Eurographics Symposium on Rendering (2004)
A. Keller, H. W. Jensen (Editors)

Sketch Interpretation and Refinement Using Statistical Models

Saul Simhon[†] and Gregory Dudek[‡]

McGill University
Quebec, Canada

Abstract

We present a system for generating 2D illustrations from hand drawn outlines consisting of only curve strokes. A user can draw a coarse sketch and the system would automatically augment the shape, thickness, color and surrounding texture of the curves making up the sketch. The styles for these refinements are learned from examples whose semantics have been pre-classified. There can be several styles applicable on a curve and the system automatically identifies which one to use and how to use it based on a curve's shape and its context in the illustration. Our approach is based on a Hierarchical Hidden Markov Model. We present a two level hierarchy in which the refinement process is applied at: the curve level and the scene level.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Line and Curve Generation

1. Introduction

One of the most natural ways a user can create illustrations is by drawing a sketch. Although almost everyone can sketch out a crude illustration, only a few of us have the artistic talent and patience to draw the refined details we wish to depict. Even those who are lucky enough to possess those abilities may not have the appropriate tools at hand. In this paper, we present a method for automatically generating elaborated illustrations from coarse outlines. This allows users to rapidly sketch a prototype of what they want and have the elaboration system provide the details and adjustments required.

There is wide array of applications where interaction with a set of functions that describe the shapes or constraints of curves is key. In particular, interfaces for simple drawings consisting of only curve strokes are often found for comics, presentation material, cel-animation, storyboard designs, system designs (sketch to prototype) and non-photorealistic pen and ink illustrations. Recent hardware advancements for stroke based interfaces (such as pen tables or tablet PC's) further justify the need for a system that can produce an intelligent rendition of the user's input.

Given the diverse set of possible drawings and applications, providing a system that only applies domain specific constraints may be overly restrictive. In this work, the knowledge of what the refinements should be, for a given domain, is extracted automatically from an ensemble of examples of the desired output. This includes refinements on a curve's shape, variations in pen thickness and coloration. As such, novel illustrations are rendered to exhibit the same *look* as the examples in the training ensemble. Furthermore, the framework developed allows for domain specific constraints to be easily combined with the learned refinement styles.

In order to develop such a system, there are two key problems we address. First, we must be able to determine and reconstruct the details of curves given only coarse and noisy inputs. This is accomplished by training a *refinement model* (i.e. a set of rules that transform a coarse curve to a refined one). Second, we must automatically select the model that should be applied on a curve from the set of possible refinement models. This is accomplished by choosing the refinement model that best transforms the curve while also satisfying high-level relationships between preceding curves that are drawn as part of the same drawing.

[†] e-mail: saul@cim.mcgill.ca
[‡] e-mail: dudek@cim.mcgill.ca

Our approach is based on the use of Markov models (**MM**s): probabilistic descriptions of how sequentially ordered states are related. Specifically, we use *Hidden* Markov models, a modeling formalism that allows us to express the relationship between aspects of a system we can observe directly (the curve drawn by the user) and variables that cannot be observed, but which determine the output (in this case, the curve the user "really wants"). Our approach is to describe an entire sketch as a hierarchy of Hidden Markov Models [FST98] such that each curve is described by one HMM. This leads to an inter-relationship between the HMMs used for individual curves (the *curve level*) and those used to specify the identity of objects within the scene (the *scene level*).

The paper is organized as follows: First, we discuss previous work. Then we present an overview of the algorithm, including a description of HMMs and our two layer hierarchy of HMMs applied to curves. We later describe in detail our method of learning and synthesis at the curve level HMM and then at the scene level HMM, showing examples for each. Finally, we conclude and discuss future work.

2. Related Work

Standard applications for hand drawn sketches are either geared toward enhanced functionality of the user interface [LM95, HZ96] or attempt to interpret a drawing by recognizing domain specific objects [KS02, SSD01]. Recognition is commonly performed using some pre-defined set of primitives or analytical constraints that are best suited for their process. While such methods are generally applied in the realm of 2D line-are objects, other work [KMM*02] provide an interactive interface for users to control strokes of a non-photorealistic rendered for 3D models.

One of the key ideas in facilitating interaction with curves is the notion of coarse to fine refinement, or *abstract* to *detailed* description. There are many notable works for manipulating curves at multiple levels, including landmark results of hierarchical and subdivision surfaces [FB88, FS94]. Typically, such approaches focus mainly on the control of individual objects that are removed from the context of their surroundings. Approaches that provide users the ability to describe whole scenes, such as the text to scene renderer in [CS01], render objects based on their context and generally avoid the intrinsic details that do not generalize in the scene. Further, in all of these methods, specialized constraints are required that are sometimes too difficult for users to construct.

Recent advancements in texture synthesis methods [HB95, WL00] suggest that we can learn the regular properties of example images and generate new ones that exhibit the same statistics. Work by Hertzmann *et al.* [HOCS02] show how such methods can be applied to synthesizing stylized curves. Curve styles are learned from the statistics of example styles and new curves exhibiting those styles are generated over the shape of an input curve. Analogies between the inputs and outputs are computed by calculating an offset over both the input curve and training examples. This offset is then used as a rigid transformation applied in parallel to a synthesis process. Likewise, Freeman *et al.* [FTP03] present an approach to learn style translations on line segments. A training set, consisting of several curve stokes, is used as a basis for reconstructing a line-art drawing. New lines are generated as a linear combination of examples consisting of the k nearest neighbors in the set.

In our work, we provide three key components that extend the approaches cited above. First, using a Hierarchy of Hidden Markov Models allows us to capture multiple stochastic functions and represent scene dynamics over various scales and context. Secondly, individual HMMs are themselves two-layered systems and allow us to model a *controllable* process. That is, the synthesis is driven by the input curve where the actual features generated are directly dependent on the shape of the input. This allows us to model examples with localized features that are tied to the shape of a given region (such as a roof ledge that extends only at the corner of the roof). Finally, typical approaches to synthesis commonly consider greedy strategies, choosing the best match at the current point. When we are given inputs or partial data, the locally best points may 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 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. Overview

Let θ be the refined curve the user is *intending* to draw and let ϕ be the coarse curve the user has actually drawn. We can say that ϕ is the result of some lossy (non-invertible) transformation of θ:

$$\phi = F(\theta) \tag{1}$$

The problem is then: how can we reconstruct θ given ϕ? Our approach is to infer a new curve by examining the similarity of the input curve with the set of examples that are currently available to the system.

We begin with example curves that serve as exemplars of the kinds of curves we wish to draw; we typically have several *sets* of such examples (e.g. fish, water, terrain). Each example in a set has associated to it a coarse curve that shows what the user would draw when their intention is to produce that particular example, i.e. the pair $\{\theta_i, \phi_i\}$. Using one of these sets, the user-drawn curve is used to *steer* a synthesis procedure and generate a new curve. The resulting curve is a locally consistent mixture of segments from the set, but not necessarily identical to any single example in the set. This is complicated by the need to account for both fine-scale details as well as large scale motions of the curve. Learning and synthesis at the curve level is described is Section 4.

While we can allow the user to manually select which model to use to refine the curve, we automate the whole process by automatically classifying the curve being drawn as belonging to one of the sets. The higher level of the hierarchy (the scene level) moderates the recognition of what sets should be used by specifying the conditional probability of drawing one type of curve after another, or one type below or above another. This scene level control is encoded in the form of a probabilistic transition diagram between types of example sets and is discussed in more depth in Section 5.

3.1. 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:

$$\Lambda = \{M, B, \pi\} \tag{2}$$

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.

There is an abundance of literature on Hidden Markov Models and the domain is frequently decomposed into three basic problems of interest:

- **Evaluation**: Given a model Λ and a sequence of observations $o_1, o_2, ..., o_T$, what is the probability that those observations are generated by that model?
- **Decoding**: Given a model Λ and a sequence of observations $o_1, o_2, ..., o_T$, what is the most likely hidden state sequence $h_1, h_2, ..., h_T$ that produces those observations?
- **Learning**: Given an observed set of examples, what model Λ best represents that observed set.

Solutions to the above three problems are key to our work. Learning allows us to model various illustration styles by simply providing the examples. Decoding allows us to synthesize a new curve (hidden states) based on a coarse user input (observation). Evaluation allows us to detect the appropriate class of illustration types that an input stroke belongs to, determining the likelihood that the input curve would be generated by the model in question.

3.2. Two Level Hierarchal Hidden Markov Model

At the first level of the hierarchy (curve level), the characteristics of our training sets are expressed statistically with each set modeled by its own Hidden Markov Model. Sample points from the refined curves play the role of the hidden states while sample points from the coarse curves play the role of the observations. Thus, the transition matrix reflects the likelihoods of generating curve segments given the previous (probabilistic local constraints) and the confusion matrix reflects the likelihoods that users would draw the particular coarse shapes when their intent is the associated refined one (Figure 1). We construct a set S consisting of N HMMs where each individual HMM is trained using a particular training ensemble:

$$S = \{\Lambda_0^0, \Lambda_1^0, ..., \Lambda_N^0\} \tag{3}$$

For example, Λ_0^0 may represent the set for terrains and Λ_1^0 the set for clouds.

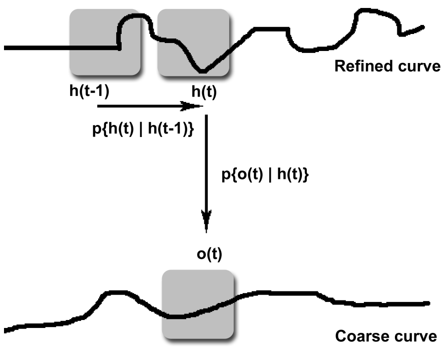

Figure 1: *For all curve segments (in gray), the transition matrix and confusion matrix store the above likelihoods. This is computed over every example in a given set.*

At the second level (scene level), we use another HMM to model semantic constraints on the models at the first level. For example, one can suggest that the cloud model in S can only be applied to a curve that lies above another curve that has been refined by the terrain model. As such, the state space of this HMM reflects all possible models in S and their relative positions. While, in principle, we can learn such constraints from labeled illustrations, for our purposes, we manually encode them in a graph (Figure 2).

Several such graphs can be used to train HMMs at the scene level of the hierarchy:

$$G = \{\Lambda_0^1, \Lambda_1^1, ..., \Lambda_W^1\} \tag{4}$$

Each model in G depicts different kinds of scenes. For example, you can have face scenes that suggest the sequence $forehead \rightarrow nose \rightarrow mouth \rightarrow chin$ or landscape scenes that suggest $grass \rightarrow (flower, above), cloud \rightarrow (tree, below)$. When we do not wish to have constraints on the order in which curves are drawn, a graph can suggest that every model can be followed by any other model, with only their relative positioning as a constraint.

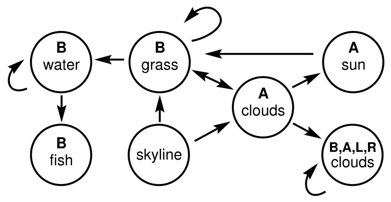

Figure 2: *Example relationships in a scene. The labels correspond to HMMs at the curve level and the letters above correspond to the allowable relative position (i.e. A: above, B: below, L: left R: right).*

4. Curve HMM

4.1. Learning Curve Styles

We train a HMM Λ_i^0 from examples of *refined curves* coupled with *control curves* (Figure 3). The refined curves depict the desired solution to be produced if a user sketches a control curve. When only the observations are available, learning is most commonly performed by algorithms such as the Baum-Welch algorithm or generalized Expectation-Maximization methods [Rab90]. In our case, we explicitly provide data for both the observation and hidden layers by a suitably normalized set of coupled curves. Therefore, we can estimate a HMM by the statistics of the training data, calculating probabilities of successive sample points along the refined curve strokes and the probabilities that they generate the sample points along the control strokes.

Figure 3: *Samples from a training set. Curves on the left show the control curves while curves on the right show the associated refined ones that include color. Typically, the shape of the control curves are filtered versions of the refined ones, in this case the filtered ones are very similar to the originals. The set is sampled uniformly with 128 samples per example for a maximum of 1024 unique states.*

4.1.1. Transition Matrix

Our HMMs operate over a multi-scale curve description in order to capture long-range interactions. A multi-dimensional state space is used for the hidden states $H_i = h_i(s,a)$ to represent a curve attribute a at scale s. The attributes we capture include the following: the shape of the curve as the tangent angle $\theta(t)$, RGB value for the color of the curve $c(t)$, the thickness of the curve $k(t)$, RGB and Alpha pixel values for texture fill $f(t)$ and the fill direction $d(t)$. (Every curve in the set is parametrized by the arc-length position t.) While all of these attributes can be represented at various scales, for efficiency, we only encode the shape at multiple scales $(\theta_s(t))$ as it generally provides sufficient multi-scale constraints. We use a wavelet representation with the *Haar basis* [FS94].

We estimate the transition probabilities of states by the statistics of successive sample points in the ensemble (i.e. the occurrence of matches of successive points). Additionally, we rate the *goodness* of the match using a Gaussian distance metric:

$$p\{H_i = H_j\} = e^{-\Delta^2(H_i,H_j)}$$
$$\Delta^2(\gamma_i,\gamma_j) = \frac{\sum_s \sum_a w(s,a)(\gamma_i(s,a)-\gamma_j(s,a))^2}{\sum_s \sum_a w(s,a)v^2(a)} \tag{5}$$

This Gaussian blur is applied on the difference of two states as a weighted sum over the scales and attributes. This avoids issues with quantization errors and also provides some degree of control and flexibility over the *mixing tendency*. The tendency to mix curve segments is determined by the value of the variance v^2. A small variance reduces the mixing tendency where the output will be closer to exact instances of the training set while a large variance allows to transition more easily, at the cost of losing some local consistency. In our experiments, we weigh most heavily on the shape attribute and the variance is empirically set for each ensemble.

Each unique state found in the training set is labeled dynamically. As such, the transition matrix M is indexed using those labels. Thus, the value $M_{i,j}$ is $p\{H_i \mid H_j\}$ and the size of M is proportional to the number of unique states found in the training set.

For training sets that exhibit stationarity $M(0) = M(1) = ... = M(T) = M$, measured over the entire curve. Otherwise, the transition matrix is calculated over a predefined window that identifies regions that exhibit regular properties. The ability to specify regions of stationarity (hence global non-stationarity) allows us to accommodate for attributes that inherently possess some global constraints. Indeed, if every curve in the ensemble constitutes a stationary stochastic signal, the ensemble can be thought of as one signal. But many cases require altering the characteristics of curves by their position in the sequence, preserving proportionality and sizing constraints over large scales. The degree of stationarity is manually selected based on the nature of the ensemble. In such cases, the input curve is normalized and $M(0)$ is still computed over the entire curve (providing invariance to the starting point of the input curve).

4.1.2. Confusion Matrix

The confusion matrix B stores the likelihood that a user would draw a certain curve segment when their intent is to produce a particular refined shape. This can be though of as a user's short-hand notation for an elaborated shape. Let $\phi_s(t)$ represent the sequence of tangent angles along the control curves. (We only consider the shape of the control curve, although in principle, it is possible to also consider other control attributes such as pressure or speed of a pen.) We estimate the probabilities of the confusion matrix B from the statistics of associated sample points $(\theta_s(t), \phi_s(t))$ in our training set. That is, for each of these coupling in our curves, we count the number of matches and then normalize. Thus, the values in the confusion matrix are the conditional probabilities $p(\phi_i|\theta_j)$ for all curve elements i and j. The confusion matrix is extended to match the dimensionality of the hidden states by replicating the probabilities over all the attributes of the hidden states (i.e. $p(\phi_i \mid H_i)$). For consistency with the transition matrix, the confusion matrix is indexed by the same state labels.

4.1.3. Initial Distribution

To complete configuration of our HMM, we must specify an initial distribution for the hidden states. We assume a uniform initial probability distribution $\pi = P_H(0)$. That is, before anything is drawn, we provide equal likelihoods to all curve candidates.

4.2. Curve Synthesis

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

$$\max_{H_i \dots H_N} p\{H(0), H(1), \dots, H(T) \mid \phi(0), \phi(1), \dots, \phi(T), \Lambda^0\} \tag{6}$$

That is, we reconstruct the most likely refined curve elements (represented by the hidden states) given sample points from the input curve (represented by the observable states). We can solve this problem using the *Viterbi algorithm* that has run-time complexity of $O(N^2T)$, where N is the number of states and T is the sequence length of the input curve. The approach consists of iterating over the sequence of sample points from the drawn curve. At each iteration, we compute the maximum likelihood estimate for a partial observation sequence and hidden state sequence up to sample point t given that the current state is H_i (i.e. the best path going through the i^{th} state at point t). This likelihood is computed for all states H_i as follows:

$$\psi\{H_i(t)\} = \max_j [p\{H_i(t) \mid H_j(t-1)\}\psi\{H_j(t-1)\}]$$

$$\psi\{H_i(t)\} = p\{\phi(t) \mid H_i(t)\}\psi\{H_i(t)\} \tag{7}$$

That is, for each sample point, we propagate the distribution over all states $\Psi(t)$ forward using the transition probabilities in M and then condition the result with the current observation. When propagating, we only consider the most likely previous state $H_j(t-1)$ that would generate the current state $H_i(t)$ and keep a back-pointer to it. For efficiency, we threshold the distribution $\Psi(T)$ and normalize, maintaining only the top n curve segment candidates.

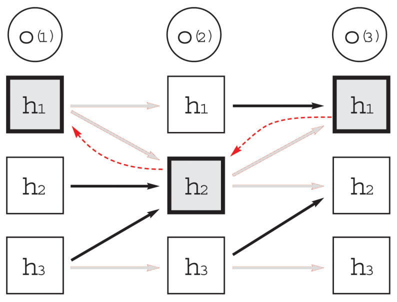

Figure 4: *Example synthesis diagram for three states $\{h_1, h_2, h_3\}$ and three input points $\{o(1), o(2), o(3)\}$. Solid arrows indicate all possible transitions, the ones shown in gray indicate the best transitions. At the last point, we choose the state with highest probability (h_1) and then backtrack through most likely transitions (dashed arrows).*

This is recursively computed over the entire sequence (up to sample point T), maintaining the distributions and back-pointers for each sample point. We then choose the state that has maximum likelihood in $\Psi(T)$ and backtrack using the back-pointers (Figure 4). Backtracking is essential for generating a consistent curve as not only does it consider the maximum likelihood links between successive states, but also implicitly propagates information from future observations back to earlier points. Choosing this maximum results in the maximum likelihood hidden state sequence that best describes the observations, given our HMM. The user can choose to scroll through the list of different solutions, selecting less likely but perhaps more preferable outputs.

4.2.1. Input Blurring

We cannot expect the user input points to match exactly with sample points from the control curves in training. This can be problematic as we will only find a non-zero conditional in B when there is an exact match. To avoid this, we blur the observation vector. The degree of the blur controls our confidence in the user's input curve.

4.3. Regularization Bias

We can take a probabilistic approach to regularization by maximizing the posterior likelihoods of a Bayesian model [Sze89]:

$$\max_{f \varepsilon \mathscr{M}} P\{f \mid D\} \propto \max_{f \varepsilon \mathscr{M}} p\{D \mid f\} p\{f \mid \mathscr{M}\} \tag{8}$$

where f is an object in class $\mathscr{M}$ and D is the coarse data. Typically the *data model* $P\{D \mid f\}$ assumes a Gaussian distribution and $P\{f \mid \mathscr{M}\}$ is a regularization bias for smoother solutions.

Our HMM can be formulated in a regularization framework. Rather than having a fixed noise model and bias, we learn these from examples. That is, in a Hidden Markov Model, the confusion matrix represents a learned *data model* and the transition matrix represents a learned regularization term. Using this framework, we can include *ad-hoc* biases to the system.

The approach consists of embedding auxiliary parameters in the candidate list data structure that are not used in matching, but only as inputs to regularization functions. Regularization functions can then be included by augmenting the energy of the candidate states:

$$E\{H_i(t)\} = -log\{\Psi(H_i(t))\} + \sum_k \lambda_k R_k(t) \qquad (9)$$

where $R_k(t)$ is a regularization constraint (such as smoothness) and λ_k is the associated weight.

In particular, we include three constraints that bias sequences that are 1) more coherent in arc-length [HOCS02] 2) reduced in the number of transitions between training examples and 3) close to the input curve. In order to enforce arc-length coherence, we include an auxiliary attribute τ that identifies at which arc-length point the state was found. Using the back-pointer, we are able to identify the sample point position of the last state that generated the current and penalize the current if out of sequence, i.e. $|H_\tau(t+1) - H_\tau(t)| > 1$. Similarly, to reduce the number of transitions, each state stores an *id* that identifies which training example it came from. Then, we penalize state sequences with different *id*s.

Finally we include a constraint that simulates a magnetic attraction between the input and output curves, preferring outputs that are closer to the input. For this, we include auxiliary parameters (x, y) that identify the Cartesian co-ordinates of the most likely path up to and including the current state. These values are computed by extrapolating the (x, y) values in previous state (identified by the back-pointer) using the current tangent angle and sampling rate. The penalty is then a function of the distance between the generated (x, y) position and the input (x_{in}, y_{in}).

There may be cases where we wish to change the amount of influence a regularization term has at different parts of the curve. This can be done by providing a regularization weight that is a function of the arc-length. In particular, to enforce closure (when a user draws a closed curve) we increase the magnetic bias when approaching the end points:

$$\lambda_{mag}(t) = k * e^{|1-\frac{2t}{T}|} \qquad (10)$$

where t is the current sample point position and T is the total number of sample points. The supplied video shows some examples of how different regularization terms affect the solution.

4.4. Curve Synthesis Examples

Figure 5 shows an example synthesis using the training set shown in Figure 3. Note how the resulting curves consists of mixtures of the training set. The outputs have the same overall shape as the sketch but exhibit the style of the training set.

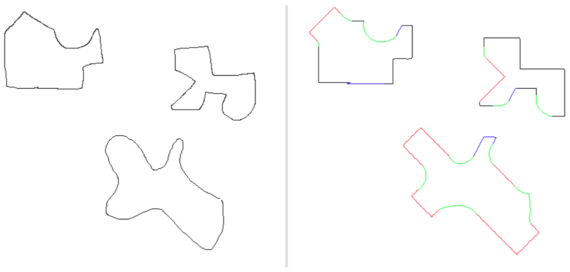

Figure 5: *Example of a synthesis using the training set in Figure 3. The left shows the inputs and right shows the resulting mixture.*

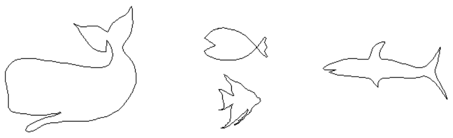

Figure 6: *Training set with examples of fish. Control curves (not shown) are blurred versions of the refined ones.*

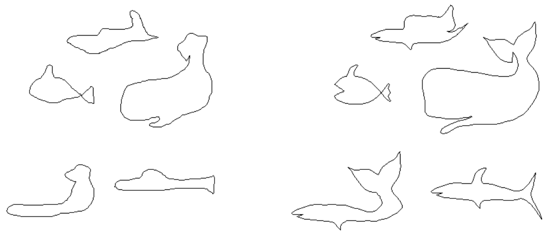

Figure 7: *Example synthesis of fish shapes. The left shows the inputs and the right shows the results. Some results are recognized as exact instances from training while others are mixed.*

Figure 7 shows the outputs when using a training set consisting of fish shapes (Figure 6). It can be seen how some of the outputs are exact matches from training while others are novel curves consisting of mixtures of segments from training. Users can control the amount of mixing by changing the variance parameter. In this case, all of the input curves were normalized over the average arc-length from the training curves. Figure 9 shows results using a leaf training set

(Figure 8). It can be seen how the generated mixtures look like leaves. Although, note some of the leaves do not exhibit symmetry in shape, a property often seen in real leaves but can sometimes be ignored in the realm of imaginative illustrations.

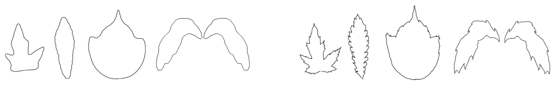

Figure 8: *A few Samples from a training set used for leaf synthesis (not all shown). The left shows the control curves while the right shows the stylized curves. curve.*

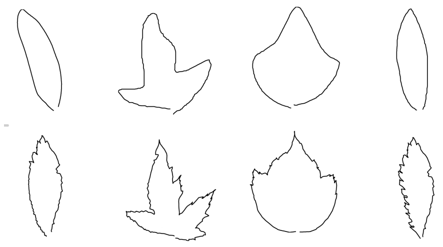

Figure 9: *Examples of synthesis using a leaf training set. Curves on the top are the input curves while curves on the bottom are the generated ones.*

The two bottom figures in Figure 10 show results when replacing the shape attribute $\theta(t)$ by a curvature attribute $\dot{\theta}(t)$. Additionally, the output curve reference frame is rotated at each sample point to align the $x-axis$ along the tangent angle of the input curve. This modification gives our system the ability to also generate outputs similar to [HOCS02].

4.4.1. Texture Filling

In the examples presented here we have chosen to specify the interior color along each curve as a supplementary attribute. A secondary statistical filling process also based on a Markov model of image properties is then applied. Thus, initiate the texture filling process from cues attached to the refined curves we synthesize. These cue pixels act as the seeds for an incremental stochastic pixel inference procedure. The set of pixels to be filled is sorted in order of the number of filled neighbors each one has. The successive unfilled pixel with the largest number of neighbors is selected and its color is drawn as the maximum likelihood value of the probability of the color for a pixel as described in [WL00]. Black pixels are reserved for borders and are not considered when filling. Each curve stroke is filled in its own image layer and the alpha value, specified in the training set, is used to overlay. Any unfilled pixel left takes on the value of a specified background.

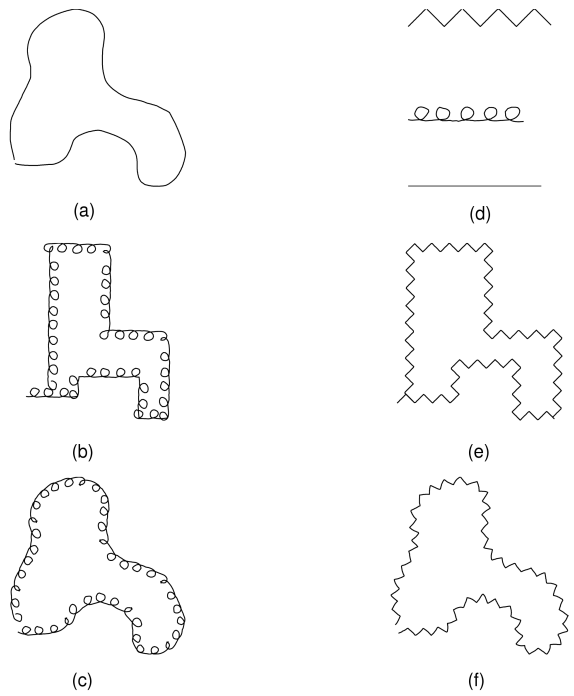

Figure 10: *Example of synthesis showing the use of curvature versus tangent angle for two stationary patterns. Figure (a) shows the input curve and Figure (d) shows the two patterns and the control curve (straight line segment). We used four versions of the patters, each rotated at 90 degrees. Figure (b) and (e) show the results using tangents and (c) and (f) show the results using curvature.*

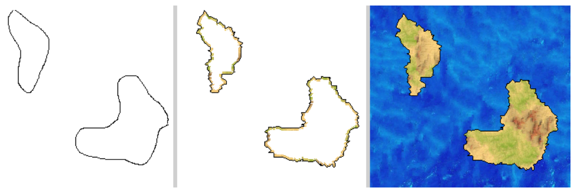

Figure 11: *Generating coastlines with texture fill. Training examples consisted of 25 coastlines (see video).The figure on the right shows the input, middle shows the output with the texture fill seeds and the right shows the resulting texture fill image.*

5. Scene HMM

The scene level HMMs encode constraints on the refinements that apply on curve segments in a scene. The constraints can suggest, for example, that backgrounds must be drawn first, followed by other objects which in turn can also

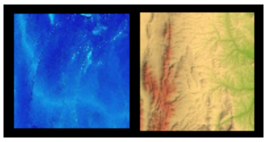

Figure 12: *Texture image used for coastlines.*

be followed by other objects, each drawn on top the previous. They can also represent typical drawing habits, such as when users draw a forehead, most likely it will be followed by a nose. They can be sequentially unconstrained, such that every curve can follow any other type of curve. In all such cases, positional constraints are embedded to further restrict the types of curves based on their relative location. High-level quantifiers are used, such as *above, below, left, right, in, out* and are evaluated relative to the edges and center of the bounding boxes of curves in the scene.

5.1. Learning Scene Constraints

A HMM Λ^1 is trained using a semantic graph that defines the high-level constraints (Figure 2). The nodes of the graph refer to both the HMMs in the curve level and an associated position (above, below, left, right, in or out). For example, if we had three models, grass, tree and cloud, we could have as many as eighteen nodes. Each edge of the graph includes a weight that identifies the probability that a user would draw the type of curve (identified by the destination node) at a relative position to the previous (identified by the source node). For example, the probability that a tree will be drawn above a cloud is very small. While this graph can sometimes become tedious to construct, it is possible to apply inference methods to remove redundancies and provide a more compact representation to the user.

The hidden states play the role of these nodes and our multi-dimensional state space can easily accommodate for the multiple attributes (model and position). The transition matrix M is captured from the edges of the graph. The confusion matrix B is the Identity matrix as we expect 1) to directly infer the likelihoods of our hidden states from the evaluation algorithms and 2) to observe the exact position of curves.

5.2. Scene Synthesis

Given a sequence of U curve strokes $\Phi(0),\Phi(1),\ldots,\Phi(U)$, a set S of N curve refinement models $\Lambda^0_1,\Lambda^0_2,\ldots\Lambda^0_N$ and a HMM trained using a particular scene graph Λ^1, we wish to determine the most likely sequence of curve-level refinement models that apply on each curve:

$$\max_{\Lambda^0_1\ldots\Lambda^0_N} p\{\Lambda^0(0),\ldots,\Lambda^0(M)\mid\Phi(0),\ldots,\Phi(M),\Lambda^1\} \tag{11}$$

Our first step is to generate U vectors $O^1_0, O^1_1, ..., O^1_U$ from the U curve strokes where each vector stores the likelihoods of all models in S. The main computational step in generating these vectors consists of evaluating all HMMs for all input curves. For the k^{th} curve, we evaluate a refinement model Λ^0_j by iterating the following over the entire arc-length:

$$\psi'\{H_i(t)\}=\sum_j p\{H_i(t)\mid H_j(t-1),\Lambda^0_j\}\psi'\{H_j(t-1)\}$$

$$\psi'\{H_i(t)\}=p\{\phi_k(t)\mid H_i(t),\Lambda^0_j\}\psi'\{H_i(t)\} \tag{12}$$

The model's likelihood is then the sum of all values in Ψ' at the last iteration. This is almost identical to the steps for decoding (Equation 7), but instead of choosing the maximum previous state, we sum the probabilities of all matching states. That is, we examine all the possible ways that the model can be used to synthesize the curve and use that as a measure of its likelihood.

While we can normalize at each iteration when decoding a HMM, evaluation requires the compound probabilities over the entire curve. However, for long curve segments, the probabilities reach very small values and are difficult to store. Thus, at each iteration we normalize and store the normalization constant. After we have evaluated all models, we normalize those normalization constants and then compound the results.

Once the vectors $O^1_0, O^1_1, ..., O^1_U$ are computed, they are used to decode the scene level HMM Λ^1. This is accomplished in the same fashion as described in the curve level HMM. The result is a solution for Equation 11, the most likely sequence of refinement models that apply on the curve strokes. We then decode each individual curve using the associated refinement model.

Since our system is Markovian, we consider the last curve that was drawn in order to determine the next. While this avoids exponential computations, it may results in an under-constrained system. However, computing the models' likelihoods based on both the shape of a curve and its context in the illustration typically provides enough constraints for a unique solution. In the case where we have redundancy in both the shape of the input curve and its context in the illustration, then the system can provide the user with a choice of the likely candidates.

5.3. Scene Synthesis Examples

Figure 13 shows example syntheses of cartoon faces. In this case, each input stroke is a face part (forehead, nose, mouth, chin and hair). We trained five refinement HMMs using several examples for each segment (a few noses, a few chins etc.). Additionally, we included a curve thickness attribute in the examples. The scene-level HMM consists of the following sequence: $head \rightarrow forehead \rightarrow nose \rightarrow mouth \rightarrow chin \rightarrow head$

Figures 14 and 16 show other full scene synthesis examples. Each scene was generated using several refinement models such as a grass model, a skyline model, a tree trunk model, a leaf model and each had its own semantic graph such as that shown in Figure 2. In Figure 16, we show the results using both a greedy strategy and the Viterbi (with back-tracking) method applied at the scene level. It can be seen how when we choose the best model by only considering the current point, all horizontal strokes below the skyline are rendered using the *grass* model, even though we recognize that the shape below the horizontal line looks best like a fish. Using Viterbi, that recognition affects the entire scene. Since a fish can only be preceded by water (Figure 2), the bottom horizontal line is thus rendered using the water model.

All of our experiments were executed on a Linux PC with a 3GHz Pentium IV processor and 1GB of RAM. The results were generated in interactive time.

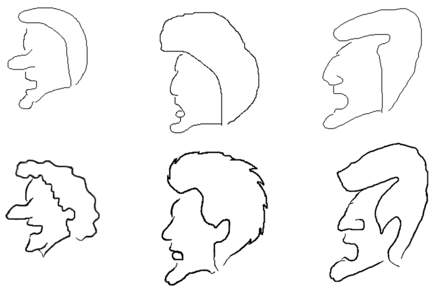
Figure 13: *Generating silhouettes of faces. The top shows the input strokes and the bottom shows the results.*

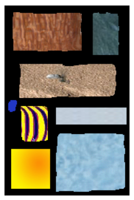
Figure 15: *Training texture used to generate the texture fill in Figure 14.*

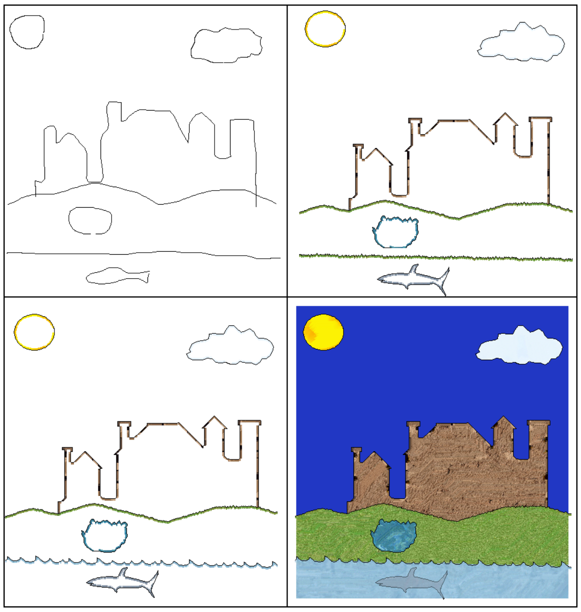
Figure 16: *Top left shows the input sketch, top right shows the output using a greedy method in the scene-level HMM, bottom left shows the output using Viterbi and bottom right shows the result using the Markovian texture filler.*

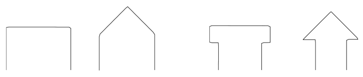
Figure 17: *Training data used for the skyline model in Figure 16. The two left shapes show the control curves and the two right shapes show the refined ones. It can be seen that from a simple set such as this, a novel skyline can be automatically classified and generated.*

6. Conclusion & Future Work

In this paper we have described an approach for the synthesis of stylized drawings from coarse outlines. This process is based on the representation of coarse to fine refinements as a Hierarchical Hidden Markov Model. The desired refinements are learned by example sets and the semantic constraints on those refinements are learned by a semantic graph. Additional constraints can be embedded using a regularization framework. Novel full colored illustrations are generated from noisy curves based on this hierarchy of constraints, including scene level, curve level and, as a post processing step, pixel level constraints.

The synthesis of novel illustrations from examples depends on mixing aspects of different examples from the same set. Excessive mixing, however, would lead to an output curve which is simply an average (in some multi-scale space) of the input curves. At present, the mixing fractions are fixed and predetermined but their automatic determination remains an open problem. We are currently examining the automatic selection of the appropriate mixing weights based on a form of cross validation.

One significant open issue is the application of global constraints to the curves being synthesized. For example, results

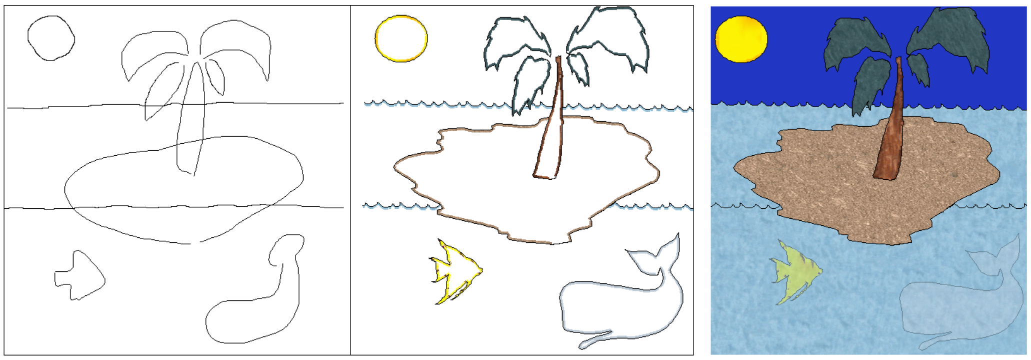

Figure 14: *Synthesis of a beach scene. Left shows the input, middle shows the generated scene including seeds for texture fill, right shows the texture filled scene using the texture shown in Figure 15.*

of the silhouettes of the leaves did not exhibit symmetrical properties or were not guaranteed to close. This also applies to the texture filling process where several filling fronts may not join in a desirable fashion. For example, filling in texture within leaves would require some specialized constraints in order to have the veins of the leaves meet at the right location. Another interesting direction for future work is to develop a method for interactively editing the output. We are examining methods that would allow the users to select regions of the resulting curves and scroll through and select different solutions.

References

[CS01] COYNE B., SPROAT R.: Wordseye: An automatic text-to-scene conversion system. In *Proceedings of ACM SIGGRAPH* (2001).

[FB88] FORSEY D. R., BARTELS R. H.: Hierarchical b-spline refinement. *Computer Graphics 22*, 4 (August 1988), 205–212.

[FS94] FINKELSTEIN A., SALESIN D. H.: Multiresolution curves. In *Proceedings of ACM SIGGRAPH* (July 1994), pp. 261–268.

[FST98] FINE S., SINGER Y., TISHBY N.: The hierarchical hidden markov model: Analysis and applications. *Machine Learning 32*, 1 (1998), 41–62.

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

[HB95] HEEGER D. J., BERGEN J. R.: Pyramid-based texture analysis/synthesis. In *SIGGRAPH* (1995), pp. 229–238.

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

[HZ96] HERNDON A. F. K., ZELEZNIK R.: Sketch: An interface for sketching 3d scenes. In *Proceedings of ACM SIGGRAPH* (1996).

[KMM*02] KALNINS R. D., MARKOSIAN L., MEIER B. J., KOWALSKI M. A., LEE J. C., DAVIDSON P. L., WEBB M., HUGHES J. F., FINKELSTEIN A.: WYSIWYG NPR: Drawing Strokes Directly on 3D Models. *ACM Transactions on Graphics 21*, 3 (July 2002), 755–762.

[KS02] KURTOGLU T., STAHOVICH T. F.: Interpreting schematic sketches using physical reasoning. In *AAAI Spring Symposium on Sketch Undestanding* (2002).

[LM95] LANDAY J. A., MYERS B. A.: Interactive sketching for the early stages of user interface design. In *CHI* (1995), pp. 43–50.

[Rab90] RABINER L. R.: A tutorial on hidden markov models and selected applications in speech recognition. In *Alex Weibel and Kay-Fu Lee (eds.), Readings in Speech Recognition* (1990), pp. 267–296.

[SSD01] SEZGIN T., STAHOVICH T., DAVIS R.: Early processing for sketch understanding. In *Perceptive User Interfaces Workshop* (2001).

[Sze89] SZELISKI R.: *Bayesian Modeling of Uncertainty in Low Level Vision*. Kluwer, 1989.

[WL00] WEI L.-Y., LEVOY M.: Fast texture synthesis using tree-structured vector quantization. In *Proceedings of ACM SIGGRAPH* (2000), pp. 479–488.