Learning Refinements on Curve-Strokes

Saul Simhon Gregory Dudek
Centre for Intelligent Machines
McGill University
Montreal, Canada

Abstract

We present a system to beautify curves: i.e. to take curves that roughly depict some property of interest and make them look more like what experts would draw. The focus of our work is in applications for artistic drawings, but our system can also be plugged into various other domains where curves and trajectories play a dominant role such as robot path planning, animation or edge-deblurring. Our approach consists of learning properties from a database of ideal example, which could be sketches or robot trajectories, and transform a coarse input curve to make it look like those in the database. The key scientific issue is: in what sense are these curves 'like' one another? In our work, this likeness is expressed statistically. Using Hidden Markov Models in combination with multi-scale methods and mixture models, we synthesize a new curve as a statistically consistent mixture of the training set that best describes the input. Additionally, our approach allows us to easily include application specific biases to the system.

1. Introduction

Curves play an important role in a wide array of domains. From applications in CAD modeling to image processing, many components consists of some kind of interaction with a set of functions that describe the shapes or constraints of curves. In particular, interfaces for simple drawings consisting of only curve strokes are often found for comics, presentation material, cel-animation, storyboard designs, system designs and non-photorealistic pen and ink illustrations. Although almost anyone can sketch rough outlines, only a few of us are lucky enough to have the artistic talent and patience to draw refined details for enhanced presentation. In this work, we consider how to model the *feel* of an ensemble of example curves that exhibit a desired *look*. We use this to develop a method for artificially augmenting coarse curves using learned refinements.

Traditionally, the tools available for creating or interacting with curves are based on application specific analytic models. For example, generating smooth curves typically consists of using models with spline basis functions. Generating motion curves can require additional complex formulations to simulate the kinematics of a moving object. Whatever the constraints, expressing them in a suitable formal framework is often challenging. Further, the processes of finding solutions can be costly, particularly since the solution techniques are often engineered for a specific context.

In contrast, this paper presents a different approach to curve synthesis. Constraints (or preferences) are expressed in terms of a set of examples that illustrate how the curves should behave. Further, these examples indicate how to elaborate an input curve from a user or high-level specialized planner (which may not be acceptable in itself) into a suitably acceptable output curve. Informally, the examples say: "if a user asks you to do something like *this* then what you should actually perform is something like *that*".

In this framework, we develop a tool that acts as a smart interface for curves. Coarse curves can be drawn (or loaded) and the system interactively augments the curves to make them *look* more like those in the database. This refinement is accomplished by synthesizing a new curve that is statistically consistent with the database examples while also considering the shape of the input curve. Our underlying approach consists of using a Hidden Markov Model (HMM). This allows us to represent measurements the system can directly observe (the users' curve) in conjunction with measurements that are hidden but somehow related to the observations (the desired refined shape).

A Hidden Markov Model is trained using a set of coupled examples, where each example consists of a refined curve associated to a *control* curve. The control curve can be thought of as what we expect the user to draw for that particular refined shape (learning a drawing habit or control scheme). After training, we can then synthesize a novel refined curve by decoding the HMM with observations from the input curve. This results in a statistically consistent mix-

ture of the training examples that best describes the observed input.

The paper is outlined as follows: first we review some related work, then describe how we train a HMM to capture the features of the training set, then we describe how we synthesize a refined curve, we further show how we can improve the system by including multi-scale curve components as plug-in attributes (while maintaining a first order Markov Model), then we show how to include specialized biases to the system and finally we discuss the results and conclude.

2. Related Work

There is ample and divers work by many authors in methods for modeling and editing curves. Much of the interesting work is also related in surface models rooted by a 2D parameterization of a curve. A traditional approach to curve modeling is to fit a specialized analytical model, such as NURBS [6, 4], over a set of data points and curve attributes. Rather than taking this specialized approach, we seek a method to learn those models from examples, providing the ability to capture a wide array of preferred interpolators in association to their control points.

This idea of learning to generalize specific examples to a broad ensemble of cases is, of course, the crux of classical machine learning [10]. Learning using Hidden Markov models is a longstanding classic research area, although to our knowledge it has never been applied to problems like this one. Likewise, although there has been some prior work on the relationship between learning and planning, most of this has dealt with more traditional plan formulation problems [13] or on learning suitable cues that control or determine plan synthesis or execution [11, 3].

Related areas where single Layer Markov Models have been applied include algorithms for texture synthesis [15, 2, 14] and motion synthesis [12, 1]. Such work suggests that we can learn the regular properties of an example and then synthesize novel outputs that exhibit the same statistics. This approach has recently been applied in curve synthesis for line-art [7] and curve hatching [8]. As single layer Markovian systems, most of these approaches do not actually learn the specific control attributes required for a *steerable* synthesis. In [7] some level of control is provided, but the synthesis itself is not biased over the absolute shape of the input curve but rather over its relative offset from the training example. Further, these methods typically use greedy type strategies for the synthesis, considering only the best match at the current point. When we are given inputs for controlling the synthesis, the best intermediate points often do not provide the global optimum. Future information often biases earlier points, for example, when drawing a vertical line, we do not know whether to apply brick features or bark features until we see the what will be drawn later. Our approach takes into account the entire sequence of inputs while also avoiding exponential run time complexity over the arc-length.

3. Learning

Our objective is to learn attributes from training examples in order to synthesize a refined curve given a coarse input. In our approach, the training examples consist of a set of coupled curves. In this coupling, one curve depicts the what a user may actually draw (the control curve) while the other curve depicts the intended refined shape (the stylized curve). What the training set attempts to say to our system is: *if the user draws something like this then the system should generate something like that*. Figure 1 shows some training examples for leaf styles. In this case, the control curves are actually low-pass filtered version of the displayed stylized curves, but in general they can be any shape to accommodate an arbitrary control scheme or drawing habit. We wish that after training, we can generate illustrations that exhibit the same *look* as the stylized example leaves.

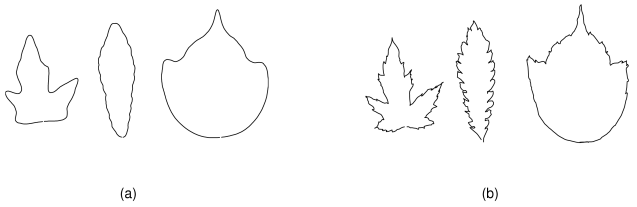


Figure 1. Some samples from a training set used for leaf synthesis. Figure (a) shows the control curves while figure (b) shows the stylized curves.

We learn local shape constrains and input biases from the examples using a Hidden Markov Model. A Hidden Markov Model encodes the dependencies of successive elements of a set of *hidden* states along with their relationship to *observable* states. It is typically used in cases where a set of states, that exhibit the Markov property, are not directly measurable but only their effect is visible through other observable states. Formally, a Hidden Markov Model Λ is defined as follows:

$$\Lambda = \{M, B, \pi\} \tag{1}$$

where M is the transition matrix with transition probabilities of the hidden states, $p\{h_i(t) \mid h_j(t-1)\}$, B is the confusion matrix containing the probability that a hidden state h_j generates an observation o_i, $p\{o_i(t) \mid h_j(t)\}$, and π is the initial distribution of the hidden states. In our work, sample points along the refined curves play the role of the hid-

den states while the sample points along the control curves play the role of the observations.

There is an abundance of literature on Hidden Markov Models and the domain is frequently decomposed into 3 critical sub-problems:

- Evaluation, where the likelihood of an HMM is evaluated for a sequence of observations: $p\{o \mid \Lambda\}$.
- Decoding, where the maximum likelihood sequence of hidden states is predicted for a given HMM and an observation sequence: $\arg\max_h \ p\{h \mid o, \Lambda\}$.
- Learning, where the transition probabilities, the confusion matrix and the initial distribution that best fit an observed set of examples are estimated.

Given only the observations, learning is most commonly performed by algorithms such as the Baum-Welch algorithm or generalized Expectation-Maximization methods. In our application, when learning we have direct access to both the hidden and observable states as given by the examples. Therefore, we can estimate an HMM by the the statistics of the training data, calculating probabilities of successive elements of the refined curves and their relationship to the control curves.

3.1. Hidden States

We represent a curve over 2D space parametrized by the arc-length. Let α represent a parametric curve $(x(t), y(t))$ where t is the arc-length of the curve from $0 <= t <= T$. Since we can, in principle, encode a function using only its derivations, we assume our curves are suitably normalized and encode them as a discreet succession of tangent angles $\theta(t)$.

Consider a stochastic process Δ as the source for a family of refined curves. Each curve is considered to be a random signal with characteristics described by the probability density function of the process. Let α denote a refined curve and $\theta(t)$ as the tangent angles of that curve. We assume that the sequence of samples $\theta(t)$ from $0 <= t <= T$ exhibit an n^{th} order Markov property, i.e. a Markov Process:

$$p\{\theta(t+1) \mid \theta(t), \theta(t-1), \ldots, \theta(t-n+1)\} \ =$$

$$p\{\theta(t+1) \mid \theta(t), \theta(t-1), \ldots, \theta(0)\}$$

This *locality* condition states that information from recent samples is sufficient to predict the next sample point. Further, the dependency is considered to be position-variant, where statistical relationships between successive points may be *non-stationary* with respect to the arc-length.

Sample points along the arc-length are represented by hidden states in the HMM. Given an ensemble of training examples, we can estimate the transition probabilities by the statistics of successive elements in predefined *stationary regions* and construct the transition matrix M where:

$$P_\theta(t+1) = M(t) \ P_\theta(t) \tag{2}$$

The transition matrix propagates the information embedded in the probability distribution $P_\theta(t)$ to predict the next distribution $P_\theta(t+1)$. For sets that exhibit stationarity $M(0) = M(1) = ... = M(T) = M$, measured over the entire signal. Otherwise, the transition matrix is calculated over fixed local regions of the curves across the ensemble of curves. The ability to specify the local regions of stationarity (hence global non-stationarity) allows us to accommodate for shapes that inherently posses some global constrains. We assume a uniform initial probability distribution $\pi = P_\theta(0)$, providing equal likelihoods to all curves at time zero.

3.2. Observable States

Sample points of the control curves are represented by observable states in the HMM. Based on the coupled association to the refined curve, an new input stroke can condition the distribution in Equation 2 and bias the synthesis according to the prescribed characteristics. Because the input stroke can be any arbitrary shape, we assume that samples of the control curve are independent. For all t and k in the domain for observation ϕ:

$$p\{\phi(t) \mid \phi(k)\} = p\{\phi(t)\}$$

That is, previous points generally do not provide information on what the next point may be. This satisfies the HMM assumption that the observable state sequence is independent over the sequence.

Let β denote the curve representing the associated control stroke and $\phi(t)$ as the tangent angles of that curve, then:

$$\alpha = \Psi \ \beta \tag{3}$$

where Ψ is some mapping that transforms the control curve to the refined one. The mapping in essence encodes the constraint relationship between the coupled pair. Given an ensemble of pairs of refined and control curves, we estimate the probabilities of the confusion matrix B from the statistics of associated sample points $(\theta(t), \phi(t))$ and form the following relation:

$$P_\phi(t) = B \ P_\theta(t) \tag{4}$$

where the elements of the confusion matrix are the conditional probabilities $p(\phi_i|\theta_j)$ for all states i and j. The confusion matrix transforms the belief vector of the hidden states

to the belief vector of the observations. This is analogous to the inverse relation of the mapping Ψ in Equation 3. However, using Bayes law, one can show that solving the decoding problem for a HMM in a maximum likelihood sense is analogous to solving for the desired transformation Ψ. (Posing this in a variational calculus form [9], it can be shown how this solves the inverse problem in a maximum likelihood sense.)

4. Synthesis

Given a set of observations and a HMM trained with a family of curves, we generate a new curve by solving for the maximum likelihood hidden state sequence:

$$\begin{gathered} \arg\max_{\theta_i \ldots \theta_n} \; p\{\theta(0), \ldots, \theta(T) \mid \phi(0), \ldots, \phi(T), \Lambda\} \\ or \\ \arg\max_{\alpha} \; p\{\alpha \mid \beta, \Lambda\} \end{gathered} \tag{5}$$

This is accomplished using the *Viterbi* algorithm for HMM decoding with a runtime of $O(N^2T)$, where N is the number of states and T is the sequence length. At each time interval, we propagate the underlying probability distribution as in Equation 2 and maintain states with maximum consistency across successive elements. For each state θ_i:

$$p\{\theta_i(t+1)\} = \max_{j:1 \; to \; n} p\{\theta_i(t+1)|\theta_j(t)\}p\{\theta_j(t)\} \tag{6}$$

We keep track of the most likely previous state $\theta_j(t)$ that generates $\theta_i(t+1)$ by storing a pointer it. The resulting distribution is then conditioned by the current observation $\phi(t+1)$ as in Equation 4:

$$p\{\phi(t+1)\} = p\{\theta_j(t+1)\}p\{\phi(t+1)|\theta_j(t+1)\} \tag{7}$$

This expresses the probability of observing $\phi(t+1)$ by going through state $\theta_j(t+1)$. We recursively compute this up to time T and produce a sequence of probability distributions $\{P_\theta(0), P_\theta(1), \ldots, P_\theta(T)\}$.

One approach of instantiating a curve is to select states with maximum probability from each distribution independently. However, selecting states in a greedy fashion can result to an inconsistent sequence, it may break the continuity of valid links between successive elements. Rather, we instantiate the state with maximum probability at time T and then backtrack by choosing the previous most likely state that would generate the current one. Backtracking is essential for generating a consistent curve as not only does it consider the links between successive states, but also implicitly propagates future information back to earlier points. Candidates that have small likelihoods at the current time instance may have more influence on the global estimate than the competing local maximum. Figure 5 shows some examples of generating leaf shapes using the leaves training set (Figure 1).

4.1. Multi-Dimensional State Space

Implementation of a first order Markov Model is generally achievable by storing the transition probabilities in a memory array. However, preliminary empirical results indicated that for typical training examples, a first order Markov Model does not capture enough information to properly generate the curves. Higher order Markov Models increase the state space exponentially and explicit storage of a transition matrix is not practical. To address this issue, we do not explicitly compute and store the transition matrix, rather, we only maintain a linked-list of candidate states with strictly positive probabilities. At each iteration, the algorithm searches the training example for matches. When all the matches are found, the probability of the next state is calculated and added to the list for the next iteration.

In order to capture large-scale structures without resorting to high-order Markov models, we represent the states of the Markov chain as a set of wavelet basis functions $\gamma(s, \tau)$ where τ is position and s is scale [5]. While the details of this are outside the scope of this paper, Figure 2 shows an example where higher scale structures are important to capture. It is easy to see how the first order assumption does not capture enough information to generate the pattern while a synthesis using higher scales produces more consistent results. In application, we found that it was sufficient to only use the multi-scale representation in the hidden layer while the observation layer is assumes a single point from the raw input. However, it is conceivable to condition the synthesis using larger patches of the control curve.

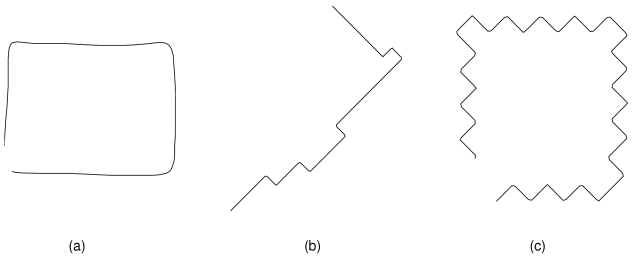


Figure 2. ***Synthesis of a zig-zag pattern. The training data consists of a set of zig-zag patterns associated to straight line segments (shown if Figure 4d. Each example is oriented orthogonally to the others, forming a rectilinear set. The training set is designated as stationary. Figure (a) shows the input curve. Figure (b) shows the synthesized curve using only first order information and Figure (c) shows the synthesized curve using the wavelet representation.***

4.2. State Blurring

For the hidden state transitions, searching for exact matches can be problematic. Quantization errors are likely to occur and can abruptly terminate the synthesis by propagating all probabilities to zero. Even with exact values, transitions can only occur at places on the training curves where the wavelet components are identical, making it difficult for the user to control the synthesis. In order to provide better 'mixing flexibility', we can blur the probability transitions, or synonymously the probability vector and then calculate the likelihood of a match. A continuous probability distribution for a continuous state space is modeled as a Gaussian mixture of the discreet distribution of the discreet state space. The probability for state γ_i is given by:

$$\begin{aligned} p\{\gamma_i\} &= \tfrac{1}{N}\sum_j p\{\gamma_j\} e^{-\frac{\Delta^2(\gamma_i,\gamma_j)}{\nu^2}} \\ \Delta^2(\gamma_i,\gamma_j) &= \frac{\sum_s\sum_\tau w(\tau,s)(\gamma_i(\tau,s)-\gamma_j(\tau,s))^2}{\sum_s\sum_\tau w(\tau,s)} \end{aligned} \tag{8}$$

Equation 8 measures the difference between two wavelet components as a weighted sum across the scales s and history τ. The variance of the Gaussian function specifies the degree of mixing. A small variance requires close to exact matches for mixing while a large variance makes it easier to transition at the cost of losing local consistency. The weight function $w(s,\tau)$ specifies the relative degree of importance for points further in history and higher in scale. Such a blur will often result in too many matches where every combination of states will produce strictly positive probabilities and contribute to the distribution. This causes computational complexities and high dimensional models may not be solved in practical time. Therefore, after normalization, we threshold over the tail of the Gaussian and re-normalize.

The input stroke used for conditioning may not provide exact matches to the training data. The input sketch is manually drawn by a human operator, hence it is overly restrictive to assume that inputs will match exactly to the user intended shape. Thus, a sigmoid function is used to blur the input. It has similar decay properties of a Gaussian function but also provides control over an almost flat region. This is ideal for noisy user input where we can suggest that the intent of the user within a given error range is equally distributed over the neighbors but decay exponentially at points further away. Thus the probability of observation o_i given input o_j is given by:

$$\begin{aligned} p(o_i|o_j) &= \tfrac{1}{1+e^{k|o_j-o_i|+s}} \\ s &= -2.197 - kp_{90} \end{aligned} \tag{9}$$

One can think of the sigmoid function as a blurred and shifted step function with blurring parameter k and a shift parameter s. A simple variable transformation allows us to specify the sigmoid shape by percentile thresholds. Empirically, we found that a 90 percentile threshold of 14 degrees and 10 percentile threshold at 38 degrees provided good results for various users. The sigmoid parameters can be thought of as the degree of user control. The smaller the shift and blur, the more we force the synthesis to adhere to the input, otherwise, the synthesis is less constrained to the inputs. For both the Gaussian and Sigmoid parameters, one can specify default values empirically and allow the user to tune them in specific cases.

4.3. Analytical Preferences

Given the compact and dynamic representation of the probability vector as a linked list, it becomes straight forward to embed additional preferences within the model. Each node in the list can include auxiliary dimensions that further condition the probability. It is important have the ability to embed the analytical biases in the model as they will also be accounted for in the synthesis backtracking phase. In this section we describe some important measures that help provide a more consistent synthesis with respect to the input curve.

When the training set exhibits regular properties, the search is configured to span across a wide range of sample points in each training curve (stationarity window). This stationarity property removes all sense of progression of points along the arc-length. The search is performed irrespective of the parametric position, possibly choosing matches arbitrarily along the points in the curve. Although, by definition of stationarity this is appropriate, in practical applications it can become somewhat problematic. The synthesis may get *stuck* at a state or a cycle through small set of states (know as absorbing states or irreducible communication classes). In such a situation, the propagated probabilities will model disjoint and self contained distributions. Conditioning over the observations and using a multi-scale model reduces the chance of this occurrence over long intervals. However, for training sets where there are many line segments that have few distinct features, such situations often occurs. To address this issue, we adopt a measure for *coherency over arc-length* as a measure of the number of out-of-sequence states in a synthetic curve [7]. To bias the synthesis for more coherent shapes, we enforce a penalty on matches that are out-of-sequence. The probability is penalized by a factor of q to help promote more coherent shapes. This penalty can easily be enforced by adding to each candidate node an arc-length parameter t and promoting matches for $t+1$.

While some degree of divergence from the input curve is necessary to fulfill the desired look, we wish to avoid situations where the generated curve diverges too far away from

the input curve. Since the state space only represents the tangent angles as a function of arc-length, there is no indication of how close the generated curve is to the input curve. Therefore, we define a measure for *spatial coherency to input* as a measure of the average distance between the input curve and the generated curve over Cartesian space. To generate more spatially coherent shapes, we include a *magnetic* force that biases the distributions to prefer points that are closer to the input. At each sample point t the probability is modified as follows:

$$p'(\gamma_i(t)) = \frac{p(\gamma_i(t))}{N(1 + kd(t)^2)} \tag{10}$$

where d is the distance between the input sample point and the resulting sample point generated by the maximum likelihood curve, up-to and including the candidate state at time t, k is the influence factor and N is a normalization constant. We including two additional dimensions (x, y) in each node to keep track of the co-ordinates of the candidate sequences (the co-ordinates are updated using the backtrack pointer). Figure 3 shows an example comparing generated curves with and without the coherency conditions.

4.4. Local Orientation Frame

For some applications, such as generating the city skyline (Figure 9), structures on the curves are not rotationally invariant. However, there are many applications where we wish to generate a single style over some arbitrary shape. Using an absolute reference frame for orientation, we would require the user to provide that style at every possible orientation. For such cases, removing the absolute bootstrapping and performing all computations over local reference frames avoids this issue. We compute the local reference frame of the user curve by applying a uniform kernel low-pass filter to the input (analogous to center of mass). The basis vectors are computed over the neighboring filtered points. For the training states, rather than using the tangent angles for our state elements, we use the curvature $(\dot{\theta}(t))$. This measures the relative change of the tangent angles and is independent of an absolute orientation bootstrap. Figure 4 shows examples of relative and absolute synthesis.

5. Results

Experiments were performed using a variety of different shapes exhibiting various properties. All of the input curves were arbitrary strokes hand drawn by a user. The parameters were empirically set once for each type of training set. For most training sets, the standard deviation of the Gaussian mixture was set to 15 degrees, for the user input blurring function, the 90 percentile threshold was set to 14 degrees and the 10 percentile threshold was set to 38 degrees. The

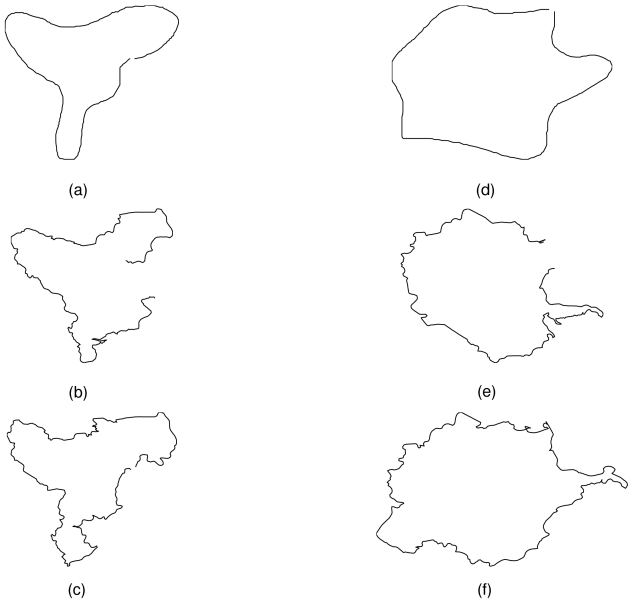


Figure 3. Generating coastlines. The training data is a set of 27 coastline patterns from geographic maps. The associated control curves consists of blurred versions of the set. Figure (a) and (d) show two input curves, Figure (b) and (e) show the corresponding synthesized curves without any magnetism and Figure (c) and (f) show the results with magnetism.

simple shapes (Figure 6) training set had a narrower Gaussian variance (50 degrees), enforcing stricter mixing condition. Most of the training sets in experiments were considered as stationary sets, except for the leaves (Figure 1). All experiments had magnetism on and, unless mentioned otherwise, all frames of reference used absolute co-ordinates. Experiments were executed on a Linux PC with a 1GHz Pentium III processor and 1GB of RAM. The results were generated in interactive-time.

Figure 5 shows example syntheses using leaf shapes. The training set (Figure 1) is non-stationary, restricting the mixture about the absolute arc-length position. Forcing this global constraint avoids mixing parts of the left side of the leaf with the right side of the leaf as there are cases that exhibit similar local consistency and control on both sides of the leaf (as in the maple leaf example). As such, each hand-drawn stroke is normalized. It can be seen how the generated mixtures look like leaves. Although, it is also easy to see that some of the leaves do not exhibit symmetry in shape, a property often seen in real leaves but can some-

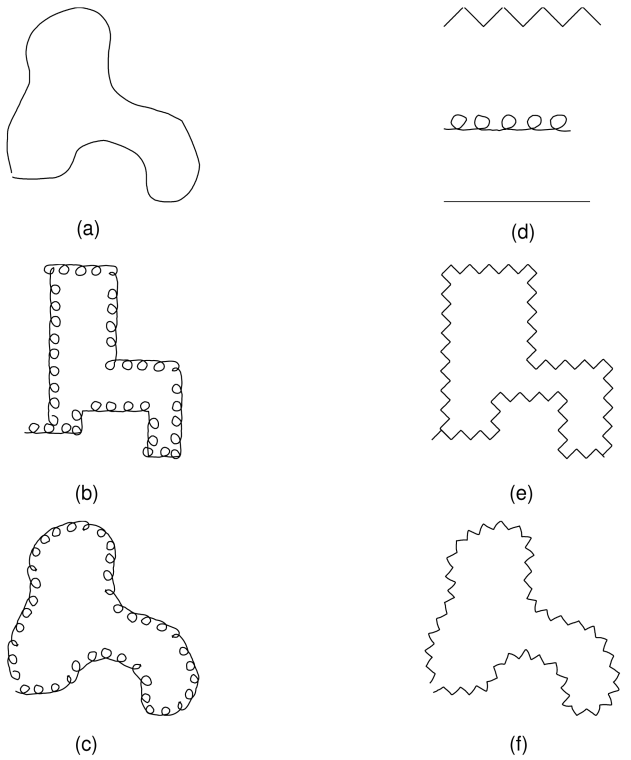


Figure 4. Example of synthesis using relative and absolute frames for two stationary patterns. Figure (a) shows the input curve and Figure (d) shows the two patterns and the control curve (straight line segment). For relative frames, each set consists of a single curve, either the saw or the curl pattern. For absolute frames, each set consists of orthogonal versions of each patters, forming two rectilinear sets. Figure (b) and (e) show the results using absolute frames and (c) and (f) show the results using relative frames.

times be ignored in the realm of imaginative illustrations. Further, when we are unlucky, we can see that in some cases the curve does not close onto itself, even with high magnetism. Such issues require large scale re-configurations of the curve which generally lead to no-linear computations.

Figure 6 displays some example shapes used in a training set consisting of simple polygons. Figure 7 and 8 display example syntheses using this set. Figure 7 shows a flow-chart generated from a noisy sketch. In this case, each stroke was generated as an exact instance from the training set. Figure 8 shows an example that results in a mixture of the training set. It is easy to see that various mixtures and shapes of different sizes can be generated in consistency with the input stroke.

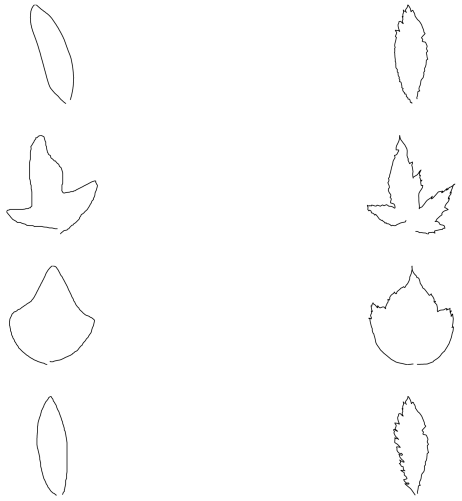


Figure 5. Examples of synthesis using a leaf training set. Curves on the left are the input curves while curves on the right are the generated ones.

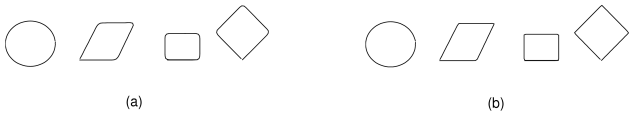
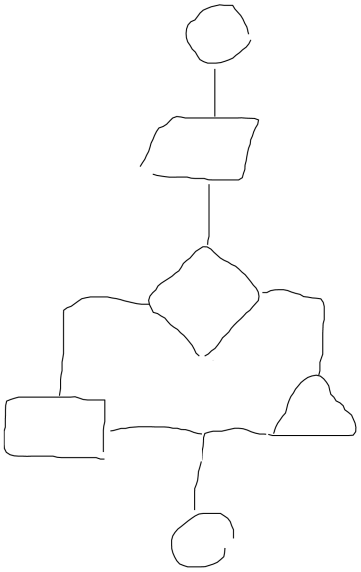


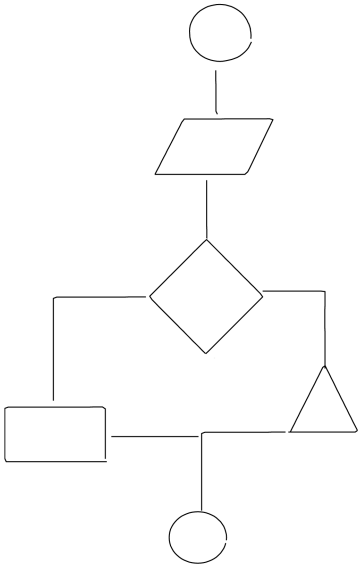
Figure 6. Some examples taken from training set consisting of simple shapes. The set is considered stationary. Both the patterns (right) and the associated control (left) are displayed.

Figure 9 shows a synthesis using various training sets. The building skyline was generated using only the two training examples shown in Figure 10. Additional sets for water, clouds and sun were used. The grassy terrain and smoke were the only curves generated with relative reference frames. One important artifact that this example displays is that even with magnetism on high, the absolute positioning of the input curve is not exactly the same as the generated curve. This is noticeable in the case where drawing the smoke out of the chimney over the input curve does not result in smoke being generated exactly at the same place over the chimney at the generated curve. However, these artifacts can be easily compensated for by manually moving the object to the desired position.

Our multidimensional state space allows us to include on only higher-scale attributes, but any desired supplementary attribute that we wish to include a constraint for. Figure 11 shows examples for learning non-holonomic paths. These

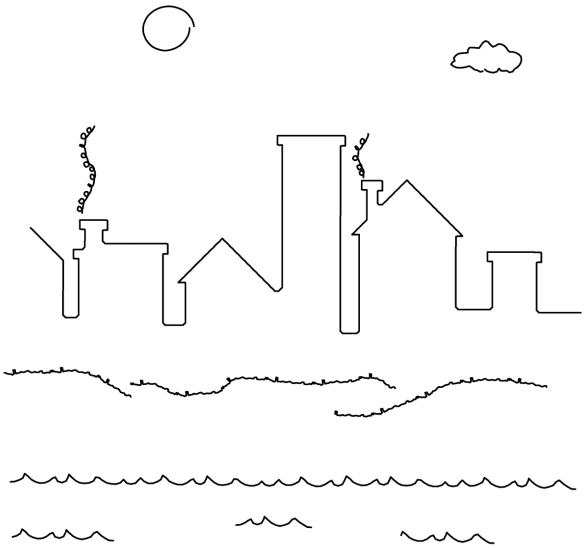


(a)



(b)

(a)



(b)

examples shows paths that include the robot axis facing direction (shown by arrow). This attribute is stored as an additional dimension in our HMM. As can be seen, some of the paths are non-trivial motions such as the parallel-parking type example. Figures 12 and 13 show how we can generate new paths that are bound by these learned non-holonomic constraints from the initially unconstrained trajectory.

6. Conclusion and Future Work

We have presented an approach to curve synthesis from learned refinement models: that is, to producing stylized curves given input curves that indicate roughly what shape the stylized curves should be. Our method learns from example, so it can be applied in a vast array of applications. Our approach is to use a Hidden Markov Model to capture both the consistency of local structure for the desired refined shape and the method of control for those curves. The refined curves are represented by the hidden states and

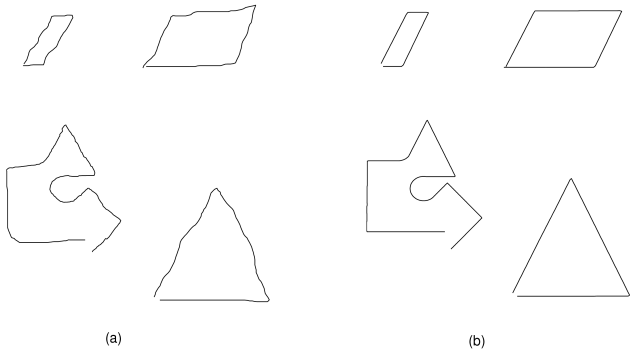


Figure 8. Example of synthesis using the simple shapes training set. Curves on the left are the input strokes while curves on the right are the generated ones. This example displays the generated curves as a mixtures of the training set. Various sizes of the same shape can be generated.

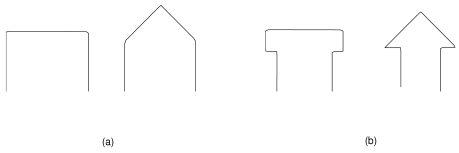


Figure 10. Training set used to for skyline. The left Figure shows the control curve and the right Figure shows the output.

the control curves are represented by the observation states. Synthesizing a new curve consists of choosing the maximum likelihood hidden stated sequence that accounts for all the observations. Experimental results display how the shape of a synthesized stroke exhibits the properties of the training data while acting in accordance to the shape of the input stroke. The model and algorithm are constructed such that they can be implemented well within physical memory and computational limitations and can be trivially integrated with supplementary analytical preferences and constrains.

The examples displayed in experimentation can be further elaborated by the inclusion of additional attributes such as pen thickness and stroke properties. These additional attributes can be directly accommodated in the model we have presented, simply as additional parametric variables encoded along the curves.

In this discussion we have assumed that when a style is generated, we know *a priori* which family of statistical biases we should apply. Given several style families with similar control schemes, in practice, it may be that in one

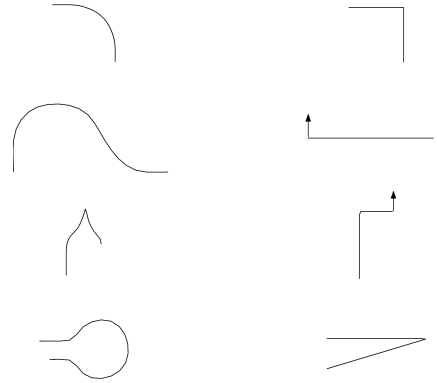


Figure 11. Samples of a training set simulating non-holonomic motions. Paths on the left display the constrained motions while paths on the right display the associated unconstrained control paths. Where specified, arrows indicate additional constraints on the direction of motion to account for the orientation of the robot. The full set consists of the above at several orientations.

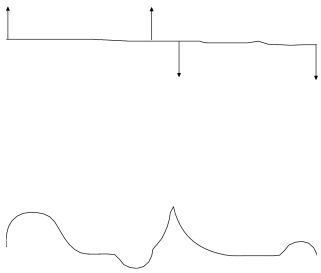


Figure 12. The above was generated using the training example for non-holonomic constraints. The input path (top) is a hand-drawn path with several desired directions of motion (shown by arrows). The resulting path (bottom) consists of parts of a D-shape turn, a U-shape turn and a parallel parking style motion in order to end up in the right motion directions at the corresponding points.

part of the illustration we want one style and in another part we expect a different style. For example, in Figure 9 we had to manually select the refinement models for each curve stroke. How to automatically incorporate two different types of bias in the system and, further, how to make a transition between them remains a topic we are still investigating.

One significant open issue is the application of global and semantic constraints to the curves being synthesized.

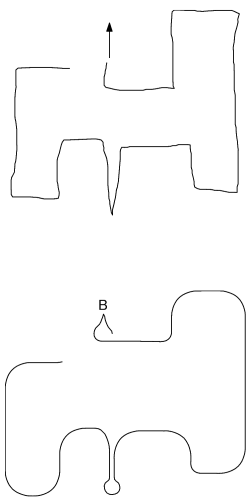


Figure 13. The figure above was generated using training examples for non-holonomic constraints. The input path (top) is a hand drawn path with a restriction on the initial direction of motion. Below shows the generated path where point (B) marks a direction reversal.

For example, when a silhouette of a face is being generated, it would be desirable (outside the science fiction domain) to assume we wish to generate a single eye, a single nose, a single mouth followed by a chin and then enforce closure. While this can be enforced by specialized constraints, a future goal is to use the current framework to learn such attributes and embed them in the synthesis.

References

[1] R. Bowden. Learning statistical models of human motion. In *IEEE Workshop on Human Modelling, Analysis and Synthesis, CVPR2000*, July 2000.

[2] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. *International Conference on Computer Vision*, September 1999.

[3] S. P. Engelson. Learning robust plans for mobile robots from a single trial. In *AAAI/IAAI, Vol. 1*, pages 869–874, 1996.

[4] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, 1992.

[5] A. Finkelstein and D. H. Salesin. Multiresolution curves. In *SIGGRAPH '94 Proceedings*, July 1994.

[6] A. R. Forrest. The twisted cubic curve: A computer-aided geometric design approach. *Computer Aided Design*, 12(4):165–172, July 1980.

[7] A. Hertzmann, N. Oliver, B. Curless, and S. M. Seitz. Curve analogies. In *13th Eurographics Workshop on Rendering*, June 2002.

[8] P. Jodoin, E. Epstein, M. Granger-Pichi, and V. Ostromoukhov. Hatching by example: a statistical approach. In *NPAR 2002 : Second International Symposium on Non Photorealistic Animation and Rendering*, June 2002.

[9] D. Keren and M. Werman. Bayesian interpolation. *ARPA Image Understanding Workshop*, November 1994.

[10] M. Learning. Tom mitchell. In *McGraw Hill*, 1997.

[11] J. Miura. Hierarchical vision-motion planning with uncertainty: Local path planning and global route selection.

[12] Y. N. W. Stefano Soatto, Gianfranco Doretto. Dynamic textures. *International Conference on Computer Vision*, pages 439–446, July 2001.

[13] X. Wang. Learning planning operators by observation and practice. In *Artificial Intelligence Planning Systems*, pages 335–340, 1994.

[14] L.-Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In K. Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 479–488. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[15] S. Zhu, Y. Wu, and D. Mumford. Filters, random fields and maximum entropy, towards a unified theory for texture modeling. *International Journal of Computer Vision*, 27(2):107–126, 1998.