Segmental Brush Synthesis with Stroke Images

Ryoichi Ando[1] and Reiji Tsuruno[2]

[1]Graduate School of Design, Kyushu University, Japan
[2]Faculty of Design, Kyushu University, Japan

Abstract

We present a new approach for synthesizing realistic brush strokes exploiting recent works of texture synthesis from stroke images. (See Figure 1). In our method, stroke images are automatically decomposed into a sequence of quad segments and stitched together along the path of user's input to produce final image. Numbers of methods using textures on digital painting have been explored; our usage of texture is novel in that the source image is typically a photo and the synthesis is fast enough to achieve realtime feedback. In contrast to previous methods, our approach allows a large variety of artistic brushes to be interactively simulated fairly so that unique media which haven't caught attention yet such as lipsticks or finger paint, are well reproduced. We shall show some artworks created using our method and demonstrate feasibility of our algorithm.

Categories and Subject Descriptors (according to ACM CCS): Feature Measurement [I.4.7]: Texture—Learning [I.2.6]: Analogies—Picture/Image Generation [I.3.3]: Display algorithms—

1. Introduction

Digital painting with textures has become one of the mainstays for simulating characteristics of artistic media. The advantages of using textures are its versatile flexibility and a reliability for authoring realistic results. Today, many commercial softwares such as Photoshop or Painter use textures for brush stroke representation. Nevertheless, modeling of specific media with textures is still not an easy task; it often involves programmers to develop individually engineered algorithm and designers to adjust sensible parameters. In the present contribution we offer a solution for the tedious task which allows designers to create stylish brushes instantly. With our method, only several example photos will be enough to learn the styles of a stroke and dynamically simulate them at realtime rate.

Physical methods may be attractive, however, they are only designed for one particular media. Some other related works of texture synthesis and early works addressed the use of images and textures to simulate visual styles of a brush; though, these methods are unlikely to be able to fulfill our goal because those methods are slow and limited to cover small variation of media. In the present contribution we exploit recent techniques of patch-based texture synthesis [EF01, KSE*03] and extend them to generate brush stroke images.

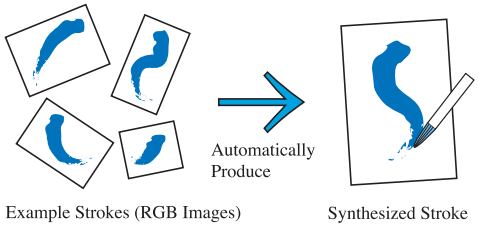

Figure 1: *Given a set of example strokes, our algorithm generates realistic strokes in response to the user's input of a brush, reproducing the characteristics of source examples.*

2. Related work

Hsu introduced *Skeletal Strokes* that abstracts arbitrary images into the path to render naturally deformed silhouettes [HL94]. They experienced successful results on illustration and calligraphy, however, the detail of texture is also stretched or shrunk with the length of the path. Ritter, et al [RLA*06] employed a pixel-based texture synthesis to simulate broad range of media. Based on "Texure-by-Numbers" [HJO*01] they embedded a novel energy function and simulated various media. Even though they devised method to achieves interactive rate, it takes few seconds to several

minutes for each stroke to synthesize full-resolution image. There is a GPU accelerated technique for general texture synthesis [LH05]; the technique is not directly extendable for our purpose. For charcoal brushes, the bitmaps of look-up table are statically stamped based on the pressure and the tilt of stylus [BSM88]. For ink-washed brushes perpendicular vectors along the spine were used to compose triangle strips and texture mapped [NM00]. These methods perform fast and are straightforward to do; though, the source textures should be created in a considerable manner depending on artist's aesthetic and intuition.

3. Our Approach

A brief workflow of our algorithm is as follows. We first compute the contour and the spines from source strokes and decompose them into a series of thin quad segments. Next, when a user input the track information of a brush, we recombine those precomputed segments into the form of a target stroke along the track of input. (See Figure 2) To measure an unnaturalness at the very local point, we introduce a visual cost function $E(\boldsymbol{x}_i)$. The function takes one multi-dimensional vector "$\boldsymbol{x}_i$" which indicates "i th segment from the beginning of a stroke along the spine." We describe mathematical details in latter section. We consider the problem as a minimization problem as

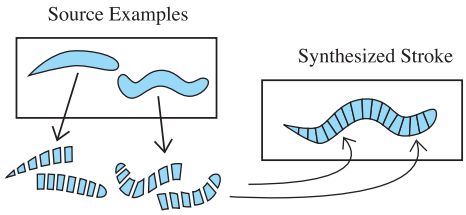

Figure 2: *Workflow of our method: We first decompose source strokes into a sequential quad segments and randomly recombine them into a new stroke image.*

$$\min_{s} \sum_{i=1}^{l} E(\boldsymbol{x}_i) \tag{1}$$

where s and l denotes the segment array of a target stroke, the length of stroke, respectively. The problem can be intuitively translated to graph theory of finding optimal length of path that connects l points. The problem is known to be efficiently solved by dynamic programming [SYJS05]. Suppose $E(\boldsymbol{x}_i)$ consists of two static and dynamic terms

$$E(\boldsymbol{x}_i) = E_s(\boldsymbol{x}_i) + \sum_{j \in N} E_d(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{2}$$

where $E_s(\boldsymbol{x}_i)$ only depends on static environment and doesn't get influenced to marginal features and $E_d(\boldsymbol{x}_i, \boldsymbol{x}_j)$ depends on marginal features where $N = (i-1, i+1)$. We abuse notation $\sum_j E_d(\boldsymbol{x}_i, \boldsymbol{x}_j)$ to be $E_d(\boldsymbol{x}_i)$ for simplicity. The minimization problem can be reduced to a recursive format

$$M_i(\boldsymbol{x}_i) = E_s(\boldsymbol{x}_i) + \min_{x_{i-1}} \{E_d(\boldsymbol{x}_i) + M_{i-1}(\boldsymbol{x}_{i-1})\} \tag{3}$$

$$M_1(\boldsymbol{x}_1) = E_s(\boldsymbol{x}_1) \tag{4}$$

where $M_i(\boldsymbol{x}_i)$ denotes a cumulative minimum energy from 1 to i. With the dynamic programming, the optimal combination of $\min M_l(\boldsymbol{x}_l)$ is then efficiently computed in $O(n^2)$ time by memorizing previously computed $M_i(\boldsymbol{x}_i)$ for each iteration. Even though the problem is solved in polynomial time, $O(n^2)$ time is still costly for realtime synthesis. In our approach, we exploit the character of feature vector distribution that features are plotted together. In other word, the feature cloud forms just as snake-like shape. This yields a more simplified iterative scheme that approximates min-energy as

$$M_i(\boldsymbol{x}_i) \approx E_s(\boldsymbol{x}_i) + E_d(\boldsymbol{x}_i) + \min_{i-1} M_{i-1}(\boldsymbol{x}_{i-1}). \tag{5}$$

In contrast to dynamic programming approach, this method is far from optimal; though, it maintains local optimality and the computation is done only in $O(\log n)$ time when search is accelerated with ANN [AMN*98]. This approach works fine since the static energy term constrains the search around its perimeter.

3.1. Segmental Decomposition

To extract spine we firstly extract a branched provincial spine. Secondly, starting from every edge of the branched spine, we iteratively shrink the spine while the number of edge more than two. To decompose a stroke into the sequential quad segments, we firstly discretize the spine path into a series of points with an arbitrary space. Next, for each point we compute a perpendicular vector and slice the region. We show an example of our decomposed stroke in Figure 3.

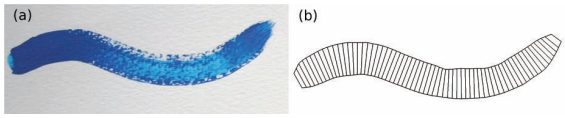

Figure 3: *The decomposition of a hairy brush: The photo of a hairy brush stroke (a) was automatically decomposed into sequential thin segments (b).*

3.2. Features

The selection of specific component feature is an open problem though, we heuristically picked features as follows: 1) color value at the neighboring segments 2) distance from head and tail 3) width of the both side of the segment 4) curvature. Each of those 4 features is a multi-dimensional feature vector; for example, width feature is 2 dimensional feature vector and color feature is 50 dimensional in our implementation. In total, the feature vector per segment is

around 60 dimensional. We illustrate those components in Figure 4. Note that in order to evaluate them fairly, those 4 features are normalized per group beforehand. We ascribe those selections to the following observations. The neighboring segments should be consistent; therefore, to make the boundaries of segments less artificial, we should place the segments in such a way that the neighboring color matches better. And the same explanation for the width of a segment. Taking those rationales into account, we design our feature metric as follows:

$$L_2^{i,j}(v_c) = (v_c^i - v_c^j)^2 \quad (6)$$

$$E_d(\boldsymbol{x}_i, \boldsymbol{x}_j) = \int_S L_2^{i,j}(v_{color})dS + L_2^{i,j}(v_{width}) \quad (7)$$

$$E_s(\boldsymbol{x}_i) = G(v_{dhead},)L_{\,2}(v_{dhead}) + \quad (8)$$

$$G(v_{dtail},)L_{\,2}(v_{dtail}) + L_2(v_{curvature}) \quad (9)$$

$$E(\boldsymbol{x}_i) = E_s(\boldsymbol{x}_i) + \sum_{j \in N} E_d(\boldsymbol{x}_i, \boldsymbol{x}_j) \quad (10)$$

where $E_d(\boldsymbol{x}_i)$ and $E_s(\boldsymbol{x}_i)$ denotes dynamic and static term of $E(\boldsymbol{x}_i)$ as described in equation 2. N is the neighboring segments at i where $N = (i-1, i+1)$. $L_2^{i,j}(\cdot)$ and S denotes L_2 norm with neighboring segments i, j or input path and overlapping region with neighboring segments, respectively. Segment colors are sampled uniformly by transforming quads into square shapes. v_{color}, v_{width}, v_{dhead}, v_{dtail}, $v_{curvature}$ and $G(x,)$ denotes the multi-dimensional component in $\boldsymbol{x}_i$ and gaussian function.

3.3. Stitching

At the core of our method is stitching of decomposed segments together into a target form of stroke along user's input of path. In the first loop, we begin synthesis from a blank space and concatenate a segment which $E(\boldsymbol{x}_i)$ is very small to the end of stroke body with the backward induction of a perimeter. In the second loop, for every segment, we further search for one with better matching with back and forward perimeter and replace with it if energy goes lower. We illustrate the pseudo code of this scheme below.

Function : SYNTHESIZESTROKE(*segments*, *path*)

```
Allocate blank segment array x
 // First loop
for each i ∈ x from head to tail
   x_i ← arg min_i E(x_i, 0, x_{i-1}, segments, path)
 // Second loop
for each i ∈ x from head to tail
   x_i ← arg min_i E(x_i, x_{i+1}, x_{i-1}, segments, path)
return (x)
```

Typically, this search term is accelerated with ANN [AMN*98] when x_{dtail} and x_{dhead} component is large enough. Overrall, our computation time is $O(\log n)$.

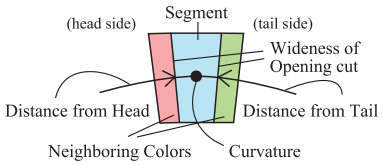

Figure 4: *Features of a segment: To represent the features of a segment, 1) neighboring color 2) distance from head and tail 3) width of the both side of the segment 4) curvature are used.*

3.4. Quilting

Barely stitched strokes often leaves artifacts around its boundaries. To blur this artifact, we employ Image Quilting technique [EF01]. Decomposed segments have inherent rectangle seams. When they're stitched randomly, those neighboring seams are often visible; this gap produces artifacts on its boundaries. By Image Quilting, we search for an optimal unnoticeable seam within overlapping region as illustrated in Figure 5. Instead of quilting in full-resolution,

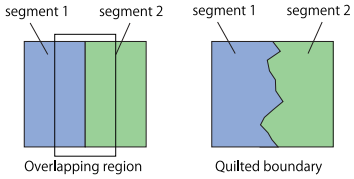

Figure 5: *Image Quilting: Two adjacent segments were stitched together in such a way that SSD goes smaller.*

we downsample pixels to accelerate approximate path. In our implementation, overlapping region is reduced to around between 5×10 and 10×20 pixels for each pair.

4. Results

Figure 6 shows brief examples of generated strokes and corresponding source photos. The cost of our overall algorithm largely depends on the number of segments of source strokes and is less subject to the resolution of source images. This is because feature vectors are defined per segment but per pixel and the rendering is done by texture-mapping on a graphic hardware. Figure 7 shows a performance of our method. As seen from the figure, our method allows users to create highly detailed strokes without consuming lots of computational cost, which is often hard for conventional texture synthesis. Figure 8 shows an artwork illustrated by our method.

5. Limitation

The proposed method includes two innate limitations. Firstly, the produced image quality heavily depends on the number of source images and its variance. For instance, if

lots of source images are imported, we can generate unpredictable plausible strokes while some remarkable artifacts are often produced. Contrarily, if few sources are given, such artifacts will not be present; though, the generated strokes tend to be trivial. Secondly, segment synthesis involves repeats occasionally. Such repeats should be suppressed by static energy term and only visible locally; though, the repeats are inevitable with segments which static energy term takes similar values.

6. Conclusion

In this paper, we presented a new framework that generates new realistic brush strokes from source stroke images. The fundamental idea of our approach was to discretize source strokes into a sequence of quad segments and recombine them into the new form of a target stroke. At the part of synthesizing strokes, we formulated the problem as a minimization energy problem and designed an iterative solver to tackle the problem. In future work, we would like to add special post effects such as fluid dynamics to increase reality in watercolor or sumie paintings.

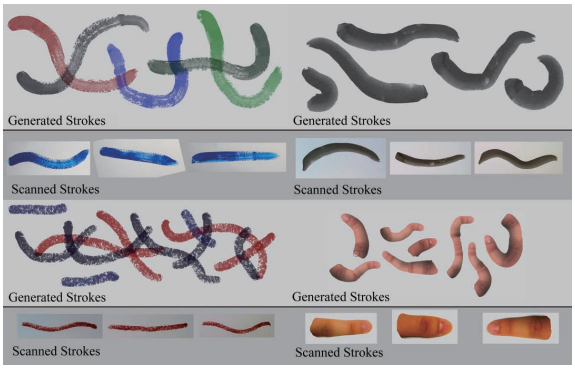

Figure 6: *Various simulated stokes: Left column: hairy brush, lipsticks. Right column: sumie, fingers along with scanned images.*

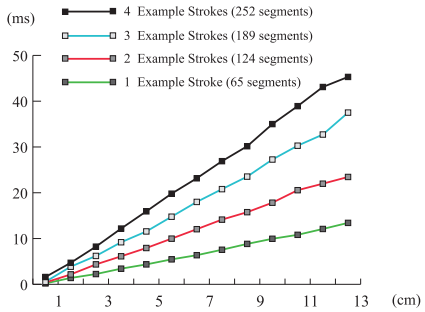

Figure 7: *Performance of our algorithm: the horizontal axis represents the stroke length and vertical axis the synthesis time on Core2Quad 2.66GHz machine.*

Figure 8: *Painting of triceratops: To paint this painting, one example charcoal brush shown under the figure was scanned. To depict the skin of the dinosaur, artist changed color interactively for each stroke.*

References

[AMN*98] ARYA S., MOUNT D. M., NETANYAHU N. S., SILVERMAN R., WU A. Y.: An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM 45*, 6 (1998), 891–923.

[BSM88] BLESER T. W., SIBERT J. L., MCGEE J. P.: Charcoal sketching: returning control to the artist. *ACM Transaction. Graphics. 7*, 1 (1988), 76–81.

[EF01] EFROS A. A., FREEMAN W. T.: Image quilting for texture synthesis and transfer. *Proceedings of SIGGRAPH 2001* (August 2001), 341–346.

[HJO*01] HERTZMANN A., JACOBS C. E., OLIVER N., CURLESS B., SALESIN D. H.: Image analogies. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2001), ACM, pp. 327–340.

[HL94] HSU S. C., LEE I. H. H.: Drawing and animation using skeletal strokes. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1994), ACM, pp. 109–118.

[KSE*03] KWATRA V., SCHÖDL A., ESSA I., TURK G., BOBICK A.: Graphcut textures: image and video synthesis using graph cuts. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers* (New York, NY, USA, 2003), ACM, pp. 277–286.

[LH05] LEFEBVRE S., HOPPE H.: Parallel controllable texture synthesis. *ACM Trans. Graph. 24*, 3 (2005), 777–786.

[NM00] NORTHRUP J. D., MARKOSIAN L.: Artistic silhouettes: a hybrid approach. In *NPAR '00: Proceedings of the 1st international symposium on Non-photorealistic animation and rendering* (New York, NY, USA, 2000), ACM, pp. 31–37.

[RLA*06] RITTER L., LI W., AGRAWALA M., CURLESS B., SALESIN. D.: Painting with texture. In *17th Eurographics Symposium on Rendering, Nicosa, Cyprus* (2006).

[SYJS05] SUN J., YUAN L., JIA J., SHUM H.-Y.: Image completion with structure propagation. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers* (New York, NY, USA, 2005), ACM, pp. 861–868.