Self-Similar Texture for Coherent Line Stylization

Pierre Bénard, Grenoble University
Forrester Cole, MIT CSAIL
Aleksey Golovinskiy, Princeton University
Adam Finkelstein, Princeton University

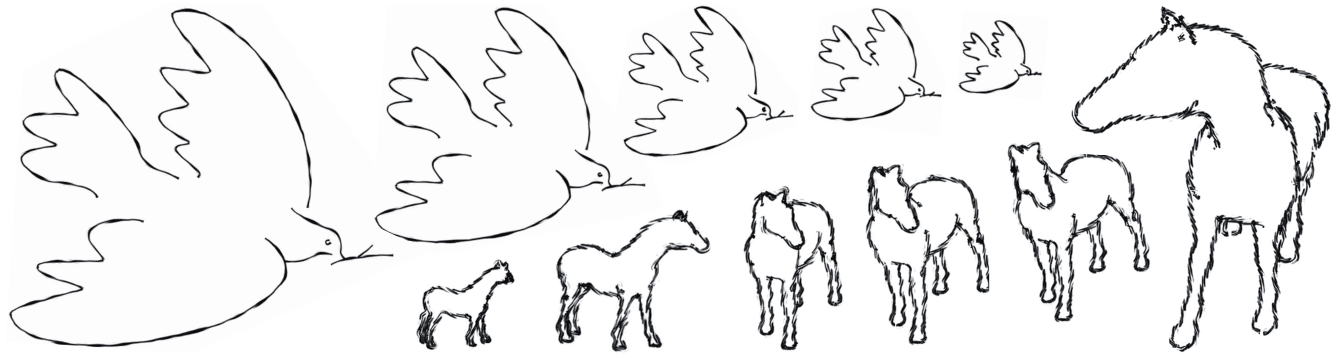


Figure 1: *Shapes drawn with stylized lines using self-similar textures. The bird (above) is a purely 2D figure, while lines are extracted from a 3D model of a horse (below). At different scales the lines appear qualitatively similar, and transitions exhibit temporal coherence.*

Abstract

Stylized line rendering for animation has traditionally traded-off between two undesirable artifacts: stroke texture sliding and stroke texture stretching. This paper proposes a new stroke texture representation, the self-similar line artmap (SLAM), which avoids both these artifacts. SLAM textures provide continuous, infinite zoom while maintaining approximately constant appearance in screen-space, and can be produced automatically from a single exemplar. SLAMs can be used as drop-in replacements for conventional stroke textures in 2D illustration and animation. Furthermore, SLAMs enable a new, simple approach to temporally coherent rendering of 3D paths that is suitable for interactive applications. We demonstrate results for 2D and 3D animations.

Keywords: Non-photorealistic rendering, line drawing, temporal coherence, artmap

1 Introduction

Stylization is an integral part of effective line rendering. Artists vary effects such as width, texture, and precision to convey weight, shape, motion, or abstraction. A dotted line might suggest an invisible object, while an oversketched contour might indicate emphasis or movement. A common approach for simulating these effects in computer graphics is to define a parameterized space curve (known here as a *brush path*), and apply a stroke texture along its length. Applied textures may include dots, dashes, or captured images of marks made with pen, pencil, or other natural media. Stroke textures create attractive imagery that can appear as though it was drawn by hand.

There are two simple policies for texture mapping a brush path with a stroke texture. The first approach, which we call the *stretching policy*, stretches or compresses the texture so that it fits along the path a fixed number of times. As the length of the path changes, the texture deforms to match. The second approach, called the *tiling policy*, establishes a fixed pixel length for the texture, and tiles the path with as many instances of the texture as can fit.

The two policies are appropriate in different cases. The tiling policy is necessary for textures that should not appear to stretch, such as dotted and dashed lines. Because the tiling policy does not stretch the texture, it is also usually preferred for still imagery. Under animation, however, the texture appears to slide on or off the ends of the path and be very distracting. In contrast, the stretching policy provides intuitive behavior under animation, but the stroke texture loses its character if the path is stretched or shrunk too far.

A third approach is the artmap method [Klein et al. 2000], which uses a set of N textures (an *artmap*), where each texture has a particular target length in pixels. At each frame, the texture with target length closest to the current path length is selected and drawn. This ensures that the brush texture never appears stretched by more than a constant factor (often $2\times$). Of course, if the path length extends beyond the length of the largest texture in the artmap, stretching artifacts will still appear. A more troublesome problem is that artmap construction is a time-consuming and painstaking process. As a result, current artmap implementations such as "Styles" in Google SketchUp [Google 2008] use as few as four textures.

This paper presents a new artmap-based approach called self-similar line artmaps (SLAMs). A SLAM is similar to a conventional artmaps, but has two additional properties: it is *self-similar* and *smoothly varying*. In this context, self-similarity implies that the same texture is repeated at two separate levels of the artmap, a property that allows for infinite zoom without visible interruption. The smooth variation property means that nearby levels of the artmap are similar in appearance, and ensures that animations using SLAMs are temporally coherent.

This paper makes two technical contributions for creating and rendering with SLAMs. First, we present an example-based synthesis approach that creates an arbitrarily dense set of textures that satisfy the SLAM conditions. We note that parametric texture synthesis (e.g. [Portilla and Simoncelli 2000]) is superior to non-parametric synthesis for this purpose, because small changes in the synthesis seed tend to produce small changes in the output texture. Second, we present a method for parameterizing view-dependent 3D lines (such as contours and suggestive contours) for rendering with SLAMs. The method is simpler than the coherent line parameterization approach of Kalnins et al. [2003] because we explicitly target arc-length parameterization for the textures.

2 Related Work

In this section we describe work related to the two main technical contributions of this paper.

Self-similar textures. With the goal of stylizing shaded regions of NPR imagery (e.g., canvas, watercolor, hatching or painterly rendering), many texture-based approaches have been proposed to maintain a quasi-constant size of the texture pattern in screen-space. Both the artmaps approach proposed by Klein et al. [2000] and the subsequent "tonal art maps" of Praun et al. [2001] rely on a specific set of mip-maps which offer coherence transitions between scales. Nevertheless, these methods can only handle a restricted range of zoom imposed by the depth of the mipmap, and exhibit noticeable blending artifacts when transitioning between levels. To address these limitations, the method of Cunzi et al. [2003] introduces an infinite zoom mechanism which dynamically regenerates the pattern by alpha-blending multiple scaled versions of the original texture. Following this approach, Bénard et al. [2009a] dynamically compute a self-similar texture for each object, which is used as a base-layer for many styles. However, they subsequently demonstrate [Bénard et al. 2009b] that this general approach perceptually degrades high-contrast structured patterns – the very kind of stylization we would like to use for lines. To better preserve the original pattern, Han et al. [2008] propose a 2D example-based *non-parametric* synthesis algorithm that generates new texture elements at multiple scales on the fly during the zoom. With their method, multiple scales of texture elements appear at every level of the zoom. Our approach also relies on texture synthesis, but uses the *parametric* method of [Portilla and Simoncelli 2000]. We precompute a continuous self-similar pyramid of textures, allowing for infinite continuous zoom while maintaining similar appearance to the original exemplar at every scale, as described in Section 3.

Coherent line stylization. Many researchers have focused their efforts on extraction and rendering of lines (see the annotated bibliography of Rusinkiewicz et al. [2008]). Nevertheless, temporal coherence for stylized lines remains a challenging problem. The state of the art system by Kalnins et al. [2003] built on the seminal work by Masuch et al. [1998] and Bourdev [1998], generalizing their approaches to provide temporal coherence for stylized lines drawn over a broad class of computer graphics models. The three key ideas of the Kalnins et al. system were: (1) propagating the line parameterization from one frame to the next to provide temporal continuity, (2) splitting screen-space continuous paths into possibly multiple brush paths corresponding to the paths used in previous frames, and (3) optimizing an energy function that includes the competing goals of uniform screen-space arc-length parameterization, coherence on the object surface, and attempting to merge multiple paths where possible. The Kalnins et al. approach is somewhat complex and brittle, largely due to the combination of optimization and splitting and merging of paths. In this work we observe that some of this complexity may be omitted for the case of line styles explicitly targeting arc-length parameterization. Coupled with the use of the SLAM datastructure described above, our system need only to compute two numbers – scale and phase – for each path in order to fully parameterize it. This leads to a simpler optimization, described in Section 4.

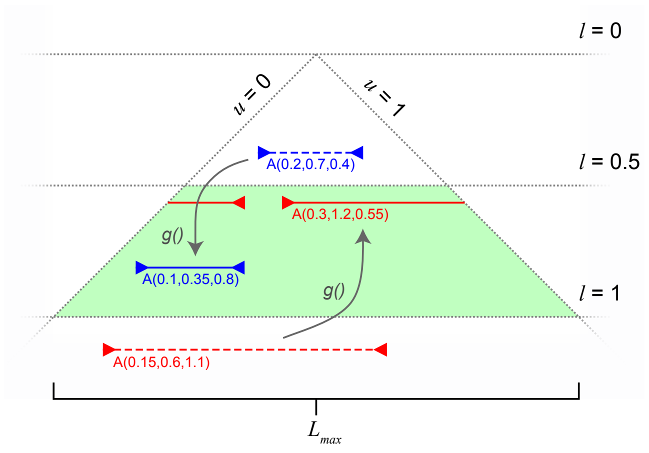


Figure 2: *Parameterization space of a self-similar line artmap. Horizontal distance corresponds to length in screen-space, vertical distance to scale. The shaded region is where $A(u_1, u_2, l)$ is defined. Paths with scale $l < 0.5$ or $l > 1.0$ (dotted lines) are mapped to coordinates with $0.5 < l < 1.0$ under the mapping function g (solid lines). Each level of the pyramid tiles seamlessly with itself, so long lines may safely be wrapped (e.g., solid red line).*

3 Self-Similar Line Art Maps

This section describes the construction of our self-similar line artmaps (SLAMs) and their relationship to conventional artmaps.

In the conventional artmap approach the number of textures required is $O(\log_\sigma L_{max})$, where L_{max} is the length of the longest path and σ is the maximum stretch factor allowed. Unfortunately, L_{max} is effectively unbounded for several interesting cases, including continuous zoom. Our new artmap construction (Section 3.1) is self-similar, allowing a small number of textures to smoothly texture paths of arbitrary length.

When σ is large, visible pops may appear at transitions. Blending adjacent artmap levels reduces popping, but tends to produce a blurry result. The solution is to create a dense artmap, with many textures and a small σ. We propose an example-based artmap synthesis approach (Section 3.2) that generates an arbitrarily dense artmap based on a single exemplar. Our synthesis approach also guarantees that each artmap level blends seamlessly into the next.

We will describe the implementation first using 1D textures. The extension to stroke textures with width is discussed in Section 3.2.2.

3.1 SLAM Definition

A line artmap can be visualized as a pyramid, with base width L_{max} and each level shrinking in size by a factor of σ. For convenience, define the artmap as a function $A(u_1, u_2, l)$, where u_1 and u_2 are the left and right texture coordinates and l is the level in the artmap, normalized such that $l = 1.0$ corresponds to texture length L_{max}. For example, $A(0, 1, 0.5)$ corresponds to the entire texture with target length $0.5L_{max}$, and $A(0.5, 1.0, 0.2)$ corresponds to the right half of the texture with target length $0.2L_{max}$. For concise notation, let $A_l = A(0, 1, l)$.

In order to support paths of arbitrary length with infinite zoom, the full artmap domain $u_1 \in \mathbb{R}, u_2 \in \mathbb{R}, l \in \mathbb{R}^+$ must be mapped somehow onto a finite set of textures (Figure 2). To accommodate arbitrary u_1 and u_2, we assume that every texture in the artmap tiles

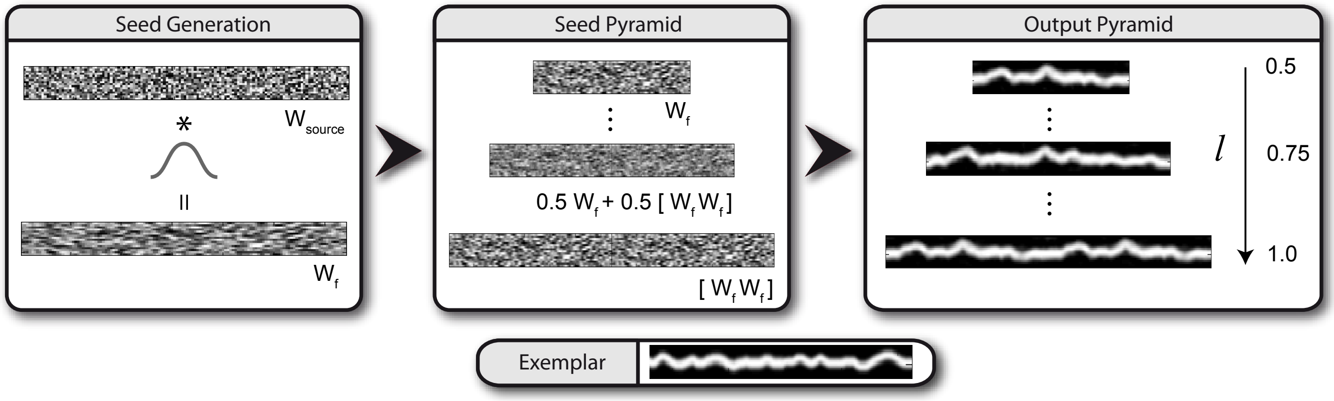


Figure 4: *Synthesis of a self-similar line artmap. An initial white noise seed W_{source} is slightly filtered to produce a filtered seed W_f. The filtered seed W_f is scaled to half its original length and placed at the top of the seed pyramid. The scaled W_f is concatenated with itself and placed at the bottom of the pyramid. Interior levels of the pyramid are created by linearly interpolating the top and bottom. Finally, each seed is used to synthesize the corresponding level of the artmap pyramid.*

seamlessly with itself. This leaves the problem of $l > 1$. The naive solution is to simply tile the path with the longest texture available (A_1), but this policy has all the negative aspects of the tiling policy.

Our approach is to change the problem by placing an additional constraint on the artmap: it must be *self-similar*. Precisely, there must exist at least one pair of textures A_x and A_y such that $A_x = [A_y A_y]$ (the concatenation of A_y with itself). If such a pair exists, then arbitrary stretching or shrinking can be handled by looping between the two similar textures. Figure 3 shows an example of such artmap for a dotted line. Note that the bottom texture is visually equivalent to a tiled version of the top texture. For simplicity, we will use $x = 1.0$ and $y = 0.5$. The mapping from the full domain to the self-similar domain is then defined by a mapping function $g(u_1, u_2, l) \in \mathbb{R}^3$:

$$\begin{aligned} g(u_1, u_2, l) &= (2^t u_1, 2^t u_2, 2^{-t} l) \\ t &= \lceil \log_2 l \rceil \end{aligned} \tag{1}$$

If the path to render has length s in pixels, the texture used is $A(g(0, 1, \kappa s / L_{max}))$, where κ is a user-defined scaling parameter. This parameterization may tile the texture, but will not cause the texture to slide on and off the path as it grows or shrinks.

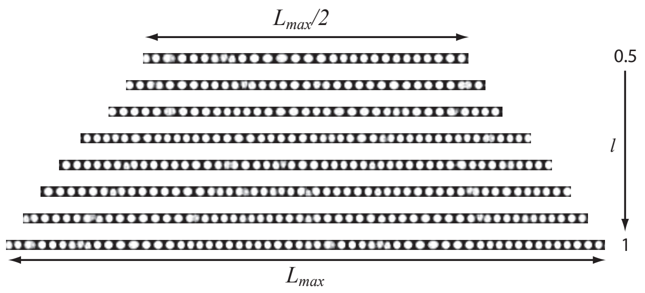


Figure 3: *Example SLAM for a dotted line texture. This set includes eight levels, but an arbitrary number may be generated. As the scale grows, the dots tend to stretch out. When they grow too large, they split into two smaller dots. Irregularly sized dots are necessary for temporal coherence; other small imperfections are the consequence of texture synthesis.*

3.2 Artmap Synthesis by Example

Given the definition of a SLAM, the challenge is to construct one without a huge amount of manual effort. The two necessary conditions for a well-behaved SLAM are that $A_{1.0} = [A_{0.5} A_{0.5}]$ (self-similarity), and that $A_l \approx A_{l+\epsilon}$ (smooth variation). A desirable third condition is that any texture extracted from the SLAM closely resembles the exemplar. Our synthesis approach satisfies the first two conditions and, for many types of textures, the third condition as well.

The basis of our approach is the parametric texture synthesis (PTS) method of Portilla and Simoncelli [2000]. Each artmap texture A_l is the result of PTS applied to a filtered white noise seed W_l. We notice that the PTS method seems to have the vital property that small changes in the seed result in small changes in the output. We exploit this property by constructing a set of seeds W_l that satisfy the SLAM conditions of self-similarity and smooth variation. Because of the continuous behavior of PTS, if the W_l satisfy the SLAM conditions, then A_l should also satisfy the conditions.

Using PTS, the quality of each individual A_l is less than what might be achieved using a modern non-parametric synthesis method (e.g.[Lefebvre and Hoppe 2006]). However, non-parametric methods tend to lack the property that small changes in the synthesis seed produce small changes in the output texture. This property arises because parametric methods are based on iteratively adjusting statistics of a random noise seed, and each of these adjustments (e.g., histogram matching) is a continuous operation. By contrast, non-parametric methods are based on iterative neighborhood matching, which produces a chain of discrete decisions that can change drastically with small perturbations to the input.

Note, however, that we do not have a formal proof of this desirable behavior of PTS. In our experiments, PTS provides continuous behavior for a variety of textures, but there may still be cases for which PTS fails to create smooth variation.

3.2.1 Construction of Synthesis Seeds

The seeds W_l are scaled and filtered versions of a fixed white noise texture W_{source}, which has length L_{max}. Ideally, all W_l have perfect white noise statistics, but perfect white noise is not necessary as PTS will coerce the statistics to match the exemplar. We therefore only need to ensure that the W_l satisfy the self-similarity and

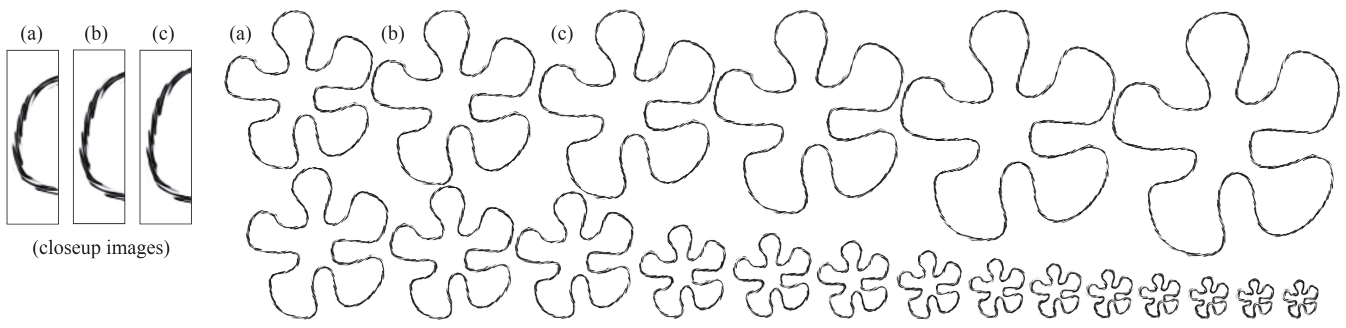


Figure 5: *Zoom sequence on a 2D vector-art flower. Note that at every frame the screen-space size of the scribbles is well preserved and that no alpha-blending artifacts are noticeable. See especially the inset closeups of the upper left three flowers for continuity, and shown in motion (with dot pattern) on the accompanying video.*

smooth variation conditions, and have roughly white noise statistics. In practice, we start from a single filtered version of W_{source}, and scale and linearly interpolate to produce each W_l (Figure 4).

The original white noise texture W_{source} is filtered with a small Gaussian filter (we use a width of 5 pixels) to produce a smoothed version W_f. This smoothing is necessary to avoid aliasing when the seed is scaled. The smoothed seed W_f is then adjusted using histogram matching to restore a uniform distribution across the range $(0, 1)$. The top level of the seed pyramid $W_{0.5}$ is simply W_f scaled to half its original length. The bottom level of the pyramid is $W_{1.0} = [W_{0.5}W_{0.5}]$, or the concatenation of $W_{0.5}$ with itself. The interior levels of the pyramid are generated by scaling $W_{0.5}$ and $W_{1.0}$ to the proper length, then linearly interpolating:

$$\begin{aligned} W_l &= \alpha W_{1.0} + (1-\alpha)W_{0.5} \\ \alpha &= 2l - 1 \end{aligned} \tag{2}$$

The seed pyramid satisfies the self-similarity condition by construction, since $W_{1.0}$ is a tiled version of $W_{0.5}$. The smooth variation condition is satisfied by the linear interpolation of the top and bottom of the pyramid.

3.2.2 Extension to 1.5D Textures

So far, we have considered only 1D textures in our description. However, all the results in this paper were generated with stroke textures with width between 8 and 32 pixels. We call these brush textures 1.5D, since they are essentially 1D textures with a finite width. A few modifications to the synthesis algorithm are necessary to accommodate 1.5D textures.

The construction of the seed pyramid is similar to the 1D case. W_{source} becomes a $d \times L_{max}$ white noise texture, where d is the number of rows. The initial filtering operation from W_{source} to W_f is only performed in 1D (the horizontal direction in Figure 4), because W_f is only scaled in one dimension. The remaining operations to produce the seed pyramid are identical to the 1D case.

The largest changes are required in the texture synthesis, because our 1.5D stroke textures violate the stationarity assumption of the PTS method. More specifically, the statistics of each row of a 1.5D texture are not identical: the top and bottom rows are usually dark (transparent), while the middle rows are usually bright (solid). Our approach is to apply different statistics to each row of the texture. While the PTS method stores and applies many different statistics of the source texture, through experimentation we found that varying only the *mean* and *variance* by row sufficed to produce good results. We therefore modify the PTS method to gather and apply the mean and variance per-row, and all other statistics per-texture. This modification is only a few lines of MATLAB code.

As illustrated by Figure 1 (top row) and Figure 5, applying 1.5D SLAMs to 2D vector-art animations ensures a strong temporal coherence of the stylization, while preserving the 2D characteristics of the stroke textures. Note in particular the quasi-constant size of the scribbles at each zoom level, and the absence of blending artifacts. This approach can also be used for fixed 3D segments, such as the edges of a cube (see the accompanying video for an example in motion).

4 Coherent Line Parameterization

Applying Self-Similar Lines Artmaps to complex curved paths – represented as a list of segments – requires the definition of a parameterization T. The most straightforward choice for T is the screen-space arclength of the path s. However, for any view-dependent feature lines (silhouettes, suggestive contours, apparent ridges, etc.) this approach produces "swimming" artifacts, as no temporal coherence is ensured from one frame to the next.

To solve this problem, we propose a simple three step screen-space parameterization scheme, which finds the parameters (ρ, ϕ) such that $T(s) = \rho s + \phi$ best fits the parameterization of the previous frame. Section 4.1 describes how we propagate the parameterization from frame f to $f + 1$ using splatting. Section 4.2 and 4.3 detail how these parameters are updated and combined with SLAMs to render strokes. Figure 6 gives an overview of the whole process.

4.1 Propagating Parameters

The first step is to propagate the parameterization of all the paths at frame f to the next frame $f + 1$. At frame f we evenly sample each visible path in screen-space at a user-defined sampling rate δ. Each sample is parameterized by its 2D arclength s_i and the current parameters of the path (ρ, ϕ).

Observing that view-dependent lines are moving in a rather smooth and continuous way in screen-space, the key idea of our approach is to extend the size of each sample and splat its corresponding parameterization into a screen-space buffer (the *parameterization buffer*). View-dependent lines at frame $f + 1$ can thus look up their closest correspondent at frame f, and the contributions of overlapping paths are added, tending to unify their parameterization. If the paths shift too far between frame f and $f + 1$ the lookup will fail, but coherence of the stylization may be unnecessary or even undesirable in that case.

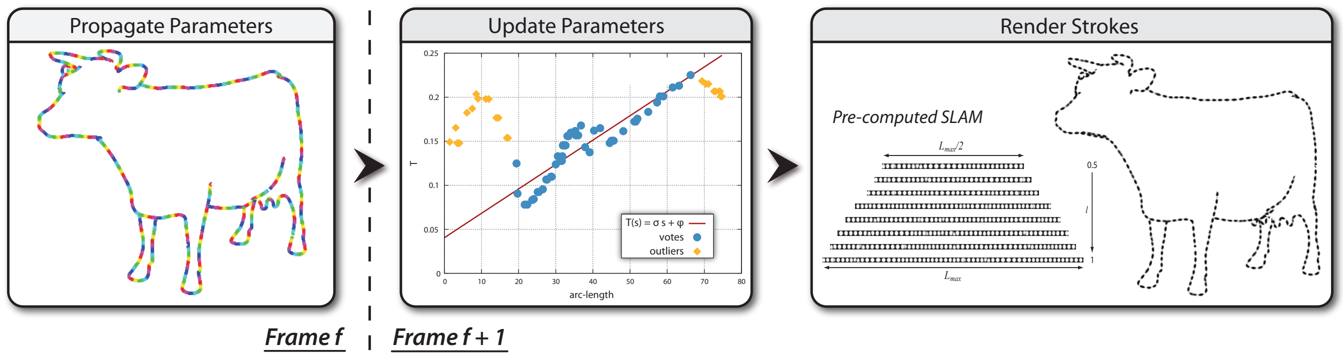
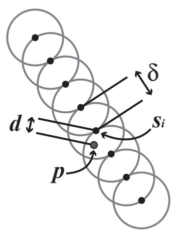


Figure 6: *Three steps pipeline for coherent line parameterization. At frame f, the parameterization of all the visible brush paths is splatted into a screen-space buffer. At frame $f+1$, this buffer is read back, and parameterization votes are recorded for each path. The new scale factor s and phase ϕ that best fit these votes are robustly estimated using RANSAC. Finally, paths are rendered according to this parameterization, choosing the appropriate level in a pre-computed SLAM.*



The sample parameterization $t_i = \rho s_i + \phi$ is written as a textured sprite of radius δ in a floating point off-screen buffer (inset left). Each location p at a distance d from the center s_i of the sprite determines a weight for the splatted parameterization t_i by:

$$w_i = clamp(1 - \frac{d}{\delta}, 0, 1)$$

We keep track of the weight by splatting w_i in another color channel. Thus, at the end of this step, each pixel of the buffer covered by j overlapping samples contains $(\sum_j w_j t_j, \sum_j w_j)$. Note that by design this weighting scheme corresponds to the exact linear interpolation of the parameterization along the spine of the line, if no paths overlap. The weight of the splat can be varied with the partial visibility [Cole and Finkelstein 2010] of the sample, or the z-depth of the sample at that point.

Unlike that of Kalnins et al. [2003], our method does not rely on an ID image (item buffer). The item buffer approach can introduce aliasing artifacts, and requires a local neighborhood search for each sample. Our splatting approach avoids this search, and also allows the additional control of varying the sprite size to trade-off coherence of parameterization against parameterization overlap. Increasing the size of the sprite makes it less likely that a path will move outside the sprite radius in a single frame, but increases the chance that separate lines will interfere with each other.

4.2 Updating Parameters

At frame $f+1$, we uniformly sample the new visible brush paths in screen-space and reproject each sample to the previous frame's parameterization buffer, using the camera from frame f. If $\sum_j w_j$ is defined at this pixel location, we record a vote:

$$v_k = \left(\frac{\sum_j w_j t_j}{\sum_j w_j}, s_k \right) \qquad (3)$$

which is a pair of values that associates the averaged parameterization at screen-space location j in frame f with the current arclength parameter s_k of this sample.

For each path, we compute the scale factor ρ and the phase ϕ that best fit the votes in the least-square sense. As the input votes can be very noisy when paths overlap, we use the RANSAC algorithm [Fischler and Bolles 1981] to robustly estimate these two parameters. This iterative method allows a trade-off between speed and accuracy: the user estimates the proportion of outlying votes (refined according to data during the computation) and a target probability for inlying votes, which control the number of required iterations.

If fewer than two votes have been recorded for a path, we parameterize the path by $T(s) = s/L$ where L is the path length, such that T ranges in $[0, 1]$. The same default parameterization is used to initialize the first frame rendering.

Note that this approach is similar in spirit to the mixed policy described in [Bourdev 1998] and used by [Kalnins et al. 2002]. As this policy mixes information from multiple paths, popping can occur when two paths merge. Our method shares this limitation, however the robust fitting operation reduces sensitivity to outlying votes, allowing paths to quickly find a common parameterization. We thus avoid the fragmentation problem encountered by [Kalnins et al. 2003].

4.3 Stroke Rendering

In the last stage, we use the path parameters to determine the appropriate SLAM level l and render the final textured stroke. To ensure that paths moving off-screen remain at the same level in the SLAM, we define l as a function of the scaling factor ρ. Unlike the length of the path, ρ does not change as the path is clipped. Taking into account the self-similarity of the SLAM, the formula is:

$$\begin{aligned} d &= \rho k L_{\max} \\ l &= \left\{ \begin{array}{ll} \omega/d & \text{if } d > 2 \\ 1/d & \text{if } d \in [1,2] \\ 1/(d\Omega) & \text{if } d < 1 \end{array} \right. \Bigg| \in [0.5, 1] \end{aligned}$$

with:

$$\omega = 2^{\lfloor log_2(1/d) \rfloor} \quad \text{and} \quad \Omega = \lfloor 2/d \rfloor$$

where $L_{\max}$ is the length of the longest texture in the SLAM and k is a user-defined scaling factor. When looping from one end of the truncated pyramid to the other, the path parameterization T has to be scaled by $1/\omega$ and Ω respectively.

A SLAM can be simply represented as a 3D texture, allowing us to make use of trilinear filtering on the GPU. We assign 3D texture coordinates to each segment according to the path parameterization and SLAM level ($w = 2l - 1$).

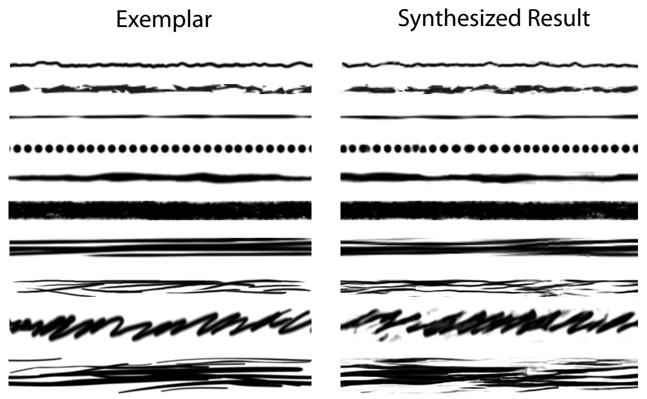


Figure 7: *Texture synthesis results. The method of [Portilla and Simoncelli 2000] produces good results for many types of brush textures, especially small or simple textures (top half). More complex textures (bottom half) are reproduced credibly, but have imperfections in detail. Other approaches to SLAM generation may be possible to improve the results further (Section 6.1).*

5 Results

Our SLAM construction process leverages the texture synthesis code of Portilla and Simoncelli [2000], and our implementation is available for download.[1] Results for several different brush textures are shown in Figure 7. For simple textures, the PST algorithm produces results almost indistinguishable from the exemplar. For more complex textures such as the scribble and the jumbled lines (bottom of Figure 7) the overall impression of the synthesized result is similar, but some details are lost. High quality SLAMs are composed of 17 gray-scale images of 1024×32 pixels (100 to 300 KB in PNG). Synthesis typically takes a few minutes.

We implemented the entire coherent line parameterization method on the CPU without fine optimizations. We use the segment atlas data-structure describe in [Cole and Finkelstein 2010] to compute the visibility of the paths and perform the final rendering of the strokes on the GPU. Nevertheless, our approach could be used with any system that renders lines as textured quads, such as [McGuire and Hughes 2004; Kalnins et al. 2002; Isenberg et al. 2002; Markosian et al. 1997]. Our system runs at interactive framerates for scenes of moderate complexity on a commodity PC with an Intel Core 2 Duo 2.4GHz CPU and 4 GB RAM. The bottleneck of our approach is the RANSAC algorithm, which varies in performance depending on the number of visible segments and the number of outliers. However, this algorithm is intrinsically parallel since the parameterization of each path can be fitted independently. Our system uses OpenMP to accelerate this step on the CPU. Although we currently do not have a GPU implementation of this algorithm, we believe it should be practical and would significantly improve performance for complex models.

Figure 8 illustrates the wide variety of styles which can be achieved applying different SLAMs on lines extracted from 3D models. Note that creating new styles only requiers to synthesis new SLAMs by example. We refer the reader to the accompanying video to appreciate the temporal coherence of this stylization during the animation. Even if some popping is noticeable when brush paths merge, our simple parameterization scheme in conjunction with SLAMs provides a convincing screen-space behavior for most textured lines.

[1] http://www.cs.princeton.edu/gfx/proj/dpix/

6 Conclusion

Self-similar line artmaps help address one of the principal challenges in stylized rendering: how to animate strokes with strong textures such as dots, scribbles, or brush marks. While SLAMs are a special case of conventional artmaps, we believe that the additional conditions of self-similarity and smooth variation extend the usefulness of SLAM textures to areas where artmaps have not seen much popularity, such as 2D animation and rendering of smooth 3D models.

The two additional conditions make an automatic method for SLAM creation more important than for conventional artmaps, since self-similarity cannot be easily obtained by just progressively simplifying a texture. Our texture synthesis scheme creates self-similar artmaps of arbitrary density, and produces good results for a range of texture types.

Our parameterization approach for rendering smooth 3D models with SLAM textures is similar in spirit to [Kalnins et al. 2003], but includes several simplifications and improvements. It handles partial visibility, which has been shown to be important for temporal coherence [Cole and Finkelstein 2010]. Since we only consider SLAM textures, we do not require a complex, piece-wise fitting scheme.

6.1 Limitations and Future Work

Our system does not satisfactorily handle several types of applications that we hope to address in future work.

We believe the quality of our synthesized textures is more than sufficient for many styles of rendering, but the quality is not perfect. As shown in Figure 7, complex textures may not be reproduced faithfully. Conventional non-parametric synthesis does not satisfy the condition of smooth variation (Section 3.1), but texture advection techniques [Kwatra et al. 2005; Bousseau et al. 2007] may provide an alternative for synthesis. In our experiments, however, the animation of the PTS textures was more visually appealing than basic advection. We believe a more promising route to higher-quality is to use the synthesized textures *themselves* as seeds for a final rendering pass using procedural strokes. For example, our synthesized dotted line texture could be used as a guide for the size and placement of particle sprites.

While our parameterization method for 3D models provides a good trade-off between performance and implementation complexity, the general problem of temporally coherent line parameterization remains unsolved. In particular, neither our method nor previous work (e.g. [Kalnins et al. 2003]) works well for complex models with many overlapping lines. Satisfactory results for complex models will likely require a new approach that does not rely solely on a screen-space buffer to propagate parameterization from frame to frame.

Acknowledgements

Thanks to Georges Winkenbach and the stylization team at Google SketchUp for helpful discussions. This work was sponsored in part by Adobe, Google, and the ExploraDoc program of the region Rhône-Alpes. Many of the images shown in this paper are rendered from 3D models courtesy of Georgia Tech, Viewpoint, Aim@Shape, and the Google 3D Warehouse.

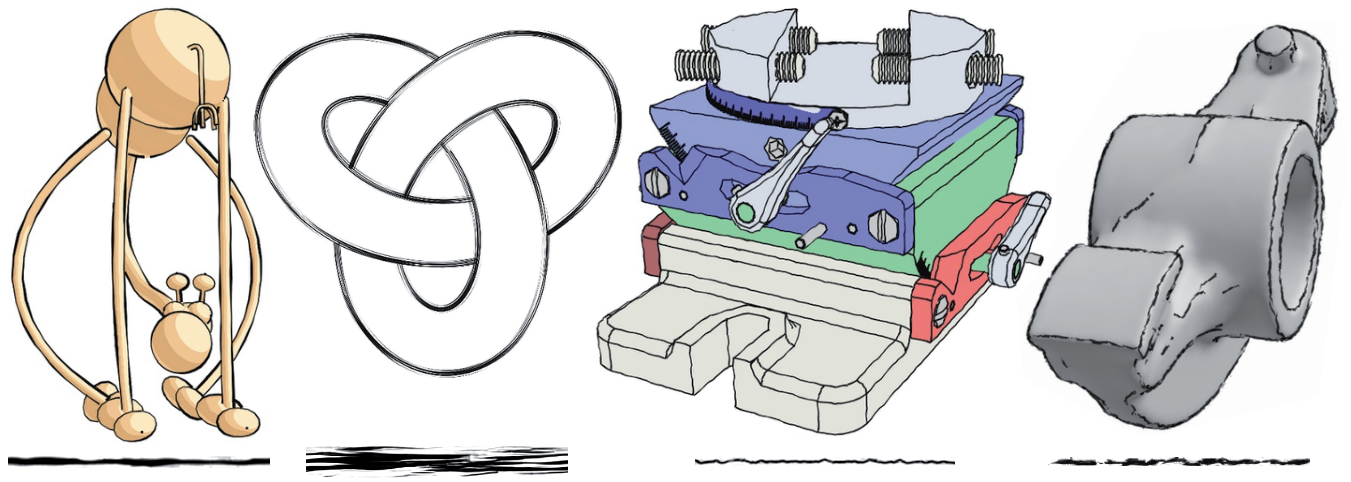


Figure 8: *Various styles obtained by applying SLAMs on lines extracted from 3D models. The bottom row shows the line texture of one level of the associated SLAM.*

References

BÉNARD, P., BOUSSEAU, A., AND THOLLOT, J. 2009. Dynamic solid textures for real-time coherent stylization. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM, 121–127.

BÉNARD, P., THOLLOT, J., AND SILLION, F. 2009. Quality assessment of fractalized NPR textures: a perceptual objective metric. In *Proc. 6th Symp. Applied Perception in Graphics and Visualization*, 117–120.

BOURDEV, L. 1998. *Rendering Nonphotorealistic Strokes with Temporal and Arc-Length Coherence*. Master's thesis, Brown University.

BOUSSEAU, A., NEYRET, F., THOLLOT, J., AND SALESIN, D. 2007. Video watercolorization using bidirectional texture advection. *ACM Transactions on Graphics 26*, 3, 104.

COLE, F., AND FINKELSTEIN, A. 2010. Two fast methods for high-quality line visibility. *IEEE Transactions on Visualization and Computer Graphics*.

CUNZI, M., THOLLOT, J., PARIS, S., DEBUNNE, G., GASCUEL, J.-D., AND DURAND, F. 2003. Dynamic canvas for immersive non-photorealistic walkthroughs. In *Proc. Graphics Interface*.

FISCHLER, M. A., AND BOLLES, R. C. 1981. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM 24*, 6, 381–395.

GOOGLE. 2008. Sketchup style builder. In *http://sketchup.google.com*.

HAN, C., RISSER, E., RAMAMOORTHI, R., AND GRINSPUN, E. 2008. Multiscale texture synthesis. *ACM Transactions on Graphics 27*, 3, 1–8.

ISENBERG, T., HALPER, N., AND STROTHOTTE, T. 2002. Stylizing silhouettes at interactive rates: From silhouette edges to silhouette strokes. *Computer Graphics Forum 21*, 3, 249–258.

KALNINS, R. D., MARKOSIAN, L., MEIER, B. J., KOWALSKI, M. A., LEE, J. C., DAVIDSON, P. L., WEBB, M., HUGHES, J. F., AND FINKELSTEIN, A. 2002. WYSIWYG NPR: drawing strokes directly on 3D models. In *ACM Transactions on Graphics*, ACM, 755–762.

KALNINS, R. D., DAVIDSON, P. L., MARKOSIAN, L., AND FINKELSTEIN, A. 2003. Coherent stylized silhouettes. *ACM Transactions on Graphics 22*, 3, 856–861.

KLEIN, A. W., LI, W. W., KAZHDAN, M. M., CORREA, W. T., FINKELSTEIN, A., AND FUNKHOUSER, T. A. 2000. Non-photorealistic virtual environments. In *ACM Transactions on Graphics*, 527–534.

KWATRA, V., ESSA, I., BOBICK, A., AND KWATRA, N. 2005. Texture optimization for example-based synthesis. *Proceedings of ACM SIGGRAPH 2005 24*, 3, 795–802.

LEFEBVRE, S., AND HOPPE, H. 2006. Appearance-space texture synthesis. *ACM Transactions on Graphics 25*, 3, 541–548.

MARKOSIAN, L., KOWALSKI, M. A., GOLDSTEIN, D., TRYCHIN, S. J., HUGHES, J. F., AND BOURDEV, L. D. 1997. Real-time nonphotorealistic rendering. In *SIGGRAPH '97*, ACM, 415–420.

MASUCH, M., SCHUMANN, L., AND SCHLECHTWEG, S. 1998. Animating frame-to-frame consistent line drawings for illustrative purposes. In *SimVis*, 101–112.

MCGUIRE, M., AND HUGHES, J. F. 2004. Hardware-determined feature edges. In *NPAR '2004: international symposium on non-photorealistic animation and rendering*, ACM, 35–47.

PORTILLA, J., AND SIMONCELLI, E. P. 2000. A parametric texture model based on joint statistics of complex wavelet coefficients. *International Journal of Computer Vision 40*, 1, 49–70.

PRAUN, E., HOPPE, H., WEBB, M., AND FINKELSTEIN, A. 2001. Real-time hatching. In *ACM Transactions on Graphics*, ACM, 579–584.

RUSINKIEWICZ, S., COLE, F., DECARLO, D., AND FINKELSTEIN, A. 2008. Annotated bibliography: Published research on line drawings from 3D data. In *SIGGRAPH 2008 Course Notes: Line Drawings from 3D Models*. http://tinyurl.com/S08LinesCourse.