# Computer-Generated Floral Ornament

*Michael T. Wong* *Douglas E. Zongker* *David H. Salesin*

University of Washington

## Abstract

This paper describes some of the principles of traditional floral ornamental design, and explores ways in which these designs can be created algorithmically. It introduces the idea of "adaptive clip art," which encapsulates the rules for creating a specific ornamental pattern. Adaptive clip art can be used to generate patterns that are tailored to fit a particularly shaped region of the plane. If the region is resized or reshaped, the ornament can be automatically regenerated to fill this new area in an appropriate way. Our ornamental patterns are created in two steps: first, the geometry of the pattern is generated as a set of two-dimensional curves and filled boundaries; second, this geometry is rendered in any number of styles. We demonstrate our approach with a variety of floral ornamental designs.

**CR Categories:** I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.4 [Computer Graphics]: Graphics Utilities—Picture description languages.

**Additional Keywords:** adaptive clip art, conventionalization, pattern generation, plant development, ornamentation, texture generation

## 1 Introduction

> *If I were asked to say what is at once the most important production of Art and the thing most to be longed for, I should answer, A beautiful House; and if I were further asked to name the production next in importance and the thing next to be longed for, I should answer, A beautiful Book. To enjoy good houses and good books in self-respect and decent comfort, seems to me to be the pleasurable end towards which all societies of human beings ought now to struggle.*
>
> — William Morris, 1892 [23]

Ornament is among the oldest forms of human expression, already well developed by the Neolithic Age [6]. Nearly all the commissioned writing of the Middle Ages was decorated with ornament, and the illuminated manuscripts of the 13th century rank among the most beautiful books ever produced.

Even the earliest printed books were often illuminated by hand, but by about 1530 such carefully crafted illumination had all but disappeared [23]. Today, documents are produced with greater ease and in greater number than ever, thanks to ubiquitous desktop publishing tools—yet, beyond the use of static "clip art" elements, these tools provide precious little support for ornamenting the page. Similarly, in architecture, ornament has historically played a critical and famous role. However, most modern buildings, despite the help of sophisticated CAD tools, are largely devoid of these beautiful decorations.

Though technological advances have virtually ignored the creation of ornament, they have at the same time provided new opportunities for its use. The dynamic nature of Web documents encourages ornament to be generated on the fly to accommodate different browser configurations and fonts. New printing processes make it feasible to print on fabric or wallpaper in small runs, raising the possibility of their custom design and production.

This paper therefore provides an early exploration into how aesthetically pleasing ornaments might be generated algorithmically. The method we describe attempts to capture the "essence" of an ornamental pattern, encoding it as a set of rules, which we call *adaptive clip art*. This encoding allows the ornament to be defined in a manner that is independent of a specific areal boundary. The adaptive clip art so described can be used to generate ornaments that are automatically tailored to any particular re-

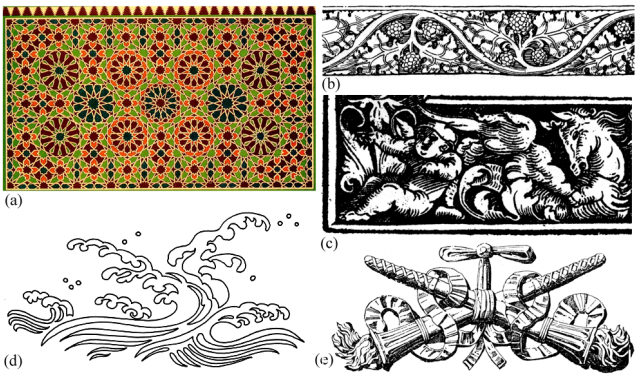



**Figure 1** Design element categories. (a) Geometric forms (after Alhambra tile) [29, plate 29]. Natural forms (b) plants (Gothic vine) [10, fig. 82], (c) animal/human forms (border detail, Germany 1518) [4, plate 30], (d) physiographic forms (17th century Japanese wave motif) [9], (e) artificial objects (Renaissance torches) [21, plate 80].

gion of interest; moreover, if the region is changed, the ornament can be regenerated to fill the new area in an appropriate way.

The automatic creation of aesthetically pleasing ornament is a monumental challenge, which we by no means claim to solve here. Nevertheless, we hope that this paper will offer some interesting new directions, with the hope that further advances may someday help in the creation of beautiful ornaments for our houses and books—and online chat rooms and web pages!

### 1.1 Problem statement

The problem space of all possible ornamental design is simply enormous. In order to approach the problem at all, we need to limit our domain. We therefore make the following taxonomy (adapted from Meyer [21]).

First, the *elements* of ornamental design can be broken down into three broad categories:

1. *geometrical elements*, such as lines, polygons, ovals, and the like (Figure 1a);
2. *natural forms*, which can be further classified as
    1. plants (Figure 1b),
    2. animal/human forms (Figure 1c),
    3. physiographic features (Figure 1d); and
3. *artificial objects*, such as shields, ribbons, or torches (Figure 1e).

Second, for our purposes we will similarly divide the *applications* of ornament into four main contexts:

A. to *bands*, which have finite thickness in one dimension and are infinitely repeating in the other (Figure 2a);
B. to *half-open borders*, which are tightly constrained along one or more edges, but open in other directions (Figure 2b);
C. to *panels*, which are arbitrary bounded regions of the plane (Figure 2c); and
D. to the *open plane*, in which the ornament typically becomes a repeating pattern (Figure 2d).

In this paper, we restrict the problem space to the case of producing floral growth within panels (case 2.1-C in the classification above). In particular, we will look at the challenging issues of structuring floral ornament according to various principles of ornamental design, such as balance, analogy, and intention—as described in Section 2. We will not, however, focus here on designs involving strict symmetries. As we shall see, the resulting design space is still quite large; however, it is at least constrained enough that we can explore a series of related approaches within the confines of a single research paper. Moreover, we expect that many of the approaches suggested here will be useful, in some form, for other cases in the taxonomy.

In the rest of this paper, we describe a number of principles of floral ornamental design, and we discuss ways in which such designs can be created algorithmically.

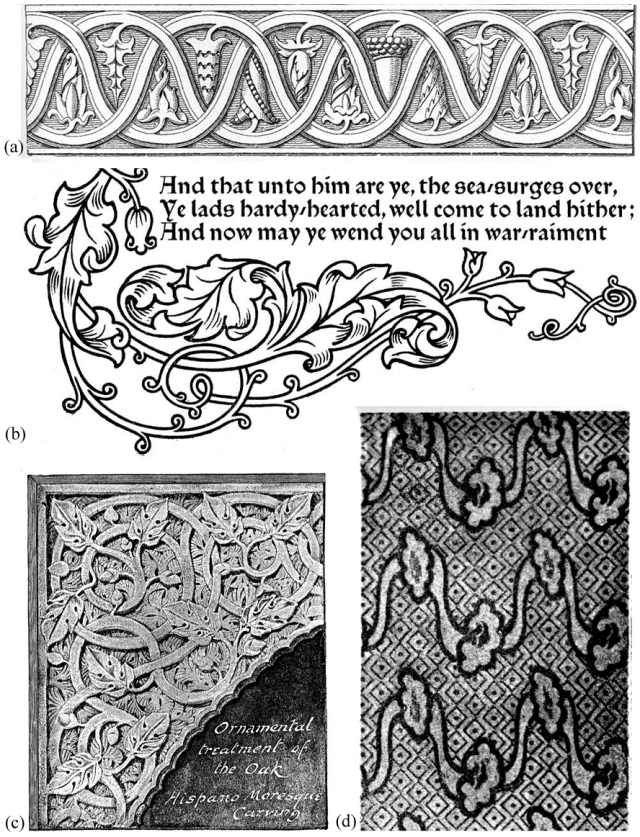



**Figure 2** Applications of ornament: (a) bands (16th century Germany) [31, plate 34] (b) half-open borders [24, opening page of chapter 7], (c) panels (oak leaf vine from the cathedral of Toledo) [10, fig. 104], (d) open plane [35, fig. 270].

### 1.2 Related work

The area of ornamental design synthesis has received relatively little attention in the computer graphics community, to our knowledge.

At SIGGRAPH '75 (the 2nd annual SIGGRAPH conference), Alexander described a Fortran program for generating the 17 symmetry patterns in the plane [1]. Grünbaum and Shephard used a more sophisticated computer program to generate periodic tilings and patterns in their landmark text on the subject [14]. However, in both of these cases, the ornamental designs produced are purely geometric and purely on the open plane.

Glassner examined the synthesis of frieze patterns, which can be used for generating textures for band ornaments [11].

Siromoney and Siromoney examined the synthesis of kolam patterns: a form of ephemeral ornament practiced in India where grains of rice are used to trace out designs forming intricate lattices [32]. Their goal, however, was to show how graph grammars could be used to generate instances of such geometric patterns, rather than to create ornament to fill a specified region.

Arvo and Kirk introduced the modeling of plant growth with environmentally sensitive automata [2], Greene examined the growth of plant-like branching structures in voxel space [13], and Prusinkiewicz *et al*. examined the generation of ornamental topiary plant forms with open L-systems [26]. The synthetic structures described in these papers were adaptive to space, but not designed to grow according to conventions of 2D ornamentation.

Smith introduced the graphics community to the modeling of plant growth with a class of parallel rewriting grammars he termed "graftals" [33]. The grammars were used to generate a branching structure, which could then be given visual character through a post-processing step. We use a similar two-step procedure to create first

the structure and then the rendering of our ornaments.

In their paper on graphical style sheets, Beach and Stone introduced the idea of procedurally generating a simple repeating border pattern that is warped to follow the path of a spline [3]. This idea was subsequently elaborated by Hsu and Lee, in their papers on "skeletal strokes," to the warping of predefined vector clip art along a path [15, 16]. Skeletal strokes—whose commercial implementation, MetaCreations Expression, we have used to render many of the illustrations in this paper—may be thought of as a rudimentary form of adaptive clip art along curvilinear paths. The work described in this paper builds on their approach by creating a higher-level mechanism for the automatic arrangement of skeletal strokes within arbitrary regions of the plane.

### 1.3 Overview

The rest of this paper is organized as follows. Section 2 surveys the key principles of floral ornamental design. Section 3 discusses how these principles can be encapsulated algorithmically. Section 4 discusses the framework of our ornamental growth engine. Section 5 presents some of our results, and Section 6 suggests areas for future research. Finally, Appendix A shows in detail some simple examples of using our system.

## 2 Principles of ornamental design

For our purposes, we will define *ornament* as the aesthetic enrichment of the surfaces of man-made objects in ways not directly contributing to their functional utility. In order to provide a sense of the richness and depth of the problems involved in creating ornament, we will briefly describe some of the principles that underlay its design. The system we have implemented so far addresses only a fraction of these principles.

Let's first look at some of the methods ornamentalists use in conveying a perception of order. We will then explore the particulars of *floral* ornamental design.

### 2.1 Order in ornament

If there is any one underlying principle of ornament, it is the conveyance of a sense of order or design [12]. Ornamentalists use three principal techniques in conveying a perception of order: repetition, balance, and conformation to geometric constraints [10, 12, 36].

#### 2.1.1 Repetition

Perhaps the most fundamental ordering principle is *repetition*. The repetition of even the simplest mark can form the basis of an ornament. When forms are repeated, they may be repeated exactly through translation and rotation (Figure 3a). Or they may be reflected about some axis, yielding *bilateral symmetry* (Figure 3c) or *glide reflection* (Figure 3b). In many patterns containing rotational symmetries, the point of radiation is positioned off-center from the design elements it controls, leading to a *bilaterally symmetric radiation* (Figure 3d).

A more subtle form of repetition is the use of *analogy*, in which similar, rhythmic controlling lines are used to place and constrain different floral or figurative elements (Figure 3e). In addition, the recurrence of almost any ratio, or *proportion*, in a design can impart a pleasing unity of form. *Color* is another powerful attribute of patterns, orthogonal to shape, that can be used to unify a design through repetition.

While designs based on rigid repetition may appeal to a clean, austere aesthetic, other patterns use *variation* within a class of forms to add organic dynamism to their composition (Figure 3h). This variation may be achieved through *alternation* of color or form (Figure 3f), or through *scaled repetition* (Figure 3g).

#### 2.1.2 Balance

The principle of *balance* requires that asymmetrical visual masses be made of equal weight. Figure 4a shows this principle applied to several compositions. We can also speak of balance in the implicit motion of lines. Crane [8] describes this phenomenon as each new

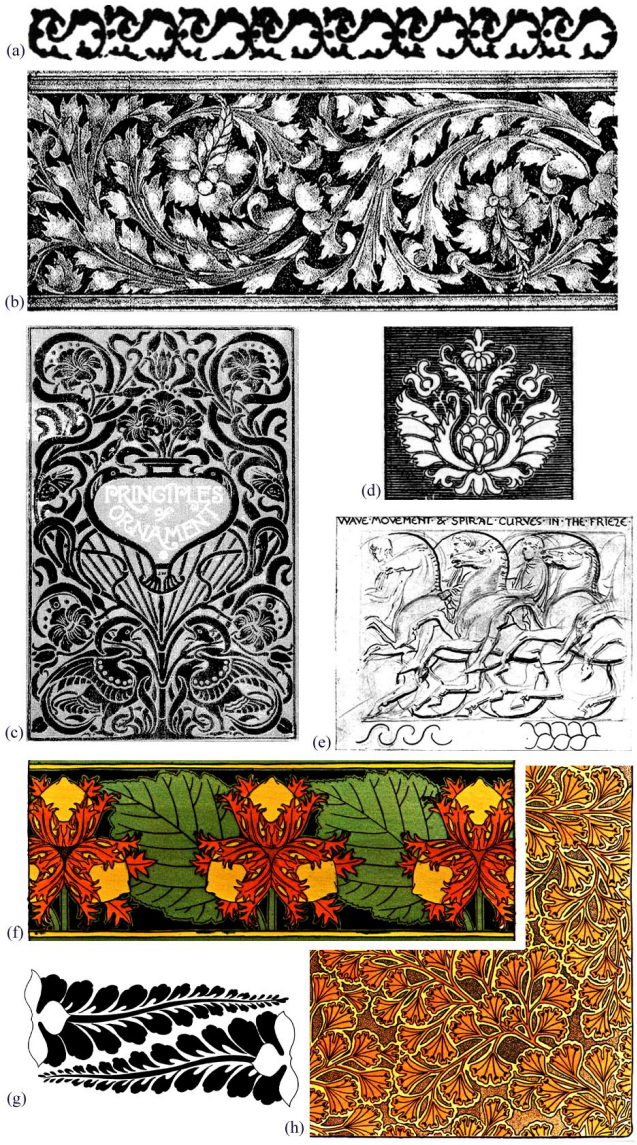



**Figure 3** Repetition: (a) simple translation [4, plate 142], (b) glide reflection [10, plate 14], (c) reflection [36, cover illustration], (d) radiation (late Gothic "pine" ornament) [10], (e) analogous (rhythmic lines in the frieze of the Parthenon) [8], (f) alternation [34, plate 78], (g) scaled [9], (h) organic variation [34, plate 27].

line posing a question that requires an answering line (Figure 4b). We can see both these principles at work in Figure 4c.

The principle of balanced masses, combined with the primal motivation for ornamentation, *horror vacui*, yields the principle of *uniform density*: ornament should uniformly fill its allotted space. In some ornaments, elements of similar mass are distributed non-uniformly in space. In this case, their unequal distribution can be balanced with different elements of a smaller scale. This type of ordering leads to a balance within and among levels of hierarchies of visual mass (Figure 4d).

#### 2.1.3 Conformation to geometric constraints

Since ornament must live within the boundaries of the objects it seeks to enrich, the design process must generally begin with a consideration of geometric constraints.

First and foremost, a careful *fitting to boundaries* is a hallmark of ornament from many cultures. Often, the period of a meandering vine, for instance, has to be adjusted not only to fit properly between the

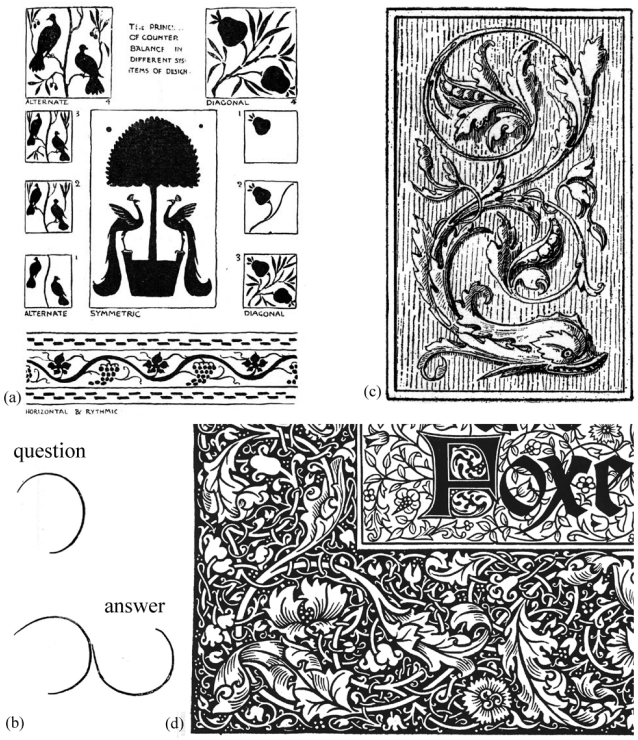



**Figure 4** Balance: (a) in composition [8], (b) question and answer within lines [8], (c) combined [36, fig. 126], (d) hierarchical [5, title page].

top and bottom edges of the panel, but also to provide appropriate positions for secondary shoots to invade other portions of the ornamented region (Figure 5a). In addition, the shapes of the design elements themselves are sometimes deformed to better fill space (Figure 5b).

In many vining motifs, elements are made to grow together tangentially. This principle of *tangential junction* lends a powerful sense of *teleological*, or ends-driven, design to the composition. For obvious reasons of structural integrity, tangential junction is also important for ornament that is "cut through" or must otherwise hang together, such as the open-work bronze basket in Figure 5c, and the sign support in Figure 5d.

A further principle ordering the layout of motifs is placement at signal geometric points such as points of maximum concavity or convexity, as in the rosettes of Figure 5e. When filling a region that has distinct corners, a design element is almost always dedicated to the task of filling each corner. When accomplishing this task with a growth motif, the growth is often coordinated by the skeleton of the region to be filled, as demonstrated by the paisley in Figure 5f.

The design of ornament frequently proceeds through the subdivision of an area followed by the filling of the divisions. Figure 5g shows the sequence of steps taken by a 19th-century textile designer from India in laying out a woodblock print. Since the act of filling may also be viewed as one of subdivision, the process may be recursively repeated, leading to a many-tiered *hierarchical composition* in the final design.

### 2.2 Floral ornament

For our purposes, we will define *floral ornament* as any ornamental design process involving plant-like growth models, such as branching structures; or plant-like elements, such as vines, leaves, or flowers.

In this section we will first examine the peculiar qualities of *growth* that distinguishes it as a progenitor of ornamental design. We will then discuss how plant-like structures can be transformed into ornamental elements through the process of *conventionalization*.

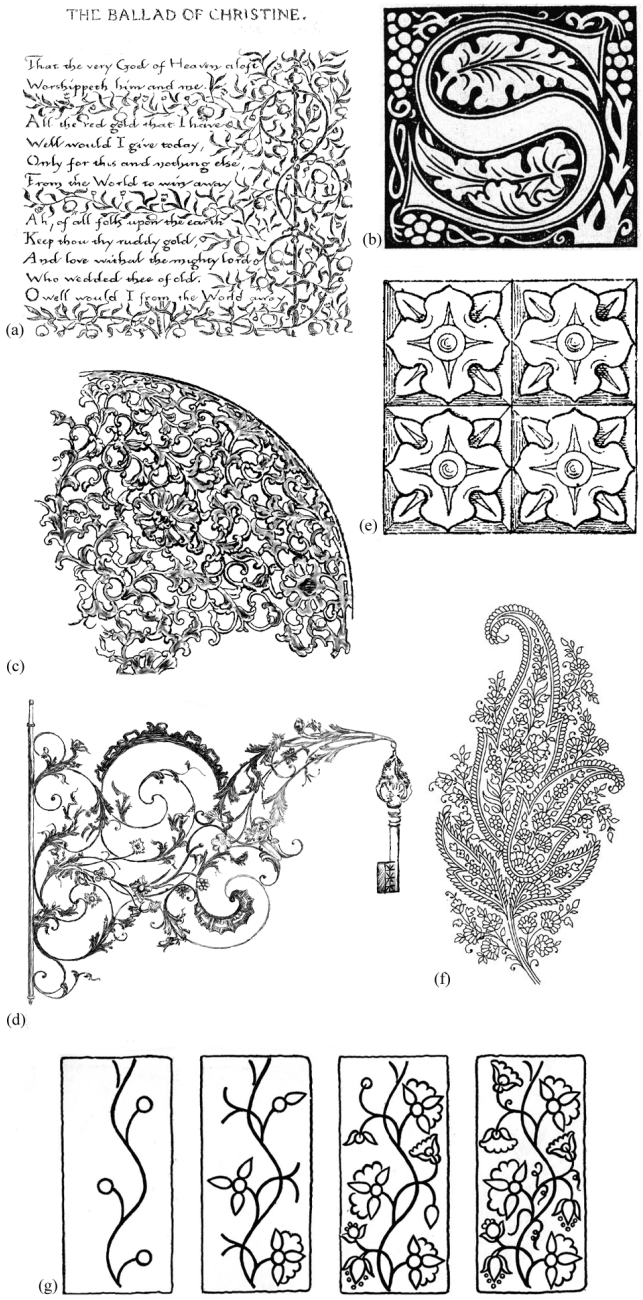



**Figure 5** Conformation to geometric constraints: (a) fitting meander period (drawn after [22, p. 35]), (b) deformation of design elements [24], (c) tangential junction (drawn after [19, p. 107]), (d) tangential junction (drawn after [12, fig. 66]), (e) signal geometric points [36, fig. 99], (f) following skeleton of a region [25], (g) hierarchical subdivision [6, fig. 213].

#### 2.2.1 Growth

To begin with, it is worth noting that most of the ornamental principles discussed so far are already principles of growth. As Owen Jones observed in the *Grammar of Ornament* [17], "whenever any style of ornament commands universal admiration, it will always be found to be in concordance with the laws which regulate the distribution of form in nature."

Growth is a particularly good source for continuous patterns that fill space and that can logically transport a design into new regions. In Figure 6, design elements are transported by linear trunks and sinuous meanders. Space is filled by smaller spiral branches and half-

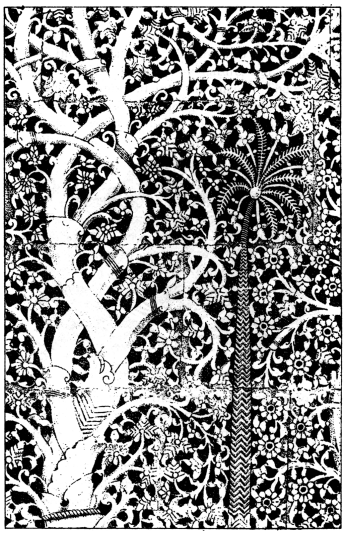


**Figure 6** Growth transporting a design [10, plate 77].

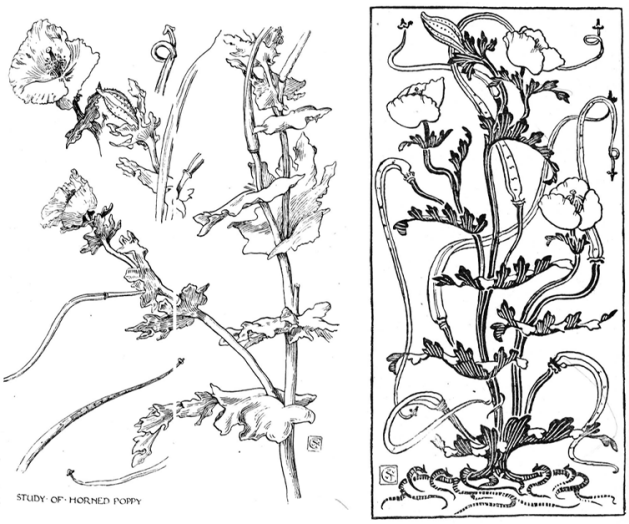



**Figure 7** Natural vs. conventional representation [8].

spiral leaves. In addition, the non-rigid repetition of forms derived from natural growth can be used to breathe life into a design.

Another issue of growth as represented in ornament is that it tends to be more highly structured, or ordered, in this context. This ordering property can be described as *intention*. Intention can be defined as the aesthetic perception of teleological growth or placement of form, discernible from multiscale features of a design: its high-level layout; its sinuous sub-motifs and their serial and hierarchical compositions; and, at the lowest level, the continuous change in curvature along a line, a line's modulation in width, and the angles of crossings of lines. In other words, intention is not just the process of growth in the absence of external influences, but rather a way of expressing growth even under such influences. Examples include growth toward pre-placed flowers, or the cooperative formation of symmetric structures, sometimes even from non-analogous locations in an overall branch structure.

#### 2.2.2 Conventionalization

While in common usage the term "convention" has a pejorative ring, implying lack of invention, in ornamental design it can have just the opposite meaning. *Conventionalization* in ornament is the development of abstractions of natural form, a highly creative process. When artists develop a conventionalization they perform a sort of inventive prefiltering of phenomenal reality followed by a creative resynthesis of form. The focus is to extract essential features of form from the vagaries of environmental influence.

In Figure 7 we see a side-by-side comparison of a study drawn from nature and a conventional representation based on that study. Note how the subtle wave of the leaf margins of the poppy get amplified and regularized in its conventionalization. Note also how the form of the seed pods has been stylized to fill space.

## 3 Approach

We will represent a given adaptive clip art pattern as a set of *elements*, which describe the geometric primitives that comprise the ornament, together with a set of *growth rules*, which describe how the elements are structured in relation to one another and to the boundaries of the panel. The growth rules are invoked by a controlling framework to produce the ornamental pattern, customized for any planar region.

L-systems would appear to be the natural choice for expressing our growth rules, as they have been used to model many plant-like structures. In the rest of this section, therefore, we will take a closer look at the use of L-systems for ornament and discuss the reasons we ultimately chose not to use them. We will then discuss the approach we took in encoding our adaptive clip art in more detail.

### 3.1 Using L-systems for ornament

L-systems were developed by biologists seeking to model the development of plants, and they have been extended by the computer graphics community [27, 28, 33] to create realistic plant images and animations. Traditional L-systems do not receive information about the environment. More recently, *open L-systems* have been introduced to allow information from the model's environment to also affect growth [20, 26]. Open L-systems are therefore a reasonable choice for encoding growth rules for ornament. As we discuss below, however, the generation of ornament differs from the growth of real plants in several significant ways that we felt limited the applicability of open L-systems in this context.

First, while floral ornaments may involve leaves, flowers, vines, and so forth, in their conventionalization these elements are often connected and arranged in ways that no plant would ever produce. Biological models are therefore not directly applicable. Indeed, we felt it would be easier, in most cases, to model the *appearance* of an ornament rather than some underlying *process* to produce it. Also, by modeling the appearance of the output directly, we felt we could have tighter control over it.

Second, the environmental feedback loop for real plant growth is indirect: the environment at a given point in space produces chemical changes in the plant that act to alter its further growth. Open L-systems model this loop by alternating "rule application" phases with "environment query" phases—productions leave symbols in the L-string to indicate where queries should be answered by the environment process. These answers can only affect productions in future iterations of the simulation. Thus, a rule for growth that incorporates environment queries must be split into a *set* of productions. We felt that in our case it would be easier to design rules in the form of *procedures*, which could both query the environment and directly act on the results of those queries in placing graphical elements of the ornament.

Finally, L-systems apply all productions to a string in parallel: each element in the string is simultaneously replaced with the result of a rule acting on the element. Rather than trying to define the semantics of parallel rule application when each rule is a procedure, we have chosen to apply our rules serially. A successful iteration of our system, then, consists of the selection of a single element, followed by the incremental growth of that element according to a certain growth rule associated with it. This process also provides an opportunity to integrate some form of global planning into both the selection of the element and the rule being applied.

### 3.2 Adaptive clip art

Adaptive clip art consists of two parts: *elements* and *growth rules*.

*Elements* correspond to the 2D geometric primitives that appear in the ornament (e.g., flowers, leaves, and stems); they are the objects upon which the growth rules operate. To provide simplicity without sacrificing the ability to draw detail, each element is defined as a collection of one or more *proxies*. A proxy is a relatively simple geometric shape that represents the element (or a part of the element) for the purposes of locating empty spaces and testing for intersections. When producing final output, a more complicated rendering procedure can be invoked. The use of proxies, therefore, keeps the details of rendering an element separate from the mechanics of positioning it in the design.

Our *growth rules* are specified as procedures. When a rule is invoked on a parent element, the code associated with that rule (the *rule body*) is executed. This code can perform environmental queries and create child elements, among other things. A support library is provided for common environmental queries and for conveniently manipulating geometrical primitives such as proxy shapes.

Finally, our framework for elaborating adaptive clip art uses a limited form of planning in selecting the element for growth on each new iteration. As described in more detail in the next section, the framework attempts first to grow the ornament into large open space, then shifts to filling in corners of the desired region.

## 4 Implementation

The current implementation consists of approximately 600 lines of Perl (the *preprocessor*) and 3,600 lines of C++ (the *framework*). The preprocessor reads a *rule file* which encodes an ornamental pattern. The preprocessor output is a C++ source file and a corresponding header file, which are compiled with the framework code to produce an executable. This executable can take a region specification and produce the ornamental pattern to fill that particular region. The output generated is a PostScript file. A default rendering is provided for every element, which simply draws each proxy of the element in outline form. The user can attach arbitrary C++ code to each element type within the rule file to generate custom PostScript output if desired. Alternatively, the PostScript output can be converted to paths and rendered with skeletal strokes [15, 16] to produce a wide variety of effects.

We will take a top-down approach to describing the implementation in the next three sections, first describing the way in which elements and rules are selected for growth, then covering the details of how they are specified.

### 4.1 Rule invocation

The main job of the framework is to decide which elements to "grow" with the rules in order to fill the given space. Let $R$ represent the region to be filled with a pattern. Our heuristic is simple: it finds the largest circle $C$ (modulo some approximation error) that does not intersect the boundary $R$ or any element of the design, and tries invoking rules on the elements within a distance $\delta$ of that circle. Elements are tried in order of their distance from the circle. When a rule succeeds (or when all possibilities are exhausted), the iteration ends and a new circle $C$ is selected.

To find the desired circle, we keep a (relatively) low-resolution buffer into which we render the proxies of already placed elements, along with the boundary of the region $R$. We start small test circles at various points within the region and increase the radius of each circle until it intersects an element or the boundary. If the inflation procedure for a given circle is stopped because the circle hits a boundary, the circle is discarded, since the circle is not adjacent to the existing ornament. If inflation is stopped by hitting an element, the circle is kept. The largest kept circle is chosen as $C$. The center and radius of $C$ are made available within rule bodies so that rules may direct their growth based on the circle's location.

To determine at which points to center the test circles, we perform a medial axis transformation (MAT) [30] using the Manhattan distance metric. A circle is centered on each pixel whose transform value is at least as great as those of its neighbors. We use these skeleton points as centers of the candidate circles to avoid having to perform the circle inflation, which is relatively slow, starting at every uncovered point in the region. The MAT is updated incrementally after each new element is placed.

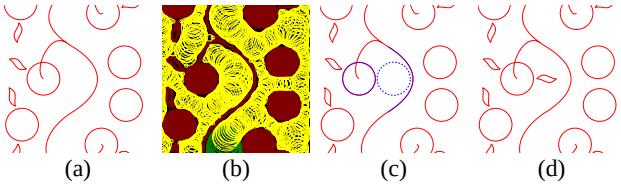



**Figure 8** One iteration of the main loop. (a) Elements already in place at the start of the iteration. (b) The render buffer, with points covered by elements and/or the region boundary (in red). Eligible empty-region circles are superimposed in yellow, ineligible circles (on the exclude list) in green. (c) The selected empty circle $C$ (dashed blue lines) and the nearby elements that are candidates for growth (thick purple lines). (d) The ornament after a rule has placed a new leaf.

It is possible that all the rules on all the elements near a given circle $C$ may fail to place new elements. In this case, $C$ would continue to be the largest empty circle available and would immediately be tried again. To prevent the algorithm from falling into an infinite loop, we keep a list of points called the *exclude list*. No circle that intersects a point on the exclude list can be selected as $C$. If all the elements near a given circle fail to produce new elements, then the center of the circle is placed on the exclude list. A point can be removed from the exclude list in one of two ways. Whenever a rule is successful in placing elements for a circle $C$, all points within $\epsilon$ of that circle's center are removed from the list. The idea is that we want to prevent a failed circle from being eligible until some change has occurred in its vicinity; then it can be tried again. The other way is for a rule body to explicitly clear the list (useful, for instance, if some state change within the rule code allows previously unavailable possibilities for placing elements).

The overall algorithm can be summarized with the following pseudocode. The *FindEmptyCircle* procedure locates the largest empty circle in the region, subject to the two restrictions above. The effects of one iteration of the main loop are illustrated in Figure 8.

```
initialize element tree with seed points
render boundary elements into buffer
compute initial MAT
initialize empty exclude list
repeat
    C ← FindEmptyCircle()
    find elements within δ of C
    try elements in order of distance from C
        try rules in order specified in rule file
            if rule succeeds, break
    if some rule succeeded
        update element tree
        render new elements into buffer
        incrementally update MAT
        remove points on exclude list within ε of C
    else
        add center of empty circle to exclude list
```

### 4.2 Elements and rules

Each design element has a *type*. The set of available types is declared in the rule file. Each element type is associated with one or more proxies, and zero or more user fields. Available proxies include circles (`circle`), arcs (`arc`), cubic Bézier segments (`bezier`), line segments (`linesegment`), etc. Each element contains a few standard fields (such as the number of children the element has), any user fields given in the element declaration, and proxy objects of the types specified in the declaration. The fields of each proxy are dependent on its type: a `circle` proxy, for instance, has `center` and `radius` fields.

Each rule file must declare the element type `seed`, with a single `point` proxy. Seed elements are placed by the framework in user-selected locations at the beginning of the run to start the ornament.

After the element declaration section of the file is the rule section. Each rule specifies what element type the rule acts on (the *parent*)

and what types of children the rule produces. The set of children created by the rule consists of the static children declared in the rule preamble, plus any dynamic children created within the rule. The only difference between static and dynamic children is how they are initialized and how they are referenced within the rule body.

The body of the rule looks very much like a block of C++ code. Anything that is legal within a C++ function is legal within a rule body. Additionally, special dollar-sign tokens provide convenient access to the fields of the parent and child elements. The preprocessor translates these tokens into C++ expressions referring into the data structures of the elements.

Each rule returns a flag to indicate success or failure. On success, the children created by the rule are permanently added to the ornament and a new iteration begins. On failure, the children elements are discarded, and the framework proceeds to try other element/rule combinations as discussed in Section 4.1.

Two detailed examples of patterns implemented with this system are given in Appendix A.

## 5 Results

Our first set of results shows four different ornamental patterns, each elaborated over two regions.

The first pattern (Figure 9) is based on a pattern taken from a Chinese vase [18, plate 47]. The pattern has two types of stylized flowers laid down in a grid pattern and connected by curving stems. The remaining space is filled with small hook-shaped curves, which themselves are adorned with smaller teardrop shapes. In addition to exhibiting constraints to geometric bounds, this example was chosen to demonstrate "intentional" growth: the large vine appears to deliver its flowers to predefined locations on the grid.

The second pattern (Figure 10) demonstrates the principle of hierarchical growth. The pattern starts from the seed points by growing the vines. It then adds the red flowers and the yellow and blue shapes, connecting them to the main vine structure with shorter subsidiary vines. Next, leaves are added, either attached to a vine or floating on their own, and finally the small double-quote-shaped structure is used to fill in small gaps. This ordering of rule phases is imposed on the system by adding "state" preconditions to each rule, so that any rule that is invoked when the program is not in the right state automatically fails.

The third pattern (Figure 11) is a somewhat less successful attempt, motivated by a William Morris willow-leaf wallpaper. There is only one rule, which grows through the empty circle by adding a curved stem with alternating leaves while preventing the leaves from overlapping too much. This pattern illustrates a shortcoming in our approach, which is that it is difficult to do significant global planning of a design. In our current system, rule invocation is controlled by the empty-space-finding algorithm, so growth always proceeds from the nearest element. For many patterns, it would be better to fill a given space with growth from a more distant element that curves so as to naturally pass through that space. Our "willow" pattern, while covering the region well, is jumbled in comparison to the more elegant original.

The fourth pattern (Figure 12) uses a motif based on an equal-angle spiral, a shape that can be seen in diverse natural forms, from the spiral of a nautilus shell to the curve of a vine tendril [7]. This same pattern is also used, with a different rendering style, to generate the border on the first page of this paper. Each spiral is composed of multiple curved segments, making heavy use of dynamic child creation, since spirals of different lengths require different numbers of segments in order to appear smooth. Each spiral curve is given an orientation opposite to that of its parent. Note how the pattern generates a rhythmic repeat with a period that is related to the changing width of the space; the pattern also simplifies as it wanders into narrower spaces. Although the rules that generate this pattern are not explicitly hierarchical, the appearance of hierarchical structuring is nonetheless formed by placing new elements in, and scaling them to, the largest empty circle adjacent to the growing ornament. The resulting ornament reveals large-scale structures placed in relation to the outline of the boundary space, with finer-scale details placed in relation to both the boundary space and the evolving ornament.

Figure 13 shows each of these four patterns again, elaborated over differently-shaped panels.

Figure 17 shows the breadth of rendering possibilities provided by the skeletal strokes technique [16]. The same spiral design is rendered with four different strokes, producing a variety of effects. Although the underlying spiral growth motif is more subtly felt in the more abstract renderings, its ordering properties structure the distribution and scaled repetition of design elements, creating an organic feel to the compositions.

## 6 Conclusion

In this paper, we have described a mechanism for encapsulating growth principles for ornamental design into "adaptive clip art" patterns.

Although we have so far implemented only a rudimentary testbed for these ideas, we envision, ultimately, a powerful interactive authoring system for designing these patterns. The artistic tool so derived—unlike most previous work in computer-generated artistic rendering—might be more than just a digital form of an existing artistic medium: it could essentially provide a new medium of artistic expression, one that yields "living," dynamic patterns that adapt to their environments. As we gain more experience with the novel parameter space of this new medium we hope to encapsulate our knowledge in high-level, interactive tools that novices and artists alike will be able to use for creating new instances of these patterns.

In addition to creating better high-level design tools, there are a huge number of other important areas for future research:

***Ornaments over manifolds.*** We would like to extend our work to creating ornaments over arbitrary manifolds. Such techniques would allow the ornamentation of 3D objects (vases, mugs, T-shirts, etc.) without the distortion that results from simply mapping a planar ornament onto the surface.

***Incorporating global planning strategies.*** Our strategy of growth towards the largest empty region is a simple, relatively local one. A more sophisticated approach might be developed to look at the design more globally and better incorporate ornamental design principles such as balance and symmetry.

***Putting an artist "in the loop."*** In applications such as web page ornamentation, the adaptive clip art must be generated purely automatically, on the fly. However, in other applications, such as wallpaper design, there is no reason not to put an artist in front of the computer to help guide the growth of the pattern and improve its appearance artistically, since both the cost of manufacturing the resulting artwork and the longevity of the finished piece are both relatively high. It would be interesting to explore semi-automatic algorithmic design processes and user interfaces for use in these situations.

### Acknowledgements

We would like to thank Przemyslaw Prusinkiewicz, Ned Greene, and Victor Ostromoukhov for many helpful discussions. This work was supported by an NSF Presidential Faculty Fellow award (CCR-9553199), an ONR Young Investigator award (N00014-95-1-0728), an NSF Graduate Research Fellowship, and industrial gifts from Microsoft and Pixar.

## References

[1] Howard Alexander. The computer/plotter and the 17 ornamental design types. In *Proceedings of SIGGRAPH '75*, pages 160–167. 1975.

[2] J. Arvo and D. Kirk. Modeling plants with environment-sensitive automata. In *Ausgraph 88 proceedings*, pages 27–33. 1988.

[3] Richard Beach and Maureen Stone. Graphical style—towards high quality illustrations. In *Proceedings of SIGGRAPH '83*, pages 127–135. 1983.

[4] Albert Fidelis Butsch. *Handbook of Renaissance Ornament: 1290 Designs from Decorated Books*. Dover Publications, Inc., New York, 1969.

[5] William Caxton. *History of Reynard the Foxe*. Kelmscott, Hammersmith, England, 1892.

[6] Archibald H. Christie. *Traditional Methods of Pattern Designing*. The Clarendon Press, Oxford, 1929.

[7] Theodore A. Cook. *The Curves of Life*. Constable and Company, London, 1914.

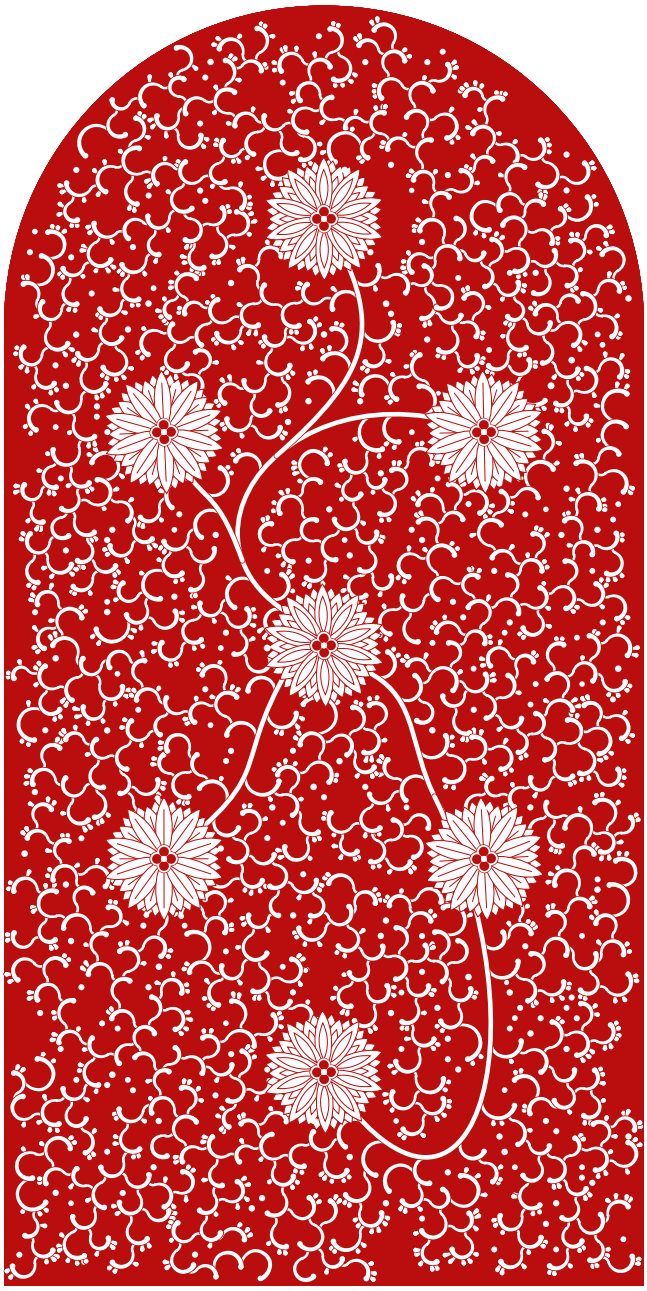


**Figure 9** Red Chinese vase pattern.

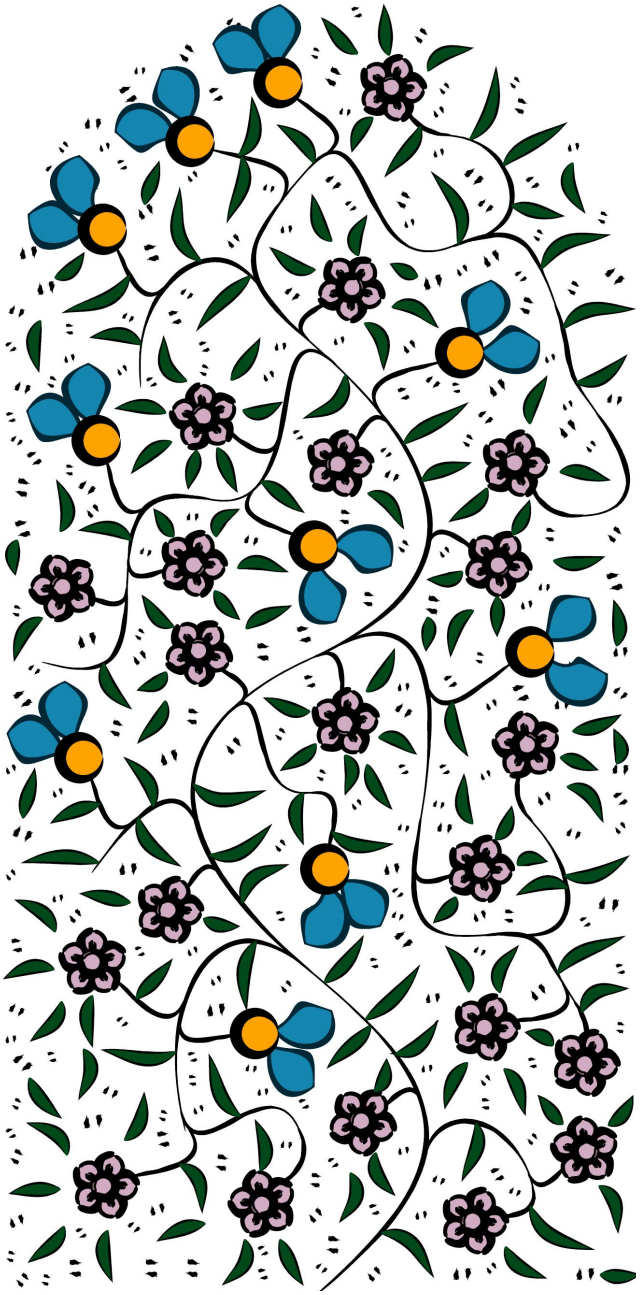


**Figure 10** Flowers and leaves pattern.


[8] Walter Crane. *Line and Form*. G. Bell & Sons, London, 1902.

[9] Joseph D'Addetta. *Traditional Japanese Design Motifs*. Dover Publications, Inc., New York, 1984.

[10] Lewis F. Day. *Nature in Ornament*. B.T. Batsford, London, 1898.

[11] Andrew Glassner. Frieze groups. *IEEE Computer Graphics and Applications*, 16(3):78–83, May 1996.

[12] E.H. Gombrich. *The Sense of Order*. Phaidon Press Limited, London, 1994.

[13] N. Greene. Voxel space automata: Modeling with stocastic growth processes in voxel space. In *Proceedings of SIGGRAPH '89*, pages 175–184. 1989.

[14] Branko Grünbaum and G. C. Shephard. *Tilings and Patterns*. W.H. Freeman, New York, 1987.

[15] Siu Chi Hsu, I. H. H. Lee, and N. E. Wiseman. Skeletal strokes. In *Proceedings of UIST '93*, pages 197–206. 1993.

[16] Siu Chi Hsu and Irene H. H. Lee. Drawing and animation using skeletal strokes. In *Proceedings of SIGGRAPH '94*, pages 109–118. 1994.

[17] Owen Jones. *Grammar of Ornament*. Day and son, London, 1856.

[18] Owen Jones. *The Grammar of Chinese Ornament*. Portland House, New York, 1987.

[19] Sherman E. Lee. *The Genius of Japanese Design*. Kodansha International, Tokyo, 1981.

[20] Radomír Měch and Przemyslaw Prusinkiewicz. Visual models of plants interacting with their environment. In *Proceedings of SIGGRAPH '96*, pages 397–410. 1996.

[21] Franz S. Meyer. *Handbook of Ornament*. Dover Publications, New York, 1957.

[22] William Morris. *A Book of verse*. Kelmscott, Hammersmith, England, 1870.

[23] William Morris. *Ideal Book: Essays and Lectures on the Arts of the Book*. University of California Press, Berkeley, 1982.

[24] William Morris and A.J. Wyatt. *The Tale of Beowulf*. Kelmscott Press, Hammersmith, England, 1895.

[25] K. Prakash. *Paisleys and Other Textile Designs from India*. Dover, New York, 1994.

[26] Przemyslaw Prusinkiewicz, Mark James, and Radomír Měch. Synthetic topiary. In *Proceedings of SIGGRAPH '94*, pages 351–358. 1994.

[27] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, New York, 1990.



**Figure 11** Willow leaf pattern.


[28] Przemyslaw Prusinkiewicz, Aristid Lindenmayer, and James Hanan. Developmental models of herbaceous plants for computer imagery purposes. In *Proceedings of SIGGRAPH '88*, pages 141–150. 1988.

[29] Auguste Racinet. *Polychromatic Ornament*. Firmin Didot freres, fils & cie, Paris, 1873.

[30] A. Rosenfeld and A. C. Kak. *Digital Picture Processing*. Academic Press, New York, 1976.

[31] Henry Shaw. *The Encylopedia of Ornament*. J. Grant, Edinburgh, 1898.

[32] Gift Siromoney and Rani Siromoney. Rosenfeld's cycle grammars and kolam. In *Graph-Grammars and Their Application to Computer Science*, pages 564–579. Springer-Verlag, Berlin, 1986.

[33] Alvy Ray Smith. Plants, fractals, and formal languages. In *Proceedings of SIGGRAPH '84*, pages 1–10. 1984.

[34] M.P. Verneuil. *Floral Patterns*. Dover, New York, 1981.

[35] Otto von Falke. *Decorative Silks*. W. Helburn, Inc., New York, 1922.

[36] James Ward. *The Principles of Ornament*. Scribner, New York, 1896.

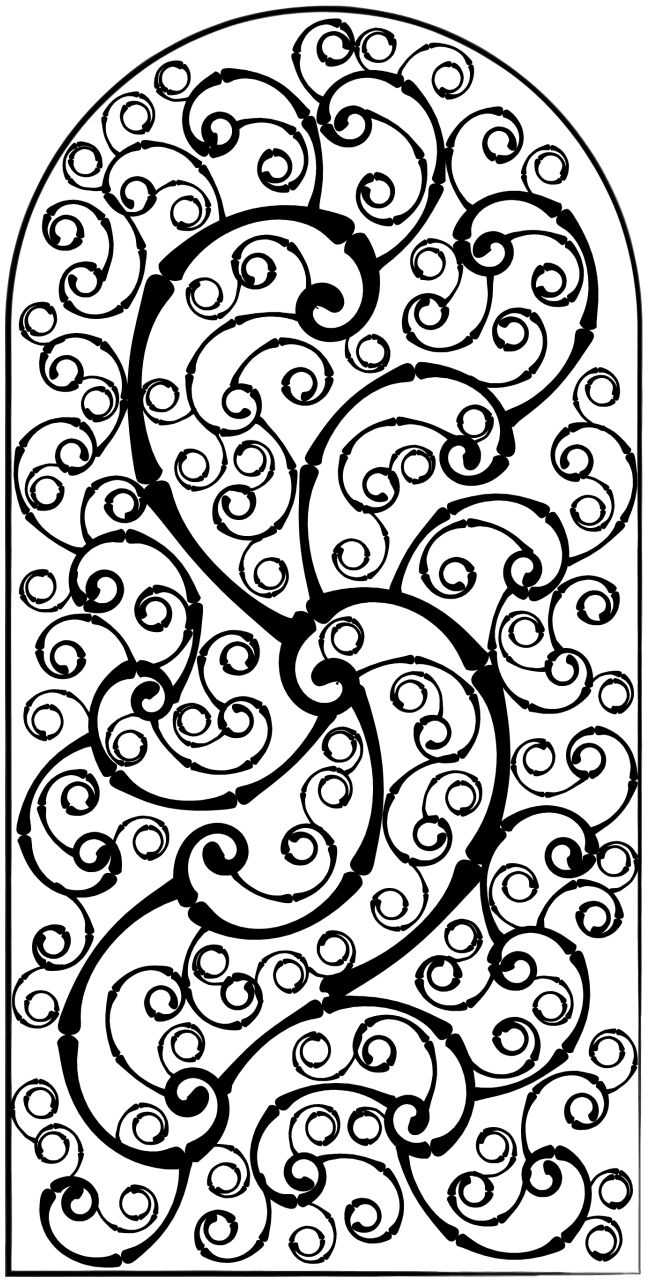



**Figure 12** Bamboo spirals pattern.

## A Examples

Our first example pattern is a simple cluster of circular dots (Figure 14). The first dot is centered on the seed point, and subsequent dots are placed adjacent to the existing ornament. Dots may be at most 3 units in radius. Table 1 explains the dollar-sign tokens used within the rules in this appendix.

```
%element
seed point     // the seed element is required
%endelement

%element
dot circle
int order;     // number each dot in order of placement
%endelement

%source               // declare a global variable
int dot_count = 0;    // to count the dots placed
%endsource
```

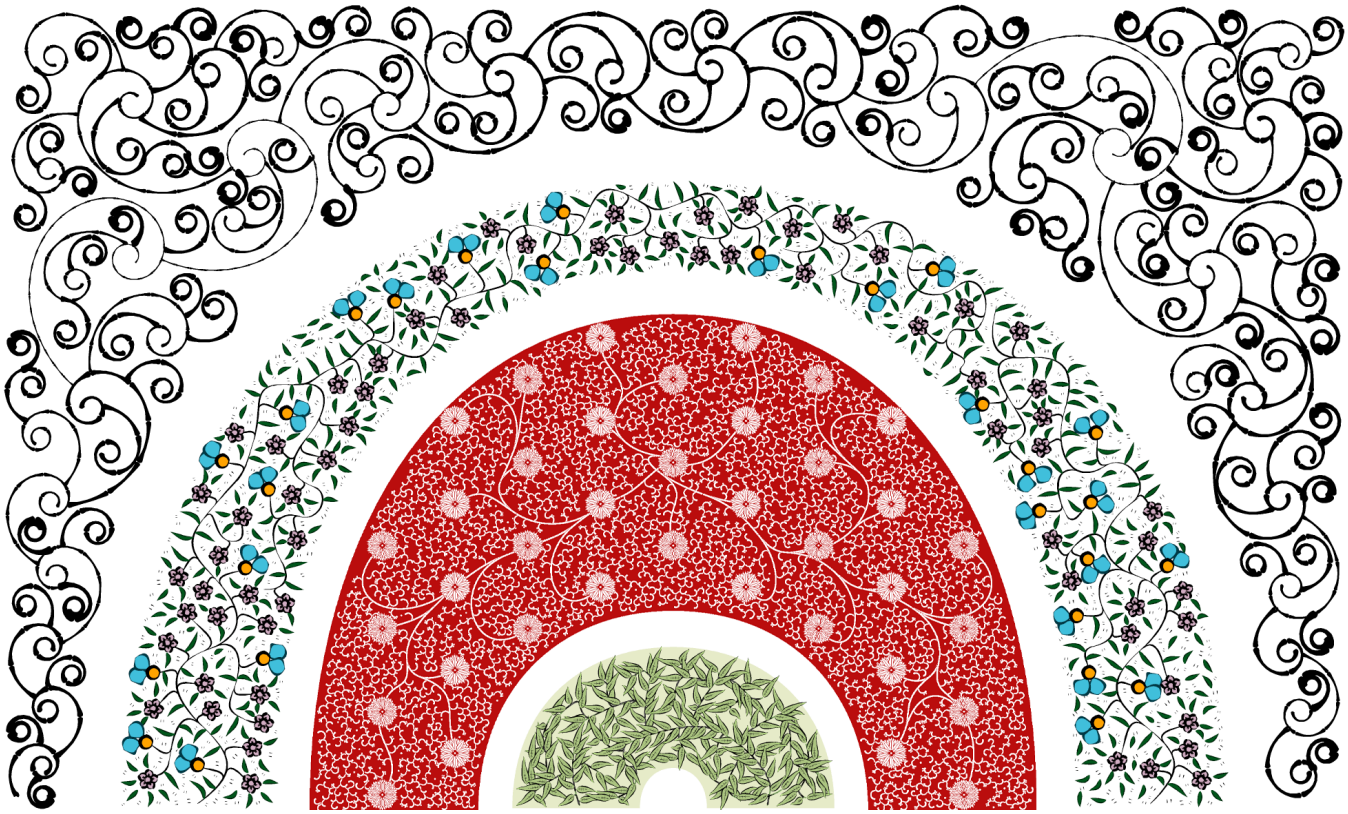


**Figure 13** All four patterns, elaborated over different shapes. The line width variation in the outer spiral arch has been reversed from that of Figure 12 to give a filigreed effect.

```
%rule
seed --> dot
{
     // place a maximum-sized dot on the initial seed
     $0.set( $, 3.0 );
     $$0.order = dot_count++;

     // prevent the seed from being used again
     $$.sterile = 1;

     return SUCCESS;
}
%endrule

%rule
dot --> dot
{
     // ignore tiny empty spaces.
     if ( $goal.radius < 0.5 ) return FAILURE;

     // determine the radius of the new dot.
     double r = min( 3.0, $goal.radius );

     // place the new dot adjacent to the parent dot.
     $0.set( $.center.offset( $goal.center - $.center,
                              $.radius + r ), r );
     $$0.order = dot_count++;

     return SUCCESS;
}
%endrule
```

The `$0.set` call is the critical line. It places the new child dot by taking the center of the parent dot (`$.center`), and offsetting it by the sum of the parent and child radii (`$.radius + r`) in the direction from the center of the parent dot to the center of the goal circle *C*.

Figure 14 shows this pattern applied to three different regions. For this example, we have colored the dots, using the `order` field, to indicate the order in which they were placed, from oldest (red) to newest (purple). This example illustrates how the empty-circle heuristic first extends the ornament along the skeleton of the region, then fills in smaller and smaller regions successively.

| token | meaning |
|---|---|
| $$ | the parent element |
| $ | proxy 0 of the parent |
| $(*k*) | proxy *k* of the parent |
| $$*j* | the *j*'th declared child element |
| $*j* | proxy 0 of the *j*'th declared child |
| $*j*(*k*) | proxy *k* of the *j*'th child |
| $$*var* | dynamic child element in *var* |
| $*var* | proxy 0 of a dynamic child |
| $*var*(*k*) | proxy *k* of a dynamic child |
| `$goal` | the empty circle *C* |

**Table 1** Explanation of dollar-sign tokens used in rule bodies.

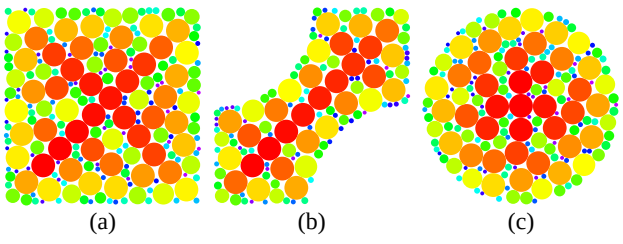



**Figure 14** A simple example applied to three different regions. Parts (a) and (b) were seeded near the lower left corner, while part (c) was seeded at the center.

A more complex example involves an arrangement of flowers, leaves, and stems. This rule file has the following declaration section:

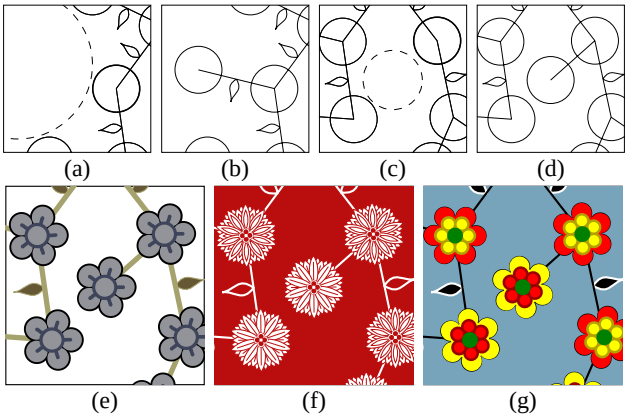



**Figure 15** Two applications of the second example rule. Part (a) shows the parent flower (bold), and the empty goal circle (dashed). The center of the goal circle lies more than 8 units away, so the rule produces (b) a flower, a stem, and a leaf. When the rule is applied in the situation of part (c), where the goal circle is smaller, only the flower and stem are produced (d). Parts (e)–(g) show the elements of part (d) in a variety of rendering styles.

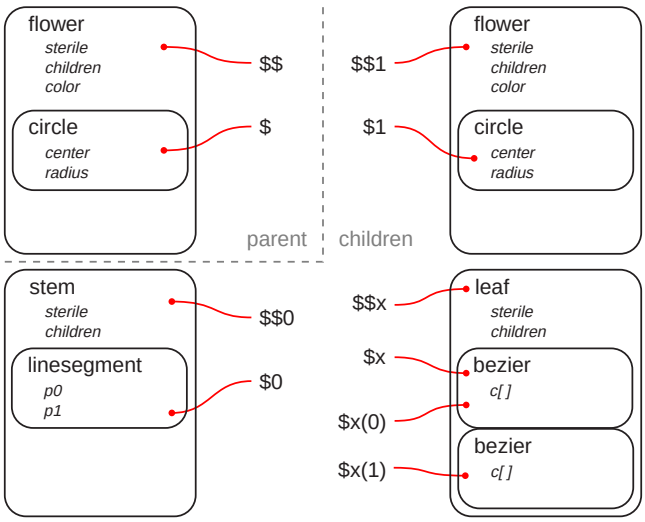



**Figure 16** Data structures and dollar-sign tokens for the rule in the second example.

```
%element
seed point          // the required seed element type
%endelement

%element
flower circle       // flower: a circle proxy
int color;          // this element type has a user field
%endelement

%element
stem linesegment    // stem: a line segment proxy
%endelement

%element
leaf bezier bezier  // leaf: two Bezier segment proxies
%endelement
```

Here three element types are declared, in addition to the required `seed` type: `flower`, which is proxied by a single circle; `stem`, which is proxied by a line segment; and `leaf`, proxied by two Bézier segments. The `flower` element contains an integer user field called `color`.

This pattern also has only two rules. One rule places a flower on top of the initial seed point, and so is invoked only once. We will omit the code for of this rule. The other rule is more interesting: it places a new flower connected to an existing flower with a new stem segment, and adds a leaf to the stem only if the stem is long enough. Thus, the number of children produced is variable (two or three). The effects of this rule are pictured in Figure 15.

Here is the preamble to the rule, which creates a set of elements whose relationships are depicted in Figure 16:

```
%rule
flower --> stem flower
* x leaf
```

The last line of the preamble tells the preprocessor that the variable `x` within the rule body will point to an element of type `leaf`. This declaration is necessary so that when the preprocessor sees a dollar-sign construction involving the variable `x`, it knows the type of `x` and can insert appropriate typecasts.

Here is the remainder of the rule:

```
{
    Direction to_goal = $goal.center - $.center;
    double distance;

    // determine how far away to place the child flower.
    // ~ on the difference between two points gives
    // the distance between them.
    distance = ~($goal.center - $.center);
    if ( distance > 10.0 )
        distance = 10.0;

    // place the flower centered "distance" units away
    // in the direction of the goal, with a radius of 3.
    $1.set( $.center.offset( to_goal, distance ), 3.0 );

    // if the new flower intersects any
    // already-placed element, cancel this rule.
    if ( intersection( $$1 ) )
        return FAILURE;

    // the stem extends from the center of the parent
    // flower to the center of the child flower.
    $0.set( $.center, $1.center );

    // if the new flower was placed far enough away
    // add a leaf as well.
    if ( distance > 8.0 )
    {
        leaf *x = new leaf;

        // the base of the leaf is the stem midpoint.
        Point leaf_base = ( $1.center - $.center ) / 2;

        // place the leaf at a right angle to the stem,
        // and make it 3 units long.
        Direction leaf_dir = to_goal + M_PI/2;
        Point leaf_tip = leaf_base.offset(leaf_dir, 3.0);

        // a leaf is proxied by two Bezier segments.
        // each is placed giving position and tangent
        // direction and magnitude.
        $x(0).set( leaf_base, leaf_dir+M_PI/4, 0.6,
                   leaf_tip, leaf_dir, 0.4 );
        $x(1).set( leaf_base, leaf_dir-M_PI/4, 0.6,
                   leaf_tip, leaf_dir, 0.4 );

        // add the newly created leaf to the
        // child set of this rule.
        new_child( $$x );
    }

    // commit the set of children to the design.
    return SUCCESS;
}
%endrule
```

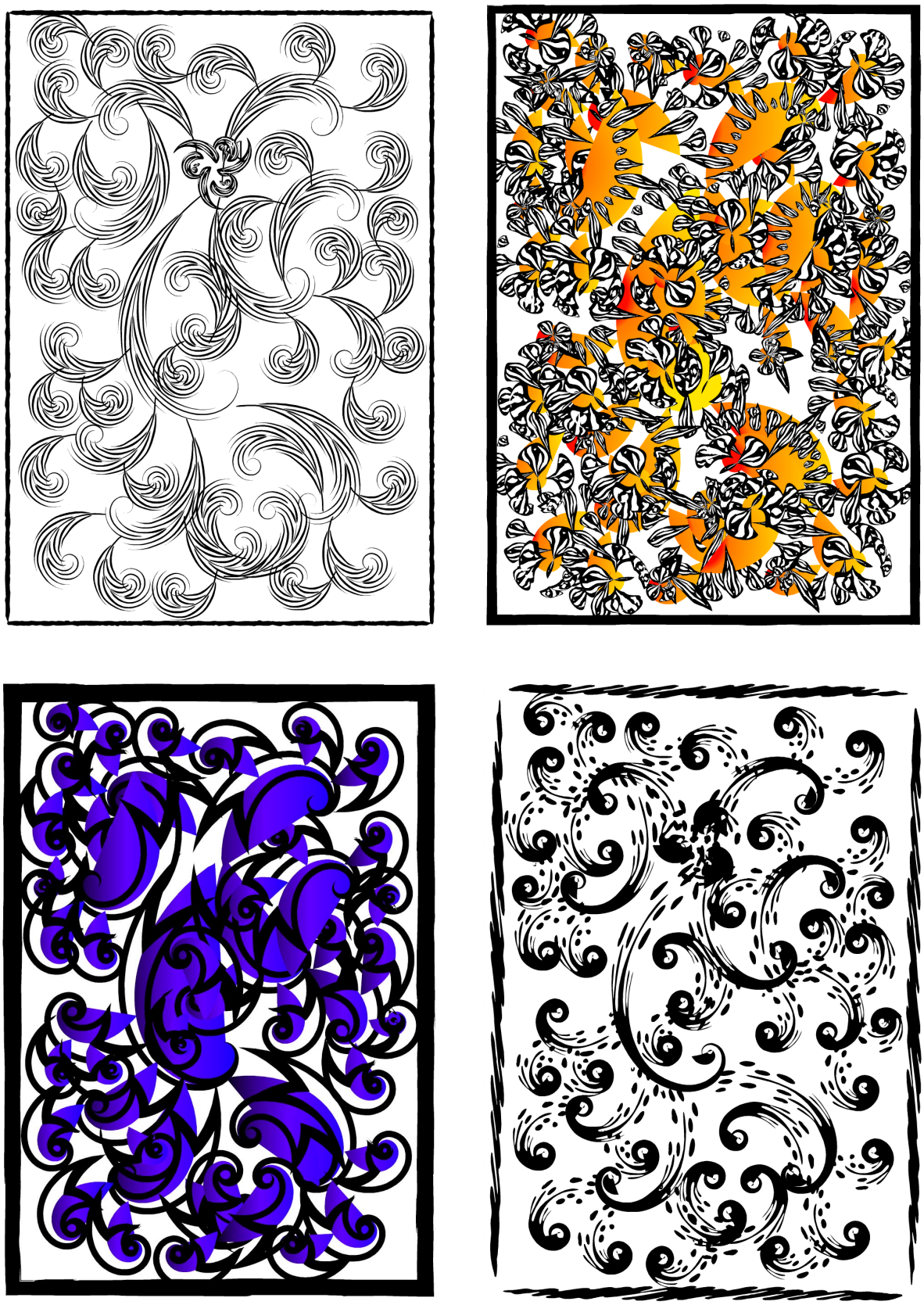


**Figure 17** Results of applying different skeletal strokes to a single design.