Automatic Pen Stroke Guidance by Example

Justin Solomon, Charlton Soesanto, and Haithem Turki
CS 221: Introduction to Artificial Intelligence

1 Introduction

Although the intricacies of generating a given stylized image may depend on the artist's vision of each component of a scene, most pen-and-ink drawings and similar stroke-based renderings have a consistent style across entire segments of the final work. For instance, a whimsical drawing may use fairly uncorrelated strokes with varying curvature, while a more accurate depiction of a scene might rely on small straight lines and checkers to communicate shading, occlusion, and other effects. Large-scale choices of stroke style are subject to the artist's vision, but often times smaller-scale texturing can become a time-consuming and repetitive task. In this project, we present a prototype of a machine learning tool to help artists spend less time generating small-scale details. Using an artist's initial strokes as training data, we train an automatic pen tool that places and shapes strokes with minimal guidance. Such a tool could be used to add detail to rough sketches or texture regions that would otherwise be frustrating to fill in.

Our tool takes as input strokes drawn by the user over a background image depicting the scene he or she intends to render; there are no restrictions on the background image, which can be a photograph, sketch, or other rendering. Strokes are collected through a simple pen-based interface and smoothed to provide more natural splines connecting control points. Then, the user uses a lasso tool to select a set of strokes from which the tool should learn a stroke technique. Finally, the same lasso is used to select an output region in which the learned stroke generator places curves imitating the style of the strokes provided by the user. This setup is illustrated in Figure 1; it is designed to allow maximum artistic control.

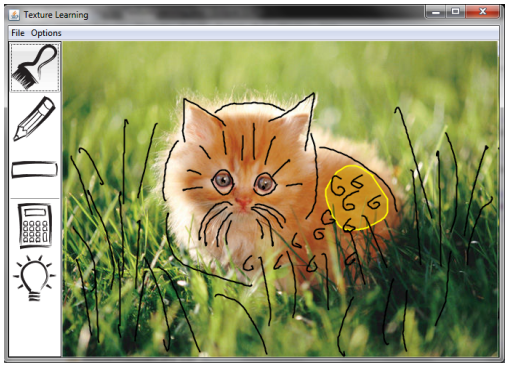


Figure 1: User interface; strokes are drawn in black, and the region of interest is a yellow polygon. The user clicks the calculator to run learning, can draw new regions of interest, and subsequently clicks the lightbulb to use the learned model.

The main technical rather than interface development is a method for supervised learning of the placement, length, and shape of pen strokes from user-provided data. We choose to model this decision-making process in two stages in analogy with how an artist might do the same: the computer (or artist) picks an initial pen position and orientation, and subsequently guides the pen along the paper. The first stage is accomplished using several probability distributions reflecting the likelihood of various pen positions and orientations given the background image, placement of existing strokes, and analogous decisions provided as training data. Then, the pen path is guided deterministically using curvatures learned from the examples; this stage is deterministic to avoid generating noisy paths.

In this paper, we present each step described above as implemented in our Java-based application and discuss several variations we implemented for experimentation and comparison. The full system reveals the strengths and weaknesses of each component of our approach, and while there is much to be done for a complete sketch-from-learning system, it shows that the general strategy is likely to yield an effective new tool.

2 Prior Work

There is a considerable literature in the computer graphics community on "non-photorealistic rendering" (NPR), in which different approaches are used to draw a scene to communicate meaning or expression rather than photographic plausibility. Here we cannot provide a review of artist-guided approaches to NPR and refer the reader to recent SIGGRAPH proceedings to review the state-of-the-art; a survey of relevant older work can be found in [4].

Most example-based "supervised" NPR techniques process and generate bitmapped rather than vector data. The most widely-cited such technique, entitled "Image Analogies," is described in [6]. This paper describes a method for generating image D from images A, B, and C using the analogy relationship $A : B :: C : D$; that is, if we somehow modified image A to get B, the method attempts to modify C in a similar way and output image D. The particular approach in [6] is a multiscale closest-point method suited to densely-sampled bitmapped representations. This paper

lies within a larger body of work related to texture synthesis and has been extended and modified in many ways; for instance, [2] provides an approach that is similar in spirit that simultaneously considers several potential examples of style and texture but still is texture- rather than vector-based. An additional extension [5] of the Image Analogies method to curve stylization provides one possible adaptation of the method to vector data, although it does not offer the degree of curve shape and placement control that we hope to achieve.

Other "stroke-based rendering" approaches are more closely-related to the work at hand in that they generate stroke curves rather than process individual pixels. Stroke-based papers mostly describe ad-hoc procedural models for rendering strokes from developers' intuitions about the painting process, as well as the physical structure of paint brushes and other implements. For instance, considerable work has been put into modeling brush tips [8]. One notable exception is [9], which synthesizes new curves from given spline examples. Our approach differs from this recent paper in several ways. Curves are assembled from pieces of the examples rather than statistical properties of how they bend and interact with a background image; the paper provides an exploratory tool that gives variations on the inputs rather than using them to match a desired output pattern or image.

3 Approach

Here we describe our approach to learning curve placement and shape models from the provided examples and then using those models to generate new curves. The learning and construction phases take place in multiple steps detailed in the individual subsections below.

3.1 Input

As input to our learning method, the user provides a background image, a set of curves or penstrokes drawn over the background, and a polygonal region of interest. All of these can be input easily using the interface depicted in Figure 1.

As usual, the background image is a simple bitmapped image that can be treated as a function $I : (x, y) \rightarrow (r, g, b)$; various blurs, edge detectors, and other filters are applied to I throughout the course of the algorithm. Each of the input curves is represented using a poly-line with vertices $(x_1, y_1), \ldots, (x_k, y_k)$; we use Catmull-Rom splines with $\tau = \frac{1}{2}$ to refine the poly-lines and add more vertices until they satisfy a user-defined minimum sampling rate. Polygonal regions of interest are also specified as poly-lines with the condition that $(x_k, y_k) = (x_1, y_1)$. The learning algorithm only considers those curves for which over 90% of the vertices are within the polygonal region of interest.

3.2 Learning Curve Placement

While we assume the output curve is continuous in the sense that the user never picks up his or her pen until starting the next curve, the choice of an initial curve location has no such locality property. Thus, we cannot use identical models for these two stages but are forced to implement two different steps.

We use a Gaussian Mixture Model (GMM) to describe the distribution of curve starting point pixels. That is, given meta-data about a pixel's location and the background image at that point, we output the likelihood that the pixel is a starting point. In particular, we use color, brightness, the Sobel edge detector, proximity to user- and automatically-placed curves, and the distance to the region of interest border.[1] Each sample curve is treated as a sample from a distribution on $\mathbb{R}^4$. These samples are used to formulate a GMM using the Bregman soft clustering method in [1], which we found more stable to our relatively low sampling rate than EM.

We use similar GMMs to describe the lengths and initial orientations of curves. We append to the pixel meta-data about the example curve starting points the length or direction angle of the curve to obtain joint distributions showing how these values depend on the meta-data; in a later step, these distributions will be conditionalized to decide on the values for a given pixel. Note that for deciding initial curve orientation, we add directional meta-data, such as the gradient of the background image and its "oriented direction," computed using the algorithm in [7].

3.3 Learning Curve Shape

While small, essentially random variations in the choice of curve placement and orientation may have minimal perceptual effects on the final textured image, perturbing the curve itself leads to tangent discontinuities, which are highly undesirable in generating a smooth output. Thus, we found the distribution approach above for curve placement ineffective for generating a curve's shape.

[1] The user is given the option of turning these on or off to help the learning algorithm decide what is important. We also give the user the option to apply Gaussian blurs to these values to decrease high-frequency changes.

Before describing our particular model, it is useful to describe a few simple notions from the differential geometry of plane curves. In this theoretical setting, curves are described as functions $\gamma(s) : \mathbb{R} \to \mathbb{R}^2$. Of course, given any monotonic function $f : \mathbb{R} \to \mathbb{R}$ the curve γ can be reparameterized using the composition $\gamma \circ f$ without changing its shape, so we add the constraint that the parameter s represents arc length; that is, $\gamma(s_0)$ and $\gamma(s_1)$ are distance $|s_1 - s_0|$ apart when traveling along γ. Then, we define the *curvature* of γ to be the second derivative $\kappa = \gamma''$. The central theorem of the differential geometric treatment of planar curves is the following:

Theorem (Fundamental Theorem of Plane Curves)**.** *Two curves γ_1 and γ_2 parameterized by arc length represent the same shape if and only if the have the same starting point, orientation, and curvature functions κ.*

This theorem shows that once we have decided where to place a pen curve, we can simply decide on its curvature function to reconstruct the curve shape. Curvature is a local differential property, which we use to our advantage: the decision of a curvature value does not need any global curve information to maintain smoothness. Also, it is easily discretized on a curve composed of short line segments as the ratio of angle defect to segment length [3], as shown in Figure 2.

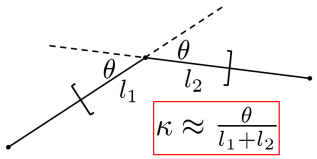


Figure 2: Set-up for computing discrete curvature at the join between two segments.

Our general approach to learning curve shape is to fit a function that takes in a number of heuristics about the location of the pen tip and the curve we have drawn so far and outputs a single curvature value, which is converted to a turning angle and used to guide the pen a fixed amount. In addition to the background image heuristics from Section 3.2 evaluated at the current pen tip, we include a number of curve shape heuristics including the curvature at previous control point, the average curvature so far, the radius of curvature $\frac{1}{\kappa}$, the distance since last curvature observed in a given range, the curvature derivative κ' (implemented using simple finite differences), the length of the curve (decided *a priori*), and the percent of the curve traversed so far. When concatenated together, these heuristics form a vector in $\mathbb{R}^N$ for $N \approx 15$ depending on user preferences regarding which heuristics to use. So in short, we want to learn a function $f : \mathbb{R}^N \to \mathbb{R}$ that takes in curve heuristics and outputs curvature. As an aside, in practice N is relatively large compared to our number of training samples, so we reduce N to approximately $\frac{N}{2}$ by projecting onto a lower-dimensional PCA basis determined from the data; this also helps avoid repeated information provided by heuristics that behave similarly.

We have reduced driving pen directions from curvature to fitting a function f given examples $f(\vec{x}_1) = \kappa_1, \ldots, f(\vec{x}_m) = \kappa_m$. There are a number of pre-existing methods from machine learning for this function estimation problem. In particular, we experimented with least squares, kernel ridge (L^2-kernelized) regression, and support vector (L^1-kernelized) regression. Surprisingly, we found that the non-kernelized least squares approach was more effective than the other two, at least without implementing cross-validation methods for choosing radial basis function radii. Since N is sufficiently large, it appears that least-squares was able to use the N values directly to formulate an output curvature, rather than using more complex nonlinear methods for which one must choose a radius of support for the different data points (a non-trivial task since there is no obvious metric on $\mathbb{R}^N$ that takes into account the semantic meanings and relationships between the heuristics).

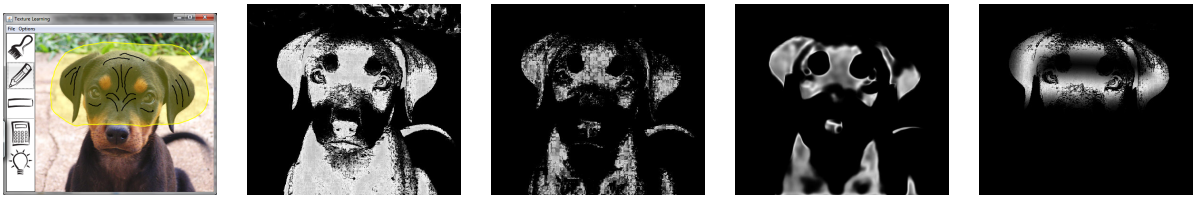
3.4 Making Use of the Learned Models

Once the learning stage is complete, the user can specify another region of interest and begin filling it with automatically-generated pen curves. This process for a single curve, which can be repeated arbitrarily, is divided into three stages.[2]

Choosing a starting point To choose a starting point for a curve, at each pixel we evaluate the curve placement heuristics to find a probability density value p_{place} representing the likelihood that the pixel is a curve starting point. We then sample a single (x, y) pair from the set of all pixels with probability proportional to p_{place}.

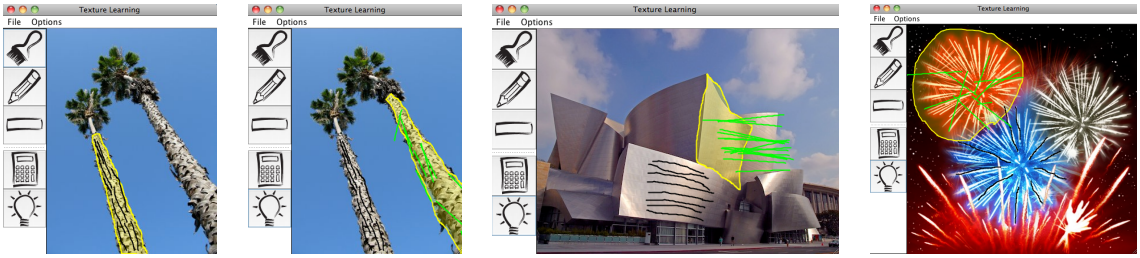
Choosing a length and initial direction For curve length heuristics $H_1, \ldots, H_n$, our learning algorithm provided a mixture of Gaussians expression for the joint distribution $P(L, H_1, \ldots, H_n)$. To decide on a particular length, we evaluate $H_1, \ldots, H_n$ at (x, y) and then conditionalize the distribution to find the single-valued distribution $P(L|H_1, \ldots, H_n)$; this conditionalization can be carried out analytically using the common formula for the

[2]We do not treat how to decide when to start or finish drawing curves but instead assume the user provides a fixed number of desired strokes. Determining this number automatically is an important topic for future research.



(a) (b) (c) (d) (e)

Figure 3: Curve placement probabilities for the example curves in (a) with various options for probability model: (b) probability from brightness and edges, (c) probability from pixel color, (d) probability from blurred pixel color, (e) probability from brightness and distance to region of interest boundary. Square artifacts are due to JPEG compression.



(a) (b) (c) (d)

Figure 4: Choice of curve placement, direction, and length rendered using line segments rather than pen curves; (a) and (b) show a training data and synthesized data pair with marked regions of interest, while (c) and (d) contain synthesized output and input segments.

conditional distribution of a Gaussian. To choose a particular curve length at (x, y), we simply sample this conditional distribution. An identical process occurs for deciding the initial curve orientation.

Curve construction We construct the curve incrementally, adding vertices one at a time until we reach the length. For each vertex, we simply compute the curve guidance heuristics for the curve built so far and evaluate the learned function f to estimate curvature. The turning angle representation of curvature allows us to determine the location of the next sample point, and the process is repeated.

4 Results

The algorithm was implemented in Java, which was chosen for its numerical methods and interfaces. We used a number of libraries to avoid re-implementing common numerical routines: EJML for matrix manipulation and factorization, DREJ for kernelized least squares, LibSVM for support vector regression, and jMEF for Bregman soft clustering (which replaced our own implementation of the EM algorithm). We wrote wrappers for these libraries to avoid exposing the particularities of their IO communication methods. Rather than adding yet another library to our implementation, we implement principal components analysis (PCA) by hand for reducing the dimensionality of the curve-drawing input; fortunately, efficiency needed not a huge concern since N is relatively small.

Figure 3 shows sample outputs of the curve placement algorithm; the probabilities p_{place} are scaled to $[0, 1]$ and shown at each pixel. This part of our algorithm was particularly effective, consistently choosing relevant starting points. As shown in the Figure, the placement probabilities were strong approximations of the relevance of different points, and using them probabilistically to find starting points was an effective strategy.

The strategies for choosing length and initial orientation were similarly successful. Figure 4 depicts choices of length and orientation using directed line segments rather than curves. In our experiments, it appears that artists are fairly consistent with their choices of length and orientation, and even the most complex checkered patterns appear to have fairly clear at most bimodal distributions of angles and lengths.

Finally, Figure 5 shows outputs of the full algorithm. Despite considerable experimentation with varied methods

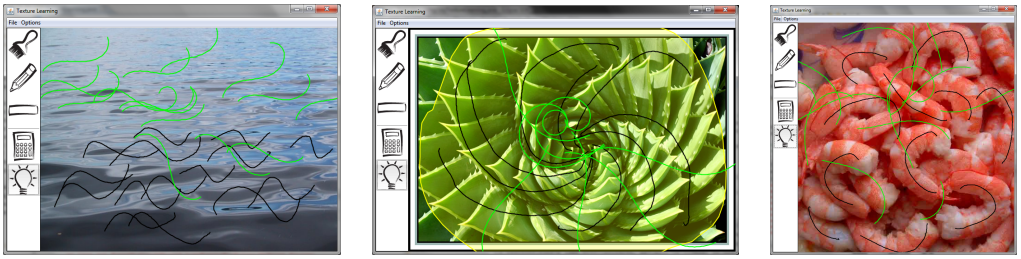


Figure 5: Examples of algorithm output; black curves are examples and green curves are automatically produced by the model. These figures show the advantages and disadvantages of the approach at hand. Linear regression only allows for fairly simplistic spiral curves. Even so, they capture basic curvature relationshipsbetween curvature and arc length, and they have reasonable shape. Placement works fairly well; for instance, the green spiral curves all start at the center of the image, and their initial directions are distributed evenly.

for function fitting, heuristics, parameter choices, and other decision, we were unable to come up with a single satisfactory approach to pen curve drawing. While refining this final step remains a topic for future research (one likely to yield interesting machine learning and geometric algorithms), we found that the simple linear least-squares method often produced satisfactory curves for simple examples, as discussed earlier. In the future, we would like to try additional nonparametric methods for learning the curvature function f that do not require as much parameter tuning; alternatively, we could try methods that automatically choose reasonable parameters so that the underfitting we tended to observe with the complex kernelized methods was not as much of a problem.

5 Discussion

We present a complete multi-step algorithm for synthesizing new pen curves from examples and a background image to portray a given scene or object. Our framework allows for a higher degree of artist control than previous approaches, allowing the user to select regions of interest and guide by placement and shape examples. Our approach makes use of several machine learning and artificial intelligence techniques, including regression, probabilistic modeling, principal components analysis, and data interpolation (for refining input curves). Although final results are mixed, they indicate that a refined version of our method shows promise for the implementation of a "smart" curve placement system.

The most obvious topic for future research is to find the optimal combination of heuristics and regression techniques to compute new curves reliably; a study attempting to solve this problem would need to compare a number of techniques side-by-side, ensuring that each has a reasonable choice of parameters. Beyond this immediate research goal, it would be useful to have a model of curve density to determine when to start and stop drawing curves in a given region. Once all of these components are in place, interaction studies could be put into place to find the interface that is most natural for artists making use of such a tool. Finally, with both a reliable back-end learning system for fabricating curves as well as an effective interface, this tool easily could be integrated with existing "paint" software as a useful brush to be used alongside more traditional pen tools.

References

[1] Banerjee, A. et al. "Clustering with Bregman Divergences." *Journal of Machine Learning Research* 6 (2005): 1705–1749.

[2] Drori, I. et al. "Example-Based Style Synthesis." *CVPR 2003* 2: 143–150.

[3] Grinspun, E. and A. Secord. "Introduction to Discrete Differential Geometry: The Geometry of Plane Curves." *SIGGRAPH Asia 2008 Courses*.

[4] Hertzmann, A.. "A Survey of Stroke-Based Rendering." *IEEE Computer Graphics* 23 (2003): 70–81.

[5] Hertzmann, A. et al. "Curve Analogies." *13th Eurographics Workshop on Rendering* (2002): 233–246.

[6] Hertzmann, A. et al. "Image Analogies." *SIGGRAPH 2001*: 327–340.

[7] Kass, M. and A. Witkin. "Analyzing Oriented Patterns." *Computer Vision, Graphics, and Image Processing* 37.3 (1987): 362–385.

[8] Lin, M. et al. "Physically Based Virtual Painting." *Communications of the ACM* 47.8 (2004): 40–47.

[9] Merrell, P. and D. Manocha. "Example-Based Curve Synthesis." *Computers and Graphics* 34.4 (2010): 304–311.