Stroke Pattern Analysis and Synthesis

Pascal Barla, Simon Breslav, Joëlle Thollot, François X. Sillion, Lee Markosian

▶ To cite this version:

Pascal Barla, Simon Breslav, Joëlle Thollot, François X. Sillion, Lee Markosian. Stroke Pattern Analysis and Synthesis. Proc. of Eurographics 2006 : Computer Graphics Forum, Sep 2006, Vienna, Austria. inria-00362883

HAL Id: inria-00362883

https://inria.hal.science/inria-00362883

Submitted on 29 Apr 2011

Stroke pattern analysis and synthesis

paper 1072

Abstract

We present a synthesis technique that can automatically generate stroke patterns based on a user-specified reference pattern. Our method is an extension of texture synthesis techniques to vector-based patterns. Such an extension requires (a) an analysis of the pattern properties to extract meaningful pattern elements (defined as clusters of strokes) and (b) a synthesis algorithm based on similarities in the detected stroke clusters.
Our method is based on results from human vision research concerning perceptual organization. The resulting synthesized patterns effectively reproduce the properties of the input patterns, and can be used to fill both 1D paths and 2D regions.

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

1. Introduction

A particularly important class of non-photorealistic renderings is that of stroke-based images. Various styles such as etchings, pen-and-ink and oil painting renderings can be thought of as stroke-based styles as described in Hertzmann's survey [Her03]. The rendered strokes can be either used to fill in 2D regions, as in painterly rendering, or to annotate 1D paths, like with some hatching patterns; in both cases, the generation of appropriate stroke arrangements for these styles remains a difficult or tedious process to date. Since the individual style of each artist has to be conserved but is not easy to translate in an algorithmic representation, we can not simply rely on procedural methods to generate stroke patterns. Finding a compromise between automation and expressiveness is then crucial for such renderings to be used by artists.

Synthesis by example appears to be the best way to address this question. However, pixel-based texture synthesis is not well suited to stroke patterns, in part because each element of a stroke pattern is individually perceptible, in contrast to pixels. Organized stroke clusters such as those found in hatchings are difficult to extract and reproduce at the pixel level. Moreover, some variation in the reproduced pattern is desirable to avoid too much regularity, and it would be difficult to achieve such variation with pixel-based texture synthesis.
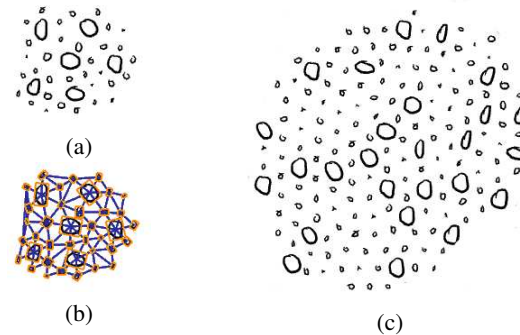

Figure 1: *Our method (a) takes as input a reference vectorized stroke pattern, then (b) analyses it to extract relevant stroke pattern elements and properties in order to (c) synthesize a similar pattern.*

We therefore propose to use a vector-based description of an input stroke pattern supplied by the user. This allows for greater expressiveness and higher-level analysis than would be afforded by a per-pixel approach. The stroke geometry is represented explicitly as connected vertices with attributes such as width and color. While this vector representation is typically less efficient to render, it has the important advantage that strokes can be controlled procedurally to adapt to changes in the depicted regions. Strokes can vary in opacity, thickness and density to depict an underlying tone, as in the WYSIWYG NPR system [KMM*02].

We target any kind of stroke patterns (stippling, hatching, brush strokes, small figures) with a quasi-uniform distribution of positions in 1D and 2D (i.e., along a path or inside a region). The strokes attributes can vary in non-uniform ways and the only parameter required from the user is the scale of the meaningful elements of the pattern. Then, in a manner analogous to texture synthesis techniques, we organise our method in two stages (see Figure 1). An analysis stage where we identify the relevant elements in terms of stroke patterns and their distribution, and a synthesis stage where these elements are placed in the image so as to reproduce an appearance similar to the reference pattern.

In the following we first review the previous work related to our goal, then explain the two stages of our method: analysis and synthesis, and last present our results and discuss future work.

1.1. Previous work

Related work can be found in two different research fields: stroke-based rendering methods that aim at reproducing various artistic styles, and texture synthesis methods that aim at generating a texture from a given sample pattern.

1.1.1. Stroke synthesis

Stroke pattern synthesis systems have been studied in the past, for example to generate stipple drawings [DHvOS00], pen and ink representations [SABS94, WS94], engravings [Ost99], and painterly rendering [Her98]. However, they have relied primarily on generative rules, either chosen by the authors or borrowed from traditional drawing techniques. We are more interested in analysing reference patterns drawn by the user and synthesizing new ones with similar perceptual properties in order to give the user as much freedom as possible in the choice of his own style.

Kalnins *et al.* [KMM*02] described an algorithm for synthesizing stroke "offsets" (deviations from an underlying smooth path) to generate new strokes with a similar appearance to those in a given example set. Hertzmann *et al.* [HOCS02], as well as Freeman *et al.* [FTP03] address a similar problem. Neither method reproduces the interrelation of strokes within a pattern. Jodoin *et al.* [JEGPO02] focus on synthesizing hatching strokes, which is a relatively simple case in which strokes are arranged in a linear order along a path. The more general problem of reproducing organized patterns of strokes has been adressed by the authors in the case of hatching and stippling using a statistical approach [Ano06]. (See the submitted version in supplemental material.) No neighborhood comparison was taken into account and the class of possible elements was reduced to single-colored points or lines. Here we target a wider range of patterns (including color patterns) and take into account the global organization of the pattern by means of neighborhood comparisons.

1.1.2. Texture synthesis

The idea of synthesizing textures, both for 2D images and 3D surfaces, has been extensively addressed in recent years. Previous work can be organized in two categories: parametric and non-parametric methods.

Parametric methods aim at giving a compact description of textures: they make use of statistical analysis to characterize an input texture by a set of parameters, and then try to synthesize similar textures in order to validate the parametric model. The reader can refer to the paper by Portilla and Simoncelli [PS00] for a good overview of parametric models. Our previous work on stroke pattern synthesis [Ano06] bears similarities to parameteric methods in that it performs a statistical analysis of properties of the input pattern (such as stroke positions, lengths, and orientations). Such methods are hard to extend to general patterns because the parameters depend heavily on the style and structure of the pattern.

On the other hand, non-parametric methods work exclusively from the reference texture [EL99, WL00, WL01, Tur01, Ash01]. Even if this representation is less compact, the synthesis results are usually more convincing and the type of synthesized textures more general. These iterative algorithms work with neighborhood comparisons between the reference and target textures. The size and shape of the neighborhoods vary from one technique to the other. Some methods work in scanline order, while others grow the texture from a central starting point; Synthesis is sometimes hierarchical. Still, one cannot directly apply pixel-based non-parametric texture synthesis to vector-based patterns, because stroke patterns are composed of individually perceived elements. We thus choose to adapt an existing non-parametric method [EL99] to vector data.

1.2. Contributions

To define a stroke pattern we draw upon research in human vision, more specifically in the field of perceptual organisation. Indeed, there is a common agreement that the human visual system, in the early stages of perception, structures 2D information into elements based on a set of criteria such as proximity, parallelism, and continuation [Pal99]. In the case of stroke patterns, this means that some sets of input strokes are perceived as single elements, and at a higher level, the distribution of elements defines the pattern.

Our first contribution is an analysis method that extracts an intermediate-level description of vector data, using perceptual organisation criteria applied to input strokes. In particular, we are able to analyse strokes of different styles: not only stippling and hatching strokes, but also brush strokes and small figures (see Figure 5 and Figure 6).

Our second contribution is a synthesis method analogous to texture synthesis, but operating on vector data. We propose a perceptually-based neighborhood matching algorithm that

allows comparisons between neighborhoods even when they have dissimilar connectivity.
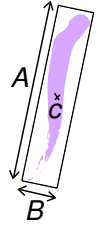
2. Analysis

The first step of our method aims at analyzing a reference stroke pattern to extract the meaningful elements that constitute the pattern, as well as their distribution. We define an element as a cluster of strokes that is perceived as a single feature by the user. Since we target a quasi-uniform distribution of elements, there is a characteristic scale of the pattern that gives the size of such elements. We believe that the choice of scale is context-dependent, and we let the user specify it. Note, however, that the whole analysis process is interactive, hence providing the user enough feedback to easily set a convenient scale. This allows us to target a wide range of patterns as shown in Section 4.

Once the scale is chosen the rest of the process is fully automatic and works as follows. We first fit an element to each input stroke; we then cluster elements iteratively using perceptual organisation criteria; finally, we relate elements to each other to derive properties about their distribution.

2.1. Element fitting

We define an element by its center and its two elongation axis. In order to fit an element to a stroke, the user can either choose to only consider the skeleton of the stroke (i.e. the gesture input by the user) or to also take into account its style (thickness, fading, tapering, etc). In the first case, we fit an element to the points that define the skeleton, while in the second case, we fit an element to the points of its contour.

An element E is constructed by fitting an oriented bounding box to the chosen set of points (skeleton or contour) that will prove useful for approximating geometric measures between elements. We first fit a gaussian distribution to the points to compute the two principal elongation directions given by the eigenvectors A and B of the points' covariance matrix. Each point is then projected onto each axis to compute the size and center c of the bounding box.

2.2. Element clustering

Now that each stroke is represented by its bounding box, we want to cluster them into elements at the chosen scale. For example, a hatch element can be made of several overlapping strokes, and a small figure (think of a flower) is often drawn using a small number of individual strokes.

To decide whether two elements can be clustered, we draw upon the work of Etemadi *et al.* [ESM*91] on perceptual line segment grouping, as we did in our previous work [Ano06],

but this time extended to bounding box elements. Two elements are clustered if they meet a proximity constraint (e.g., for a flower), or if they meet a continuation constraint (e.g., for a hatch element). These tests are performed against the user-defined scale ε, that represents the minimum distance at which two elements are perceived separately. When two elements are clustered, we merge their respective strokes and fit a new element using the method explained in the previous section.

Clustering is performed by a greedy algorithm that processes the strokes in the order they have been drawn. The fitted elements are first placed into a queue. At each step, an element E^* is popped and every other element E_i in the queue is tested for clustering based on an element pair comparison. If the test is successful, we merge E_i into E^* and remove E_i from the queue. After all the elements of the queue have been tested, if any clustering occured, E^* is pushed back into the queue. Otherwise, E^* is added to the output list of clustered elements. The algorithm repeats until the queue is left empty.
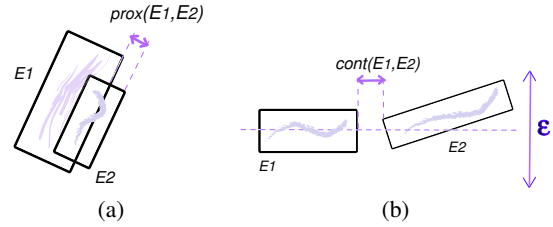

Figure 2: *Element clustering uses two perceptual measures: (a) Proximity and (b) Continuation.*

2.2.1. Proximity measure

The proximity between two elements E_1 and E_2 is computed using Hausdorff distances so that nested, or very close objects are clustered together (see Figure 2(a)):

$$prox(E_1, E_2) = \min(d_H(E_1, E_2), d_H(E_2, E_1))$$
$$d_H(E_1, E_2) = \max_{q_1 \in E_1} \left(\min_{q_2 \in E_2} (d(q_1, q_2)) \right)$$

where $d_H(E_1, E_2)$ is the directed Hausdorff distance. In practice, it is computed using point-line distances between the bounding boxes. If $prox(E_1, E_2) < \varepsilon$, then E_1 and E_2 are clustered.

2.2.2. Continuation measure

Continuation has to be checked on all pairs of axes between the two elements E_1 and E_2. For each of the four configurations, we first have to ensure that the elements are near-collinear; then we compute a continuation measure. Without loss of generality, we only consider the measures of E_2 relative to the axis A_1 of E_1. E_2 is near-collinear to E_1 iff:

$$\forall p \in E_2, d(p, A_1) < \varepsilon/2$$

If the above condition is met, then a continuation error is computed as the gap between the projected points of E_1 and E_2 on A_1 (see Figure 2(b)):

$$cont(E_1, E_2) = \min_{p \in E_1^*, q \in E_2^*} (d(p, q))$$

where E_i^* is the set of points of E_i projected on A_1, $i = 1, 2$. In practice, we also use the bounding boxes to speed up this computation. If $cont(E_1, E_2) < \varepsilon$ for any of the four configurations, then E_1 and E_2 are clustered.

2.3. Element distribution

Having identified the elements of our reference pattern, we can now extract connectivity information among them in order to characterize their distribution (see Figure 1(b)). We use the center position of each element. For 1D patterns, we extract the neighbors along a chain, while for 2D patterns, we extract a Delaunay triangulation and keep only edges that are part of at least one unskewed triangle (i.e. a triangle that do not has an angle greater than $\frac{2\pi}{3}$). The pattern input by the user is supposed to be uniform, but in practice, the distribution of elements is only close to uniform. We measure locally this variation by computing a shift vector S_{ref} that expresses the displacement between an element's position and the barycenter of its neighbors' positions. We will use those measurements to add variation to a synthesized pattern in Section 3.3.

3. Synthesis

Thus far, we analysed the reference stroke pattern in order to get a higher-level, perceptually meaningful description of it. In this section, we show how to take advantage of this knowledge during synthesis. Since we want to address the synthesis of texture-like patterns, it makes sense to take inspiration from the texture synthesis litterature. In our approach, we draw comparisons with Efros and Leung's pioneering paper [EL99]: like them, we use a causal synthesis procedure that starts with an element at the center of the pattern and expands it outward using neighborhood comparisons on the previously synthesized elements.

Our method exhibits some important differences though: contrary to the distribution of pixels on a grid, our element positions are not supposed to be aligned. This has an impact on the neighborhood comparison procedure that has to match relevant neighbors between the reference and target patterns. Moreover, elements are easily identifiable and perceived in isolation, thus their comparison should consider the whole set of their parameters: orientation, length, width and color. To do that, we draw inspiration from the field of perceptual organisation once again and show how a trade-off between variation and fidelity can be obtained. Algorithm 1 summarizes the synthesis process.

Algorithm 1 Stroke pattern synthesis

$D_{tar} \leftarrow$ InitialiseDistribution(D_{ref})
$E_{tar}^s \leftarrow$ GetCenterElement(D_{tar})
for each element E_{tar} in D_{tar} growing outward from E_{tar}^s
do
$\quad E_{ref}^* \leftarrow$ FindBestMatch($E_{tar}, D_{tar}, D_{ref}$)
$\quad E_{tar} \leftarrow$ SynthesizeElement($E_{ref}^*, D_{tar}, D_{ref}$)
end for

We first present our neighborhood comparison in Section 3.1, before describing in Section 3.2 how it is applied iteratively to create the target pattern. Finally, in Section 3.3, we explain how the user can add variation to the synthesized pattern while keeping a strong similarity with the reference.

3.1. Synthesizing one element

Let E be an element in the synthesized pattern and assume for the moment that all elements in the pattern except for E are known. Let $\omega(E)$ be a neighborhood around E. To assign properties to E, as in [EL99], a set of neighborhoods Ω similar to $\omega(E)$ is extracted from the reference pattern using various perceptual measures described below. Then one of the neighborhoods in Ω is randomly chosen and the center element of the picked neighborhood is used for E.

Neighborhood comparisons are more complex for stroke pattern synthesis than for texture synthesis for two reasons: the number and position of neighbors vary in our distributions, and elements are more complex entities than pixels. The computation of the similarity between two neighborhoods ω_{ref} and ω_{tar} is thus performed in two steps. First, relevant elements of this pair of neighborhoods are found. Second, a set of perceptual organisation measures is tested against perceptual similarity thresholds for the whole candidate reference neighborhood.
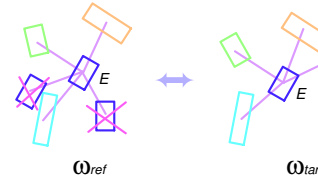

Figure 3: *Neighborhood comparison: Pairs of relevant elements are extracted based on their position.*

For the determination of relevant elements, we only consider pairs of closest reference and target elements by comparing their positions, see Figure 3. This heuristic locally matches the distribution of element positions between the reference and target patterns. We keep pairs of elements $E_{ref} \in \omega_{ref}$

and $E_{tar} \in \omega_{tar}$ such that

$$E_{tar} = \arg \min_{E \in \omega_{tar}} (d(E_{ref}, E))$$

$$E_{ref} = \arg \min_{E \in \omega_{ref}} (d(E_{tar}, E))$$

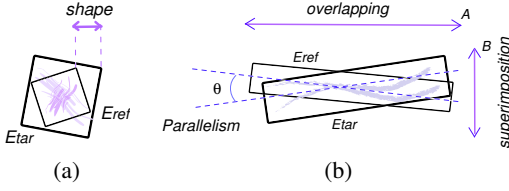where $d(E_1, E_2)$ is the euclidean distance between the centers of E_1 and E_2.

Figure 4: *Element neighborhood matching uses various perceptual measures: (a) isotropic elements; (b) anisotropic (elongated) elements.*

Once the elements are matched, we compute a set of four different perceptual measures that can be organized in two categories (see Figure 4): a *shape-matching* measure that compares two elements as point sets by computing a symmetric Hausdorff distance; and a set of three measures - *parallelism, overlapping and superimposition* - that compares elements using the higher-level description extracted during the analysis. The shape-matching measure is useful when there is an ambiguity between the principal and secondary axes of elongation of an element (e.g., a circle).

Each measure is computed independently on each pair of elements in their respective frame; i.e., the element centers are first aligned before the following measures take place:

$$shape(E_{ref}, E_{tar}) = \max(d_H(E_{ref}, E_{tar}), d_H(E_{tar}, E_{ref}))$$

$$par(E_{ref}, E_{tar}) = |\theta(A_{ref}, A_{tar})|$$

$$ov(E_{ref}, E_{tar}) = \max(\frac{|A_{ref}|}{|A_{tar}|}, \frac{|A_{tar}|}{|A_{ref}|})$$

$$sup(E_{ref}, E_{tar}) = ||B_{ref}| - |B_{tar}||$$

For the computation of the shape measure, we approximate the directed Hausdorff distance d_H using bounding boxes as previously. The parallelism measure is simply taken to be the norm of the angle θ between the two elements' principal axes. Overlapping is the maximum ratio of lengths between the target and reference principal axes. And superimposition is the difference in thickness (length of the secondary axis) between the target and the reference.

In addition to these geometric measures, any attribute can also be taken into account during the synthesis. We illustrate this ability with a simple color distance:

$$col(E_{ref}, E_{tar}) = d_{RGB}(C_{ref}, C_{tar})$$

where C_{ref} and C_{tar} are the colors of E_{ref} and E_{tar} in RGB.

All the measures are then averaged over the element pairs

of ω_{ref} and ω_{tar} to give a set of perceptual measures between neighborhoods. They are then tested against a set of perceptual thresholds. These thresholds control the amount of selected candidate neighborhoods: they have to be sufficiently large to provide enough candidates, but small enough to avoid incoherences. In our experiments, we use $\sigma_{shape} = 0.1L$ where L is the average length of the reference elements, $\sigma_{par} = \frac{\pi}{20}$, $\sigma_{ov} = 1.5$, $\sigma_{sup} = 0.1L$ and $\sigma_{col} = 0.15$. We observed that our algorithm is robust to small variations in these thresholds. Two neighborhoods are then considered similar relative to a given measure m iff $m(\omega_{ref}, \omega_{tar}) < \sigma_m$.

The measures are finally combined to determine the similarity of the candidate reference neighborhood to the target one. If the colors or any other attribute does not match, we simply discard the matching. Otherwise, we test whether the neighborhoods match by considering them as sets of points or sets of elements. In our approach, we thus use the following combination

$$col \text{ and } (shape \text{ or } (par \text{ and } ov \text{ and } sup))$$

In Section 4, we show and comment examples that exhibit the role of each measure: hatching strokes are more discriminated by parallelism, overlapping or superimposition, while small figures rely mainly on the shape measure; color is independent of the shape of elements, but further refines the above measures.

3.2. Synthesizing a pattern

As mentioned previously, our synthesis algorithm begins at the center of a uniform distribution similar to that of the reference one. Hence, we first build a distribution of element positions that we call seeds, and connect seeds together to get neighborhood relationships. To this end, as in our previous method [Ano06], we use Lloyd's method [Llo82] in 1D and in 2D: it is an iterative algorithm that starts with a random distribution of seeds. Then, at each step, a Voronoi diagram of the seeds is computed, and each seed is moved to the center of its Voronoi region. It converges to a centroidal Voronoi tesselation, close to regular. The only parameter of the method is the number of seeds, that we set to $N_{tar} = N_{ref} . \mathcal{A}_{tar}/\mathcal{A}_{ref}$, where $\mathcal{A}_{ref}$ and $\mathcal{A}_{tar}$ are the areas of the reference and target regions respectively. Finally, when the algorithm has converged, for 1D patterns, we extract the neighbors along a chain, while for 2D patterns we extract a Delaunay triangulation and keep only the edges which are part of an unskewed triangle, in order to avoid degenerate edges at the border of the triangulation.

In the previous section, we have discussed a method of synthesizing an element when its neighborhood elements are already known. Unfortunately, this method cannot be used directly for synthesizing the entire pattern since for any element, only some of its neighbors will be known during the propagation. Like in Efros and Leung's approach, the

element synthesis algorithm must be modified to handle unknown neighborhood elements. This can be easily done by only matching on the known values of $\omega(E)$ and normalizing the error by the total number of known elements. This heuristic appears to provide good results in practice.

3.3. Adding variation

The patterns synthesized with our method exhibits strong similarities with the reference, since they consist of elements that have been copied from it. One might also wish to introduce some amount of variation relative to the reference pattern. In order to add such a variation, we developed a post-processing mechanism that slightly changes the parameters of the synthesized elements and is controllable by the user via a slider. For each element, and for each of its parameters independently, we select a set of similar values in the reference pattern. E.g., we select a set of orientations close to the synthesized element's orientation. Then, we pick one value from this set and use it in place of the parameters of the synthesized element. This mechanism lets us exchange parameters between similar reference elements without producing elements that are too different from those of the reference pattern.

Another noticeable difference between our synthesized patterns and their reference is the distribution of positions: while the distribution of a synthesized pattern can be considered uniform, this is not the case of the pattern input by the user. The variation present in the input might be desired by the user, and we thus propose a heuristic to reintroduce variation in the distribution of element positions as a postprocess. For each synthesized element E_{tar} that has $n \geq k$ neighbors, we get the reference shift vector S_{ref} of the corresponding E_{ref}, computed during the analysis (see Section 2.3); Then we position all the E_{tar} in parallel at the barycenter of their neighbors, and translate them by $S_{tar} = (S_{ref}\mathcal{A}_{E_{tar}})/(n\mathcal{A}_{E_{ref}})$ where $\mathcal{A}_{E_{ref}}$ and $\mathcal{A}_{E_{tar}}$ are the areas of the neighborhoods of E_{ref} an E_{tar} respectively. In practice, we use $k = 2$ for 1D patterns and $k = 4$ for 2D patterns.

4. Results

We show here some results of our synthesis method using various types of elements in 1D and 2D. Computation times are of the order of a second for 1D patterns; and between 5 and 10 seconds for 2D patterns, depending on the neighborhood size.

Figure 5 shows 1D synthesis. A simple example is shown in Figure 5(a) where curved hatching strokes are drawn with sketchy gestures and are properly analysed and synthesized. Our postprocess that adds variation to both position and element parameters is illustrated in Figure 5(b) with a simple hatching pattern; Notice how the vertical positions of elements are reintroduced in the synthesized pattern. We then show how we can reproduce smooth variations in the parameters of the elements along the 1D path. In Figure 5(c), we use a 5-ring neighborhood synthesis to reproduce the smooth change in the orientation of elements along the path; Here the parallelism measure plays a major role in the synthesis. Figure 5(d) shows the influence of the neighborhood size on the quality of the result. Synthesised patterns with 1-ring, 3-ring and 5-ring neighborhoods are shown from top to bottom: the smooth change in stroke length is only well captured with the 5-ring neighborhood. Here, the overlapping measure is discriminant. Figure 5(d) shows a smooth change in element thickness captured with a 5-ring neighborhood, and made possible by our superimposition measure. Finally, we show at the bottom an example using brush strokes of alternating colors: the synthesized pattern exhibits the same alternance, thanks to our color measure.

Figure 6 shows 2D synthesis. Figure 6(a) shows an example of bars oriented in multiple directions. Here, our synthesis method is able to reproduce the complex relations among similar elements. It can also synthesize elements that are less similar, as in Figure 6(b) with water drops, small figures like the flowers of Figure 6(c) or alternating two different kinds of elements as in Figure 6(d). Finally, the variation of element positions is illustrated in Figure 6(e), where the distribution obtained with Lloyd's method is modified to be more similar to the input pattern.

5. Discussion and future work

5.1. Analysis

Our analysis method can extract a wide range of elements (stipples, hatches, brush strokes, small figures), but our element representation (a center and two axes) is too simple to correctly extract long curved strokes. Moreover, we do not target structured patterns, such as a brick wall, where the overall organization should be extracted along with each element. We thus plan to address these two issues in the future by modifying our element model and adding multiple levels of analysis to be able to capture more structured patterns. Another issue is the use of additional perceptual criteria such as closure or junctions in order to perform a deeper interpretation of the input pattern.

In our approach, the analysis stage is user-assisted in order to determine the scale of the pattern. Since the clustering of elements is dependent on this scale and is implemented with a greedy algorithm, it can produce flickering on rare occasions when the user interactively modifies the scale. However, this has no impact on the final synthesis result. On the other hand, if one wants to consider scanned drawings as input, it would make sense to extract both the elements and the scale automatically. The greedy nature of the clustering algorithm might then lead to problematic behaviors.

Figure 5: *1D synthesis results. (a) A simple hatching example that uses sketchy strokes; (b) another hatching example with a uniform distribution of elements (on top), and the same pattern after variation have been added (at bottom); (c) a smooth change of element orientation is analysed and synthesized with a 5-ring neighborhood; (d) a smooth change of element length is analysed and synthesized with increasing neighborhood sizes (from top to bottom: 1-ring, 3-ring and 5-ring neighborhoods); (e) a smooth change of element thickness is analyzed and synthesized with a 5-ring neighborhood; (f) the alternance of strokes colors is captured and reproduced in the synthesized pattern.*

5.2. Synthesis

Our synthesis method currently generates quasi-uniform distributions of elements via the Lloyd algorithm. In the future we will target non-uniform distributions that can take into account density variations within the pattern. We also plan to take into account the whole element shape rather than only its center position in the distribution definition. The heuristic we used for finding relevant neighbors also suffers from a limitation: it can happen that no pairing is found between the reference and target neighborhoods. However, in prac-

tice, the even distribution produced by Lloyd's method prevents this worst case scenario from happening.

Another interesting point is the ability to take into account attributes of the input strokes during synthesis. We only investigated color, but other attributes such as thickness, opacity or texture might give convincing results. Finally, our variation technique is tailored to stroke pattern synthesis and has thus no equivalent in texture synthesis. We plan to extend this ability to take into account the shape of the strokes.

5.3. Extension to synthesis on surfaces

Our system works in the picture plane and we plan to extend it to synthesis on surfaces in the future. However, stroke-based rendering raises several specific questions in terms of rendering. Indeed, the strokes need to be of roughly constant size in 2D if one wants to maintain the same style for every viewpoint. Therefore an LOD mechanism has to be defined, and we believe that this can be achieved with a dedicated synthesis algorithm. The automatic generation of mipmaps or Tonal Art Maps [PHWF01] would be a simple way to address synthesis on surfaces. However, we believe that direct synthesis on surfaces will open more interesting avenues: for example, the rendering could be done by varying the attributes of each synthesized stroke depending on the viewing and lighting conditions. By keeping the analogy with texture synthesis, it should be possible to devise such a method in the same way we did in the present paper.

6. Conclusions

We presented a novel approach to the analysis and synthesis of vector-based stroke patterns along a 1D path or inside a 2D region. Our method makes use of perceptual organisation findings to interpret a reference input pattern, and later synthesize a similar one.

The synthesis algorithm is directly inspired by texture synthesis algorithms. It makes use of neighborhood comparisons and is able to take into account additional attributes of the input strokes like color.

The synthesized patterns are very similar to the reference ones, and the user can also add variation to the results while keeping a reasonable fidelity to the reference pattern.

References

[Ano06]　ANONYMOUS: Interactive hatching and stippling by example. In *submitted to GI* (2006). 2, 3, 5

[Ash01]　ASHIKHMIN M.: Synthesizing natural textures. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics* (New York, NY, USA, 2001), ACM Press, pp. 217–226. 2

[DHvOS00]　DEUSSEN O., HILLER S., VAN OVERVELD C., STROTHOTTE T.: Floating points: A method for computing stipple drawings. *Computer Graphics Forum 19*, 3 (2000). 2

[EL99]　EFROS A. A., LEUNG T. K.: Texture synthesis by non-parametric sampling. In *IEEE Int. Conf. on Computer Vision* (1999), pp. 1033–1038. 2, 4

[ESM*91]　ETEMADI A., SCHMIDT J., MATAS G., ILLINGWORTH J., KITTLER J.: Low-level grouping of straight line segments. In *British Machine Vision Conf.* (1991), pp. 119–126. 3

[FTP03]　FREEMAN W. T., TENENBAUM J. B., PASZTOR E. C.: Learning style translation for the lines of a drawing. *ACM Trans. Graph. 22*, 1 (2003), 33–46. 2

[Her98]　HERTZMANN A.: Painterly rendering with curved brush strokes of multiple sizes. In *Proceedings of SIGGRAPH 98* (1998), pp. 453–460. 2

[Her03]　HERTZMANN A.: A survey of stroke-based rendering. *IEEE Computer Graphics and Applications 23*, 4 (July/August 2003), 70–81. Special Issue on Non-Photorealistic Rendering. 1

[HOCS02]　HERTZMANN A., OLIVER N., CURLESS B., SEITZ S. M.: Curve analogies. In *Rendering Techniques 2002: 13th Eurographics Workshop on Rendering* (June 2002), pp. 233–246. 2

[JEGPO02]　JODOIN P.-M., EPSTEIN E., GRANGER-PICHÉ M., OSTROMOUKHOV V.: Hatching by example: a statistical approach. In *Proceedings of NPAR 2002* (2002), pp. 29–36. 2

[KMM*02]　KALNINS R., MARKOSIAN L., MEIER B., KOWALSKI M., LEE J., DAVIDSON P., WEBB M., HUGHES J., FINKELSTEIN A.: WYSIWYG NPR: Drawing strokes directly on 3D models. *ACM Trans. on Graphics 21* (July 2002), 755–762. 1, 2

[Llo82]　LLOYD S. P.: Least squares quantization in pcm. *IEEE Trans. on Information Theory 28*, 2 (1982), 129–137. 5

[Ost99]　OSTROMOUKHOV V.: Digital facial engraving. In *Proceedings of SIGGRAPH 99* (1999), pp. 417–424. 2

[Pal99]　PALMER S.: *Vision Science : Photons to Phenomenology.* MIT Press, 1999. 2

[PHWF01]　PRAUN E., HOPPE H., WEBB M., FINKELSTEIN A.: Real-time hatching. In *Proceedings of ACM SIGGRAPH 2001* (2001), pp. 579–584. 8

[PS00]　PORTILLA J., SIMONCELLI E. P.: A parametric texture model based on joint statistics of complex wavelet coefficients. *Int. J. Comp. Vision 40*, 1 (2000), 49–70. 2

[SABS94]　SALISBURY M. P., ANDERSON S. E., BARZEL R., SALESIN D. H.: Interactive pen-and-ink illustration. In *Proceedings of SIGGRAPH 94* (1994), pp. 101–108. 2

[Tur01]　TURK G.: Texture synthesis on surfaces. In *Proceedings of ACM SIGGRAPH 2001* (2001), pp. 347–354. 2

[WL00]　WEI L.-Y., LEVOY M.: Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2000), ACM Press/Addison-Wesley Publishing Co., pp. 479–488. 2

[WL01]　WEI L.-Y., LEVOY M.: Texture synthesis over arbitrary manifold surfaces. In *Proceedings of ACM SIGGRAPH 2001* (2001), pp. 355–360. 2

[WS94]　WINKENBACH G., SALESIN D. H.: Computer-generated pen-and-ink illustration. In *Proceedings of SIGGRAPH 94* (1994), pp. 91–100. 2
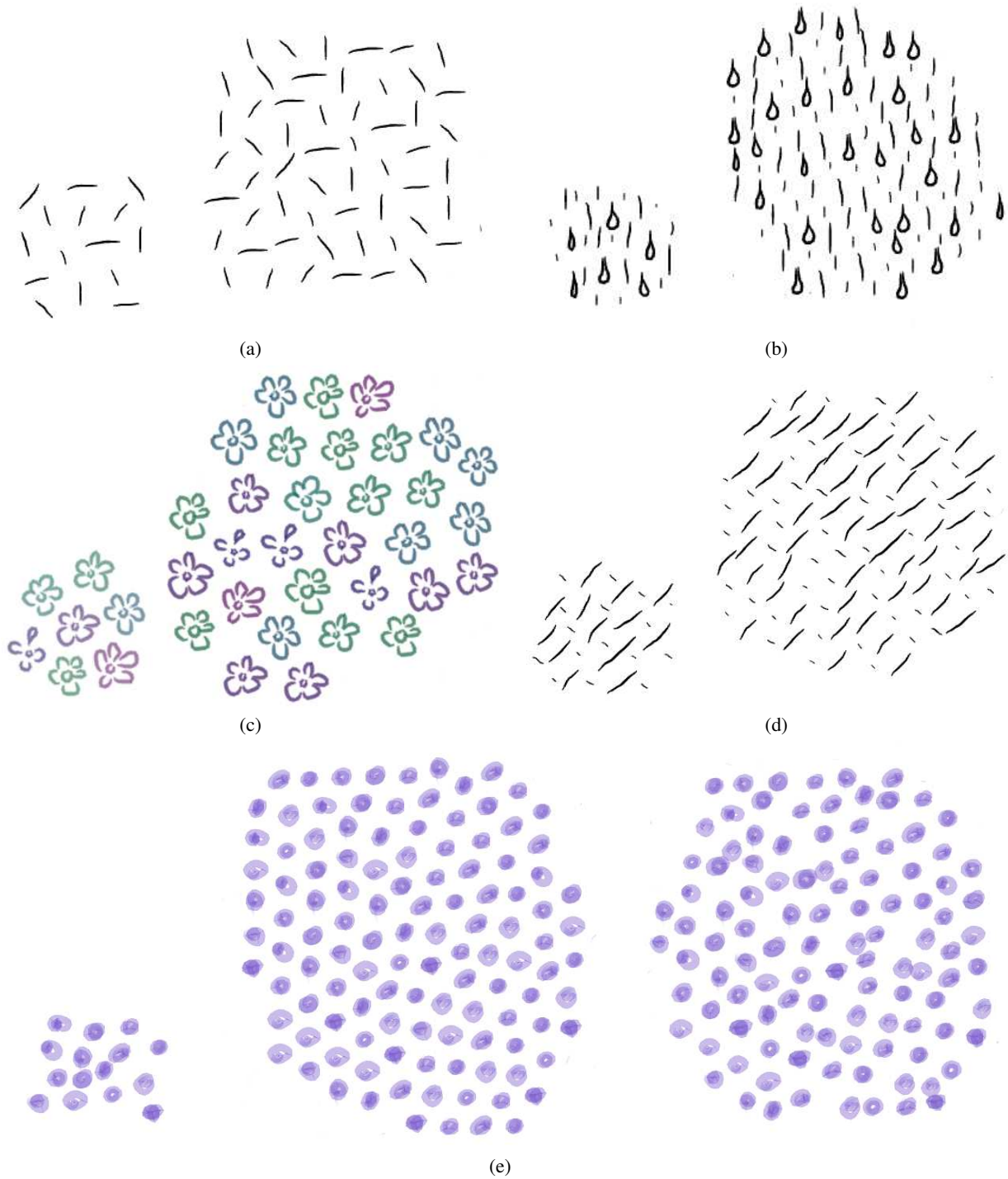
(a)

(b)

(c)

(d)

(e)

Figure 6: *2D synthesis results. (a) A pattern of hatching strokes in multiple directions is synthesized using a 3-ring neighborhood; (b) elements of various nature (water drops, hatches) are analysed and synthesized with a 3-ring neighborhood; (c) small figures like flowers of different colors can be analyzed and synthesized by our method using a 2-ring neighborhood; (d) a pattern composed of two different types of hatching strokes is synthesized with a 2-ring neighborhood; (e) the addition of variation in the position of elements is able to break the uniform distribution of Lloyd's method.*