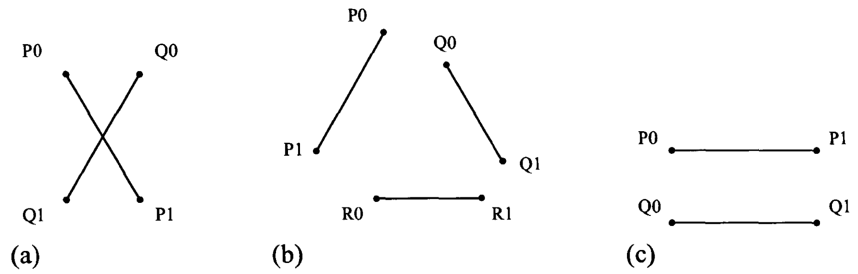


Figure 4–2. Maximal lines. Three possible spatial relations among maximal lines.

Another way to describe the notion of maximal lines is in terms of the embedding relation. If a line, L1, is embedded in another line, L0, then L1 is not maximal. The notation of within, designated by <=, captures this relation. Formally we can say a line, L1, is embedded in L0 if and only if every maximal element in L1 is embedded in a maximal element in L0, i.e., L1 < L0=. In shape grammars this is also known as the part, or sub–shape, relation. So, simply stated, a shape is embedded when it is part of another shape.

There are several ways to represent curves, and hence lines, in space. Explicit, implicit and parametric forms are well known with each having its benefits and drawbacks. The parametric form is the most flexible as it may represent a line in any position in 3D and is easy to manipulate in computations. Using the parametric form also makes it easy to join pieces of curves together giving the appearance of a fluid stroke. This aspect is particularly important for a curvilinear design language in general and mine in particular.

Conceptually, a parametric line is movement along a vector over time and in 2D can be generalized as

$$(x, y) = (f(t), g(t))$$

where t is time, f(t) is horizontal motion and g(t) is vertical motion. For a finite segment defined by two points, P0 and P1, over the unit interval, the parametric line representation for the general case is

$$P(t) = P0 + t(P1-P0) \text{ for } 0 >= t <= 1 \qquad (1)$$

and for the 2D case is

$$P(t) = P0(x) + t(P1(x) - P0(x)) \qquad (2)$$

$$P(t) = P0(y) + t(P1(y) - P0(y)) \quad (3)$$

Figure 4–3 shows a visual representation of the parameterized line where t is a fraction of the distance from P0 to P1. For example, P(.5) indicates the midpoint of P0P1. The vector representation is opportune not only for lines. An element is maximal if and only if it embeds everything continuously along its same vector. This understanding is important when we think about how to represent maximal curves within the constraints of a computer implementation and is explored later in this chapter.

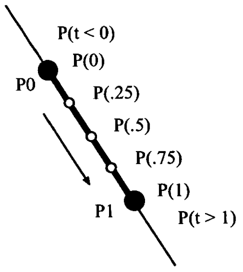


Figure 4-3. Parametric line. P0P1 indicates a finite segment and P(t) a fractional distance between P0 and P1. P(0) = P0 and P(1) = P1.

Not only is this representation useful when the shapes become curvilinear, it also makes issues such as embedding and shape difference, i.e. erasing, easier to compute programmatically as the following section shows.

Maximality, Embedding and Erasing with Parametric Lines

Recall from the definition of a maximal line that we need to determine if two lines are collinear and if they are non–disjoint. This section describes how the parametric line applies to these requirements as well as describes how the parametric representation facilitates finding embedded shapes and erasing shapes.

Collinearity

The use of parametric lines makes reducing two lines to their single maximal representation straightforward. Once we have determined that two lines are both collinear and non–disjoint, we simply adjust the boundary parameters appropriately.

First, we test two lines for collinearity using the area of a triangle approach. Three points are collinear if the triangle that they define has a zero area. We take the four points that define the two lines in question three at a time for a total of four cases to test. To calculate the triangle's area, we use the familiar determinant form:

$$\text{Area} = 1/2 * \begin{vmatrix} x0 & x1 & x2 \\ y0 & y1 & y2 \\ 1 & 1 & 1 \end{vmatrix}$$

Disjointness and reparameterization to construct the maximal line

Knowing two lines to be collinear, we can then test for disjointness by considering the bounding boxes of each line. If the bounding boxes overlap or share a boundary, then the lines are not disjoint and may be reduced to a maximal representation. Recall from Figure 4–1 that there are five cases that require reduction

a) one line is completely embedded in the other and there are two shared points, i.e., the two lines are the same

b) one line is completely embedded in the other and there are no shared points

c) one line is completely embedded in the other and there is one shared point

d) one line is partly embedded in the other and there are no shared points

e) the two lines only share a boundary at exactly one point

Referring to the parametric line equation (1), we use the t values to determine the minimum and maximum points of the two lines and in this way test for embedding. First, find the parametric equation of the line in question, and then find t for the given points on the line. The minimum and maximum coordinates become the t=0 and t=1 values for the resultant maximal line. Figure 4–4 serves to illustrate these concepts in general while Figure 4–5 shows the test bed data for our maximal line function and a specific worked example.

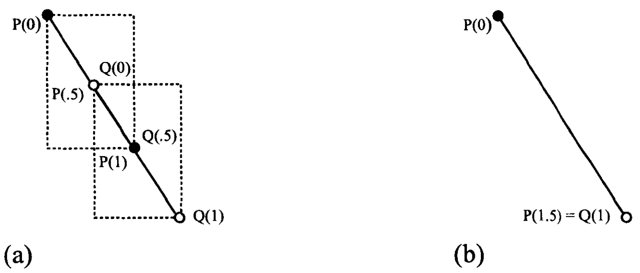
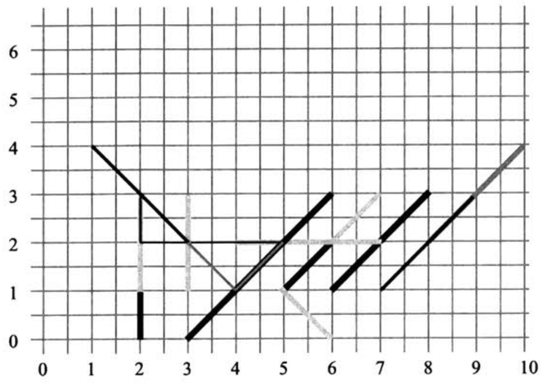


Figure 4–4. Testing disjointness and reduction to a maximal line using the parametric line representation. (a) the overlapping bounding boxes of two collinear lines where P(0) < P(.5), P(.5)=Q(0), P(1) = Q(.5), P(1.5) = Q(1). The minimum value for t is at P0 while the maximum value is at Q1. (b) a reparameterized line which is now maximal.



(a)

line	P0, P1
lxco1	(2,2) , (5.2)
lxco2	(4,2), (7,2)
lxco20	(7,1), (9,3)
lxco21	(8,2), (10,4)
lxco3	(2,1), (2,2)
lxco4	(2,3), (2,2)
lxco9	(2,0), (2,1)
lxco5	(3,1), (3,3)
lxco6	(5,1), (6,2)
lxco7	(6,2), (7,3)
lxc10	(6,0), (5,10)
lxco8	(6,1), 8,3)
lxc11	(3,0), (6,3)
lxc12	(4,1), (5,2)
lxc13	(4,1), (3,0)
lxc14	(4,1), (6,3)
lxc15	(4,1), (3,2)
lxc16	(4,1), (1,4)

(b)

Worked Example

Given

P(t) = P0 + t(P1-P0)

lxco20 is defined by P0P1 where

P0 = (7,1) and P1 = (9,3)

lxco21 is defined by P2P3 where

P2 = (8,2) and P3 = (10,4)

1) find the parametric equation for the line

x = 7 + t(9-7)

y = 1 + t(3-1)

2) substitute in the relevant x and y values and solve for t

P2x => 8 = 7 + 2t => t = 0.5

P2y => 2 = 1 + 2t => t = 0.5

P3x => 10 = 7 + 2t => t = 1.5

P3y => 4 = 1 + 2t => t = 1.5

3) order the points according to t values

P0(0), P2(0.5), P1(1), P3(1.5)

4) keep the max and min values to define the maximal line such that MaxLin x = (7,1), y = (10,4)

(c)

Figure 4–5. Data test bed for maximal lines. (a) visualization (b) data (c) worked example. Line lxco21 is partially embedded in lxco20. Upon completion of the maximal line procedure, lxco20 is reparameterized to the full length of the two lines and lxco21 is discarded.

Each time a shape is drawn on the canvas, the maximal lines must be re–calculated.

Registration points

Recall that a shape grammar consists of a design state; call it the canvas, C, and a set of shape rules in the form A -> B. We can apply the rule if there is some transformation of the shape A, $\tau(A)$, within C. Where $\tau(A)$ occurs, we can replace it with $\tau(B)$. Informally, we can articulate this as see a shape, erase it and draw a new one. Formally, these are the operations recognize shape, shape difference and shape union. Specifically

C, the canvas

A -> B, the shape rule

$\tau(A) <= C$, we see the shape, A, within the Canvas, C

$(C - \tau(A)) + \tau(B)$, we erase the shape A and draw the shape B

In order to carry out these steps in a computer implementation, an additional device assists us: registration points. Registration points can provide more specificity about how we can recognize a shape and apply a rule [57]. To find the registration points both of a shape rule and shapes on the canvas, we must extend all maximal lines in search of their intersection points with other maximal lines.

Again using the ability to reparameterize a line, we extend the maximal lines by some constant, k, representing the distance that the line should be extended. Specifically we restate the line equation as,

$$P(u) = P0 + u(P1-P0) \text{ for } (0 - k) <= u <= (1 + k) \qquad (4)$$

The value of k should be set in relation to the canvas size.

In order to find the intersection of two extended lines, we again use the notion of overlapping bounding boxes. If two bounding boxes overlap, the lines may intersect within that overlapping bound. To determine more precisely where this occurs, we then divide each line in half, i.e. find P(.5) and Q(.5), and examine the four resulting lines and their bounding boxes. The cases that do not result in an overlap can be discarded, as can any overlapping cases that contain disjoint lines. The remaining case contains the intersection and we continue dividing and checking recursively until we find an intersection within some acceptable tolerance, epsilon. Figure 4–6 illustrates the general concept of the divide–and–conquer algorithm for finding registration points while Figure 4–7 shows a specific application of registration points in shape replacement.

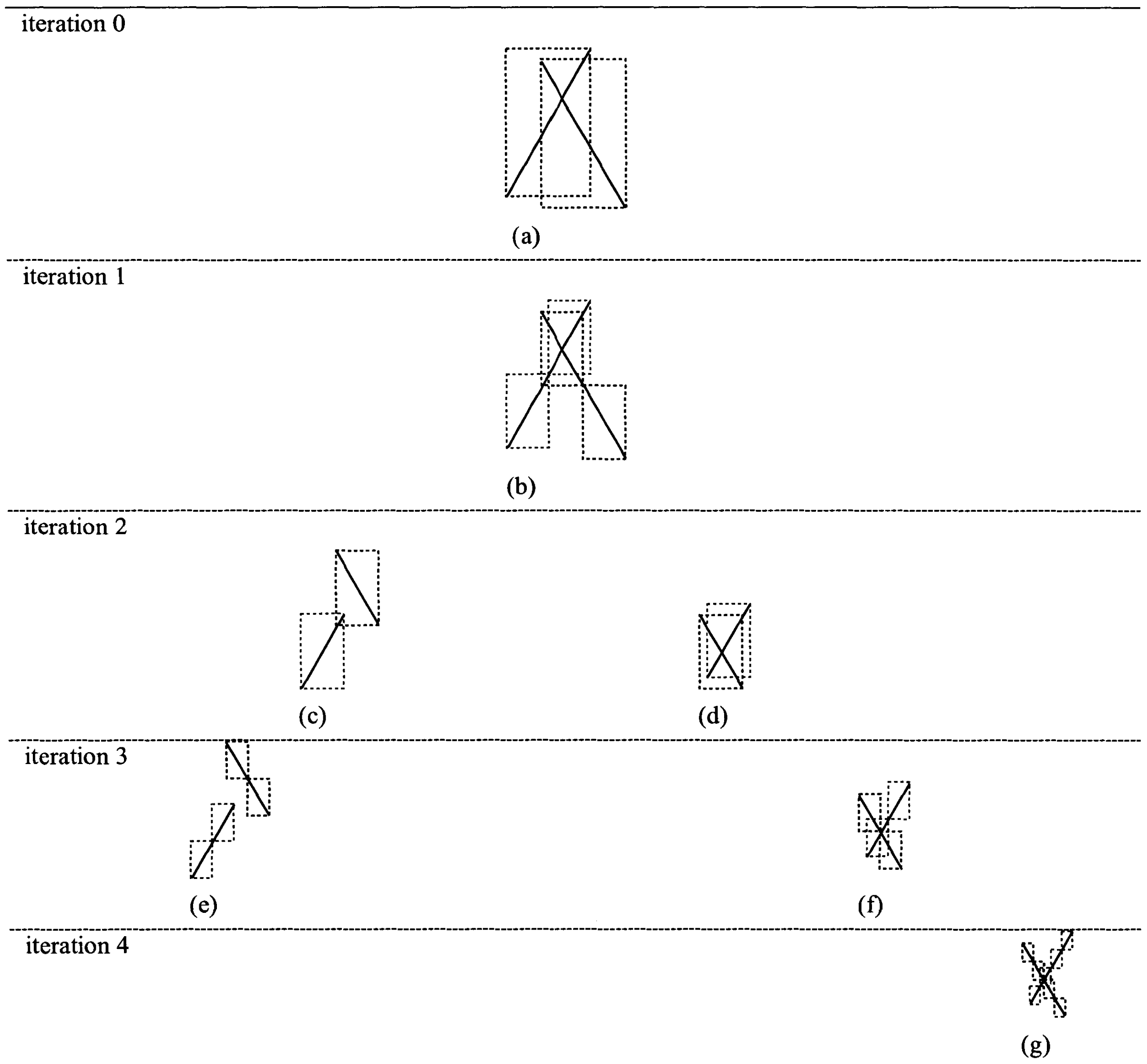


Figure 4–6. Recursive sub–division search for the intersection of two lines. (a) the initial state (b) sub–division and bounding box location. (c), (e) this branch is eventually discarded as it contains disjoint lines. (d), (f) Overlapping bounding boxes with non disjoint lines (g) the search continues until the remaining overlapping bounding boxes are within the tolerance of epsilon, e, in each of x and y. The box becomes very small such that the intersection point may be defined as (x, y) = |x(max)-x(min)| , |y(max)-y(min)|. Note that this is a general procedure that, with minor modifications, will work for curves as well.

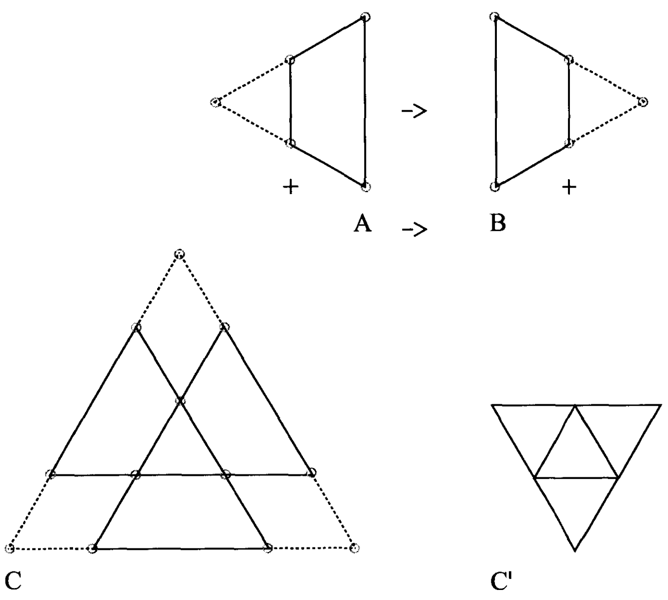


Figure 4–7. Use of registration points to apply shape rule. The dotted lines indicate extensions of the maximal lines while the circles indicate the registration points. The shape rule, A->B, flips a quadrilateral portion of an equilateral triangle about the marker, +. After three applications of the rule, the result is C'

Embedding

Having computed the maximal lines and registration points for all rules and the current design, we are now prepared to recognize both explicit and implicit shapes on the canvas. The parametric line equation allows us to determine the value of t for any registration point on a line where it is embedded. We use the same programming techniques as those for reducing lines to their maximal representation, namely those for collinearity and non disjointness. However, we check for embedding by restricting our search to lines that are completely within another as in cases (b) and (c) of the reduction rules. Specifically, L1 is embedded in L0 when

L1 < L0 where there are no shared points

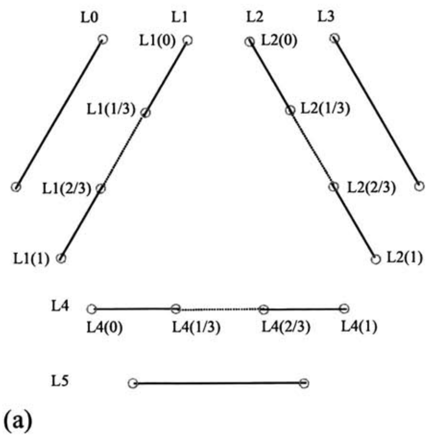
L1 < L0 where there is one shared point

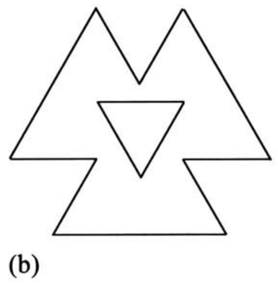
Note that the case of L0 = L1, two shared points, has been removed by prior calculation of maximal lines.

Erasing

Erasing, or shape difference, is another reparameterization manipulation. Once a shape, A, has been successfully recognized on the canvas, we must first erase it as in (C - τ(A)) before we can draw the new shape, B. Using the shapes in Figure 4–8C as an example, we see that the overall design contains six (6) maximal lines, L0 thru L5, as defined in Figure 4–8a. We consider a rule that recognizes an equilateral triangle and rotates it 180 degrees about its center of mass. Such a triangle is embedded in lines L1, L2, and L4. Specifically, the points L1(1/3)=L2(1/3), L1(2/3)=L4(1/3), L4(2/3)=L2(2/3) form a triangle. The middle third of each of L1, L2, and L4 must be erased and then redrawn as in Figure 4–7b. Twelve (12) maximal lines now represent the resulting design.

We used the data in Figure 4–9 to test our procedures for maximal line construction, embedding and erasing and were able to recognize explicit and implicit shapes, draw (perform shape union), and erase (perform shape difference). Specifically, we used a rule to find squares repeatedly and rotate them 45 degrees about center. The methodologies described here also conveniently allow for almost perfect shapes to be recognized, erased and drawn. For example, as the square gets rotated, it loses its perfect 90-degree angles due to rounding errors, but with respect to our eye the angles stay right. In cases such as this, the computer implementation will respect the eye.





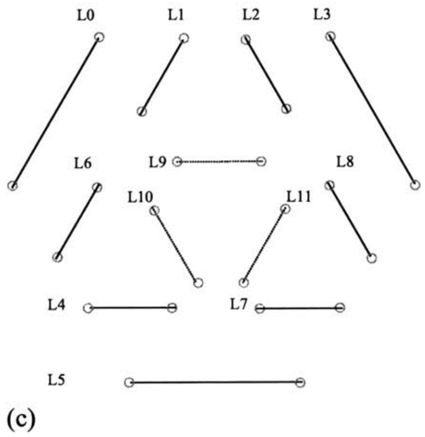
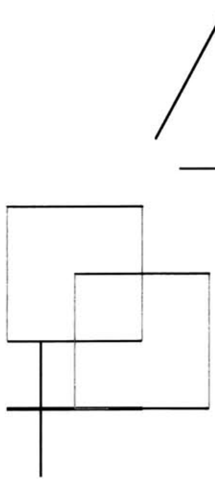


Figure 4–8. Recognizing, erasing and drawing shapes embedded on the canvas. (a) an exploded view of Figure 4–7C shows the maximal line representation of the design before applying the shape rule (b) the result of applying the shape rule that rotates an equilateral triangle about its center of mass (c) the maximal line representation, still in exploded view, of the design after applying the shape rule.



(a)

line	P0, P1
l0	(100, 550) , (200, 550)
l1	(200, 550) , (200, 450)
l2	(200, 450) , (100, 450)
l3	(100, 450) , (100, 550)
l10	(250 , 500) , (250 , 400)
l11	(250 , 400) , (150 , 400)
l12	(150 , 400) , (150 , 500)
l13	(150 , 500) , (250 , 500)
l45	(210 , 600) , (262 , 696)
l67	(290 , 696) , (348 , 600)
l89	(334 , 578) , (228 , 578)
lx0	(100 , 400) , (200 , 400)
lx1	(150 , 400) , (175 , 400)
lx2	(125 , 350) , (125 , 450)

(b)

Figure 4–9. Test bed for shape recognition, draw and erase. This is the dataset before finding all the maximal lines. (a) visualization (b) data points

Curves from Visualization to Representation

As an artist, nearly every mark that I make is a curve. But, making a mark, or a curve, with a pencil is different than with a stylus and tablet connected to a computer. While it is true that any mark that I make is drawn over some time interval, the computer representation of a mark is acutely dependent on this fact. A computer mark, more specifically a curve, will be defined by a series of points that is specifically defined as a function of time. The exact number of points also depends upon the precision desired and the complexity of the curve path. Clearly, a complex curve with high precision will have more points than the same complexity curve with less precision.

Geometric representations for curves are nontrivial as there are many types to choose from each with their advantages and disadvantages. Any representation will be good at solving some problems and less good for solving others. The requirements for a computer implementation of a curvilinear shape grammar where the artist's mark is paramount are:

a) the curve must be easy to draw, yet afford complex shapes

b) curve intersections must be easy to evaluate

c) curve reparameterization must be possible without changing the shape of the curve

The most useful way to represent a curve is as a spline, or piecewise parametric polynomial curve. A variety of spline types exist such as Bézier, B-Spline and Hermite curves [37], [41]. As this section describes below, Béziers are particularly appropriate for representation of the artist's mark.

A Bézier curve is a parametric polynomial function that interpolates its control points. The polynomial can be of any degree. Further, the curve will contain two end points plus a number of control points. Where n is the curve degree, n - 1 is the number of control points such that the total number of points is n + 1. The polygon defined by the n + 1 points is call the convex hull while the line that joins the points (P0 ... Pn) is the control polyline. Typically, the points of the curve are defined over the unit interval, 0 <= t <= 1, such that P0 corresponds to t = 0 and Pn to t = 1. The curve segment is of the form

$$P(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^{i} P_i \ , 0 <= t <= 1 \qquad (5)$$

where n is the degree of the curve, P is the point and t is time. For the linear, first degree form of the curve, we recognize the parametric line formula from (1)

$$P(t) = (1 - t)P0 + tP1$$

Similarly the quadratic form is

$$P(t) = (1 - t)^2P0 + 2(1 - t)P1 + t^2P2 \qquad (6)$$

a visual example of which is in Figure 4–10. The cubic form, visualized in Figure 4–11, is

$$P(t) = (1 - t)^3P0 + 3(1 - t)^2tP1 + 3(1 - t)t^2P2 + t^3P3 \qquad (7)$$

As we can see from the figure drawings, lower–order curves such as the quadratic form, are not particularly flexible as they do not curve much beyond the shape of the letter 'C'. For our purposes the cubic is better able to match the requirements of the artist's mark. Higher–order Béziers are possible, but not considered here because they become more complex to compute. Indeed, cubic Béziers are popular in graphics programs expressly because they give a reasonable compromise between computational issues and smoothness.

Further, cubic Béziers have many properties that we require to represent the artist's mark in shape–grammar computations. Specifically [37], [41] they

a) lie within their convex hull and generally resemble the shape of their control polyline

b) do not require slopes as inputs and can therefore be constructed from data points

c) are invariant under affine parametric transformations such that the reparameterization from the unit interval to any other interval results in the same curve with some exceptions discussed later in this chapter

d) are easy to draw by sampling the time parameter

e) have easily computed intersections

Figure 4–12 illustrates the five (5) types of Bézier curves with their control polylines [54], [60] less the two (2) degenerate cases of a straight line and a point. Importantly, these types hold only for local examination of the curve i.e., for smaller intervals of t. Under global examination, where the t interval becomes rather large, the number of types reduces. These types will become important as we explore their role in shape grammar computer implementations

Although a single cubic Bézier can generate a number of interesting shapes, their real expressive power comes from joining them piecewise. When two curves are joined, they are said to be continuous. The degree of continuity, however, depends upon the conditions at the joint. Here, we are primarily concerned with G0 and G1 continuity. G0 continuity only ensures that the two curves join and says nothing about the smoothness of the two curves at the joint. G1 ensures smoothness across the joint as it has the additional requirement that the curves have a common tangent line at the joint. Examples of G0 and G1 continuity are in Figure 4–18 e, e'. The concept of continuity comes into play when we create piecewise Béziers from hand–drawn data as well as in the method for defining maximal curves. We will look at continuity again in the following sections.

The ability to join cubic Béziers piecewise meets our requirement to draw complex freeform curves. It also affords the ability to approximate shapes such as a circle (Figure 4–13) or an ellipse. This is in addition to the ability to construct n-gons as bezigons – 2D shapes that use Bézier curves for the edges of the shape.

This section has given an overview of a curve representation, the cubic Bézier that has the characteristics required to model a hand–drawn mark. The next section will explore these properties as they relate to the computer implementation of a shape grammar.

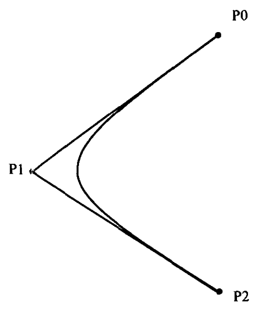


Figure 4–10. Quadratic Bézier curve. P0 and P2 are the end points and P1 is the control point. Note that the curve lies completely within the convex hull formed by the points P0, P1, P2. Also note that the curve is not particularly flexible.

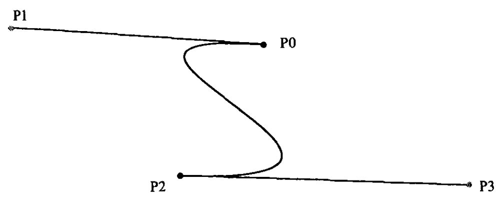


Figure 4–11. Cubic Bézier curve. P0 and P2 are the end points and P1 and P2 are the control points. Note that the curve lies completely within the convex hull formed by the points P0, P1, P2, and P3. Also note that the curve is particularly flexible and well suited for hand–drawn representations.

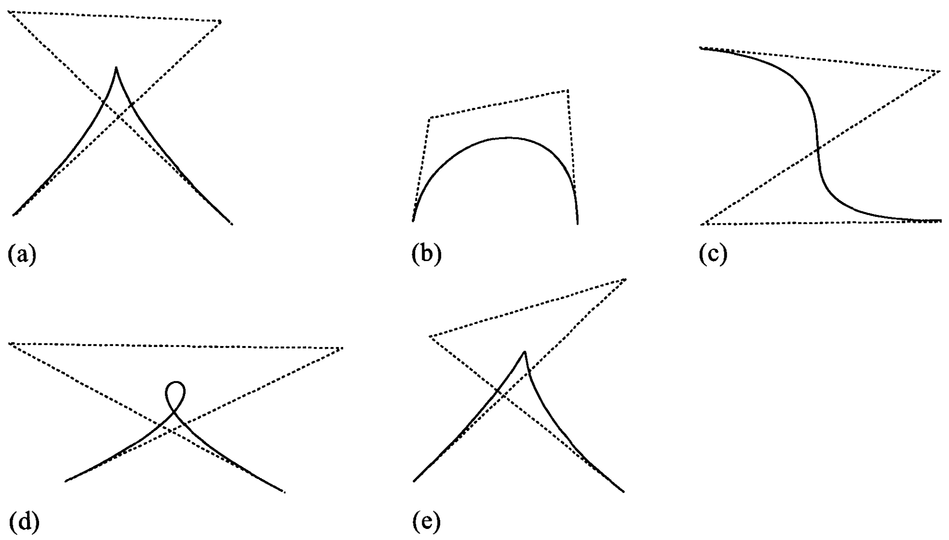


Figure 4–12. Cubic Bézier types. The dotted lines represent the control polyline for each curve. (a) cusp (b) arch (c) single inflection point (d) loop (e) two inflection points, or bump

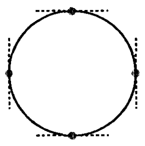


Figure 4–13. Circle approximated by four (4) cubic Béziers. It is not possible to represent a mathematically perfect circle, but the results are visually correct. The gray circles represent the joints between the Bézier pieces and the dotted lines are the tangents across the joints.

Extending the Theory from Lines to Curves

When extending the shape–grammar computer implementation to curves, we consider the same devices required for lines. Specifically this section discusses maximal–curve construction, embedding and finding the registration points on a canvas of curvilinear shapes.

Extending curves and finding registration points

Recall that to find the registration points among rectilinear shapes, we needed to extend the lines. Similarly when searching for registration points among curves, we must extend the curves. Even though we see five (5) types of curves in Figure 4–12, curve extensions reduce this number to four (4) as an arch will always extend to one of the other types. To extend the curves, we use formula (7) and adjust the t interval as desired. For example, using the interval -6 <= t <= 6 extends two arch curves as in Figure 4–14. Note that they degenerate to other curve types.

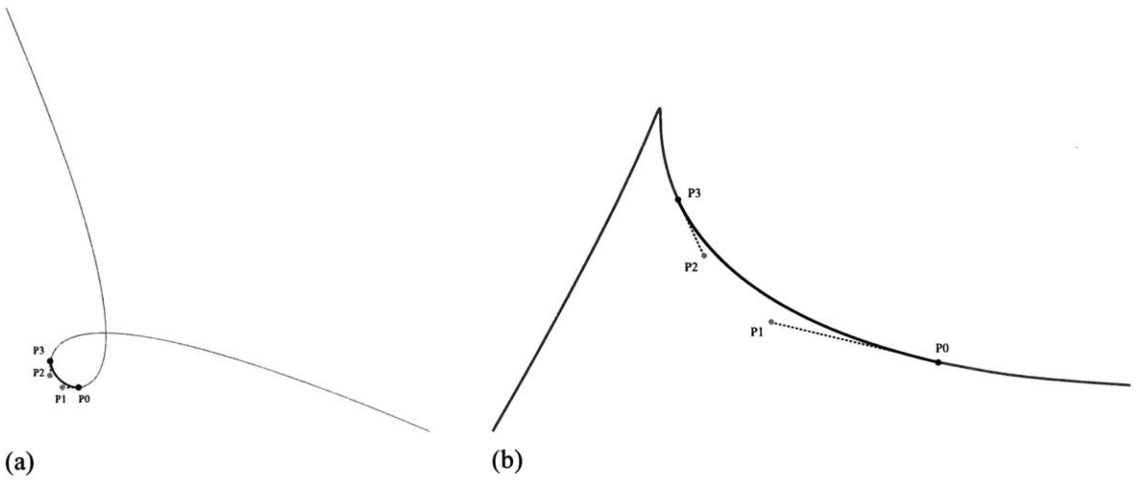
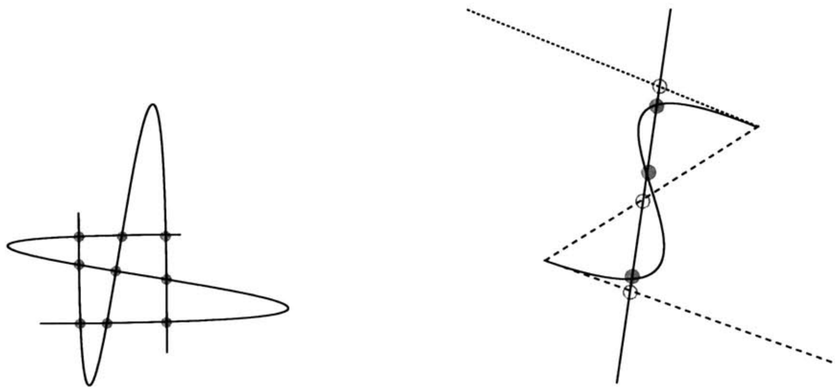


Figure 4–14. Extended Bézier curves. (a) shows an arch, P, which degenerates to a loop. (b) shows an arch, P, which degenerates to a bump.

Now that we can extend the curves, we must calculate the registration points by finding the intersections of the curves on the canvas. Using the sub–division algorithm of de Casteljau [41] we apply the same idea to finding Bézier intersections as we did to finding line intersections. Rather than check the bounding box of the line, however, we check the convex hulls of the curves for overlap. If convex hulls do not overlap, then the curves do not intersect. If the convex hulls do overlap, we check the flatness of the curves within the hull. If the curves approximate a flat line, then we find the intersection using the method for line intersections illustrated in Figure 4–6. If the curves do not approximate a straight line, we continue to recursively subdivide the curves checking for overlapping convex hulls and nearly straight curves.

Two Béziers can possibly intersect in nine (9) places compared to two lines, which can only intersect at one point. Also, the variation diminishing property [38] of cubic Béziers tell us that a line will intersect a Bézier at most the number of times the line intersects the Bézier's control polyline as in Figure 4–15.



(a) (b)

Figure 4–15. Bézier intersections. (a) two Béziers will intersect each other at most nine (9) times as shown by the filled circles. (b) A straight line will intersect a Bézier at most the number of times that it intersections the control polyline shown as the dotted line. The unfilled circles show the three (3) intersections of the line with the control polyline while the filled circles show its intersections with the curve.

Recall, however, that before searching for registration points, the drawing must be reduced to its maximal elements. The next sub–section proposes such a procedure for curves.

Maximal curves and embedding

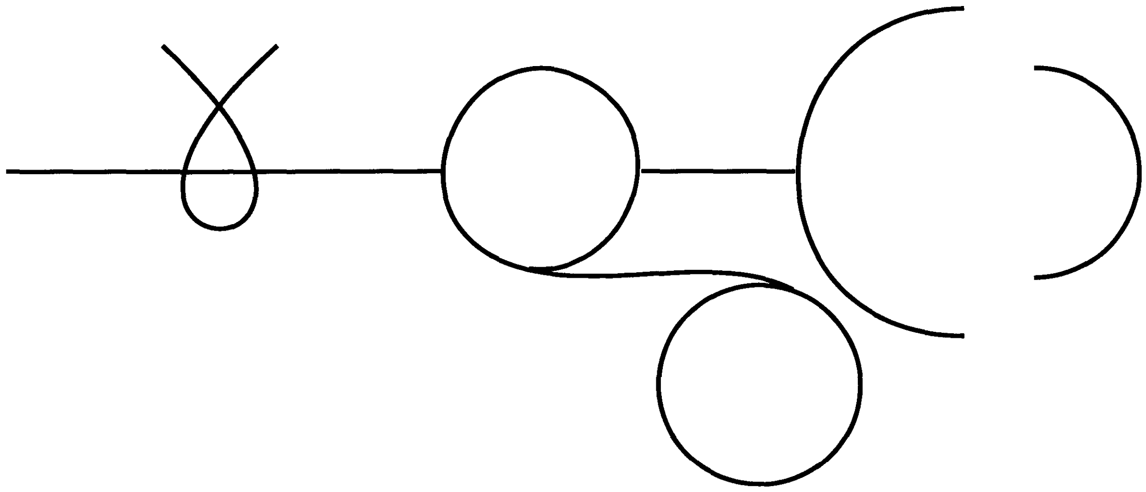


Figure 4–16. Canvas of curves.

Figure 4–16 shows a hypothetical canvas of curves and lines. In order to carry out shape grammar computations, each curve must have a unique representation on the canvas; it must be maximally represented. To solve this problem, we will represent the curves on this canvas as cubic Béziers.

In order to compute maximal lines, we depend on the notions of collinearity and non disjointness to reduce lines to their maximal representation. In the world of curves, continuity is the analog to collinearity and non disjointness from the line world. As stated in the section above, two curves are non disjoint if they have G0 continuity. For our purposes, we will say that two curves have the property of cocurvilinearity if they exhibit G1 continuity. Indeed, if we consider lines as a degenerate case of a cubic Bézier, collinearity meets the requirements for G1 continuity and by default for G0 continuity as well. Referring to the types of Bézier curves in Figure 4–11, maximal curves will always be of one of these types. Complex curves can be achieved, however, by joining these types piecewise. Note that an arch is very likely to be embedded in another curve type as it degenerates to one of the other four types when extended. Given this, Figure 4–16 shows an example drawing containing two maximal curves: a loop and a single inflection.

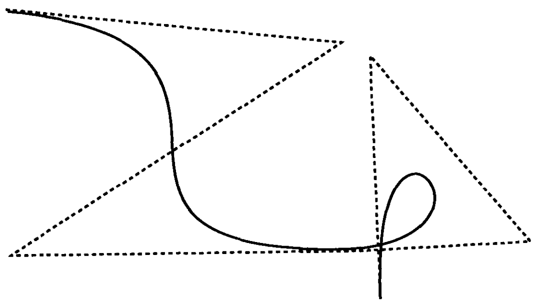


Figure 4–17. Maximal curves. Two Bězier curves, an inflection and a loop, are joined piecewise in a maximal representation. The control polyline is dotted.

Before showing the decomposition of Figure 4–16 into maximal curves, however, it is opportune to restate the reduction rules in terms of the cubic Bězier representation as in Figure 4–18. As with lines, five cases require reduction.

a) one curve is completely embedded in the other and there are two shared points, i.e., the two curves are the same

b) one curve is completely embedded in the other and there are no shared points

c) one curve is completely embedded in the other and there is one shared point

d) one curve is partly embedded in the other and there are no shared points

e) the curves lines only share a boundary at exactly one point

The method for actually reducing two curves to their maximal representation is similar in spirit to that of line reduction. The details, however, are different enough to warrant detailing here and are taken case by case.

Given two curves, P and Q, as in Figure 4–18 and the formula for a cubic Bézier as in (7)

(a) If P0 = Q0 and P3 = Q3 and the control polyline of each curve is identical, then the curves are identical. Discard Q and keep P as the maximal curve. We will call this the test of control polylines.

(b) If Q0 and Q3 are both embedded in P, then divide P into three (3) curves R, S, T from the intervals defined by P0 to Q0, Q0 to Q3 and Q3 to P3, respectively. If R, S or T is identical to Q by the test of control polylines, then discard Q, R, S, and T and keep P as the maximal curve.

(c) If Q0 is embedded in P and Q3=P3, then if P2P3 is collinear with Q2Q3, divide P at Q0 to form two (2) curves R and S. If either R or S is identical to Q by test of control polylines, then discard Q, R, and S and keep P as the maximal curve.

(d) If Q0 is embedded in P and P3 is embedded in Q, divide P at Q0 into curves R, S and divide Q at P3 into curves T, W. If S is identical to T by test of control polyline, then join P and Q and evaluate according to case (e) below.

(e) If P3 = Q0 and there is G1 continuity at the shared point and if at least one of P or Q is of type arch, then join the curves to make curve R. Now compare P with R and Q with R as per each of cases a - d above. If none of the requirements is met, discard R and keep P and Q as maximal curves

To illustrate how each of these tests might fail, again refer to Figure 4–18.

(a') P0=Q0 and P3=Q3, but the control polylines are not identical

(b') Q0 and Q3 are both embedded in P, but none of R, S, or T is identical to Q

(c') Q0 is embedded in P and Q3=P3, but P2P3 is not collinear with Q2Q3

(d') Q0 is embedded in P, but P3 is not embedded in Q

(e') P and Q have G0 continuity, but not G1 continuity

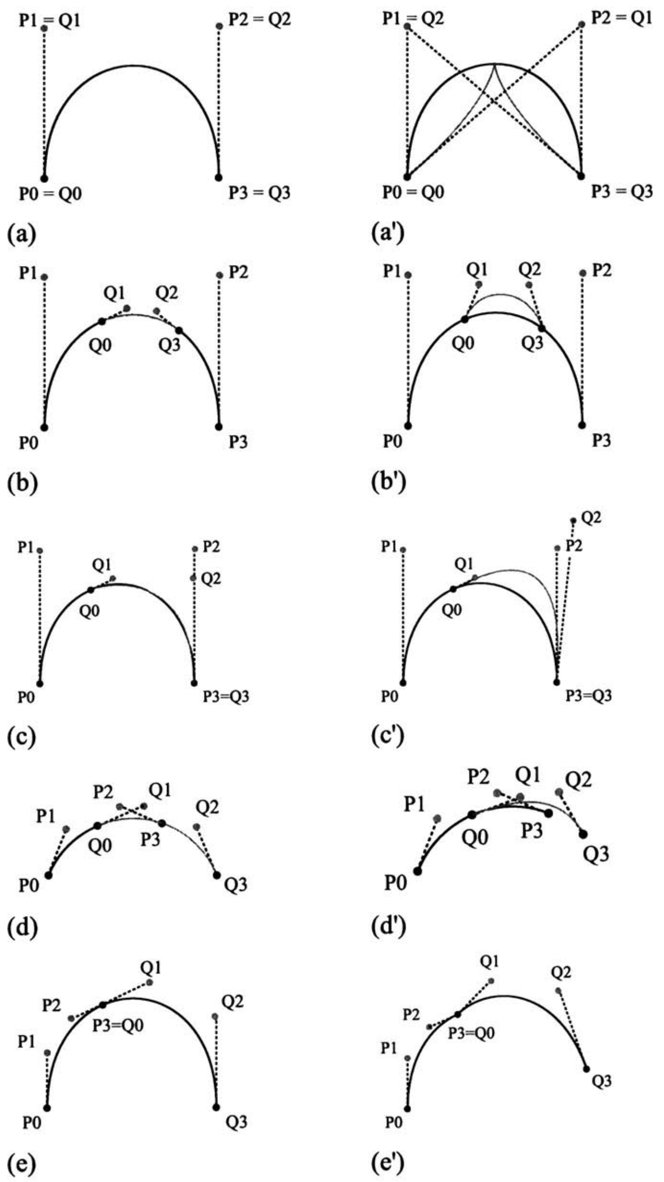


Figure 4–18. Reduction rules for curves. (e) shows G1 continuity between P and Q. (e') shows only G0 continuity between P and Q.

Now we are ready to evaluate Figure 4–16 and decompose it to its maximal curve representation. Figure 4–19 shows the drawing represented as 13 maximal curves. A close up of one area shown in Figure 4–20 reveals two maximal curves, one of type arch and the other of type single inflection point. We know that P and Q are maximal because when we join them to create R, as in Figure 14–20b, neither P nor Q is embedded in R.

To join the curves P and Q to make curve R, we once again use de Casteljau's algorithm. Below we will see an example of finding an embedded curve and how to handle its reduction to a maximal element.

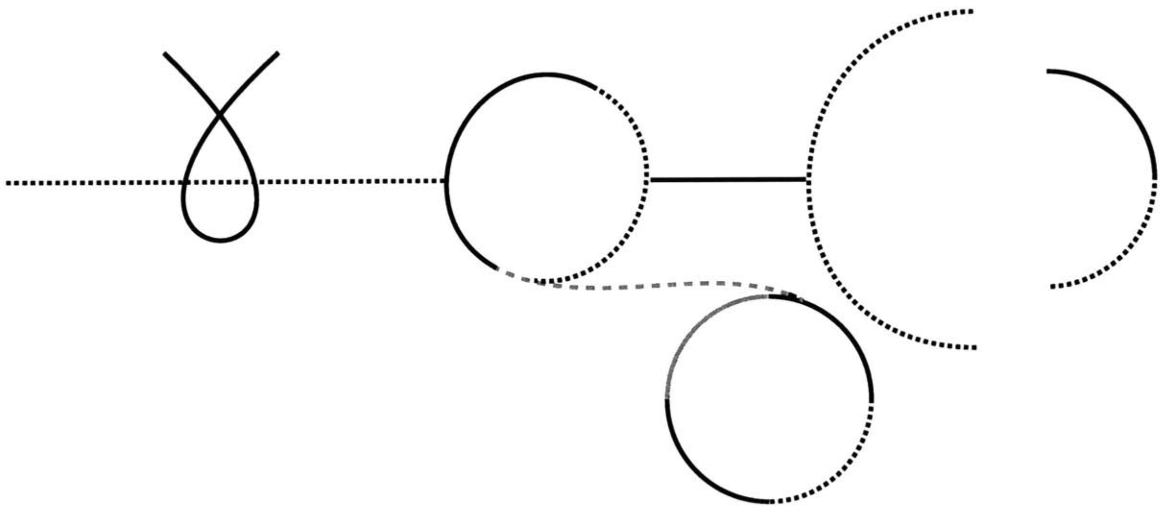


Figure 4–19. A maximal element representation of a curvilinear drawing. Thirteen (13) maximal curves represent the drawing in cubic Bézier form.

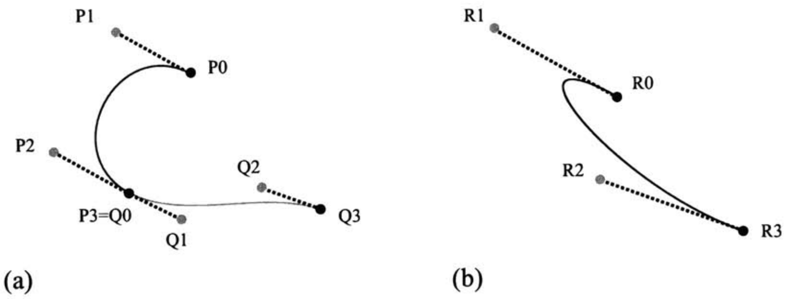
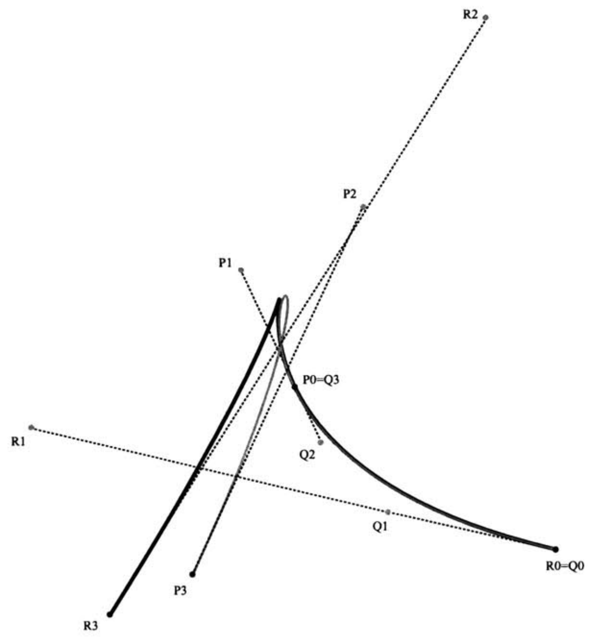


Figure 4–20. Magnification of portion from above drawing. A piecewise Bézier that cannot be further decomposed and is therefore maximal. (a) the two maximal pieces arch and loop (b) a failed attempt at further decomposition

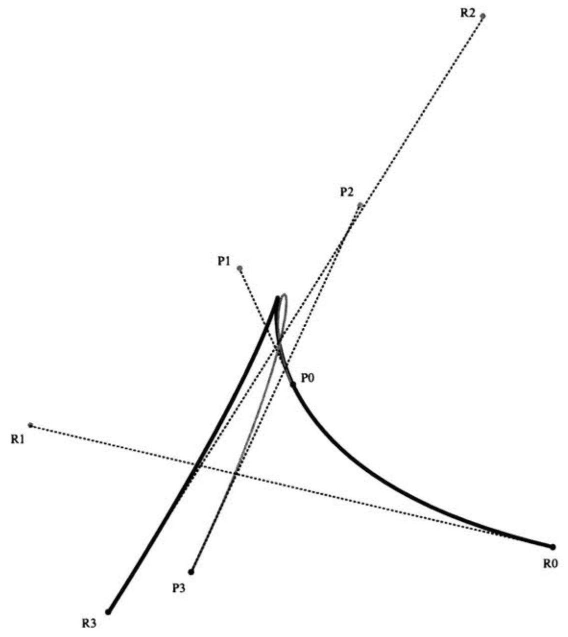
Problem of non-uniqueness

Figure 4–21a illustrates a more complex example of embedding and construction of a maximal curve. We see the arch curve, Q, embedded in the bump curve, R. We also note that the curve P, the loop, exhibits G1 continuity with Q. This scenario presents two options. The first is to accept Q as embedded in R and define P and R as the maximal curves as in 4–21b. The second is to join Q and P into a reparameterized P and to subtract Q from R, causing R to be reparameterized as well as in 4–21c. In order to maintain the uniqueness required by reduction to maximal elements, one of the options must be selected over the other.

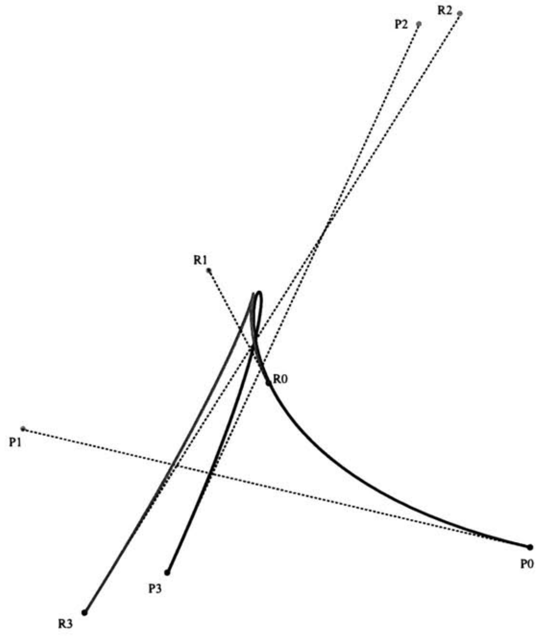
Other interesting cases arise when a curve R is only partially embedded over the G1 continuous joint of curves P and Q. In such cases, the amount of curve R that is not embedded may be considered trivially small so as to be discarded. But, this judgment depends on the intent of the overall design and should be handled on a case–by–case basis.



(a)



(b)



(c)

Figure 4–21. Curve embedding. Q is the arch, P is the loop, and R is the bump. (a) Q is embedded in R. Q and P are G1 continuous. To maintain uniqueness either option (b) or (c) must be selected. (b) Q is embedded in R so delete Q. P and R are now maximal curves. (c) Q and P are G1 continuous. Divide Q from R thus reparameterizing R. Join Q with P. P and R are now maximal curves.

Shape recognition, drawing and erasing

Now that we have detailed the methods for computing maximal curves, handling embedding and finding registration points on a canvas of curves, we can attempt to find implicit and explicit shapes.

Returning to the canvas of curvilinear shapes, we will recognize an implicit shape in C, erase it, $C-\tau(A)$, and draw a new shape, B. Figure 4–22 offers a close–up view of the drawing and the shape rule in question.

Shape recognition starts much the same way for curves as for lines. For curves, however, there are two phases to the recognition. The first is only concerned with the endpoints of shape A. Once the spatial relation of the points is found on the canvas, the second phase of recognition begins. If any of the points on the canvas is embedded in a curve at other than the end points, i.e., if the point is a registration point, then the curve must be divided using de Casteljau's algorithm. Clearly, a registration point is embedded in at least two curves. At this point, recognition shifts to the second phase. We compare the control polylines of the shape rule with those of the curves found on the canvas. If the shapes of the polylines are similar, then a shape is recognized. Now, A may be erased and B drawn. Erasing of curvilinear shapes is handled in much the same way as the case for lines. Once a shape is recognized, the maximal curve needs to be reparameterized in the usual way.

In the example of Figure 4–22, P is divided to make P, R, S, T and W. Then, the control polyline of the piecewise shape A, represented by curves A and A', is compared to the control polyline of each of the newly divided curves. The polyline of A is recognized as similar to that of S joined with T. A is erased and B is drawn.

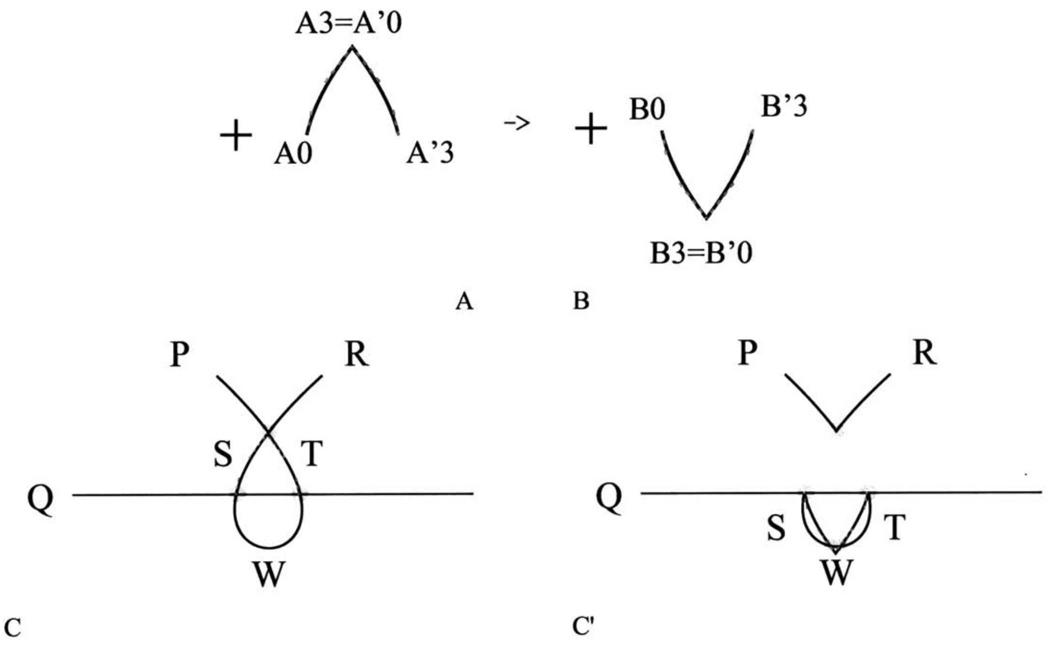
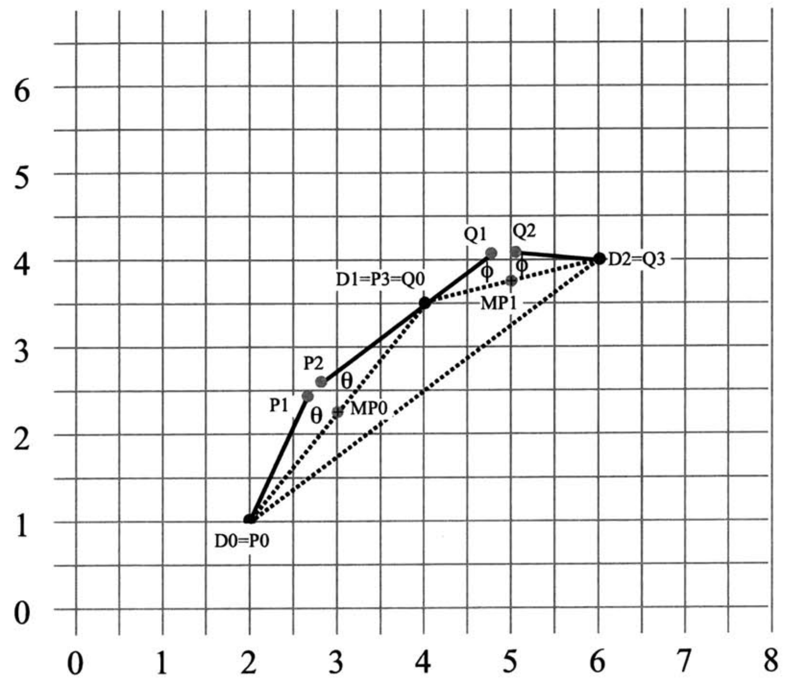


Figure 4–22. Curvilinear shape recognition, erasing and drawing. The gray circles indicate registration points. (A) the left hand side of the shape rule defined by two maximal curves (B) the right hand side of the shape rule defined by two maximal curves (C) the canvas before applying the rule (C') the canvas after applying the rule.

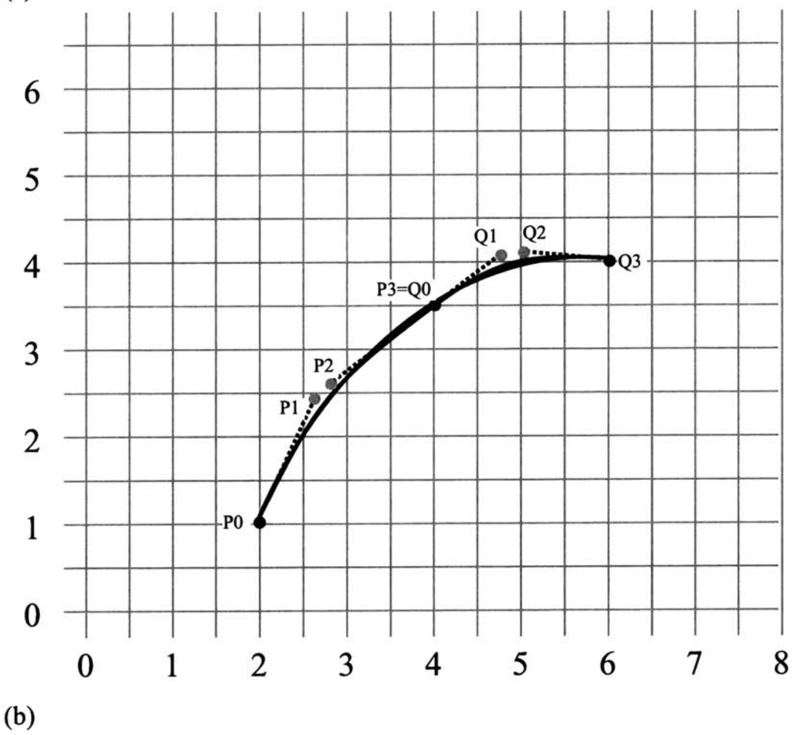
Translating Hand–Drawn Data to Piecewise Cubic Bézier

In order for the artist's mark to be represented in a form that will allow its computation as described above, we must translate the series of data points that make up a stroke into some usable form. Since the piecewise Bézier has many properties that respect the gesture of a mark, we take our hand–drawn data and convert it to a piecewise curve. Before doing this however, it is wise to reduce the amount of data that hand–drawn strokes generate. We consider the connection of all the data points of a stroke to be a polyline. Then, we cull the data set by simply applying a distance threshold. There are, however, much more sophisticated techniques [9, 29] such as those used in cartographic applications, but they are somewhat overzealous for our application. Once we have a polyline, we must convert it to a piecewise Bézier.

Referring to Figure 4–23, a minimum of three (3) drawn points, D0, D1, D2, is required to covert a polyline to a piecewise Bézier. These points will convert to two pieces of a joined Bézier curve with G1 continuity. The points D0 and D2 define the unit cord, which is then displaced to become the tangent of D1. Then each cord, D0D1 and D2D1, is split at the midpoint and rotated according to the angle defined between the tangent at D1 and the relevant cord, either D2D1 or D0D1. This process continues for all the points in the stroke until a multi–piecewise Bézier is defined as illustrated in Figure 4–24. Now the piecewise curve is ready to be redrawn in terms of its maximal elements.



(a)



(b)

Figure 4–23. Constructing a piecewise Bézier from hand–drawn data. (a) D0, D1 and D2 are the given hand–drawn points. P1, P2, Q1 and Q2 are constructed. (b) the final piecewise representation

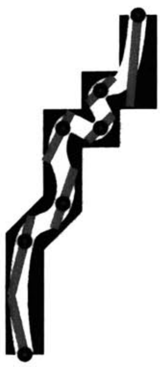


Figure 4–24. Raw hand–drawn stroke data and its conversion to a piecewise Bézier. The black polyline is the original data; the white piecewise Bézier is the curve representation before maximization. Tangents to the joints of the pieces are shown in grey and circles mark the joints themselves.

Computer Implementation Environment

Choosing an implementation environment for representing a shape grammar is not an easy task. The fundamental choices are between imperative and declarative languages. Many imperative languages exist to choose from, each replete with tempting graphics libraries. Some examples are Java, C, C++, etc. For declarative languages, the list is much shorter and includes Prolog, Lisp and SQL. With imperative languages, instructions are in list form and must execute in a particular order. This one statement at a time model means that each statement is having an effect on the changing content at a memory location. In contrast, declarative languages describe a set of conditions and let the system figure out how to fulfill them using a general mechanism like logical unification [39]. This model is based on stating the relationship between inputs and outputs and essentially is performing pattern matching. When considering the artist's mark, we must be able to express abstractions because the drawing is the computation [58]. This implies that declarative methods will be the most successful. Even in declarative languages, however, the mechanisms are symbolic and not visual. The description of the visual world of the artist will always lose something in translation to the symbolic realm. Also worth noting, imperative methods will always be needed for pesky things like graphics library management and user interfaces.

The implementation experiments described in this chapter were carried out in SWI–Prolog running over X11 on a Macintosh laptop with 1.33 GHz processor and 256Mb RAM. Visualizations were confirmed in the Adobe PostScript description language.

In this chapter we have shown techniques for representing a shape grammar in a computer implementation. For the rectilinear case we show methods that have been tested on representative data. Based on understanding of the rectilinear case and motivated by the specific grammar of Chapter 3, a curvilinear method is proposed and implementation "sketches" demonstrated.

Chapter 5 Conclusions and Future Work

The primary motivation for this work was to produce insights in the computational analysis and synthesis of a specific artistic body of work that is still evolving stylistically. The secondary motivation was to outline a software roadmap for continued work in the integration of shape grammar theory with computer graphics practice. As my research evolved, it became clear that the above motivations could be classified as two possibly incompatible concerns: shape visualization and geometric representation.

From the perspective of the artist, visualization and representation will always be one and the same. When working in the analog domain of the pencil, the artist conceives the mark and can quickly represent it on the canvas with a gesture. With similar ease, the artist can recognize shape on the canvas and transform it. The problem space is fluid and the solution path is immediate, intimate and continuous. No need exists to grapple with intermediaries such as symbolic mathematical specifications.

When working in the digital domain, however, the problem space remains fluid, but the solution process is not as spontaneous. This is particularly the case when the artist chooses to employ algorithmic techniques, or rule–based systems, in their work. Computations that are produced via programming techniques require a separation of the visualization and representation processes. The notion of visualization remains largely the same, but representation becomes symbolic and in so doing erodes the immediacy—the continuity—of the mark making process. The text of Chapter 4 makes this point quite dramatically. For every stroke that the artist desires to place on the canvas, there are numerous ways to represent it mathematically. And once represented, the mark still needs to go through a number of conditioning steps to make it easier to compute with. Essentially, the initial, immediate symbolic representation is insufficiently elegant to work with from a mathematical perspective and must be further massaged. Take, for example, the process for translating the gestural hand–drawn stoke into a piecewise Bézier. This type of transformation activity would never concern the artist during the creative process. Indeed, imagine for an instance that any artist's mark, no matter how crudely drawn, lacks elegance the moment it touches the canvas. The notion is absurd and is good motivation to not burden the artist with issues of symbolic representation. Yet, the requirement persists at the behest of the computer's discrete approach to problem solving and inevitably the computer artist risks being similarly constrained.

An additional objective of the research, then, became the idea of bridging visualization and representation issues for a specific body of work when confronted with the requirement to complete a computer implementation. Motivated by the primary objective of the research, I defined a shape grammar largely outside the confines of a computer implementation. The analysis required to define the grammar instigated a more complete understanding of my artistic process and using the grammar as a synthesis tool helped in refining the very style that I was analyzing. Essentially, the activity of achieving formal algorithmic understanding required me to look more objectively and critically at what I was already drawing and painting. Armed with the understanding afforded by the grammar, discoveries on the implementation side ensued. Specifically, the research meets its secondary objective by demonstrating incremental improvements to existing rectilinear methods and further contributes to the field in the area of curvilinear methods by highlighting issues and proposing possible solutions given a specific representation approach.

Future Work

The future of this research could take a number of directions. The text of Chapter 4 gives insights into the challenges that await in developing a 2D drawing system to support curvilinear shape grammar techniques. Extensions to that system would include color usage and eventually 3D representation. Ultimately, however, the way that humans are able to recognize and transform shape can never be fully supported in the current day's computer architectures. In short, the systems would need to be re–built from the ground up.

With respect to user interfaces to support an environment for the immediacy of the artist's mark as a computation device, the playing field is still relatively open and challenging. An exciting research area in sketch and gesture recognition is an excellent place to look for possibilities. The research of Arvo [2] and LaViola [31] is particularly inspiring. Displays are also a critical part of the equation for the computational artist. The notion of the display as substrate is not too distant in the future as early work in the area of paper–like displays demonstrates [19].

In summary, the artist's ability to define computational devices in the analog domain that support their aesthetic understanding can only lead to their increased awareness and abilities. And perhaps it will also alter the way they work; that has certainly been the case here. In addition, as artists become more facile with and encouraged by analog computational approaches, the demand for the digital realm to follow will only increase. We have already seen proof of this in the realms as they exist today where creative demands continually push the boundaries of our tools.

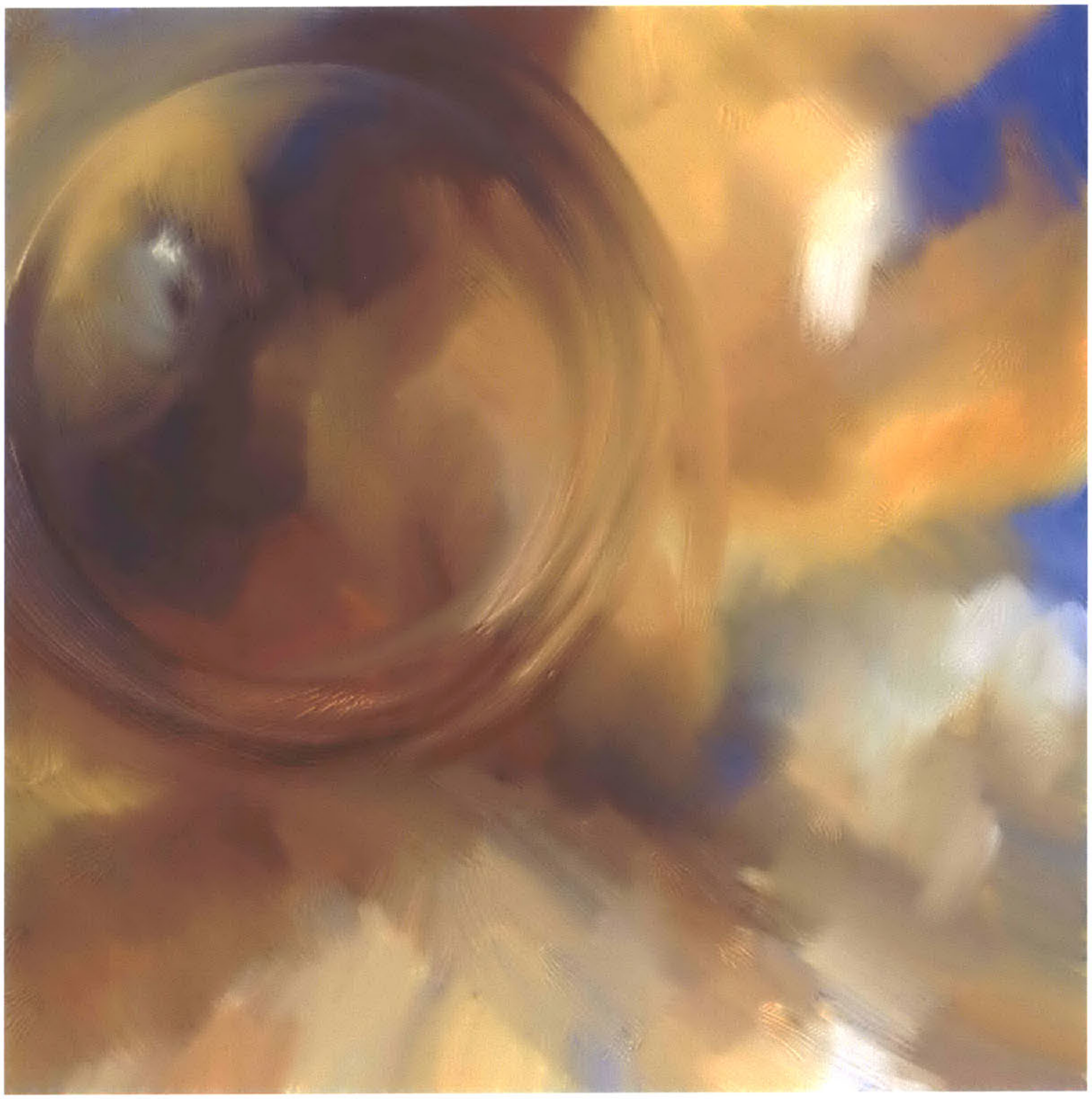


Figure 5–1. Questo non e' un cerchio.

Bibliography

1. ANTONSSON, E.K. and CAGAN, J., *Formal engineering design synthesis.* 2001, Cambridge, UK ; New York: Cambridge University Press. xxv, 470 p.

2. ARVO, J. and NOVINS, K. *Appearance-preserving manipulation of hand-drawn graphs.* in *Computer graphics and interactive techniques in Australasia and South East Asia Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia.* 2005. Dunedin, New Zealand: ACM.

3. BENTLEY, P. and CORNE, D., *Creative evolutionary systems.* 2002, San Francisco, CA. San Diego, CA: Morgan Kaufmann ; Academic Press. xxxi,576 p.

4. BURKS, A.W., *Essays on cellular automata.* 1970, Urbana,: University of Illinois Press. xxvi, 375 p.

5. CANDY, L. and EDMONDS, E.A., *Explorations in art and technology.* 2002, New York: Springer. xvi, 304 p.

6. CHASE, S.C., *Modeling designs with shape algebras and formal logic.* 1996. p. xxi, 188 leaves.

7. CHOMSKY, N., *Syntactic structures.* 1957, 's-Gravenhage,: Mouton. 116 p.

8. COHEN, H., *The further exploits of Aaron, Painter*, Stanford Humanities Review (unpublished).

9. DOUGLAS, D.H. and PEUCKER, T.K., *Algorithms for the reduction of the number of points required to represent a digitised line or its caricature.* The Canadian Cartographer 1973. **10**(2): p. 112-122.

10. DUARTE, J.P. and MASSACHUSETTS INSTITUTE OF TECHNOLOGY. DEPT. OF ARCHITECTURE., *Customizing mass housing : a discursive grammar for Siza's Malagueira houses.* 2001. p. 536 p.

11. DYE, D.S., *A grammar of Chinese lattice.* 1937, Cambridge (Mass.),: Harvard University Press. 2 v.

12. EMMER, M., *The Visual mind : art and mathematics.* Leonardo book series. 1993, Cambridge, Mass.: MIT Press. xvii, 274 p.

13. EVANS, T.G., *A heuristic program to solve geometric-analogy problems.* 1963. p. 199 leaves.

14. FRANKE, H.W., *Computer graphics, computer art.* 2nd, rev. and enl. ed. 1985, Berlin ; New York: Springer-Verlag. xii, 177 p.

15. FREEMAN, W.T., TENENBAUM, J.B., and PASZTOR, E.C., *Learning style translation for the lines of a drawing.* ACM Transactions on Graphics (TOG), 2003. **22**(1): p. 33 - 46

16. FU, K.S., *Syntactic methods in pattern recognition.* Mathematics in science and engineering ; v. 112. 1974, New York,: Academic Press. xi, 295 p.

17. GANGNET, M., HERVÉ, J.-C., PUDET, T., and VAN THONG, J.-M. *Incremental computation of planar maps.* in *Proceedings of the 16th annual conference on Computer graphics and interactive techniques.* 1989: ACM SIGGRAPH Computer Graphics.

18. GIPS, J. and STINY, G., *Production systems and grammars: a uniform characterization.* Environment and Planning B: Planning and Design, 1980. **7**: p. 399-408.

19. HENZEN, A., AILENEI , N., DI FIORE, F., VAN REETH, F., and PATTERSON , J. *Sketching with a low-latency electronic ink drawing tablet.* in *Computer graphics and interactive techniques in Australasia and South East Asia Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia.* 2005. Dunedin, New Zealand: ACM.

20. JOWERS, I., PRATS, M., EARL, C., and GARNER, S. *On curves and computation with shapes.* in *Generative CAD Systems Symposium 2004.* 2004. Pittsburgh, Pennsylvania: Carnegie Mellon University.

21. KANEFF, S. *Picture language machines.* in *Proceedings of a conference held at the Australian National University.* 1969. Canberra: Australian National University.

22. KIRSCH, J.L. and KIRSCH, R.A., *The structure of paintings: formal grammars and design.* Environment and Planning B: Planning and Design, 1986. **13** p. 163-176.

23. KIRSCH, J.L. and KIRSCH, R.A., *The Anatomy of Painting Style: Description with Computer Rules.* Leonardo, 1988. **21**(4).

24. KNIGHT, T.W., *Transformations of the De Stijl art: the paintings of Georges Vantongerloo and Fritz Glarner.* Environment and Planning B: Planning and Design, 1989. **16**: p. 51-98.

25. KNIGHT, T.W., *Color grammars: designing with lines and colors.* Environment and Planning B: Planning and Design 1989. **16**: p. 417-449.

26. KNIGHT, T.W., *Transformations in design : a formal approach to stylistic change and innovation in the visual arts.* 1994, Cambridge ; New York: Cambridge University Press. xvii, 258 p.

27. KNUTH, D.E., *METAFONT : the program.* 1986, Reading, Mass.: Addison Wesley Pub. Co. xv, 560 p.

28. KNUTH, D.E., *The METAFONTbook.* 1986, Reading, Mass.: Addison-Wesley. xi, 361 p.

29. KRAAK, M.J. and ORMELING, F., *Cartography : visualization of geospatial data.* 2nd ed. 2003, Harlow, England ; New York: Pearson Education. ix, 205 p., [208] p. of plates.

30. LAUZZANA, R.G. and POCOCK-WILLIAMS, L., *A Rule System for Aesthetic Research in the Visual Arts.* Leonardo, 1988. **21**(4).

31. LAVIOLA JR, J.J. and ZELEZNIK , R.C., *MathPad2: a system for the creation and exploration of mathematical sketches.* ACM Transactions on Graphics (TOG) Special Issue: Proceedings of the 2004 SIGGRAPH Conference, 2004. **23**(3): p. 432-440.

32. LI, A.I.K. and MASSACHUSETTS INSTITUTE OF TECHNOLOGY. DEPT. OF ARCHITECTURE., *A shape grammar for teaching the architectural style of the Yingzao fashi.* 2001. p. 52, [51] p., [38] leaves of plates.

33. LIMING, R.A. and HUDSON, L., *Practical analytic geometry with applications to aircraft.* 1944, New York,: Macmillan. xvi, 277,[249], 273 p.

34. LORAN, E., *Cézanne's Composition: Analysis of His Form with Diagrams and Photographs of his Motifs.* 1970: University of California Press.

35. MCCORDUCK, P., *Aaron's code : meta-art, artificial intelligence, and the work of Harold Cohen.* 1991, New York: W.H. Freeman. xvi, 225 p.

36. MCGILL, M.C. and MASSACHUSETTS INSTITUTE OF TECHNOLOGY. DEPT. OF ARCHITECTURE., *A visual approach for exploring computational design.* 2001. p. 139 p.

37. MORTENSON, M.E., *Geometric modeling*. 1985, New York: Wiley. xvi, 763 p.

38. MORTENSON, M.E., *Computer graphics : an introduction to the mathematics and geometry*. 1989, New York, N.Y.: Industrial Press. x, 381 p.

39. O'KEEFE, R.A., *The craft of Prolog*. Logic programming. 1990, Cambridge, Mass.: MIT Press. xix, 387 p.

40. REICHARDT, J. and INSTITUTE OF CONTEMPORARY ARTS (LONDON ENGLAND), *Cybernetic serendipity; the computer and the arts*. 1969, New York,: Praeger. 101 p.

41. SHIRLEY, P. and ASHIKHMIN, M., *Fundamentals of computer graphics*. [2nd ed. 2005, Wellesley, Mass.: AK Peters. xv, 623 p.

42. SIMON, H.A., *The sciences of the artificial*. 3rd ed. 1996, Cambridge, Mass.: MIT Press. xiv, 231 p.

43. SIMS , K. *Artificial evolution for computer graphics*. in *International Conference on Computer Graphics and Interactive Techniques Proceedings of the 18th annual conference on Computer graphics and interactive techniques*. 1991.

44. SMITH, R.H., *Graphics; or, The art of calculation by drawing lines, applied especially to mechanical engineering, with an atlas of diagrams*. 1889, London and New York,: Longmans, Green, and co. xxi, 257 p.

45. STERLING, L., *The Practice of Prolog*. Logic programming. 1990, Cambridge, Mass.: MIT Press. 312 p.

46. STERLING, L. and SHAPIRO, E.Y., *The art of Prolog : advanced programming techniques*. 2nd ed. Logic programming. 1994, Cambridge, Mass.: MIT Press. xxxix, 509 p.

47. STINY, G., *Pictorial and Formal Aspects of Shapes and Shape Grammars and Aesthetic Systems*. 1975, University of California: Los Angeles.

48. STINY, G., *Pictorial and formal aspects of shape and shape grammars*. 1975, Basel ; Stuttgart: Birkhäuser. xv, 399 p.

49. STINY, G., *Ice-ray: a note on Chinese lattice designs*. Environment and Planning B: Planning and Design, 1977. **4**: p. 89-98.

50. STINY, G., *Shapes are individuals*. Environment and Planning B: Planning and Design, 1982. **9**: p. 359-367.

51. STINY, G., *Weights.* Environment and Planning B: Planning and Design, 1992. **19**: p. 413-430.

52. STINY, G. and GIPS, J. *Shape Grammars and the Generative Specification of Painting and Sculpture.* in *Proceeding of IFIP Congress 71 in Ljubljana and subsequently published in The Best Computer Papers of 1971* 1971: Auerbach 1972.

53. STINY, G. and GIPS, J., *Algorithmic aesthetics : computer models for criticism and design in the arts.* 1978, Berkeley: University of California Press. xi, 220 p.

54. STONE, M.C. and DEROSE, T.D., *A geometric characterization of parametric cubic curves.* ACM Transactions on Graphics (TOG), 1989. **8**(3): p. 147-163.

55. SUTHERLAND, I.E., *Sketchpad, a man-machine graphical communication system.* 1963. p. 176 leaves.

56. SUTHERLAND, I.E., *Sketchpad: a man-machine graphical communication system.* 1963, Lexington, Mass.,: M.I.T. Lincoln Laboratory. xi, 92 p.

57. TAPIA, M., *From Shape to Style. Shape Grammars: Issues in Representation and Computation*, in *Department of Computer Science.* 1996, University of Toronto: Toronto.

58. TAPIA, M., *A visual implementation of a shape grammar system.* Environment and Planning B: Planning and Design, 1999. **26**(1).

59. TODD, S. and LATHAM, W., *Evolutionary art and computers.* 1992, London: Academic Press. 224 p.

60. VINCENT, S., *Fast Detection of the Geometric Form of Two-Dimensional Cubic Bézier Curves.* Journal of Graphics Tools, 2002. **7**(3): p. 43-51.

61. VON NEUMANN, J. and BURKS, A.W., *Theory of self-reproducing automata.* 1966, Urbana,: University of Illinois Press. xix, 388 p.

62. WATT, A.H., *Fundamentals of three-dimensional computer graphics.* 1989, Wokingham, England ; Reading, Mass.: Addison-Wesley. xvi, 430 p., [416] p. of plates.

Appendix A First Formal Grammar and a Derivation

Rules 1 and 2 address the requirement to loosely follow the form of an implied shape where rule 1 creates the exterior shape and rule 2 the interior shape. The various forms of rule 3 provide a system for closing segmented shapes in a simple way, namely the closing of exterior and interior shapes independently (rules 3b, b.2, d). Drawings resulting from these rules more closely resemble the implied, non–drawn shapes. Rule 3c is responsible for joining interior and exterior shapes as part of the closing process. It is this rule that creates shapes that do not resemble the initial implied shapes. Rules 4, 5, 9 and 10 provide depth information and shape embellishment. Rule 8 is very similar to the basic Ice Ray rule [Stiny 19xx] in that it divides a shape into pleasingly balanced smaller shapes. The end of this appendix presents a possible design outcome in figure A–34 using these rules to generate the design. Throughout the rules the marker, , denotes segments of a segmented shape and the preferences for connecting other segments to it. α is the preferred angle of connection while β is the second choice.

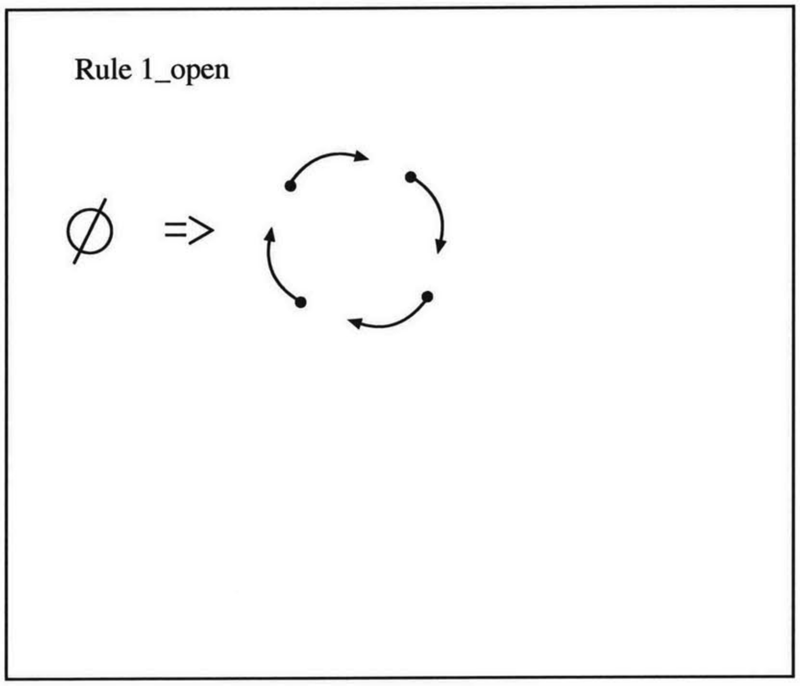


Figure A–19. Rule 1_open replaces the empty shape with a segmented shape based on an implied primitive. In this case, the implied shape is a circle, but may be any other simple shape such a triangle, square, spiral. The dot and arrowhead serve as labels for subsequent replacement rules.

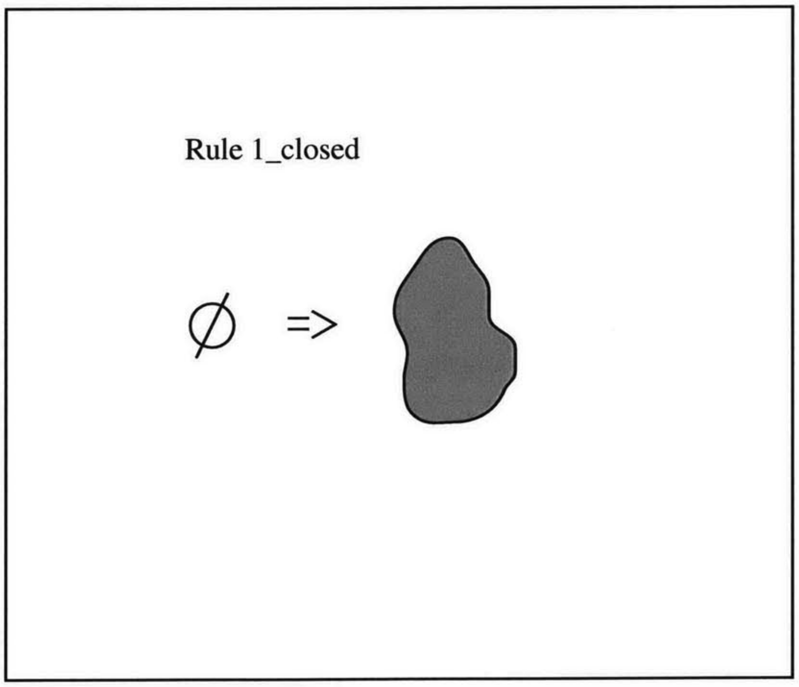


Figure A–20. Rule 1_closed is essentially the same as Rule 1_open with the difference that a closed form replaces the empty shape. In this case the implied replacement shape is a rectangle.

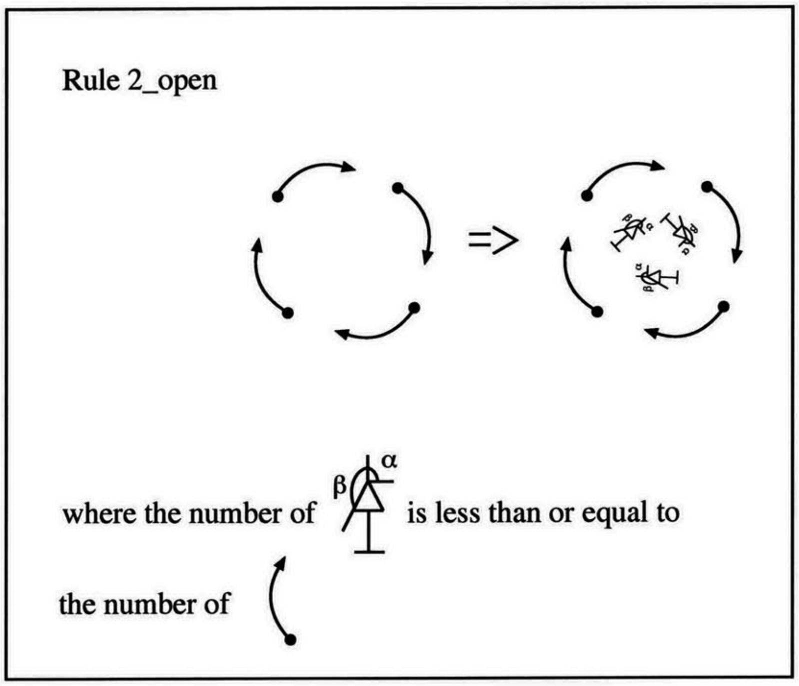


Figure A–21. Rule 2_open replaces a segmented shape with two segmented shapes, one within the other. The dot, dash, angle marker and two arrowhead styles serve as labels for subsequent replacement rules.

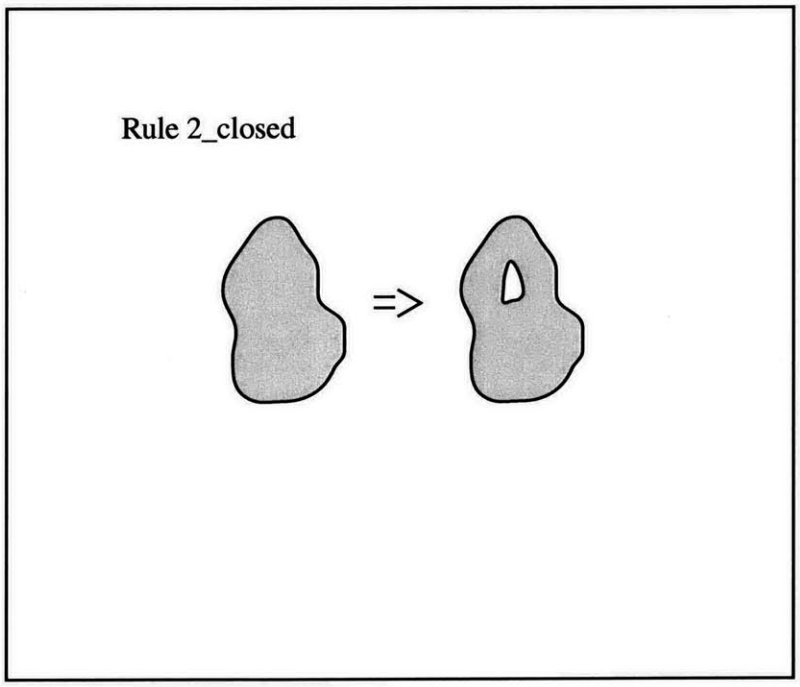
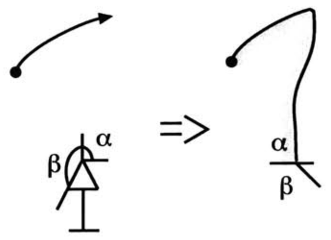


Figure A–22. Rule 2_closed replaces a non–segmented shape with two closed shapes, one within the other. In this case, the outer shape is an implied rectangle while the inner shape is an implied triangle.

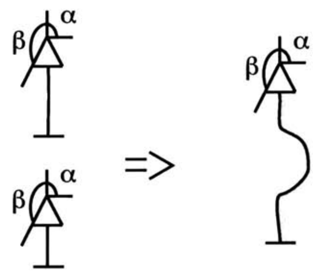
Rule 3a



connect within angle α 1st choice
connect within angle β 2nd choice
without intersecting any other line segment
all must be gone before moving to Rule 3c

Figure A–23. Rule 3a shows the joining rule for two segmented shapes with an inner and outer relationship. Stating preferences for angles ensures that the two forms do not cross over each other in undesirable ways. The shade area represents the interior, or filled area, of the shape as subsequent rule applications occur.

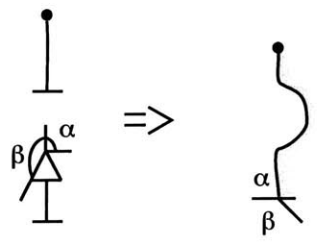
Rule 3b



connect within angle α 1st choice
connect within angle β 2nd choice
without intersecting any other line segment
all must be gone before moving to Rule 3c

Figure A–24. Rule 3b shows the joining rule for segments of an inner shape. The shaded area represents the interior of the shape that will ultimately be closed by repeated rule application.

Rule 3b.2



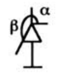
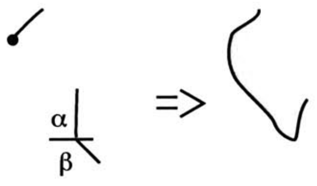
connect within angle α 1st choice
connect within angle β 2nd choice
without intersecting any other line segment
all  must be gone before moving to Rule 3c

Figure A–25. Rule 3b.2 shows the joining rule for segments of an inner shape. The shade area represents the interior of the shape that will ultimately be closed by repeated rule application.

Rule 3c



connect within angle α 1st choice
connect within angle β 2nd choice
without intersecting any other line segment

Figure A–26. Rule 3c shows the joining of tail portions of exterior and interior shape segments. The shaded area represents the inside edge of the form.

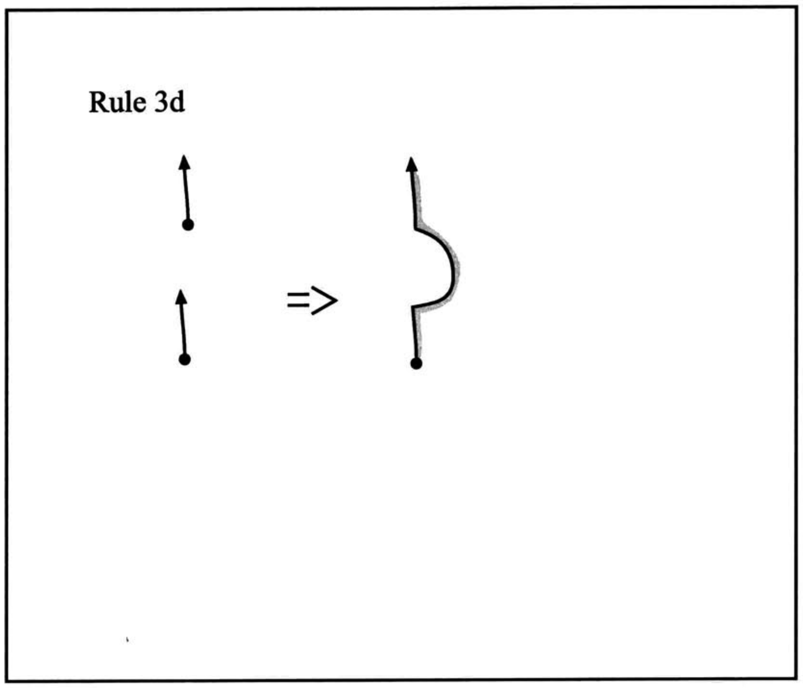


Figure A–27. Rule 3d show the joining rule for segments of an outer shape.

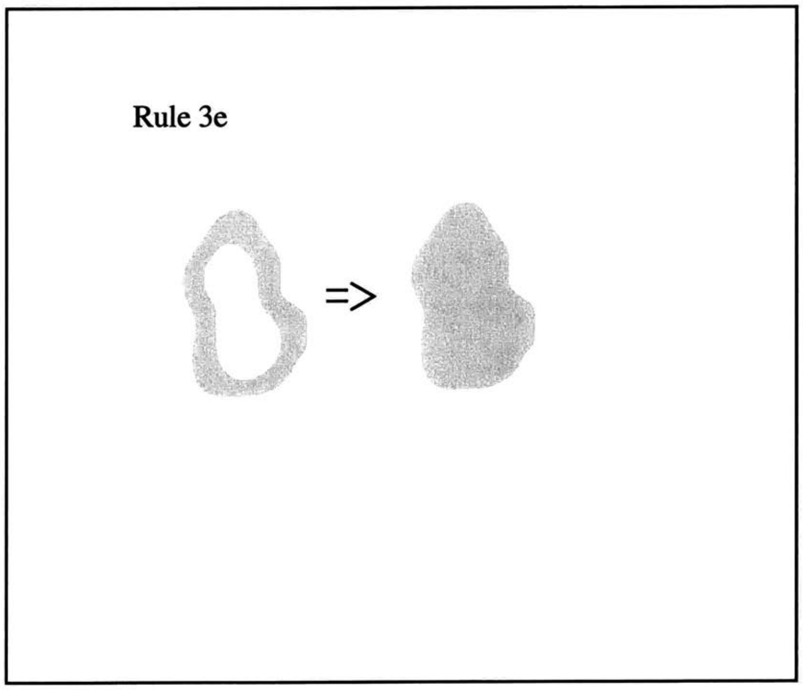


Figure A–28. Rule 3e shows the completion of filling a closed shape.

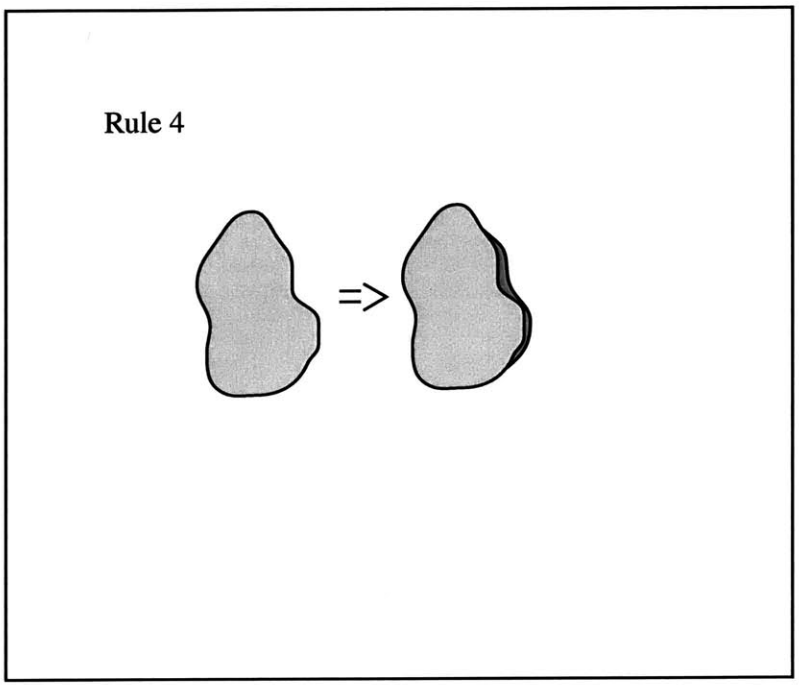


Figure A–29. Rule 4 shows the ability to add a decoration to the exterior of a shape much like a drop shadow effect to create depth to the form.

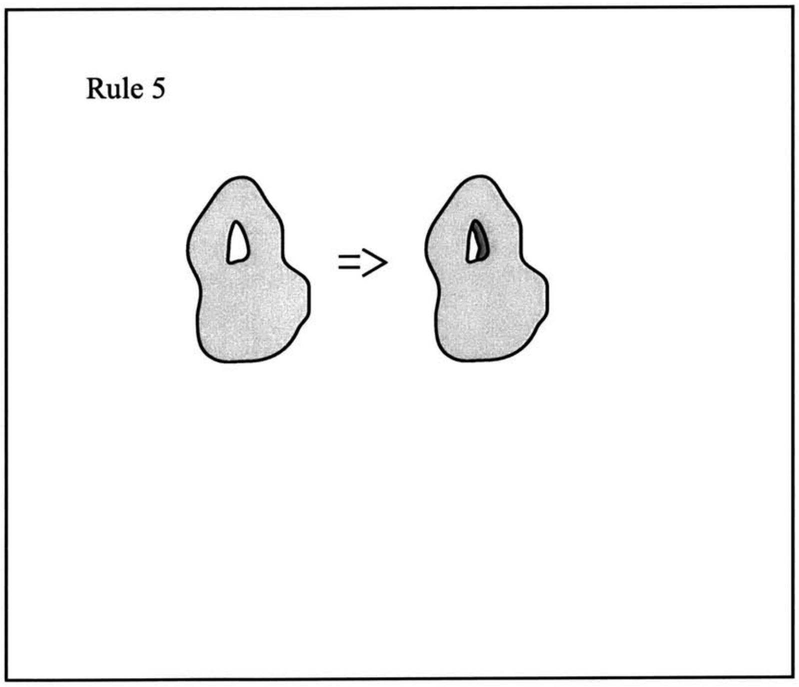


Figure A–30. Rule 5 shows the ability to add a decoration to an interior shape much like a drop shadow effect to create depth to the form.

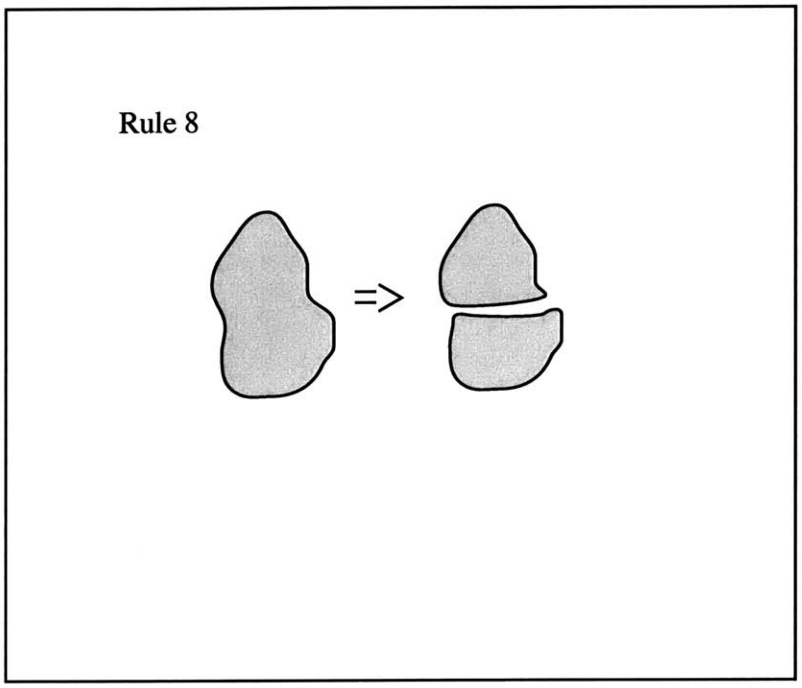


Figure A–31. Rule 8 shows the ability to split a closed shape into two shapes.

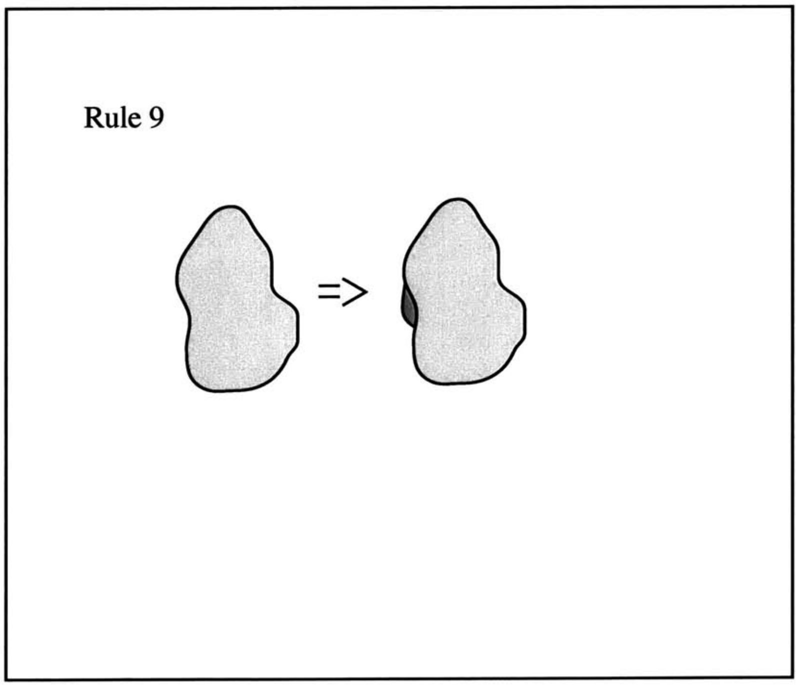


Figure A–32. Rule 9 is a variation of Rule 4 that allows further decoration to the exterior of a closed shape.

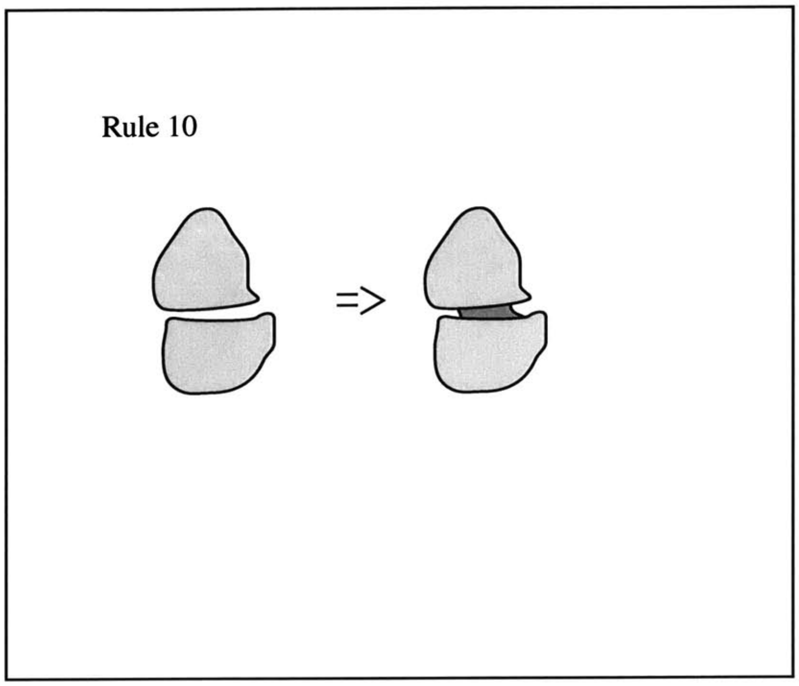


Figure A–33. Rule 10 allows a decoration to be added between two closed shapes.

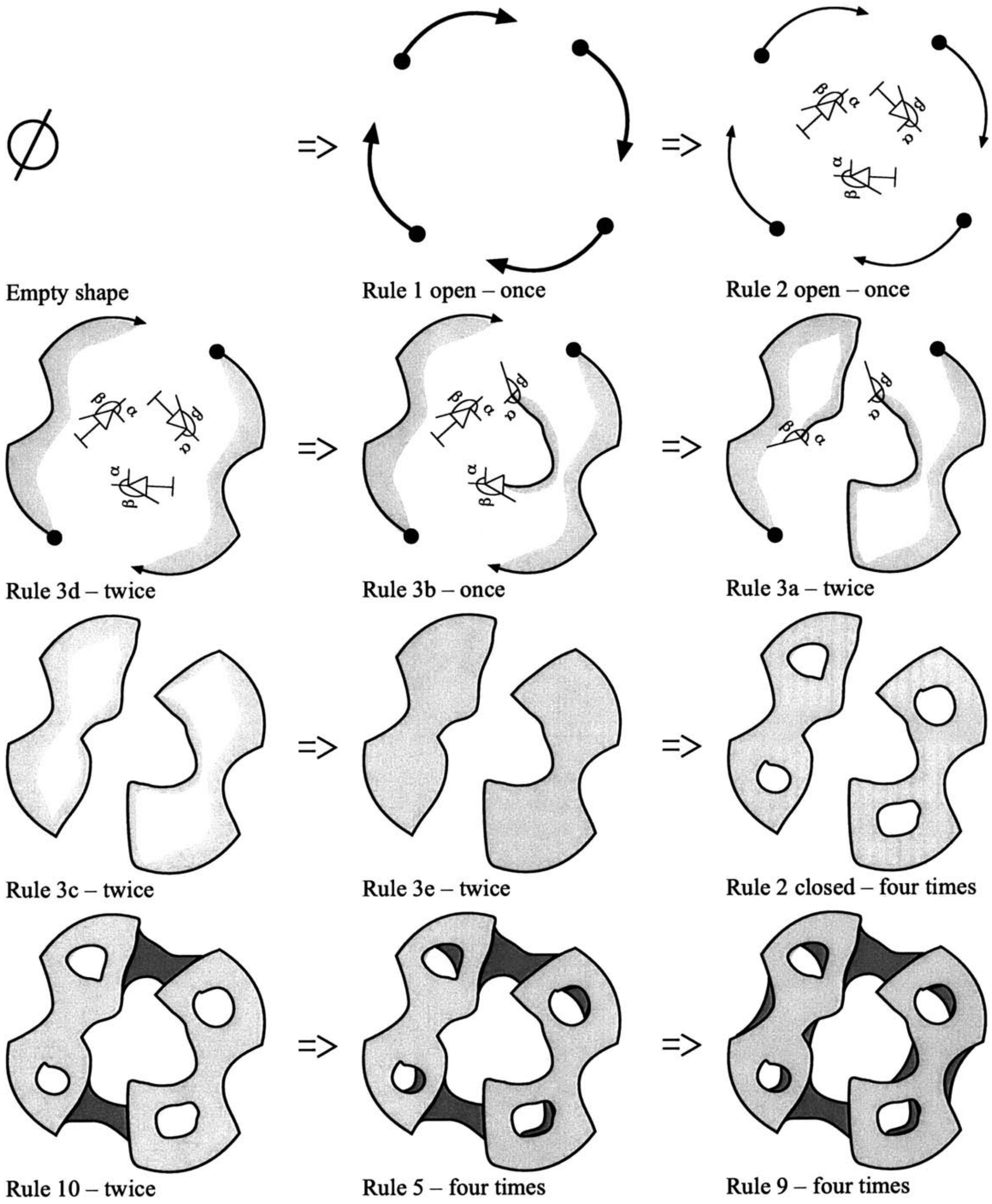


Figure A–34. Derivation of completed shape using initial design language and based on implied circle within a circle.