EUROGRAPHICS 2008 / G. Drettakis and R. Scopigno
(guest editor)

Volume 27 (2008), Number 3

An Example-based Procedural System for Element Arrangement

Takashi Ijiri[†], Radomír Měch[‡], Takeo Igarashi[†*], and Gavin Miller[‡]

[†]The University of Tokyo, [‡]Adobe Systems Incorporated, [*]PRESTO JST

Abstract

We present a method for synthesizing two dimensional (2D) element arrangements from an example. The main idea is to combine texture synthesis techniques based-on a local neighborhood comparison and procedural modeling systems based-on local growth. Given a user-specified reference pattern, our system analyzes neighborhood information of each element by constructing connectivity. Our synthesis process starts with a single seed and progressively places elements one by one by searching a reference element which has local features that are the most similar to the target place of the synthesized pattern. To support creative design activities, we introduce three types of interaction for controlling global features of the resulting pattern, namely a spray tool, a flow field tool, and a boundary tool. We also introduce a global optimization process that helps to avoid local error concentrations. We illustrate the feasibility of our method by creating several types of 2D patterns.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Color, shading, shadowing, and texture. I.3.4 [Computer Graphics]: Paint Systems.

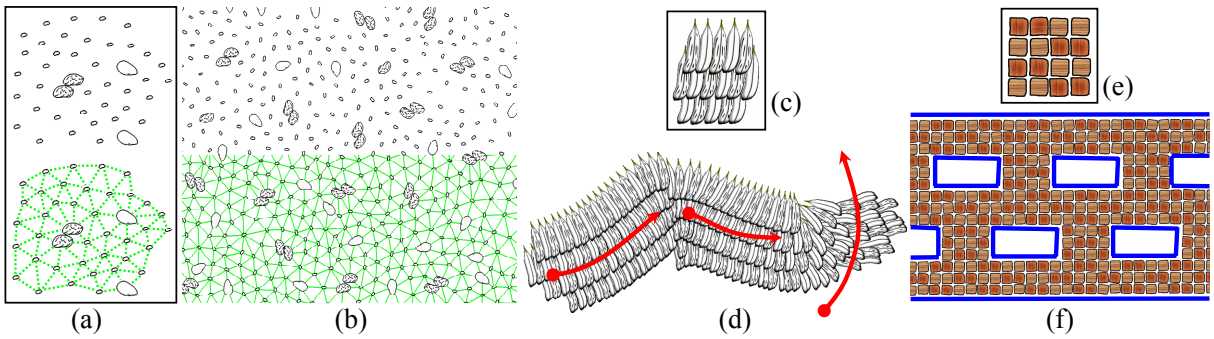

Figure 1: *Our system takes the reference arrangement input by the user and analyzes the local structure by constructing the connectivity (a). We then synthesize a larger pattern which has a similar local relationship and topology (b). The user also can specify a local growth area, an underlying flow field or a boundary (c-f). Red strokes in (d) indicate user-specified flow field strokes and blue lines in (f) are user-specified boundary lines.*

1. Introduction

We can often observe 2D arrangements of elements in either man-made or natural environments, such as tiles, wallpaper, hatching strokes, decorative arts, fabrics, flowers, honeycombs, animal fur, waves, and so on. The synthesis of arrangement patterns is not only useful for texture generation and non-photorealistic rendering (NPR), but it also poses an interesting challenge in computer graphics. These arrangements have a large variation; some of them may have regular or near-regular features throughout, others are not locally regular but they distribute irregular features uniformly (irregular uniform). To synthesize the arrangement patterns, the system has to preserve local spatial relationships between elements in the global distribution.

Two different groups of methods exist for generating general texture patterns; texture synthesis and procedural modeling. Pixel based texture synthesis is not well suited to the problem of element arrangement, because not each pixel but each element is individually perceptible in the element arrangements. The spatial relationships between elements are more important than those between pixels. In contrast, procedural modeling systems deal with an element as a unit module. The system starts from an initial feature, and progressively replaces and adds modules based on local generation rules. However, procedural frameworks

are limited to patterns, in which elements have explicit local structure such as trees. For instance, it is difficult to define the growth rules for irregular uniform arrangements.

In this paper, we present a new technique for synthesizing element arrangement patterns from a user-input example. The main idea is to combine texture synthesis methods based-on a local neighborhood comparison [EL99] and procedural modeling systems based on a local growth [PL90; WZS98]. Given the reference pattern, we construct connectivity to obtain the immediate neighborhood information of each element (Figure 1a) [BBT*06]. The synthesis process starts with a single seed and expands the pattern outward by placing a new element one by one. In this process, we apply local neighborhood comparison; we choose a reference element whose neighborhood is the most similar to the target neighborhood of the previously synthesized pattern. Figure 1(b) shows a synthesized pattern, in which almost all elements are uniformly distributed but two gray blobs appear to be adjacent.

To support an intuitive synthesis process, we provide three types of interaction tools for controlling global features of the resulting pattern; a spray tool for specifying the growth area, a flow field tool for specifying an underlying flow field (Figure 1(c) and (d)), and a boundary tool for limiting growth (Figure 1(e) and (f)). We also introduce a global shape optimization of the distribution in synthesis process that helps to avoid errors' concentrating local area and to get smooth result.

Barla et al. introduced a method to synthesize stroke patterns dealing with a single stroke or a cluster of strokes as a unit element [BBT*06]. The main difference between their system and our system is in the synthesis process. Their system constructs a global distribution of seeds by Lloyd's method [Llo82] at the beginning and places synthesized strokes at the seed positions. In contrast, we place elements one-by-one, generating a global distribution gradually. We also keep a local topology of each element. This difference provides several advantages and disadvantages. One advantage is the ability to create a regular or near regular pattern that is difficult to be created by Lloyd's method. We can also provide the spray and flow field tool; we think it is difficult for Barla's method to combine these controls since their method fixes the global distribution at the beginning. One disadvantage of our approach is the difficulty of obtaining an optimized distribution. While Barla's system generates an optimal distribution by Lloyd's methods, it is sometimes difficult for our system to obtain a globally optimal distribution since we grow each element locally.

In recent decades, many procedural modeling systems have been proposed for creating plants, buildings or decorative art design. Although they have achieved impressive results, most of them require the user to write scripts to define the local growth rules, which makes it difficult for novice users to control the resulting model. In contrast, our system tries to obtain the rules from examples. We believe that our example-based procedural modeling system provides a more intuitive way to design element arrangements.

2. Related Work

In this section we briefly review representative work from procedural modeling, texture synthesis, and NPR.

Procedural modeling: Recently a lot of procedural modeling systems have been developed for different purposes. L-System was originally formulated by Lindenmayer [Lin68] to simulate the growth of plants and was introduced to the computer graphics community later [PL90]. L-Systems have been extended to simulate a wide variety of interactions between plants and their environments [MP96; PJM94]. Wong et al. applied procedural modeling to the creation of floral ornaments [WZS98]. Procedural modeling has been used in architectural models of cities [PM01], and buildings or facades [WWSR03; MWH*06]. While these systems have achieved impressive results, they require the user to write unintuitive scripts to define rules. Deussen et al. use a predefined set of rules, represented by icons. Instead of writing a script the user connects icons together and modifies their parameters [DL99]. This system, however, still requires the user to choose rules from the predefined set and, as with any procedural system. It is often not obvious what the resulting structure will be for a given set of rules. Also, the rules are fixed and the system is suitable only for modeling plants and trees. We are more interested in an example-based system in which the user only specifies a small piece of the desired pattern.

There are researchers focusing on interactions with particular types of procedural models. Esch et al. used a flow field to modify street pattern [EWMZ07]. Ijiri et al. apply free hand strokes to specify the growth direction of main axes of trees [IOI06].

Texture synthesis: Texture synthesis techniques that take an example image and generate a new texture are roughly categorized into four groups; frequency domain methods, tiling methods, pixel-based methods, and patch-based methods. i) Heeger and Bergen used frequency domain techniques [HB95], but they can only deal with specific types of textures. ii) The tiling methods are used for a fast texture synthesis [CSHD03] or a near-regular texture synthesis [LLH04]. Some researches generate large object distributions by combining Poisson disk distributions and the tiling [CSHD03; LD05]. However these methods can only create irregular uniform patterns and it is difficult for them to combine the flow field control. iii) The pixel-based approaches work with neighborhood comparisons between the example and generated (target) textures [EL99; WL00], and various extensions have been developed [Ash01; HJO*01; Tur01]. iv) The patch based approach [KSE*03], which stitches the reference image along optimal seams can generate images quickly. These pixel and patch based techniques achieve convincing results for more general textures. However, it is still difficult to directly apply these techniques to element arrangements because each element is perceived individually. The local relationship between elements is more important than that between pixels.

Non-photorealistic rendering: Stroke pattern synthesis is a popular topic in NPR field. Deussen et al. generated

stipple drawings [DHOS00], Winkenbach et al. [WS94] and Salisbury et al. [SABS94] generated a pen-and-ink effect by using stroke textures. These systems synthesized stroke patterns based on generating rules that were selected by authors or observed on traditional drawings.

Some example-based systems have been published recently. Hertzmann et al. presented curve analogies which learns a style of an example and modifies a target curve to have a similar style [HOCS02]. Jodoin et al. generated hatching patterns by examples [JEG*02]. They focused on a relatively simple case, in which all strokes are aligned in a linear order on a curve. Our system is particularly inspired by stroke pattern analysis and synthesis [BBT*06] that synthesize a vector-based 2D pattern from an example by extending a texture synthesis technique (See section 1).

3. Overview

Our system takes an arrangement of input elements which we call *a reference arrangement* and generates a new larger pattern which is similar to the reference. We suppose that the reference arrangement uses symbols from a predefined set. Each symbol in this set has a symbol id. In addition, each symbol in the reference arrangement has a unique sub-id that is used in the synthesis process to differentiate between symbols of the same id. Since our focus is on element arrangements, we do not synthesize shapes of symbols. Instead, we allow the user to introduce random noises to modify the scale, rotation, and color of each synthesized element so as to get a larger variation [BA06].

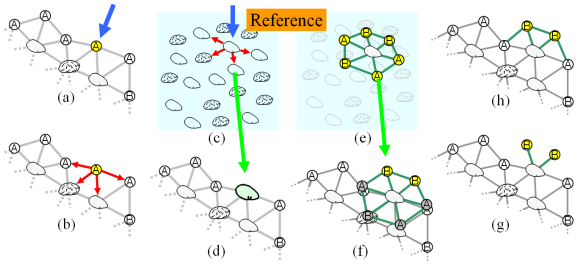

Figure 2: *An overview of the local growth process. Each circle with a symbol id is a seed. The symbol id of the white stone is A and that of the gray stone is B.*

Given a reference pattern, we analyze local relationships between neighboring elements by triangulating the reference pattern. The synthesis process begins with placing a single seed. We iteratively replace a seed with a reference element and several new seeds, similarly to the local growth of an L-System [PL90]. In each iteration step, we also construct connectivity for new elements and seeds. Figure 2 shows an example of a single iteration step. The system first chooses a seed (a), checks its neighborhood (b), and finds a reference element which has the most similar neighborhood condition to that of the seed (c). Next the system replaces the target seed with the copy of the found reference element (d). Finally, we get copies of the neighboring elements of the found element (e) (f), place unpopulated parts of them as new seeds (g), and construct edges (h). We also introduce a global relaxation after each growth step in order to get a smooth pattern.

In the local growth process, we introduce three different modes. 1) In the *non-rotation mode* the system uses the reference element without rotation. 2) In the *rotation mode* the system searches the best fitting reference element considering the rotation, resulting in a better distribution. 3) In the *flow field mode* the system rotates the reference to adjust its local coordinate to the user-specified underlying flow field direction. This allows control of the global flow of the synthesized arrangement.

To support creative design activities, we introduce three types of tools for controlling global aspects of the resulting arrangement. 1) The spray tool activates seeds under the cursor area. This tool allows the user to paint the arrangement pattern [IOI06; RLAD06]. Our algorithm is fast enough to return immediate feedback. 2) The flow field tool allows the user to design flow fields by drawing a set of strokes. The system uses the directions of the input strokes as constraints and interpolates 2D space by radial basis functions [EWMZ07]. 3) The boundary tool allows the user to draw a set of boundary strokes which stop the local growth.

4. Analysis

Our approach differs from those used in the texture synthesis that deals with pixels aligned on a grid. Our target is an element arrangement, in which elements are not supposed to be aligned. Thus, we need to consider a "spatial" neighborhood characteristic of each element.

To extract such a neighborhood feature, we use a similar idea to the one presented by Barla et al. [BBT*06]. Given an input reference arrangement of elements, we extract the connectivity by Delaunay triangulation. We use the center position of each element. Delaunay triangulation often generates skewed triangles around the boundary, thus we remove skewed triangles which have angles greater than $2/3*\pi$ and we keep only edges which are a part of at least one un-skewed triangle. We then register relative positions, ids, and sub-ids of immediate neighborhoods of each element. We do not use elements around the boundary of the reference arrangement, since they do not have enough neighborhood elements.

When regular or near regular arrangements are input, Delaunay triangulation often generates undesired connec-

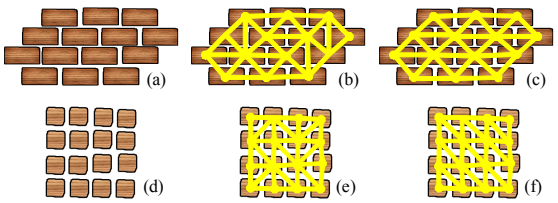

Figure 3: *The system automatically generates connectivity (b) (e) for user input reference arrangement (a) (d). The user can manually modify the topology to reconstruct underlying regularity (c) (f).*

tions. Figure 3 shows examples. In the top row, several generated edges connect blocks over one row (b). In the bottom row a near-regular arrangement is input (d); however generated edges have no regularity (e). Since our synthesis algorithm is strongly affected by local connectivity of each element, these connections often break the regular pattern. Thus, we allow the user to correct the connection after the triangulation. The user can flip an edge by clicking it (c) and (f). This simple interface is enough to get desired connectivity in our experience.

5. Synthesis by local growth

We begin with an element at the center of the pattern and expand it outward by placing a new element one by one based on neighborhood comparisons on the previously synthesized elements. However, our target element arrangements are not always supposed to have regular structures as in the case of pixels in a grid. The position for placing a new element and its neighborhoods is unclear. Also, the size and orientation of each element may differ. These are major differences between the texture synthesis technique and our method. To tackle this problem, we apply a procedural system that defines a local growth of seeds based on connectivity of elements [PL90]. In each growth step, we not only replace a seed with the best fitting reference element, but we also place new seeds by copying the immediate neighborhood of the chosen reference element.

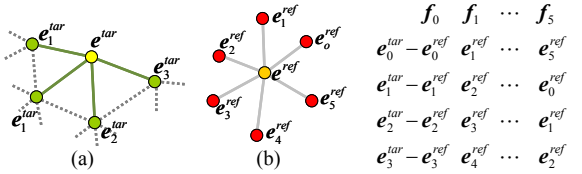

Figure 4: *An example of matching patterns. The system constructs a set of matching patterns between neighbors of the target seed (a) and that of a reference element (b).*

5.1 Seeding

The seed is a candidate position of a new element. Optionally a seed has a symbol id associated with it. This id forces the seed to become a certain element but it could be any element of the same id from the reference arrangement.

We begin the synthesis process by placing a single seed. We sort seeds by the distance from the initial seed and choose the nearest one in each growth step. In the spray tool mode, we generate an initial seed at the cursor position when the user starts painting and grow all the seeds which are in the user-specified distance from the cursor during painting.

There are some arrangement patterns in which elements have different importance. Wong et al. [WZS98] designed floral ornament by placing the larger elements at first and filling spaces with the smaller elements later. We mimic this effect by specifying the priority of seeds based-on their symbol ids. The system only grows seeds with the highest or same priority in their neighboring area (i.e. we define the neighboring area as a circle with the radius 5 * l, where l is the average length of the edges in the reference arrangement). For example, in Figure 1 (b), we specified the higher priority to the gray stones and white stones, and the lower priority to the small stones which we used as fillers only.

5.2 Finding the Best Matching Element

Let $\boldsymbol{e}^{tar}$ be a selected seed in the synthesized pattern and $\boldsymbol{w}(\boldsymbol{e}^{tar})$ be a set of its immediate neighborhood elements. Similarly we note $\boldsymbol{e}^{ref}$ and $\boldsymbol{w}(\boldsymbol{e}^{ref})$ for reference elements (Figure 4). We define an error function based on differences between $\boldsymbol{w}(\boldsymbol{e}^{tar})$ and $\boldsymbol{w}(\boldsymbol{e}^{ref})$, and examine all reference elements so that find one which minimize the error function:

$$\min_{\boldsymbol{e}^{ref} \in reference} ERROR(\, \boldsymbol{w}(\boldsymbol{e}^{tar}), \boldsymbol{w}(\boldsymbol{e}^{ref}) \,). \qquad (1)$$

Note that if the target seed has a symbol-id we examine only corresponding reference elements.

Neighborhood comparisons for the element arrangement are more complex than for the texture synthesis because the number and position of neighboring elements vary depending on distributions. We compute the difference in two steps, similar to those in [BBT*06]. First, we construct a matching $\boldsymbol{f}$ which matches each element of the target neighborhood to an element of reference neighborhoods. There are several candidate matching patterns so we need to construct a set of matching patterns $\boldsymbol{F}$. Second we calculate differences for the all relevant pairs and take summation of them with respect to each $\boldsymbol{f} \in \boldsymbol{F}$.

To build a set of matching patterns $\boldsymbol{F}$, we consider only cases which have no skipping and flipping. We first sort elements $\boldsymbol{e}_i^{tar} \in \boldsymbol{w}(\boldsymbol{e}^{tar})$ and $\boldsymbol{e}_j^{ref} \in \boldsymbol{w}(\boldsymbol{e}^{ref})$ in counter-clockwise order. For target neighbors, we start with the element which is on the boundary (Figure 4a). We choose $\boldsymbol{e}_0^{tar}$ and $\boldsymbol{e}_0^{ref}$ as an initial pair, and match the subsequent elements $\boldsymbol{e}_i^{tar}$ and $\boldsymbol{e}_i^{ref}$ incrementally, so as to obtain one correspondence $\boldsymbol{f}_0$. In the similar manner we obtain $\boldsymbol{f}k$ by choosing $\boldsymbol{e}_0^{tar}$ and $\boldsymbol{e}_k^{ref}$ as an initial pair. In the case of Figure 4, we obtain 6 matching patterns. This is different from the approach in [BBT*06] where they choose the nearest element based on the distance and they can skip some neighbors in between.

We assumed that the size of $\boldsymbol{w}(\boldsymbol{e}^{tar})$ is equal to or less than that of $\boldsymbol{w}(\boldsymbol{e}^{ref})$, because the target seed is under growth and the number of its neighboring elements is relatively small. However, we occasionally observe exceptional cases. For instance, all neighboring elements of the seed have already been placed due to the difference of the priority or the use of the spray tool. When the size of $\boldsymbol{w}(\boldsymbol{e}^{tar})$ is greater than $\boldsymbol{w}(\boldsymbol{e}^{ref})$ we simply ignore such reference elements. These heuristics work well in practice.

Once we obtain a set of matching patterns, we can define the error function with respect to each $\boldsymbol{f} \in \boldsymbol{F}$ as follows:

$$\boldsymbol{error}(\boldsymbol{w}(\boldsymbol{e}^{tar}),\boldsymbol{w}(\boldsymbol{e}^{ref})) = \sum_{i,j\in\boldsymbol{f}} w_1\,\boldsymbol{d}(\boldsymbol{e}_i^{tar},\boldsymbol{e}_j^{ref}) + w_2\,\boldsymbol{id}(\boldsymbol{e}_i^{tar},\boldsymbol{e}_j^{ref}) + w_3\,\boldsymbol{subid}(\boldsymbol{e}_i^{tar},\boldsymbol{e}^{ref}). \quad (2)$$

The $\boldsymbol{d}(\boldsymbol{e}^{tar}, \boldsymbol{e}^{ref})$ is a Euclidean distance between the relative positions of $\boldsymbol{e}^{tar}$ and $\boldsymbol{e}^{ref}$. The $\boldsymbol{id}(\boldsymbol{e}^{tar}, \boldsymbol{e}^{ref})$ returns 1 if the symbol id of $\boldsymbol{e}^{tar}$ is different from that of $\boldsymbol{e}^{ref}$, or otherwise it returns 0. These two terms measure the local spatial relationship. The $\boldsymbol{subid}(\boldsymbol{e}^{tar}, \boldsymbol{e}^{ref})$ is used to avoid the same element recursively appearing again and again. This function returns 1 if sub-ids of $\boldsymbol{e}_i^{tar}$ and $\boldsymbol{e}^{ref}$ are the same, or returns 0. w_1, w_2, and w_3 are weighting values (e.g. if the weight w_1 is large the structure of the reference arrangement is better preserved in the resulting patterns.). We set $w_1 = 1.0$, $w_2 = 1.0$, and $w_3 = 100.0$ to obtain the results in this paper. Finally we can define the error function:

$$ERROR(\boldsymbol{w}(\boldsymbol{e}^{tar}),\boldsymbol{w}(\boldsymbol{e}^{ref})) = \min_{\boldsymbol{f}\in\boldsymbol{F}} \boldsymbol{error}(\boldsymbol{w}(\boldsymbol{e}^{tar}),\boldsymbol{w}(\boldsymbol{e}^{ref})). \quad (3)$$

5.3 Rotation Modes

We introduce three modes to generate different results for different purposes; the non-rotation mode generates patterns that preserve the orientation of the reference neighborhood, the rotation mode changes the orientation of the reference neighborhood and adjusts it to the local target neighborhood, resulting in a better distribution, and the flow field mode generates patterns following the user specified flow field. We achieve these effects by slightly changing the process of the finding best matching elements (Figure 5).

In the non-rotation mode, we find the best fitting element without rotating the reference pattern; we solve the error function (1) as it is. In the rotation mode, we consider the best fitting rotation when calculating the error function (2). In other words we minimize the positional difference by rotating the reference elements;

$$\min_{\theta} \sum_{i,j\in\boldsymbol{f}} \boldsymbol{d}(\boldsymbol{e}_i^{tar}, \boldsymbol{R}_\theta(\boldsymbol{e}_j^{ref})), \quad (4)$$

where $\boldsymbol{R}_\theta(\boldsymbol{e}_j^{ref})$ rotates the relative position of $\boldsymbol{e}_j^{ref}$. We apply the shape matching problem introduced in [MHTG05]. In the flow field mode, we consider the user specified flow field when finding the best match. We first obtain the orientation of the flow field at the target seed position. We then rotate the whole reference pattern so as to adjust its local x coordinate to the orientation. We use this rotated reference when finding the best match.

5.4 Local Growth

Now we found the best fitting reference element, and optionally the best fitting rotation in the rotation mode or the flow field rotation in the flow field mode.

The system first replaces the target seed with the found reference element (Figure 2d), including the symbol id and the sub-id. The system next places the new seeds by using the neighboring elements of the found element (these elements form the so called *ring shape*). We overlay a copy of the reference ring shape with the target seed neighborhood (Figure 2f), and then we delete every neighboring element which was paired to a target neighboring element when calculating the error function. The remaining elements will form new seeds with corresponding symbol ids (Figure 2g). Finally, we construct edges to connect the new seeds.

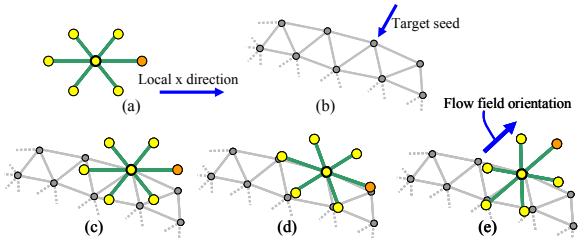

Figure 5: *The system fits a reference element (a) to a target seed (b) differently depending on three rotation modes; non-rotation (c), rotation (d), and flow field mode (e).*

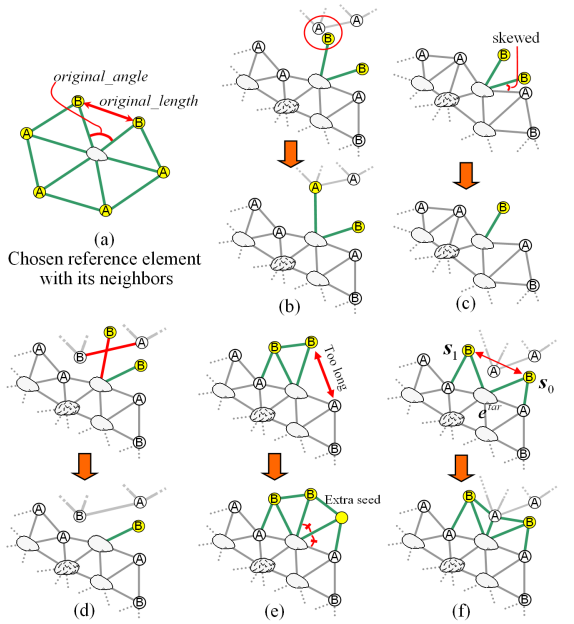

Figure 6: *The heuristic approach used to avoid undesired structures.*

To avoid too dense or sparse distribution or collisions of edges, we introduce several heuristics during the local growth process (Figure 6). After obtaining new seeds, we check the neighboring area of each new seed. If we find an existing element or seed in a distance l^{short} from the seed, we delete the seed and connect the edge to the found object (b). We choose a half of the shortest edge length in reference as l^{short}. We next check the angle between new edges, if the angle is less than 0.5 * *original_angle*, we delete the new seed and edge (c). The *original_angle* is the corresponding angle in the chosen reference ring shape (a). We also check for a collision of the edge of the new seed; if it is detected, we remove the seed and the edge (d). When constructing new edges around new seeds, we check the length of each edge. If an edge is longer than l^{long}, we generate an extra seed without a corresponding symbol id (e).

We place the seed at the end of a line segment which bisects the corresponding angle and whose length is the average of two adjacent edges (e). We define l^{long} = 1.5 * *original_length*, where the *original_length* is the length of the corresponding edge in the ring shape (a). Then we check the edge collisions again. If a collision is detected, one or more objects must be in a triangle constructed by $\boldsymbol{e}^{far}$, s0 and s1, and we construct edges as shown in (f).

5.5 Relaxation

Although we place a reference element with its neighbors that have the most similar shape to the target neighbors, the error accumulates in each local growth step because the system cannot always find an optimally fitting element which makes the error value zero. Here, we introduce a relaxation process that modifies the positions of the previously synthesized elements so as to keep the local feature of the synthesized pattern as similar as possible to that of the reference pattern. For each synthesized element, we adjust its ring shape to that of the corresponding reference element.

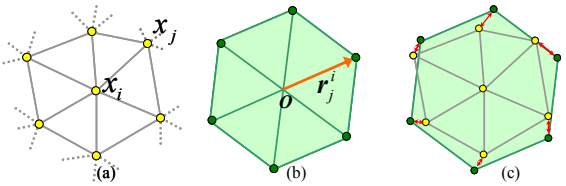

Figure 7: *We fit an immediate neighborhood shape of each vertex (a) to its desired shape (b). We minimize a gap between each vertex and its desired position (c).*

Let $\boldsymbol{x}_i$ be a position of a synthesized element $\boldsymbol{e}_i^{ref}$, and $\boldsymbol{w}(\boldsymbol{x}_i)$ be a set of its neighboring positions. We also refer to the relative position of the corresponding reference ring shape as $\boldsymbol{r}_j^i$, where $\boldsymbol{r}_j^i$ is corresponding to $\boldsymbol{x}_j \in \boldsymbol{w}(\boldsymbol{x}_i)$ (Figure 7(a) and (b)). We then minimize a distance between each neighbor's positions $\boldsymbol{x}_j$ and the position of the corresponding reference ring shape (Figure 7c):

$$\min_{\boldsymbol{x}} \sum_i \sum_{j \in \boldsymbol{w}(\boldsymbol{x}_i)} \| \boldsymbol{x}_j - (\boldsymbol{x}_i + \boldsymbol{r}_j^i) \|^2. \qquad (5)$$

Since this representation is translation invariant, we should predetermine the position of one element in order to have a unique minimizer; we simply hold the initial element at the initial position. If there are user-specified boundaries, we constrain elements at their generated positions.

$$\boldsymbol{x}_k = \boldsymbol{x}_k^0 \quad k \in constraints. \qquad (6)$$

This quadratic minimization problem with linear constraints can be solved in closed form. We apply the Lagrange multiplier method. Note that there are elements which lose an edge or get an extra edge during growth. These elements no longer have same topology as the corresponding reference element. All seeds do not have corresponding reference elements yet. For these objects, we use their current neighboring shapes as their target ring shapes.

6. Results and Discussion

Synthesis is fast enough to return immediate feedback when the user paints using the spray tool. It took about 5 seconds to synthesize 1000 elements on a Windows XP notebook PC with Intel Core 2 Duo CPU. The bottleneck of the synthesis is the relaxation process; it took less than a second to synthesize 1000 elements without the relaxation.

Figure 8 shows synthesized arrangements with reference patterns in the three different classes; a regular pattern (a), a near regular pattern (b), and an irregular uniform distribution (c). In the case of (b), we introduced small randomness to the scale, rotation, and color of each symbol. Since our system attempts to keep local spatial relationship and topology between neighbors, it can cover these three classes in the same framework. Figure 8 (d) is a pattern synthesized by the rotation mode. This example indicates that our system successfully synthesizes a pattern in which gray stones are linked each other in rows.

Figure 9 illustrates different effects of three rotation modes. The system takes the same input example (a) and generates different results in non-rotation (b) and rotation (c) mode. In both case, two gray stones appear close to each other. While an orientation of each two stones is retained in the non-rotation mode, it is locally rotated in the rotation mode. The user can also control the arrangements by specifying underlying flow fields. When the user draws strokes by using the flow field tool (red curves in (e)), the system generates flow fields along to the strokes (e). The system then takes this flow fields as well as a reference arrangement (d), and it synthesizes a new pattern (f) in which scales are aligned along to the flow. Our system can change the topology to keep the uniform distribution. Two scales at which topology changes happen are highlighted by red circles (f). We show the resulting connectivity in (g).

Finally, we show results created by using three UI tools. In Figure 1 (d), we control the orientation of feathers by using the flow field tool and the growth area using the spray tool. In Figure 1 (f), we limit the growth area of the blocks by the boundary tool. We also painted four characters "Euro" with an arrangement pattern by using the spray tool and flow field tool in Figure 10. In these feathers and scales examples, we arranged 3D objects which have a rounded shape and are slightly slanted, so that they naturally express an overlapping effect. These effects are difficult to create with a 2D vector based approach [BBT*06] because it is required to specify the rendering order of elements explicitly. We also have a capability to modify the local orientation or size of each element after obtaining the arrangement. This type of control is difficult for pixel based texture synthesis.

7. Conclusion and Future Work

We presented a new system for synthesizing 2D element arrangements by combining non-parametric texture synthesis techniques based on local neighborhood comparison and procedural modeling systems based on local growth. Our

system takes a user-input reference arrangement, analyses local relationship between reference elements by constructing connectivity and synthesizes a larger pattern by locally growing a seed one by one. Since we attempt to keep the connectivity (i.e. topology) between neighboring elements, our system can generate various regular patterns and irregular uniform patterns. To support creative design activities, we present three UI tools which allow control of global features of the synthesis. We also introduce the relaxation process in order to adjust a ring shape of each synthesized element to be closer to that of the corresponding reference element.

Our current system considers only immediate neighborhoods, so we would like to extend our matching function to handle more than 1-ring elements in the future. While our flow field mode generates fine results with relatively simple flow fields, if complicated flow fields containing several peaks or edge lines are given, the system often fails to keep the local topology at the peaks or the edges and breaks local spatial relationships. Another limitation is local error concentration. Even if we introduce the relaxation, our synthesis algorithm is based on local best matching, thus it is difficult to obtain globally optimized results. We believe that this problem can be solved by hierarchical representation introduced in texture synthesis frameworks [WL00; Tur01]. Future work might include more complicated patterns such as tree structures [PL90; WZS98] or 2D arrangements which vary the scaling of each element.

Finally we would like to emphasize that our approach is one of the first trials to combine the example based system and procedural modeling system. We hope that our technique could be a first step towards an answer the problem of inverse procedural modeling, which is one of the biggest problems in procedural modeling research.

References

[Ash01] ASHIKHMIN, M.: Synthesizing Natural Textures. In *Proc. the Symposium on Interactive 3D Graphics 2001,* 217-226.

[BA06] BAXTER W., ANJYO K.: Latent Doodle Space. *Computer Graphics Forum*, 25 (2006), 3, 477-485.

[BBT*06] BARLA P., BRESLAV S., THOLLOT J., SILLION F., MARKOSIAN L.: Stroke Pattern Analysis and Synthesis. *Computer Graphics Forum*, 25 (2006), 3, 663-671.

[CSHD03] COHEN, M. F., SHADE, J., HILLER, S., DEUSSEN, O.: Wang tiles for image and texture generation. *ACM Trans. Graph.,* 22(2003), 3, 287-294.

[DHOS00] DEUSSEN O., HILLER S., VAN OVERVELD C., STROTHOTTE T.: Floating points: A method for computing stipple drawings. *Computer Graphics Forum*, 19 (2000), 3, 40-51.

[DL99] DEUSSEN O. AND LINTERMANN, B.: Interactive Modeling of Plants. *IEEE Computer Graphics and Applications*, 19 (1999), 1, 56-65.

[EL99] EFROS A. A., LEUNG T. K.: Texture synthesis by nonparametric sampling. In *IEEE Int. Conf. on Computer Vision* (1999), 1033-1038.

[EWMZ07] ESCH G., WONKA P., MÜLLER P., ZHANG E.: Interactive procedural street modeling, *SIGGRAPH 2007 Sketches*.

[HB95] HEEGER, D. J., BERGEN, J. R.: Pyramid-based texture analysis and synthesis. In *Proc. SIGGRAPH '95* (1995), 229-238.

[HJO*01] HERTZMANN, A., JACOBS, C. E., OLIVER, N., CURLESS, B., SALESIN, D. H.: Image Analogies. In *Proc. SIGGRAPH '01*(2001), 327-340.

[HOCS02] HERTZMANN A., OLIVER N., CURLESS B., SEITZ S. M.: Curve analogies. In *Proc. the 13th Eurographics Workshop on Rendering*(2002), 233-246.

[IOI06] IJIRI T., OWADA S., IGARASHI T.: The sketch L-System: global control of tree modeling using free-form strokes. In *Proc. SmartGraphics 2006*, 138-146.

[JEG*02] JODOIN P. M., EPSTEIN E., GRANGER-PICHE M., OSTROMOUKHOV V.: Hatching by example: a statistical approach. In *Proc. NPAR 2002*, 29-36.

[KSE*03] KWATRA, V., SCHODL, A., ESSA, I., TURK, G., BOBICK, A.: Graphcut textures: Image and video synthesis using graph cuts. *ACM Trans. Graph.* 22 (2003), 3, 277-286.

[Lin68] LINDENMAYER A.: Mathematical models for cellular interactions in development, I & II. *Journal of Theoretical Biology* 18, 3 (1968), 280-315.

[Llo82] LLOYD S. P.: Least squares quantization in pcm. *IEEE Trans. Information Theory* 28, 2(1982), 129-137.

[LD05] LAGAE A., DUTRE P.: A procedural object distribution function. *ACM Trans. Graph.*, 24(2005), 4, 1442-1461.

[LLH04] LIU Y., LIN W. C., HAYS J. H.: Near regular texture analysis and manipulation. *ACM Trans. Graph.*, 23(2004), 3, 368-376.

[MHTG05] MULLER M., HEIDELBERGER B., TESCHNER M., GROSS M.: Meshless deformations based on shape matching. *ACM Trans. Graph.*, 24(2005), 3, 471-478.

[MP96] MĚCH R., PRUSINKIEWICZ P.: Visual models of plants interacting with their environment. In *Proc. SIGGRH '96*(1996), 397-410.

[MWH*06] MULLER P., WONKA, P., HAEGLER, S., ULMER, A., GOOL, L. V.: Procedural modeling of buildings. *ACM Trans. Graph.*, 25(2006) 3, 614-623.

[PJM94] PRUSINKIEWICZ P., JAMES M., MĚCH R.: Synthetic topiary. In *Proc. SIGGRAPH '94*(1994), 351-358.

[PL90] Prusinkiewicz P., Lindenmayer A.: *The Algorithmic Beauty of Plants.* Springer–Verlag, New York, 1990. With J. S. Hanan, F. D. Fracchia, D. R. Fowler, M. J. M. de Boer, and L. Mercer.

[PM01] PARISH Y. I. H., MULLER P.: Procedural modeling of cities. In *Proc. SIGGRAPH '01*(2001), 301-308.

[RLAD06] RITTER L., LI W., AGRAWALA M., CURLESS B., SALESIN D.: Painting with Texture. In *Proc. 17th Eurographics Symposium on Rendering* (2006), 371-376.

[SABS94] SALISBURY M. P., ANDERSON S. E., BARZEL R., SALESIN D. H.: Interactive pen-and-ink illustration. In *Proc. SIGGRAPH '94*(1994), 101-108.

[Tur01] TURK G.: Texture Synthesis on Surfaces. In *Proc. SIGGRAPH '01*(2001), 347-354.

[WL00] WEI L-Y., LEVOY M.: Fast Texture Synthesis using Tree-structured Vector Quantization. In *Proc. SIGGRAPH'00* (2000), 479-488.

[WS94] WINKENBACH G., SALESIN D. H.: Computer generated pen-and-ink illustration. In *Proc. SIGGRAPH '94* (1994), 91-100.

[WWSR03] WONKA P., WIMMER M., SILLION F., RIBARSKY W.: Instant architecture. *ACM Trans. Graph.*, 22 (2003), 3, 669-677.

[WZS98] WONG M. T., ZONGKER D. E., SALESIN D. H.: Computer-generated floral ornament. In *Proc. SIGGRAPH '98* (1998), 423-434.

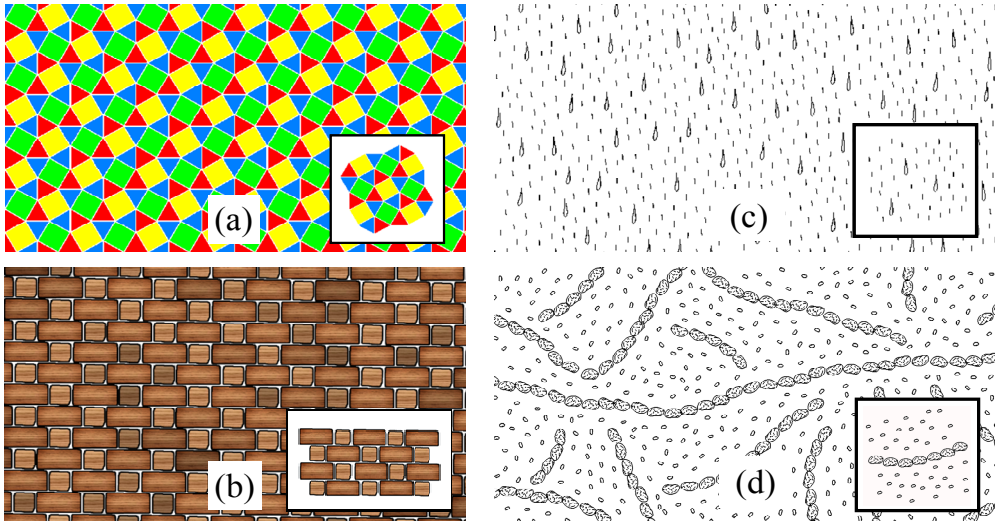

Figure 8: *Synthesized arrangements and input example patterns (right button for each figure).*

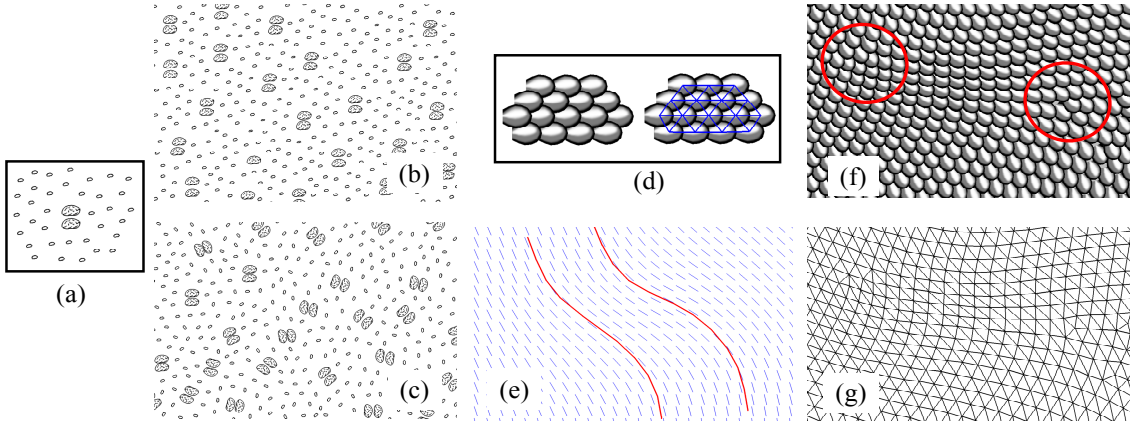

Figure 9: *Effects of three rotation modes. The system takes the same reference pattern (a) and generates different results in the non-rotation (b) and the rotation (c) mode. In these examples, we assign the higher priority to the gray stone. In the flow field mode, the system takes user-specified reference (d) and flow field (e), and synthesizes a pattern along the flow (g). The red circles highlight the positions at which the local topology changes. (g) shows the resulting connectivity.*

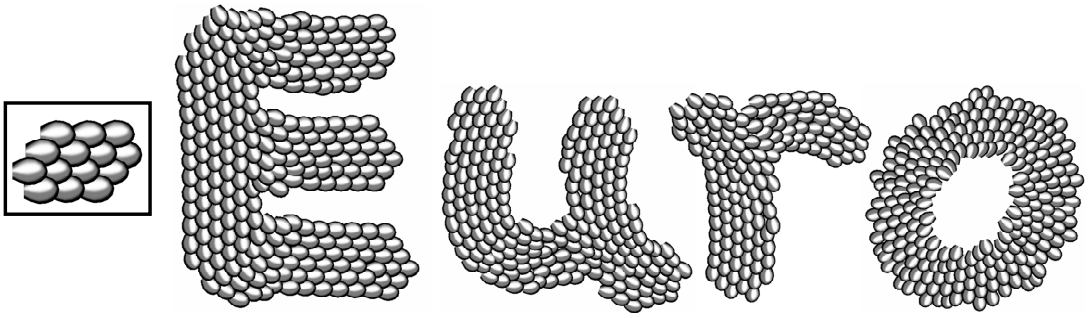

Figure 10: *Four characters "Euro" painted with scales. We first designed the underlying flow fields and then specified the growth area using spray tool.*