A MACHINE LEARNING FRAMEWORK FOR THE CLASSIFICATION AND REFINEMENT OF HAND DRAWN CURVES

Shlomo Saul Simhon

Department of Computer Science
McGill University, Montréal

6 February 2006

A Thesis submitted to McGill University
in partial fulfilment of the requirements for the degree of
Doctor of Philosophy

Your file Votre référence
ISBN: 978-0-494-25255-0
Our file Notre référence
ISBN: 978-0-494-25255-0

ABSTRACT

This thesis presents a machine learning framework for the automatic classification and refinement of curves. The proposed framework is composed of both a representation and a family of algorithms for making inferences from examples, given suitable guidance from a user. The underlying computational paradigm taken consists of applying Hidden Markov Models to a wavelet representation of the curves of interest, each of which is presented as part of a pair of examples. The learning framework is exemplified by developing a gesture-based interface for two distinct applications: robot path planning and sketch beautification. For each, it is demonstrated that we can learn constraints on curves from a set of examples and apply them to augment rudimentary gesture information from a human operator. Further, it is demonstrated that we can identify what class of curves the human input belongs to, allowing us to automate the curve refinement process for unclassified inputs. Finally, in cases where gesture information is given in the form of an image, it is also shown that the same methodology can be used to detect and extract the most likely parametric curve from the image.

There are three key issues that are addressed for the classification and refinement of curves. First, we must establish the way in which the input, training and output curves *look* like one another. In the framework presented, this likeness is expressed statistically using Hidden Markov Models that extend over multiple curve attributes (such as curve thickness or color) and scales. Second, when attempting to infer a curve, we must also determine the way in which the surrounding curves should

affect the inference. Using a hierarchy of Hidden Markov Models, we can impose and exploit probabilistic interactions between multiple curves that make up an entire scene. Finally, in addition to the learned constraints, we must also determine a method for combining user-defined constraints with the Hidden Markov Models. It is shown that we can reformulate the Hidden Markov Models using a regularization framework and allow for the seamless integration of *ad hoc* biases to the learned models.

RÉSUMÉ

Cette thèse présente une structure d'apprentissage informatisée automatique de classification et de raffinement de courbes. La structure proposée est composée à la fois d'une représentation et d'une famille d'algorithmes afin de créer des inférences à partir d'exemples, selon des instructions pertinentes d'un usager. Le paradigme d'informatisation de ce travail est d'appliquer les Modèles Cachés de Markov à une représentation ondulatoire des courbes d'intéret, chacune desquelles étant démontrée par une paire d'exemples. La mise en pratique de cette structure est représentée par deux applications disctinctes: planification du parcours d'un robot et embellissement de croquis. Pour chacune d'entre elles, il est démontrque nous pouvons apprendre les contraintes des courbes à partir d'exemples et les appliquer afin de perfectionner l'information gestuelle rudimentaire d'un opérateur humain. En outre, il est démontré que nous pouvons identifier à quel type de courbe appartient l'entrée d'informations fournie par la personne humaine. Tout ceci nous permet aussi d'automatiser le nettoyage de la courbe pour des entrées non classifiées. Finalement, dans les cas où l'information gestuelle est traitée sous forme d'image, il est démontré que la meme méthodologie peut etre utilisée afin de détecter et extraire les courbes paramétriques les plus probables à partir de l'image.

Il existe trois points principaux qui sont adressés quant à la classification et le raffinement de courbes. Tout d'abord, il faut établir la manière par laquelle les courbes initiale, déntrainement et finale se ressemblent. Avec la structure préée, cette similarité est exprimée statistiquement grace aux Modèles Cachés de Markov qui prennent

en compte les multiples attributs de la courbe (tels que son épaisseur ou sa couleur) et les échelles. Ensuite, lorsque l'on essaie d'inférer une courbe, nous devons aussi déterminer de quelle manière les courbes environnantes affectent l'inférence. Grace à l'utilisation d'une hierarchie des Modèles Cachés de Markov, il est possible d'imposer et d'exploiter les interactions probables entre les multiples courbes qui forment une scène entière. Finalement, en addition aux contraintes apprises, nous devons aussi déterminer une méthode qui puisse combiner les contraintes définies par l'usager avec les Modèles Cachés de Markov. Il est démontré que nous pouvons reformuler les Modèles Cachés de Markov par l'utilisation d'une structure de régularisation et permettre l'intégration aisée des polarisations aux modèles appris.

ACKNOWLEDGEMENTS

I am grateful to have had the opportunity to work with my adviser, Gregory Dudek, who has taken me under his wing and given me the opportunity to take part in some of the most wonderful research in Computer Science. Greg is both an **amazing** adviser and great friend. He has given me the encouragement and confidence that this thesis would not have been possible without. I cannot begin to express how much I have learned and grown while working with him. The countless discussions we had, both on academic and personal facets, and the many problems we worked on together are all invaluable and greatly cherished. Thanks for everything you've done Greg!

I would like to thank members of my committee and all other faculty members at McGill who I've had the opportunity to either take courses with or get feedback about my work. In particular, I would like to thank James Clark, Frank Ferrie, Mike Langer, Doina Precup and Sue Whitesides for their insightful feedback and discussions.

Many thanks to Luz Abril, Eric Bourque, Paul Di Marco, Matt Garden, Dimitri Marinakis, Ioannis Rekleitis, Junaed Sattar and Rob Sim who are both my friends and lab-mates. I am grateful for our friendships and for all the technical assistance. I will miss our weekly lab-meeting/lunch duo!

Finally, I thank all my family for their support; my parents and brothers who have always been there for me. I especially thank Stacy, whose love and encouragement has helped me overcome all obstacles. Thanks for putting up with me throughout this long endeavor!

This thesis would not have been possible without the financial support from FCAR (Fonds de Recherche sur la Nature et les Technologies) and IRIS/Precarn.

for Stacy, Sage, Samson and my angel Shoshana Shalem

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

Introduction

Human interactions can be characterized by a set of signals or cues that typically result in ambiguous, noisy or incomplete expressions. Cues such as words, body language or tone of voice are thus subject to interpretation by the recipient who, based on an assumed context and prior knowledge concerning the subject, attempts to infer missing information and disambiguate the message being conveyed [**17**]. Can computer systems similarly interpret such informal expressions? This thesis examines the problem of how to represent and use knowledge in order to deal with ambiguities, noise or missing information. This is a longstanding research problem in Artificial Intelligence (AI) which has been extensively investigated in contexts such as Robotics. The focus of this thesis lies in aspects of human-computer interaction for robotic and graphical applications; extending and applying AI techniques to develop a *smart user interface* which can interpret the rudimentary hand gestures used for sketching and robotic control.

While it is common in human dialog to say "I know exactly what you mean" (and in most cases be correct), a computer system simply "knows exactly what it knows". That is, the process of high-level interpretation that is implied by the statement "I know what you mean" is generally lacking when interacting with a computer system. This is evident in many of today's systems where there is a large gap between the level of abstraction possible in human-computer interactions and the level of abstraction

which humans are normally accustomed to when interacting amongst themselves. To accomplish what one "means" typically requires substantial low-level specifications, fine tuning and specialized expertise or talent. Whether for a computer program, a schematic drawing or simply a set of point-and-click actions, the acceptable human inputs for producing the desired outputs are usually constrained by a rigid set of low-level requirements.

An important concept in AI is that of abstraction; i.e. methods that attempt to provide more compact representations for a problem of interest. The process of abstraction is often used in commonsense reasoning to eliminate unnecessary details from information involved in some task [**32**]. While finding an appropriate abstract representation is an essential problem, conversely, decoding an abstract model is an even larger challenge. The challenge is to automatically compute the missing values of low-level parameters, which are larger in number with potentially complicated interdependence, from the given values of high-level parameters, which are fewer in number and more suitable to the user (or agent). Indeed there are many cases where this is trivially accomplished. For example, consider the image of a circle. One does not need to specify every pixel on the screen, but rather to simply identify a center and radius and the circle can be rendered automatically. On the other hand, consider designing a new sixteenth-century style chapel. One requires a substantial amount of meticulous specification of the parameters which will fully define the desired output. How to provide the appropriate abstraction model and to subsequently acquire the knowledge required to decode that model remains an open problem.

Reducing the required amount of user intervention while, at the same time, providing enough expressive power is a key sub-problem in abstraction. There are growing efforts in the fields of Robotics and Computer Graphics, where end-users are challenged by increasingly complex tasks, to both implicitly and explicitly address this issue. For instance, there has been substantial work in Robotics where the goal is to automatically control low-level motor parameters in order to provide a sufficient set of high-level behaviors [**12, 4, 64**]. (The most popular methods are known

as hierarchical control, reactive control and subsumption.) Similar ideas have been applied in Graphics and Animation where the low-level control points required for producing complex scenes are abstracted by higher-level models with more intuitive controls [**3, 28**]. Depending on the domain of application, the methods and degree of abstraction can vary substantially.

In this thesis, a gesture-based interface is developed for two applications of interest; sketching and robot control. For the sketching application, a sketch beautification system is developed that transforms rough hand-drawn curves to produce refined versions of them [**80**]. For the robot control application, a path-planning system is developed that automatically generates kinematically correct trajectories from rough goal trajectories (that are otherwise invalid) [**77**]. These two seemingly different applications share in common aspects of abstraction; to take the coarse inputs and automatically compute the low-level details that satisfy some desired constraints or preferences. This thesis presents a generic framework (suitable to both applications) for modeling the knowledge required to both define the validity of the low-level output and associate that output with a high-level layer for control. The following outlines the important criteria that are considered in developing this framework:

Simplicity: A key criterion for the framework is that of simplicity. The resulting system must simplify what is otherwise a complex task to the user. However, the focus is not to directly address aesthetic issues such as the usability of an interface, the layout preferences or difficulty in system set-up and configuration. While all these are important components to consider, the main problem of interest is how to reduce the amount of data or expected accuracy required to produce the desired end-results. Thus, the user-friendliness of the system relates to the convenient and intuitive forms of input that can be used in order to perform difficult tasks.

Controllable: The framework must provide a suitable control scheme. Unlike many AI applications, the goal is not aimed at developing a completely autonomous system, rather, the behavior of the resulting system should be

dependent and tightly coupled to the human input. This semi-autonomous and collaborative approach emphasizes the aspects of human-computer interactions in the system. Users must have the ability to easily *steer* the system toward the desired results.

Customizable: The way in which the system behaves may vary from one user to another. Users have different preferences and abilities, some may wish to have a certain degree of control while others may wish to change the control scheme altogether. The framework must therefore be flexible in the way it handles the inputs, allowing for custom configuration that accommodates personal preferences.

General: The framework should be general, with limited domain specific assumptions. Supplementary *ad hoc* constraints that are suitable for specialized applications must be easily integrable into the system. The framework must also support parameters in arbitrary dimensions such that it can represent domains with multiple degrees of freedom.

Example-Based: Having to manually model the desired behavior of the system for each application is a cumbersome task. Instead, the framework should have the ability to learn from examples the range of valid outputs for a particular application and its relationship to the expected inputs.

Expressive Power: The range of possible outputs that the system can produce for a given domain should not be overly restrictive. The provided learning framework must adequately generalize specific examples to a broad ensemble of cases.

The framework developed is based on Markov Models: probabilistic descriptions of how sequentially ordered states are related. Specifically, we use *Hidden* Markov Models (HMMs), a modeling formalism that allows us to express the relationship between aspects of a system that can be observed directly (the coarse input from the user) and variables that cannot be observed, but which determine the output (what the user "really wants"). This doubly stochastic model combined with several

other techniques discussed later form the core of the framework satisfying the criteria described above.

1. Gesture-Based Input

Gesture-based interfaces provide users with a natural method for interacting with computer systems. There is a wide array of hardware devices that attempt to accurately capture human gestures. Examples range from expensive wearable sensor systems that capture multiple degrees of freedom to simple point and click devices (Fig. 1.1). Pen-based devices such as pen tablets, tablet PCs or electronic white-boards are becoming increasingly popular tools for users of varying degrees of expertise. While such systems may soon become common HCI devices, driving the demand for novel application, the research dealing with processing and analyzing pen strokes is still in its infancy.

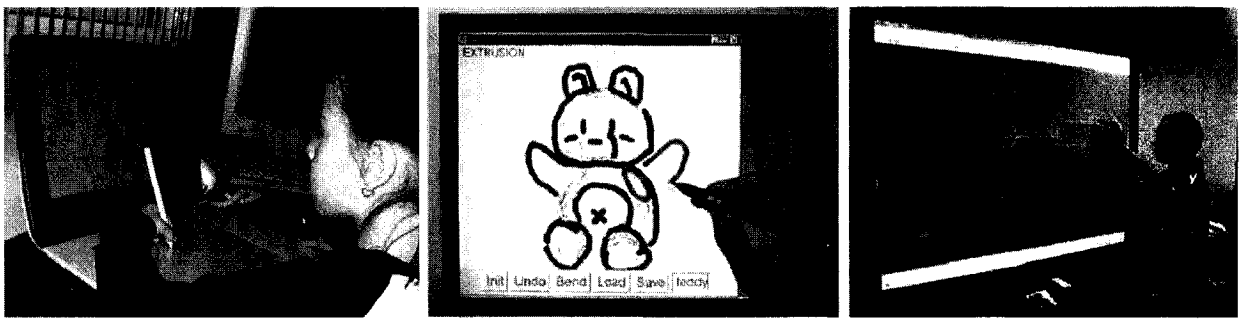


FIGURE 1.1. Examples of gesture-based hardware devices. The left image shows a child using a simple electronic pen tablet, middle shows a tablet PC (from teddy [**41**]) and right shows an electronic whiteboard (from T. Stahovich *et al.* [**50**]).

In the framework presented here, a signal processing approach is taken to process pen-based inputs. Every pen stroke drawn by the user can be considered as a stochastic signal, represented by a parametric function over time (or traveled distance). In general, the parametric functions can be of arbitrary dimensionality, encoding multiple attributes such as pen position, speed or pressure. Continuous functions that map a one-dimensional space to an n-dimensional space are referred to as *parametric curves*. As such, throughout this thesis, all user inputs to the system are referred to

as curves (e.g. the drawn outlines of a sketch or the coarse trajectory of a robot). Similarly, all of the resulting outputs are also referred to as curves, parametric functions that compute the desired application specific output (mapping attributes such as position, motor speed, coloration or thickness).

2. Sketching

Drawing a sketch is one of the most common and versatile ways to convey information. Sketches are often found in comics, presentation material, cel-animation, storyboard designs, system designs (sketch to prototype) and non-photorealistic pen-and-ink illustrations (Fig. 1.2). Despite the natural ease of drawing a sketch, the creation of high-quality good-looking sketches remains time consuming and skill dependent. In fact, a search on the web for the phrase "I can't draw" returns roughly 74,000 hits (using the Google search engine, August 28, 2005). While almost everyone can sketch a crude illustration, only a few have the artistic talent and patience to draw the refined details they wish to depict. Even those who are lucky enough to possess those abilities may not have the appropriate tools at hand.

In many cases, sketches are used as a first-order presentation of concepts [**93**]; to quickly construct a coarse visualization of an idea. The purpose is not to produce a physically or cosmetically correct illustration but rather to easily capture an idea that may or may not materialize later in a more comprehensive design. Such sketches typically consist of a set of curves that are disproportionate, noisy and coarse. When these sketches pass the "drawing board" phase or if they need to be presented more clearly to other viewers, the intricate task of preparing a more refined version takes place. Existing tools to accomplish this in a digital domain include software applications for CAD, diagramming, desktop publishing, image editing and vector graphics.

While the initial sketch typically lacks the details required for an unambiguous interpretation, it is meant to provide sufficient information for a human observer to *envision* the artist's original intention and construct a *Mental Model* [**18, 45**]. This interpretation is dependent on both the shapes of the curves and their context in

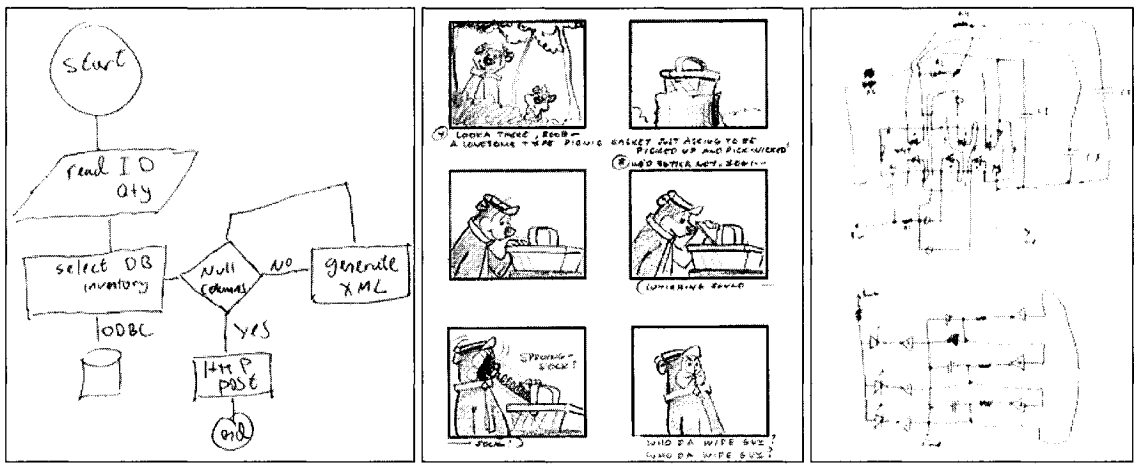


FIGURE 1.2. Sample sketches. The left shows a flow chart, the middle shows a storyboard and the right shows a circuit design.

the sketch (Fig. 1.3). For example, in one context, a rough circle may be a coarse representation of a gear, while in another, it may represent the head of a stick figure. The resulting Mental Model in combination with the tools at hand are used in a skill dependent feedback process to produce the final drawing.

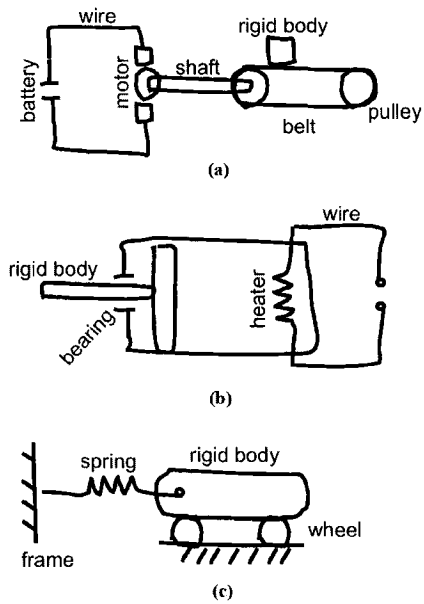


FIGURE 1.3. Sample sketches demonstrating ambiguities (e.g. in one case the zig-zag shape is a resistor while in another its a spring, T. Stahovich *et al.* [**50**]).

Given the diverse set of possible types of drawings and contexts, can a universal system be developed that automatically infers the refined version of a sketch? A system that uses only specialized domain specific constraints may be overly restrictive. Rather, the knowledge of what the refinements should be, for a given domain, must be extracted automatically from an ensemble of examples that show samples of the desired output.

This leads to two key problems that must be addressed in a trainable setting; the curve refinement problem and the curve classification problem. First, the system must be able to learn the desired types of curve refinements and then transform a coarse curve to exhibit the desired style (treating each curve in the sketch independently). This is accomplished by applying a *curve refinement model*; a set of learned rules that transform a coarse curve to a refined one. Second, for each curves in the sketch, the system must automatically select most appropriate refinement model that should be applied from the set of all possible models (there can be several such models trained under different contexts or styles). This is accomplished by choosing the refinement model that best transforms the curve while also satisfying high-level relationships between other curves in the sketch (using semantics that identify the types of curves in a sketch). For example, suppose a curve is best represented using a refinement model for a tree (the semantic label), the refinement models applicable to other curves above it should be ones for clouds, leaves, birds, etc.

The approach taken to address these problems consists of modeling probabilistic constraints on curves in a sketch using a two-level hierarchy of Hidden Markov Models [**25**]. Each HMM in the first level describes the refinements for a class of curves and each HMM in the second level describes the high-level constraints for a class of scenes. This leads to a relationship between the HMMs used on individual curves, the *curve-level HMMs*, and those used to specify the identity of objects within the entire scene, the *scene-level HMMs*. The scene-level HMM encodes probabilistic constraints on the application of the curve-level HMMs based on the type of sketch that is being drawn. When a curve is refined using a particular curvel-level HMM, the refinements

applicable on the neighboring curve must be compatible. For example, we should never refine a curve to look like a fish when a previous curve below it has been refined to look like a tree. While this approach does imply a sequential ordering of curves, it is possible to either ignore the sequential ordering constraints (described in Chapter 3), dynamically sequence the curves based on their proximity or extend the framework to use Markov Random Fields.

While the HMMs provide a powerful framework for learning local constraints on curves, their application alone may not be sufficient for the desired behavior of the system. Additional domain specific constraints may be required to complement the learned ones and further enhance the way a curve is synthesized. In order to allow for such supplementary constraints (*ad hoc* biases) to be plugged into the system, the curve-level HMMs are reformulated using a regularization framework (discussed in Chapter 4). This is especially important in cases where the prior distributions are not easily available and explicit analytical functions must be used to further bias the distribution. For example, it may be desirable to include a constraint to enforce curve closure, which is too difficult to model using only HMMs but can easily be incorporated as an regularizing bias using analytical functions.

Developing a *smart* sketch-based interface in this fashion can facilitate many of the difficult and laborious drawing tasks. Such an interface can be used for technical drawing and diagramming, where primitive shapes such as lines and arcs can make up a training set. Instead of using the traditional point-and-click technical drawing interfaces, the user can naturally sketch out a novel diagram while the smart interface infers the new shapes that are *stylistically similar* to the training set. Similarly, the interface can also be used for creating web-art and can potentially be extended to animation. One potential difficulty in the usability of such a system is that the training set must be carefully constructed in order provide the desired types of outputs. Though this is a one time set-up step, users must have a good idea on the types of outputs that can be produced for a given training set.

2.1. Segmentation. If the sketch is drawn using a digital medium (such as a tablet, pointing device or PDA) then the pen strokes may be available to the system. An interesting problem arises when the original pen strokes are not directly available but have already been rendered to an image. Can the original pen strokes be detected and extracted from the image? This issue is also relevant to many modern image editing applications where the available tools for transforming objects are applied under the assumption that the objects have already been isolated. Manually extracting individual pen strokes from images that are noisy or include occlusions can be a tedious task.

Curve segmentation is a well established researched topic in the fields of Computer Vision and Image Processing, with particular applications to contour detection and object recognition. Approaches to this problem range from band-pass filtering techniques to curvature heuristic-based methods [**21**]. In this work, the learning framework is used to extract parametric curves from images of sketches. The approach consists of searching through all the possible parametric curves that explain the image and pruning those that are inconsistent with the learned constraints [**79**]. The user can then simply click near an end-point of the desired pen stroke and the system automatically extracts it based on its similarity in style with the training set.

3. Robotic Control

In motion planning (e.g. for a robot), there are typically two types of constraints to consider on the paths that can be taken: *extrinsic* constraints, imposed by the environment or other external factors, and *intrinsic* constraints, imposed by the physical characteristics of the vehicle itself. Finding valid trajectories that satisfy both the mechanically imposed constraints and environmental constraints can be a difficult task. Further, motion planning is complicated not only by the need to generate these paths, but also by the need to initially model whatever constraints may be imposed by a particular vehicle or task.

Traditionally, intrinsic motion constraints have been modeled using analytic methods that constrain the differential geometry of the set of admissible paths. The constraint equations for motion are complex relations that attempt to simulate the dynamics or kinematics of a mobile robot based on its mechanical design. In particular, *non-holonomic* constraints refer to limitations on the allowed derivatives of the path, and planning in the presence of such constraints is often difficult (an automobile is a common example of a vehicle with such constraints, as it is unable to move perpendicular to the direction in which it is facing, Fig. 1.4). There are two approaches taken for analytically modeling these constraints: one based on *forward kinematics* and the other based on *reverse kinematics*. Forward kinematics refers to the modeling formalism used for computing the trajectory a robot can take given a sequence of model parameter values (which directly translate to motor commands). A regular sequence of such parameter values for an articulated robot is often referred to as a *gait*. The reverse kinematics based approach consists of deriving explicit models that can solve for the sequence of parameter values (or gaits) that result in the robot traversing a desired goal trajectory. Path planning typically entails either solving the inverse kinematic models or solving an optimization problem over the forward kinematic constraint equations.

Motion constraints are not only used for modeling a robot's internal mechanical configuration but are also used to model extrinsic motion preferences. In some applications, equations are constructed to model task specific motion requirements, such as a sweeping pattern for full floor coverage or a suitable behavior to scan the environment using a narrow-beam sensor. Specialized paths also occur in various specialized contexts; in the classic 1979 film "The In-Laws" Peter Falk instructs Alan Arkin to run along a "serpentine" path while heading for a goal that is straight ahead. Additionally, in applications such as obstacle avoidance, motions are not only related to the robot's pose but are also a function of the perceived environment. In all of these examples, the underlying core problem consists of finding a valid transformation between two components: 1) the idealized "raw" path that directs the robot to a

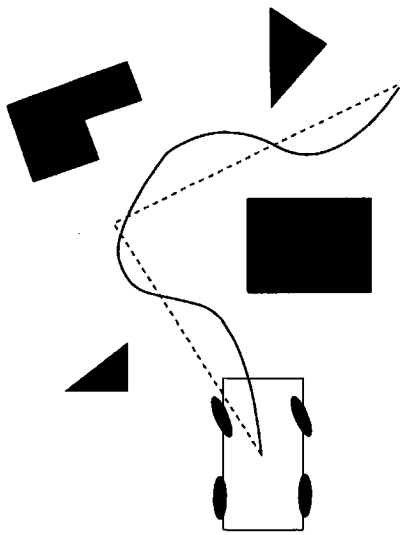


FIGURE 1.4. Example trajectory for a car-like robot. Instead of the simpler trajectory (dashed path), due to the limitation on the turning radius a more complex trajectory must be taken (solid path).

goal without taking into account certain preferences or constraints, and 2) the refined path that attempts to reach the goal while also satisfying the system constraints.

Whatever the constraints, expressing them in a suitable formal framework is often challenging. Further, the processes of finding allowable solutions can be costly, particularly since the solution techniques are often engineered for a specific context. In contrast, this thesis presents a radically different approach to path planning. The presented machine learning framework is used to simulate the motion constraints without having to explicitly model them. The constraints (or preferences) are expressed in terms of a set of examples that illustrate how the robot is permitted to move. Further, these examples indicate how to elaborate a coarse input path from a user (which is typically not acceptable in itself) into a suitable acceptable output path. Informally, the examples say: "if a user asks you to do something like *this* than what you should actually perform is a maneuver like *that*". This novel approach to path planning is referred to as *analogical path planning* [**78**], wherein paths are generated by analogy with previous observed acceptable paths and without an analytic model.

In similarity with the sketching application, Hidden Markov Models are used to both model the constraints on the allowable paths and provide a layer for high-level control. Further, using the regularization framework, these HMMs are dynamically biased in order to account for environmental constraints. Obstacle avoidance is performed by computing a distance transform over boundaries in the environment and combining the resulting field with the trained HMM. This biased HMM constrains the configuration space of the robot resulting in trajectories that satisfy the desired motions and avoid obstacles.

Because the system learns from examples, it can be applied in a variety of domains. This avoids having to extract and analytically model constraints for each desired task or mechanical configuration that we may wish to control. One can simply demonstrate how a robot can move and subsequently the system can automatically produce new paths based on these motions. One area of application is tele-operated robotics. A human can guide the robot to areas by simply sketching a coarse path. The system can then refine that path based on the learned specification of the robot and generate a new valid path analogous to the goal. Another application is to complement high-level planners to relieve them of the burden of non-holonomic (complex) path planning. Our system can take in as input the paths generated by such planners and augment them to avoid obstacles while maintaining a desired behavior during motion.

4. Contributions

The main contribution of this thesis is the development of a machine learning framework applied to a novel gesture-based interface for creating illustrations and controlling a robot. The novelty stems from the idea that coarse gestures can be used to steer a Markovian-based synthesis procedure and produce new refined outputs. The synthesis procedure is rooted on the idea that stitching and blending fragments from a training set can results in new examples that can also be considered part of the set. While many of the models used in this thesis are already well established

(such as HMMs, wavelets), they have never been used together in the fashion presented nor have they been applied as a unified approach for the applications presented here (sketch beautification, curve extraction from images and robotic control). The following lists the contributions of this thesis in more detail:

- A novel sketch beautification system is developed where curve transformations are represented by a hierarchy of probabilistic models.
- A novel gesture-based robot path-planning system is developed (called analogical path planning) where robot trajectories are produced by analogies with the the training set.
- The same framework used for the sketching and robotics application is also used to develop a novel method for segmenting images.
- A novel two-level Hierarchical Hidden Markov Model is developed where the first level of the hierarchy (curve-level) models probabilistic constraints on individual curves and the second level of the hierarchy (scene-level) models probabilistic constraints over entire scenes. A third level process is also developed (pixel-level) to synthesize textures.
- Using dynamic programming, an efficient algorithm is developed for synthesizing full colored illustrations from the hierarchy of models. Unlike most approaches taken in this domain, the algorithm is not based on a greedy strategy but rather takes into account the entire sequence of inputs, both at the curve-level and at the scene-level, before committing to a final solution.
- Hidden Markov Models are used in combination with wavelets to efficiently capture long-range probabilistic constraints from the training data. This simulates a higher order Markov process without exponentially increasing the state space.
- A regularization framework is combined with the HMMs in order to integrate supplementary analytical biases whose net biasing effects are otherwise too difficult to automatically learn.

- A dynamic labeling scheme is developed to represent training sets in continuous domains using discrete samples. The scheme applies on multi-dimensional samples and can simultaneously encode multiple curve attributes (such as curve color or thickness), multiple input attributes (such as pen pressure or speed) and multiple scales. In this fashion, the state space has at most N elements where N is the number of unique sample points form the training set.
- Learning curve transformations is simplified by providing training sets that consist of a pair of examples (i.e. labeled samples). Rather than having to use computationally complex learning algorithms such as Expectation Maximization, learning is performed by simply analyzing the statistics of the training set. Using training sets in this fashion also allows users to explicitly customize the way curves are transformed.
- Using Gaussian basis functions, a method for controlling the sensitivity to make transitions between training examples is developed. The variance of the Gaussian controls the degree of mixing examples where at one end of the spectrum the outputs are forced to resemble exact instances of the training set while at the other end of the spectrum arbitrary mixtures can take place.
- Using a sigmoid blur, a method for controlling the sensitivity of the system to the user input is developed. The parameters of the sigmoid regulate how sensitive the system is to the user input.

The thesis includes a theoretical framework, an instantiation in software and experimental validation.

5. Outline

This thesis is organized as follows:

Chapter 2: Relevant work in the fields of Computer Graphics, Vision and Robotics are presented and discussed in context to the applications of interest.

Chapter 3: An overview of the framework is presented, including a discussion of the curve representation, Markov Models, multi-scale methods, Hidden Markov Models and the the Hierarchical Hidden Markov Model.

Chapter 4: The method for learning curve refinement models and applying them on individual curves is presented.

Chapter 5: The framework is customized for the sketching application and results are presented using various training sets.

Chapter 6: The framework is customized for the robotics application, including an additional function for obstacle avoidance. Results are presented for a simulated robot.

Chapter 7: The method for learning scene refinements and applying them on entire sketches is presented. The curve extraction algorithm is also presented and the results for sketch refinement and curve extraction are shown.

Chapter 8: A conclusion and discussion of future work is presented.

CHAPTER 2

Related Work

This chapter reviews the related work in the fields of Computer Graphics, Image Processing and Robotics. While all of these areas together are beyond the scope of this thesis, there are sub-domains of applications that address similar issues from different vantage points. Section 1 describes recent efforts for developing convenient curve and surface models, a well established topic in Computer Graphics. Section 2 describes some existing sketching systems where the goal is to provide a natural interface for creating illustrations. This is followed by a discussion of image processing systems that attempt to detect and extract curves from images. Section 3 reviews relevant work in robot path planning and animation, where the control of elaborated systems is crucial.

1. Curves and Surfaces

In Computer Graphics applications, one of the most widely used mechanisms to construct an illustration consists of manually laying out curves and surfaces. This is accomplished by specifying a set of control points and using an *interpolating function* that defines the geometry between the points. First-order functions (e.g. polylines) produce piecewise linear approximations to the desired shapes while higher-order functions provide a smoother approximation with fewer vertices. Common curve and surface representations are based on third-order tensor products known as NURBS

(Nonuniform Rational B-Splines) [**27**]. Such models provide good expressive power, local support, up to second-order parametric continuity and invariance to affine transformations (i.e. transformations need to be applied only on the control points). Implicit models [**74**], defined by a function $f(\mathbf{x}) = 0$ for all points $\mathbf{x}$, are an alternative form of representation that are less widely used due to their computational complexity and memory requirements. They are most commonly used in applications where point classification is critical such as collision detection, constructive solid geometry and shape blending [**92**].

In general, parametric interpolation consists of hard-coded smooth functions $\{x(t), y(t), z(t)\}$ for a curve or $\{x(t,s), y(t,s), z(t,s)\}$ for a surface over a domain such as $[0, 1]$. These functions alone are often not intuitive enough to provide a natural interface for constructing and manipulating shapes. Important details are usually blurred-out during interpolation and can only be preserved by manually adjusting an excessive number of control points. For example, in order to represent shapes of arbitrary topology, these models must be partitioned into a collection of patches and explicitly stitched together [**24**]. A large number of parameters are introduced to stitch adjacent patches and enforce geometric continuity conditions. This is further complicated in cases where the designers wish to interactively edit the model on a regular basis. Instead, approaches such as hierarchical modeling, multi-scale methods, subdivision schemes or functional minimization can be used to further extend the functionality of the underlying parametric models.

One of the key ideas for facilitating user friendly interactive models is the notion of coarse to fine control, or *abstract* to *detailed* representation. This idea has been explored in early work by Forsey and Bartels [**28**] in which hierarchical B-splines are developed. Rather than having the user interact with a single control layer, large- or small-scale edits can be made by manipulating control points at the corresponding levels in the hierarchy. Similarly, Salesin and Finkelstein [**26**] develop a multi-resolution curve representation using *wavelets*. A curve is decomposed into n resolution levels

using a multi-scale basis function based on B-splines. (The 0'th degree B-spline reduces to the Haar basis.) Curves may then be modified at multiple levels of detail, such as changing the overall form of the curve while preserving its detail or vise versa.

Instead of providing the user with multiple levels of representation, another approach consists of attempting to automatically determine those areas of a curve (or surface) that require higher resolution. In this approach, the sampling resolution is related to the geometry of the shape rather than a level of some hierarchical representation. Parametric subdivision schemes, first introduced by Doo *et al.* [**19**] and Catmull *et al.* [**15**], consist of repeatedly refining an initial control mesh until a satisfiability criterion is met. Applying subdivision rules on the smooth basis functions results in piecewise smooth shapes that maintain the character of sharp features such as creases, corners and darts while reducing the number of control points at smoother areas.

Such multi-resolution approaches provide enhanced flexibility and control and a designer using them can easily interact with pre-fabricated families of shapes with fixed topology. However, once the topology is established, it becomes a tedious task to modify it. Weltch and Witkin [**99**] describe a variational calculus approach to free-from shape design by representing a surface as the solution of an energy minimization problem. The control points of B-spline basis functions are dynamically computed to satisfy a desired objective function (smoothness) in conjunction with the potentially varying shape constraints. Topological changes are managed with heuristic based facet splitting and merging techniques. Users can pin-down, cut, extrude and merge the shapes.

In general, functional minimization techniques can be used with arbitrary objective functions, including physics-based formulations where the objective is to simulate or approximate the dynamics of real world systems. Such formulations typically include specialized constraints that define an application specific behavior for the desired shape. For example, Terzopoulos and Fleischer [**87**] model flexible surfaces such as cloth by connecting a grid of points with springs, dash-pots and plastic slip units.

When applying forces at different positions, a new cloth-like shape is computed by solving an energy minimization problem. Baraff *et al.* [**83**] model the shattering of brittle objects using a set of point masses connected by linear constraints. Forces are cascaded through the lattice using Lagrange multipliers, producing cracks when surpassing a predefined threshold. A similar approach is taken to develop a smart floor-plan designing system [**36**]. Finding good approximation models with computationally tractable solutions is a key hurdle to overcome in these physics-based systems.

2. Sketching

Though most of the approaches described in the section above result in computationally complex solutions, the main intent is to create an illusion of simplicity for the designer. The ideal representation should produce the impression of a continuously malleable shape having no fixed control points. Indeed this vision is synonymous with the core ideas behind may of todays smart sketching systems, where the aim is to simulate the simplicity experienced in pen-and-paper drawings. There are a variety of related sub-goals that sketching systems attempt to achieve, including modeling, beautification, recognition, synthesis and classification. The recent approaches taken to deal with these problems is the subject of this section.

Traditionally, methods for pen-based modeling use specialized constraints that are based on rules and preferences for a given domain. Such methods attempt to estimate the parameters of geometric objects that best fit the data points acquired from a pen-stroke. The difficulty in solving for this stems from both the noisy nature of most data sets and the ambiguities present in the object models themselves (different parameter values can result in a similar shape). This has been demonstrated by several authors. Banks and Cohen [**8**] develop a system for the real-time fitting of B-splines to hand-drawn curves. The approach, based on earlier work by Lyche and Morken [**59**], is to first consider the initial curve sample points as B-spline knots and then iteratively remove the knots that have least influence the shape. Davis *et al.* [**75**] describe a

method that uses both curvature and speed of a pen-stroke to detect vertices for a hybrid geometric model consisting of a combination of polylines and Bezier curves. Novins and Avro [**5**] apply a continuous morphing procedure to interactively morph segments of a hand-drawn curve with the best fit shape from a set predefined basic primitives such as lines, arcs and boxes. Igarashi *et al.* [**41**] develop a system where 2D contours of shapes are sketched out with the intent of producing 3D surfaces that pass through them (Fig. 2.1). A 3D mesh is automatically "inflated" from the flat image by first triangulating the contour using the control points and then computing the shape's spine along the triangulated plane. New vertices and edges are attached to the spine joints and are elevating in proportion to their distance from the surrounding edges. Topological edits can also be performed by sketching out extrusions or cuts from the drawn shape. Mesh beautification systems [**40**] or line-drawing beautification systems [**42**] can be used in cases where the resulting objects do not satisfy preferences such as uniformity, smoothness, perpendicularity, congruency and symmetry. Though these and other similar approaches have proven to be very successful, the extent of their application is strictly limited by the expressive power of the geometric models used.

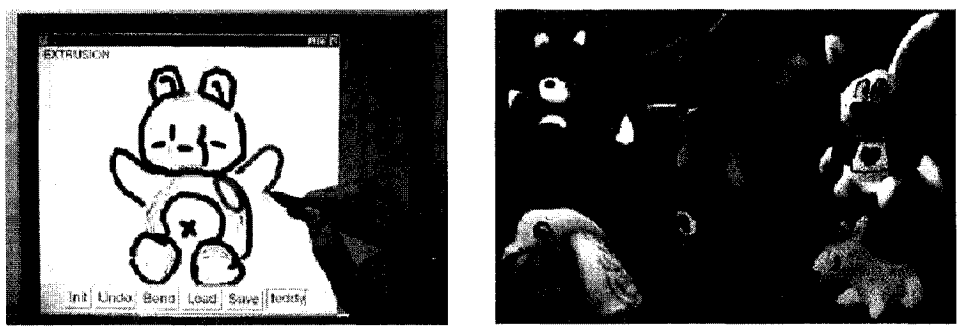


FIGURE 2.1. Example interface and results from [**41**]. Left shows the interface and right shows 3D example results.

Another research direction involves the use of a training set that shows examples of the types of curves that the user is expected to draw. By simply providing the appropriate examples, these systems can dynamically adapt to both the personal drawing habits of a user and the desired domain of application. Rubine [**71**] expands

on this idea to develop a gesture-based interface for drawing, editing and writing text. The core of the work consists of a trainable recognizer that classifies gestures using a linear discriminator on features from the input gesture and the target examples. Landay and Myers [**52**] further make use of this recognition system to develop a sketch-based interface for designing and creating graphical user interfaces. Their system recognizes predefined curve strokes as UI widgets and further groups them based on predefined preferences on spacial relationships. A similar approach is taken by Zeleznik *et al.* [**103**], where example gestures are used as quick-hand notations of 3D objects and editing commands. Lipson and Shpitalni [**57**] develop a system for reconstructing 3D polyhedra from 2D line drawing by learning from examples the correlation of connected lines in 3D space to their planar projection.

The methods mentioned above provide flexibility in the types of gestures that are recognizable. Once recognition is performed, the outputs consist of some parametric variations of predefined primitives (such as a line, a cube or a character). Recent advancements in texture synthesis and image restoration methods [**37, 98, 31**] suggest that we can learn the regular properties of example images and generate new ones that exhibit the same statistics but are not exact duplicates of the original. Work by Hertzmann *et al.* [**38**] show how this stochastic approach can also be taken to stylize hand-drawn curves (Fig. 2.2). In their work, curve styles are learned from the statistics of example styles and new curves exhibiting those same styles are synthesized along the shape of the input curve. Analogies between the inputs and outputs are computed by calculating an offset between the best matching segments of the input curve and training examples. This offset is then used for a rigid transformation on the best candidate match. Likewise, Freeman *et al.* [**29**] present a example-based method to stylize line segments. Novel curves are generated as a linear combination of the k nearest neighboring examples in the training set. In work by Kalnins *et al.* [**46**] these ideas are extended to automatically synthesize stylized silhouettes of 3D object, part of a comprehensive interactive system for gesture-based annotations of non-photorealistic rendering styles on 3D objects. The framework presented in

this thesis is similar in spirit to such methods where, using a probabilistic approach, users can controllably synthesize novel outputs that are similar to, but are not exact instances of, examples in the training set.

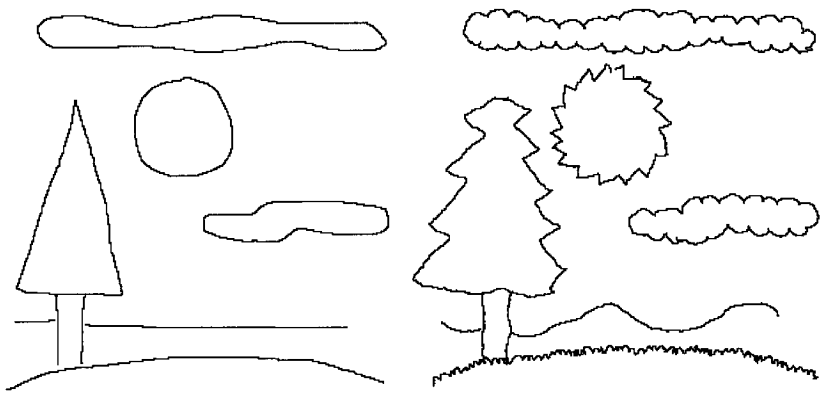


FIGURE 2.2. Example curve analogies from [**38**]. Left shows the input curves and right shows the output.

Another important function of a sketching system is to determine if the interpretation of the current object is compatible with the interpretation of the surrounding objects. In principle, this problem is similar to the ones addressed by the methods discussed above. Rather than only considering constraints on individual pixels (or sample points), the pixels are grouped to form objects and constraints are then applied on these objects. Several researchers take this into consideration and develop systems that attempt to resolve ambiguities by examining the compatibility of the placement of objects using their semantics. Alvarado and Davis [**1**] describe a system for recognizing and disambiguating shapes in mechanical drawings. Included in the system are constraints that are applicable to individual objects, with preferences for temporal coherence, simplicity and high recognition confidence, and constraints on how these objects relate to one another, with preferences for valid mechanical layout and physically feasible configurations (Fig. 2.3). Similarly, Kurtoglu and Stahovich [**50**] identify the physical compatibility of sketch components in mechanical drawings and remove interpretations of object that are in incompatible classes. In later work by Alvarado *et al.* [**2**], shape description grammar rules are used to define high-level

objects by patterns of low-level primitives. A hierarchical recognition system coupled with a Bayesian network is developed to recognize the domain specific objects based on the interpretation that best fits the grammar rules. In work by Viola and Shilman [**76**], an A^* algorithm is used to search through the set of possible groupings of neighboring pen-strokes to find the optimal interpretation. The search is based on an underestimate measure for segment compatibility, which can be computed using arbitrary recognizers.

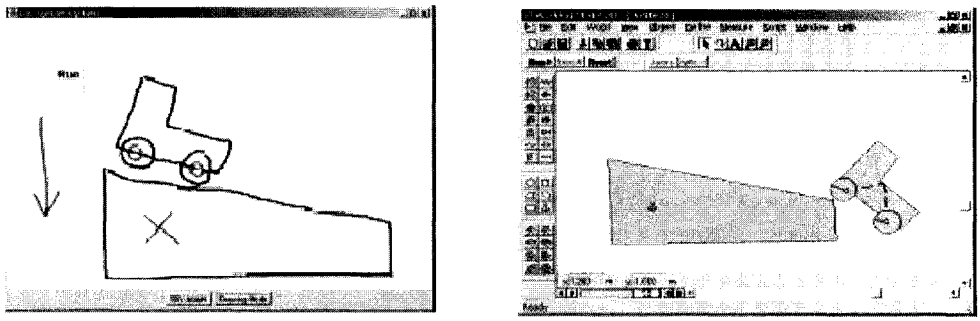


FIGURE 2.3. Example sketch interpretation from [**1**]. Left shows the input sketch and right shows the physical interpretation.

In this thesis, three key ideas are developed that extend the sketch based methods cited above. First, using a Hierarchy of Hidden Markov Models, the system can capture the interaction of multiple stochastic functions in order to represent scene dynamics over various scales and contexts. Second, the individual HMMs themselves are two-layered systems, where one layer models the output generation process while the other ties in controllability to that process. That is, the synthesis is driven by the input such that the actual features generated are directly dependent on the shape of the input. This allows us to model examples with localized and non-stationary 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, many of the existing approaches to synthesizing novel outputs consider greedy strategies, always choosing the best match at the current point. When we are given inputs or partial data, the locally best points may not contribute to 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. Using a dynamic programming algorithm, the system takes into account the entire sequence of inputs while also avoiding excessive run-time complexity.

2.1. Curve Extraction. If the stroke sequencing is not directly available but has already been rendered in the form of an image (i.e. a digital scan of a hand-drawn picture), then we must first attempt to extract the curves from the image. There is an abundance of literature concerning the extraction of curves from images. The literature typically deals with several distinct processes: edge detection, curve grouping, and segmentation. The latter two (grouping and segmentation) refer to the process of extracting meaningful connected curves from data that may be confusing, cluttered or incomplete.

Curvature information is a key heuristic for building curves from noisy data. A standard approach in the presence of ambiguous data is to select the curve that minimizes a "goodness measure" based on minimum curvature, minimum absolute curvature or minimum variation in curvature. Such goodness measures can be posed as energy functionals, procedural rules, or decision trees. Work by Ullman and Sha'ashua [**94**] use locally connected networks to determine saliency for smoothness, continuity, and curve length. Similarly, Jacobs [**43**] develop a method for extracting curve segments based on a convex saliency measure. Earlier work by Lowe shows how a curve can be extracted by applying perceptually inspired grouping rules with properties such as proximity, collinearity and parallelism [**58**]. Estrada and Jepson [**23**] use predefined geometry-based affinity measures to evaluate the quality of line segment junctions. All of these approaches have proven to be very powerful, but they are based almost universally on an attempt to obtain generic domain-independent grouping strategies, typically using rules inspired by visual psychophysics [**48**].

Another approach is to use probabilistic methods in order to maintain the likelihoods of a set of possible solutions. These likelihoods are typically computed using both hard-coded conditional biases (such as a preference on curvature) and learned conditional biases (computed on the fly using representative exemplars). Taking this

approach, Williams and Jacobs [**100**] develop a method for contour extraction where a prior probability on the shape of a boundary is computed using paths of particles that undergo a random walk in the image. August and Zuker [**7**] describe the notion of *curve indicator fields* as generic models for producing edge likelihoods. In particular, their experiments employ a Markov random field model for contour enhancements. This thesis takes a similar approach where, using the HMM learning framework, probabilistic constraints are applied to rank the candidate pen strokes found in images. Unlike many of the previous methods, the presented method captures features over multiple scales using a wavelet representation.

While the problem of extracting curves from images has been a long standing research topic in the domain of computer vision, there is a variety of recent work that is more focused on sketches and employ similar image based principles for recognizing and grouping sketch components. For example, Saund [**72**] develops a method to rank candidate paths that form perceptually closed contours. The approach consists of applying local preferences on candidates with both tightly closed paths and smooth paths. In related work, Saund *et al.* [**73**] develop a sketch editing application that includes image analysis techniques for the separation of foreground from background and a method for finding and selecting "perceptually sound" grouping of sub-regions of a sketch. Images such as that shown in Fig. 2.4 can then be interacted with by simply selecting the desired elements of the image.

3. Motion Planning

The key problem in the sketch refinement consists of determining the appropriate methods for producing the preferred output while reducing the required amount of user intervention. Generating a sketch, which consist of producing curves while taking into account the user input and the desired type of output, is a similar problem to generating a path, where a robot must be driven using a trajectory that adheres to a set of constraints. Many of these constrains stem from either the mechanical configuration of a robot, a desired task-specific motion or the surrounding environment.

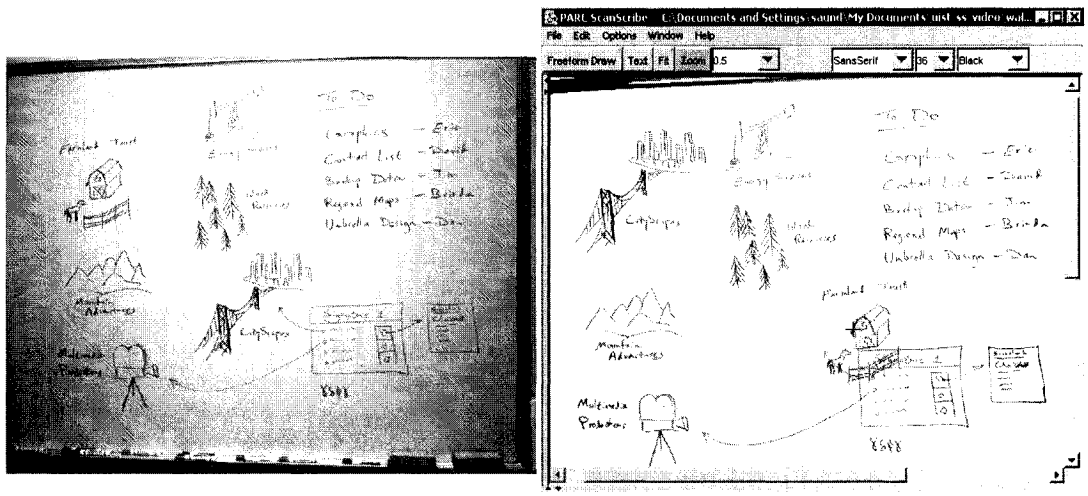


FIGURE 2.4. An example sketch editing action from [**73**]. Using the sketch editing application, elements of the sketch can be extracted, grouped and modified.

Path planning for a mobile robot has been extensively examined by many authors. One of the key ideas in the area is the notion of path planning under non-holonomic constraints, where the velocity of the robot $\dot{q}$ is constrained by its pose q:

$$G(q, \dot{q}) = 0$$

Specifically, path planning using a bound on the turning radius of the vehicle [**53**] is the subject of interest in this thesis. Notable work in the field includes that of Dubins [**20**] and Reeds and Shepp [**70**] on optimal trajectories. Much of this work deals with the quest for an optimal path (or trajectory) under a motion constraint which is expressed analytically (for example a derivative constraint). Prevalent solution techniques include analytic solutions (or expressions regarding their bounds), search methods that seek to optimize a path, and planners that start with a path of one form and seek to refine it.

In particular, a classic approach to the application of non-holonomic constraints is to find an (optimal) unconstrained solution and then apply recursive constrained path refinement to the sub-regions to achieve an admissible plan [**53**]. This is also

typical of probabilistic motion planning methods [**54**]. Similarly, jerky paths are sometimes smoothed using energy minimization methods [**82, 55**].

This thesis shares that common spirit in that the presented system takes an initial path as input and produces a refined path as its result. While traditional methods such as those cited above typically accomplish path refinement based on highly specialized constraints, typically in the domain of differential geometry, the method presented learns from examples of acceptable paths. That is, the desired constraints or preferences are indicated by showing the appropriate refinements that should be applied in specific cases.

This idea of learning to generalize specific examples to a broad ensemble of cases is, of course, the crux of classical machine learning [**63**]. Learning using Markov models is a longstanding classic research area, although, to our knowledge, it has never been applied to problems like this one. 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 [**97**] or on learning suitable cues that control or determine plan synthesis or execution [**22**].

Uncertainty introduced when executing commands in the real world is a foremost challenge in robot path planning and navigation. This is often addressed using closed loop processes and probabilistic models that maintain a distribution over system states [**89, 81**]. These distributions (or beliefs) can reflect the likelihoods of a robot's current pose, the existence of obstacles in the environment and the progress of a particular task. Planning long-term strategies is generally computationally intractable when considering all possible states. Approaches that deal with such issues include approximate methods that reduce the belief space [**10**], tree methods that exploit similarities in neighboring belief vectors [**65**] and temporal abstraction techniques (planning over higher level actions) [**85**].

3.1. Animation. Motion planning in virtual environments reduces the complexities that can occur when dealing with the uncertainties introduced in real world executions. However, the difficult challenge of controlling and synthesizing realistic

motions remains an open problem. People are naturally skilled at perceiving subtle anomalies of motion [**44**], exaggerating any artifacts produced when designing the animation sequence. Further, animation sequences of an articulated figure are typically made up of motions that exhibit many singularities in the velocity vector ($f(\dot{x}) \rightarrow \infty$), imposing on the designer the tedious task of carefully configuring the large number of required *key-frames*. (Key-frame animation [**84**] is a common approach in designing animations in which, akin to the parametric methods described in Section 1, an animation sequence is generated by interpolating over a selected set of control points.)

Example-based methods avoid these problems by providing a mechanism to reuse pre-fabricated libraries of motion clips, typically blending and transitioning motion snippets according to some procedural rules and matching functions. Witkin and Popovic [**102**] apply time-warping techniques that blend the motion signals from the training set with the specified key-frame points. In work by Wang *et al.* [**51**], each example motion clip is modeled by a Linear Dynamic System and transitions between these clips are moderated by a higher-level transition matrix. Their system controllably synthesizes novel motions by searching for the most locally consistent *mixture* of the clips in the database that results in a sequence that passes near the specified key-frames. A similar approach consists of automatically identifying good transition points in the clips and then explicitly storing them in a graph where the nodes represent the clips and the edges represent the allowable transitions [**49**]. Paths can then be controllably generated by searching the graph for motions that satisfy user defined criteria (such as "stay close to this path" or "go near these key-frames"), minimizing an error function. Hertzmann and Brand [**11**] develop a system to learn motion styles from a database of pre-classified families of motions (i.e. examples for ballet, modern dance, running etc.). Examples of each style are used to train a Hidden Markov Model. The set of Hidden Markov Models are then parametrized by a style parameter s to produce an all encompassing Stylistic Hidden Markov Model (SHMM). The user can specify the desired amount of each particular style (by setting weights

on the style parameter) and the system synthesizes a new motion that exhibits the desired styles (by selecting the maximum likelihood sequence of the SHMM states). Van de Panne *et al.* [**88**] develop a sketch based system for controlling an animated sequence. In their work, a hand-drawn path is segmented to recognizable primitive gestures that are associated with particular motions. The resulting compound motion consists of first extracting parameters from the recognized gesture segments (i.e. the start position, end-position, speed of motion, scale, etc.), then smoothly interpolating the associated motions while rescaling them according to the extracted parameters.

Signal processing approaches have also been used in example based motion editing. Unuma *et al.* [**95**] apply Fourier transforms to the signals produced from the sequence of joint angles of an articulated figure. Based on frequency analysis of the data, they capture global qualitative factors such as "brisk" or "tired". These factors are then used in linear combinations to interpolate and combine characteristics to create new motions. Bruderlin and Williams [**13**] similarly applied Gaussian and Laplacian filter pyramids and time-warping techniques over the example motion signals to provide equalizer-like tools for motion editing. Pullen and Bregler [**68**] use multi-band filters to match candidate database motion signals with sparse and incomplete key-frames (typical in a quick and dirty design process where the key-frames are far apart and each key-frame may be missing some parameter values). The match is further constrained by learning the correlation of joint angles, providing better solutions when key-frames have many missing joint angle specifications.

The main approach of these techniques is to take a few examples and build some knowledge base about the valid motions. Alternatively, this knowledge can consists of specialized constraints that attempt to simulate real physical laws. Witkin and Kass [**101**] propose a method for animation where a set of differential equations are used to describe the dynamics of real world physical factors such as gravity, friction, muscle forces, etc. An objective function that specifies how the motion should be performed is optimized subject to the constraints. Some physics-based methods [**91, 39**] dynamically vary the objective function in accordance to high-level behavior

models. In most cases, these methods, also known as space-time constraint methods, result in nonlinear differential equations that are often computationally intractable and extremely sensitive to initial conditions. Hybrid systems attempt to avoid this by combining key-frame techniques and nonlinear approximation methods [**33, 67**]. Typically, *intra-frame* constraints (constraints between joint angles) are modeled by using object kinematics and *inter-frame* constraints (constraints between frames) are models by an interpolating function. Lee and Shin [**56**] take this approach by using inverse kinematics and hierarchical B-splines to morph an existing motion clip to one that adheres to specified key-frames.

CHAPTER 3

Framework Overview

This chapter presents a brief overview of the learning framework, including the pen-stroke representation, the problem definition and the approach. A review of Markov Models, Wavelets and Hidden Markov Models is also presented, followed by an introduction to the Hierarchical Hidden Markov Model.

1. Pen Stroke Representation

The curve refinement system accepts as input a pen stroke for controlling a low-level synthesis process. The path of the pen stroke is represented by a curve over 2D space parametrized by the arc-length. (Though in principle the refinement system can also be applied to 3D curves.) Let the mapping $\alpha : \mathcal{R} \rightarrow \mathcal{R}^2$ represent a parametric planar curve $\{x(t_c), y(t_c)\}$ where $t_c \in \mathcal{R}$ is the arc-length of the curve over the range $0 <= t <= T$. The tangent angle along the curve can be computed by the following:

$$\theta_c(t_c) = tan^{-1}\frac{\partial y}{\partial x} \tag{3.1}$$

In this thesis, all curves are approximated using a discrete representation. A zeroth-order discrete representation of α can be produced by sampling the continuous curve using a uniform sampling resolution. The resulting points can then be used as vertices for a polyline. A first-order discrete representation of α thus consists of a starting point p_0, the sequence of all edge lengths $r(t)$ and the edge angles $\theta(t)$ of the

polyline, where $p_0 \in \mathcal{R}^2$, $r : \mathcal{Z}^* \rightarrow \mathcal{R}$, $\theta : \mathcal{Z}^* \rightarrow \mathcal{R}$ and $t \in \mathcal{Z}^*$ (i.e. an absolute chain code). A second-order discrete representation of α consists of the starting point p_0, starting direction $\theta_0 \in \mathcal{R}$ and the sequence of all edge lengths $r(t)$ and exterior angles $\Delta\theta(t)$ (i.e. a relative chain code). Depending on the application requirements, either a first-order or second-order representation can be used. When using the first-order representation, the system is invariant to the initial starting point and hence the shape can be reproduced over rigid orientation-preserving transforms. When using the second-order representation, the system is invariant to the initial starting point and direction and hence the shape can be reproduced over all rigid transforms. In both cases, all curves are sampled uniformly over the arc-length, $r(t) = r \in \mathcal{R}$.

In principle, these curves can be used to represent arbitrary signals. They represent not only the input pen stroke, but the examples in the training sets and the synthesized output as well. The curves can be generalized to functions that support application specific input and output attributes, such as pen-pressure, pen-speed, robot trajectory, a motor command or a curve's thickness, and can be extended to higher-dimensional spaces to simultaneously support multiple attributes (i.e. $\alpha : \mathcal{R} \rightarrow \mathcal{R}^m$).

2. Problem Definition

Let α denote a *refined curve*, the curve the user seeks to produce. Let β denote a *coarse curve*, the curve the user has actually drawn (the path of the pen-stroke). A set of refined curves is referred to as a *refined scene.* and set of coarse curves is referred to as a *coarse scene* The curve β can be thought of as the curve resulting from some lossy (possibly non-invertible) transformation of α:

$$\beta = F(\alpha) \tag{3.2}$$

The *refinement problem* is to reconstruct α given the noisy and coarse user input β. This is an *ill-posed* problem, where there is insufficient information to solve for a unique solution (i.e. the problem is under-constrained) [**90**]. Before it can be solved,

one must first define the way in which α can be inferred. That is, restoring the well-posedness of the problem requires a set of restrictions that limit the class of admissible solutions.

The underlying idea in this thesis is that the knowledge required to uniquely infer α can be acquired in two ways: by using pre-defined analytical functions engineered for a specialized domain and by using pre-classified examples that show the types of outputs the user intends to produce. The problem then becomes three-fold. How can the system learn from the examples the appropriate constraints? What are the analytical functions and how should they be combined with the learned priors? How can α be generated given β and this prior knowledge? Indeed, this is a classical inverse problem given *a priori* knowledge. Though unlike many of the existing approaches to this problem (such as variational regularization using a quadratic stabilizer), it cannot be assumed that the output is smooth, nor that the solution space is convex. In contrast, the system must reconstruct the high frequency features that are assumed missing in the input curve.

3. Approach to the Refinement Problem

The curve elaboration framework is based on a two-level hierarchy of Hidden Markov Models. As a supervised learning problem, the goal is to acquire a multi-level generative model that captures from data a *refinement function* for augmenting rudimentary hand-drawn curves. The first level of the hierarchy, called the curve-level, models the refinements that are applicable on individual hand-drawn curves. The second level of the hierarchy, called the scene-level, imposes constraints between the allowable types of curves that make up an entire scene.

A curve-level HMM is trained using a set of example curves that serve as exemplars of the kinds of curves the users wish to produce; there are typically several *sets* of such examples (e.g. fish, water, terrain, trajectories) and hence several models are trained. Each example in a set has a coarse curve associated to it that shows what the user would draw when their intention is to produce that particular example, i.e. the

pair $\{\alpha_i, \beta_i\}$ for $i = 1...N$ examples. Each β_i can be thought of as a user's short-hand notation (or glyph) for the elaborated shape α_i. Using one of these sets, a hand-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 is not necessarily identical to any single example in the set. A locally consistent mixture of curve segments refers the sequence of curve segments where the values of all neighboring samples are also found in the training set under the same sequential ordering. This is complicated by the need to account for additional user-defined analytical functions and for both fine-scale details as well as large scale motions of the curve. Learning and synthesis at the curve level is described in more detail in Chapter 4.

While the user can manually select which training set to use to refine the curve, the whole process is automated by classifying the curve being drawn as belonging to one of the sets. The scene-level of the hierarchy 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. These constraints are encoded in the form of a probabilistic transition diagram over the curve-level models. The scene-level of the hierarchy is discussed in more depth in Chapter 7.

4. Curve Classes

Each training set is made up of examples that are pre-classified in the same *family*. The notion of families of examples is not new and has been applied in various settings, including in work for stylized motion synthesis where examples are said to be admissible to the same family if they have "some generic data-generating mechanism in common" [**11**]. Similarly, in this thesis it is assumed that there exists some underlying generation process that results in a variety of distinct examples that have similar characteristics. It is up to the users to subjectively identify a family of examples based on the type of outputs they wish to produce. It is then up to the system to attempt to infer the underlying process that can produce new instances in that family.

(The terms a "family of curves" and a "class of curves" are used interchangeably in this thesis.)

There are two necessary but not sufficient conditions for identifying when curves are similar and hence belong in the same class:

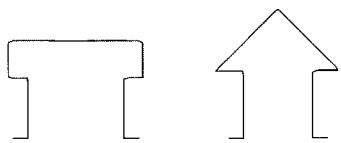
- All curves in a class have the same user defined *semantic annotation.*
- Any locally consistent mixture of curve segments in a class results in a new curve that can also be considered part of the same class.

The first condition states that examples can be in the same class only if they have the same label (which is manually specified). This semantic annotation is used to identify curves that are subject to the same high-level constraints. For example, leaves and flow-chart symbols are used in different contexts, have different applicable high-level constraints and should therefore be in separate classes. It may be the case that an example can have multiple labels. It is then duplicated, with each copy given a single label and assigned to the appropriate class. It may be possible to extract this automatically from labeled scenes, though this problem is outside the scope of this thesis.

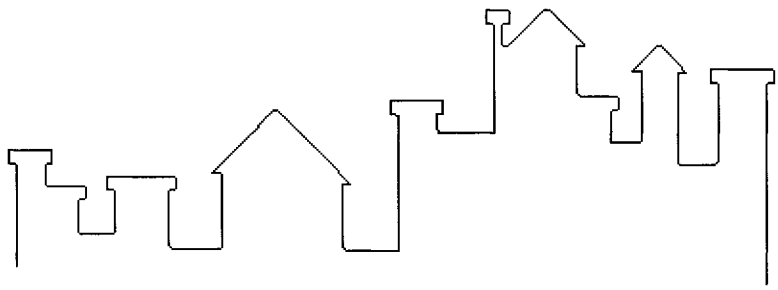
The second condition states that new curves having the same local shape as those in the set can also be considered to be part of the same set. (An assumption readily used in many of the recent state-of-the-art texture and curve synthesis methods [**98, 38**].) As such, the examples form a set of basis functions for the types of outputs that can be produced. Under this criterion, the richness of a set can be quantified using cross validation techniques where examples are first removed from the set, followed by an attempt to reconstruct them using the remaining examples.

As an example, consider the class of shapes comprising of roof-top segments. New roof-top shapes can be produced by taking locally consistent mixtures of segments from the original set. The degree of mixing, the location of mixing and the scale in which the consistency is enforced are all crucial parameters that must be considered in producing the desired output. Figure 3.1 shows a roof-top class with two members and the result of a manually controlled mixture of segments from the set (the new output

is also considered to belong in the same class). Each curve in this training set has very distinct features and contributes crucially to the shapes that can be produced. The statistic of this set would then result in a distribution with low entropy and by using cross-validation, it is easy to see that no example can produce the other.



(a) An example training set.



(b) A new curve made up of locally consistent mixtures of segments in the set.

FIGURE 3.1. Two examples classified in the same family. Each example is labeled as a *roof-top* and new locally consistent mixtures result in other examples that can also be considered as roof-tops.

5. Curve Synthesis using a Markov Model

The main idea behind the curve synthesis framework is to model local probabilistic constraints on the desired shapes in a given curve family. (This is further extended in Chapter 4 to include other curve attributes such as thickness or color.) It is assumed that a stochastic process Δ is the common curve generating source for a family of refined curves. This process generates a sequence of sample points where the value of the points represent the realized states of the process (i.e. the current state refers to the value of the current sample point). Each curve is thus considered to be a random signal with characteristics described by the probability density function of the process. Let α denote a curve and $\theta(t)$ denote the tangent angles of that

curve parametrized over the arc-length t (i.e. the first-order representation). It is assumed that the sequence of samples $\theta(t)$ from $0 <= t <= T$ for all curves exhibit an n^{th}-order Markov property, i.e. Δ is a Markov process:

$$p\{\theta(t+1) \mid \theta(t), \theta(t-1), \ldots, \theta(t-n+1)\} = p\{\theta(t+1) \mid \theta(t), \theta(t-1), \ldots, \theta(0)\} \quad (3.3)$$

This *locality* condition states that information from recent sample points is sufficient to compute the likelihood for the next candidate points. How far back in history does the model need to account for? This is dependent on the nature of the training set and the scale of the desired features of interest.

An n^{th}-order finite state Markov process Δ is defined by the finite state space $\mathcal{H}$ (the set of all values that the process can produce), the transition matrix M (which stores the likelihood of having a transition from any state in $\mathcal{H}$ to any other state in $\mathcal{H}$, as shown in Eq. 3.3) and an initial probability distribution π over the state space $\mathcal{H}$. There are several other factors that characterize the long term behavior of a Markov process:

Absorption: A state is said to be *absorbant* if the transition probability of leaving that state is zero.

Communication Class Structure: The states in any Markov process can be grouped together such that for any two states h_i and h_j that belong to the same *communicating class*, it is possible, starting from h_i to get to h_j and starting from h_j to get back to h_i.

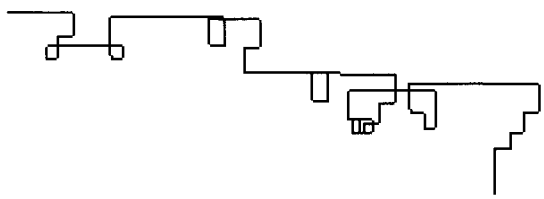
Periodic: A state h_i has period d if, given that the system at time zero is in state h_i ($X_0 = h_i$), the system can return to state h_i at time n ($X_n = h_i$) only when n is a multiple of d.

Irreducible: A Markov process is said to be *irreducible* if starting from any state it is possible to get to any other state (i.e. one communication class).

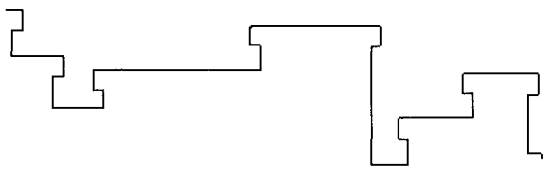
Ergodicity: If the process is irreducible and a-periodic, then it is said to be *ergodic* and guarantees a unique stationary distribution (i.e. there exists a unique eigenvector of M with an associated eigenvalue of 1).

The characterization of the process helps identify the types of outputs we should expect. For example, in texture mixing applications [**9**], a resultant process that has many recurrent (absorbant) communication classes may produce a synthesis that is not representative of the target texture. The procedure can get stuck in a small disjoint sub-region of the texture, generating pixels that do not express the entire texture. It is demonstrated later how this effect can be reduced by effectively blurring the transition matrix and providing a steering mechanism to bias the process.

Figure 3.2 shows two random realizations from two related Markov processes over curve elements; a first-order process and a higher-order process. The transition probabilities of the processes are computed by the statistics of consecutive tangent angles of an example in the roof-top training set; the left curve in Fig. 3.1(a). It is easy to see that large-scale structures are only captured under the higher-order assumption.



(a) Results from a first-order Markov Process.



(b) Results from a high-order Markov Process.

FIGURE 3.2. Realizations of two Markov processes trained using the left rooftop example in Fig. 3.1(a).

5.1. Non-Stationarity. For training sets that exhibit regular properties, uniform local constraints set at the appropriate scale are sufficient to capture the desired structures, which in turn can be realized at any point along the curve. However,

there are many interesting examples that contain transitory characteristics, where the probabilistic constraints on successive points are *non-stationary* along the curve. In such cases, the locality condition is position-variant and hence the transition matrix M becomes a function of the arc-length t; $M(t)$. This generalization provides more flexibility in the types of examples that can be synthesized, having the option to impose global constraints in cases where the absolute location of features is important. (For example, if the training set consists of examples of leaves, when one side of the leaf is being processed the constraints from the other side do not necessarily need to be considered.)

6. Multi-Scale Representation

As shown in Fig. 3.2, the order of the process has significant impact in the way the output is produced. For many applications, a high-order Markov assumption is crucial for a satisfactory synthesis, though the implementation can be somewhat problematic. The size of the transition matrix grows exponentially with the order of the process, becoming impractical to suitably store it under physical memory limitations. There are two observations that can be exploited in order to deal with this problem. First, the resulting matrix is generally sparse, where the number of non-zero entries is not larger than the number of sample points in the training set. One can then apply matrix compression techniques or exploit space-time tradeoffs and perform on-line computation of likelihoods (discussed later). Second, using the appropriate filter, one filtered sample point can be representative of several unfiltered sample points, providing a shorthand summary for the region. The latter approach is the subject of this section while the former is described in Chapter 4.

In order to efficiently capture the structure of a curve at various scales, a wavelet representation is used. Not only does this representation address the implementation issues described above, it also allows to control the scale at which constraints are enforced. (It may be desirable to produce mixtures of examples that are similar at a coarse scale, though differ significantly at the fine scale.) The fundamental idea in a

wavelet representation is that functions can be reconstructed by linear combinations of *basis functions*. Indeed, this idea lies at the heart of Fourier analysis, where any oscillatory function can be represented by a combination of sines and cosines. For a wavelet representation however, the basis functions must be functions of both time and scale. This complements the Markov Model as not only does it provide a spectral decomposition of a signal but also indicates where those spectral components exist, providing the ability to sequentially order points at multiple sub-bands.

There are many research areas that use wavelets, including topics such as speech, music, time-scale analysis and sampling theorems. The idea of multi-scale analysis is not new and has been explored as early as 1909 (Haar, 1909 [**35**]). Since then, similar ideas have been applied in Communications (Gabor, 1946 [**30**]) and Quantum Mechanics (Aslaksen and Klauder, 1969 [**6**]). Though only recently has a unifying theoretical formalism been reconciled, rooted by the works of various authors including Grossmann and Morlet [**34**], Marr and Hildreth [**61**], Meyer [**62**] and Mallat [**60**]. Wavelets are now commonly used in Computer Vision and Graphics and have been proven to be extremely useful in multi-resolution editing of curves [**26, 16**].

6.1. Continuous Wavelet Transform. The multi-scale representation $h(s, \tau)$ for a continuous signal $\theta(t)$ is a function of scale and time (or arc-length position) consisting of a convolution of the signal with wavelet basis functions:

$$h(s, \tau) = \int \theta(t) \Omega_{s,\tau}(t) dt \tag{3.4}$$

That is, the function $\theta(t)$ is decomposed by a set of basis functions $\Omega_{s,\tau}$ parametrized by scale s and translation τ. Figure 3.3 shows an example signal and its multi-scale representation.

The inverse wavelet transform is given by the following:

$$\theta(t) = \int \int h(s, \tau) \Omega^*_{s,\tau}(t) ds d\tau \tag{3.5}$$

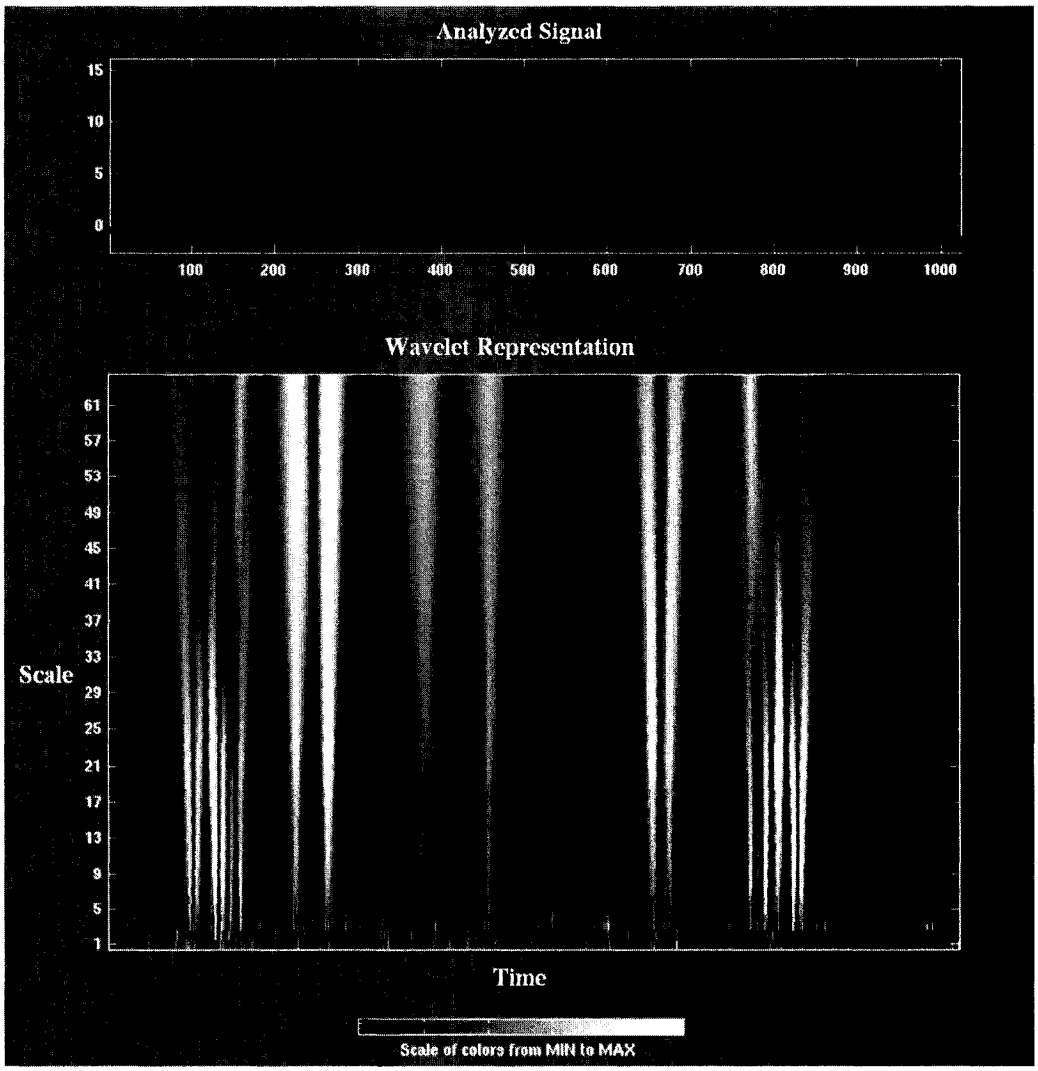


FIGURE 3.3. Wavelet representation of a signal.

where $\Omega^*_{s,\tau}$ is the complex conjugate of $\Omega_{s,\tau}$ such that:

$$\langle \Omega_{s,\tau} | \Omega^*_{s',\tau'} \rangle = \int \Omega_{s,\tau}(t) \Omega^*_{s',\tau'}(t) dt = 1 \tag{3.6}$$

when $s = s'$ and $\tau = \tau'$. The basis functions are said to be orthogonal if the integral in Eq. 3.6 evaluates to zero whenever $s \neq s'$ and $\tau \neq \tau'$.

6.2. Properties of Wavelet Basis Functions. For a candidate function to formally be considered as a wavelet basis, it must adhere to the *admissibility* and

regularity conditions. The admissibility condition implies that the function must be oscillatory (with band-pass like spectrum, the integral of the function over all time must be zero and the square integral of the Fourier transform of the function, divided by the frequency, over all frequencies must be finite) while the regularity condition implies the function should be compactly supported (the low-order moments must vanish). Different wavelet families have different tradeoffs over how compactly the basis functions are localized in space and how smooth they are.

A prototype function, also known as the mother wavelet $\Omega(t)$, is first developed to satisfy the above criteria, then the wavelet basis functions are translated and dilated variations of the mother wavelet. Computing the transform over every scale s and translation τ is an overly redundant and computationally intensive process. Instead discrete steps are taken where the basis at one scale is dilated typically by a power of 2 of the previous:

$$\Omega_{s,\tau}(t) = 2^{\frac{-s}{2}}\Omega(2^{-s}t - \tau) \tag{3.7}$$

where s and τ are integers. The term $2^{\frac{-s}{2}}$ is used for energy normalization.

6.3. Haar Basis. Due to its simplicity and efficiency, a *Haar* wavelet (rectangular function) is used for producing the multi-scale representation for the input and training set curves:

$$\Omega(t) = \begin{cases} 1 & 0 < t \leq \frac{1}{2} \\ -1 & \frac{1}{2} < t \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

The traditional Haar basis consists of the Haar mother wavelet with a dilation factor of 2^s and a translation factor of $2^{-s}\tau$. Figure 3.4 shows the shape of the Haar wavelet with examples of translation and scaling.

In this work, because orthogonality is not a pressing factor for the application of interest, the basis functions can be scaled and translated by any user-defined step. There are several reasons for allowing such flexibility. Primarily, the use of wavelets in the HMM framework is not for compression encoding and reconstruction purposes,

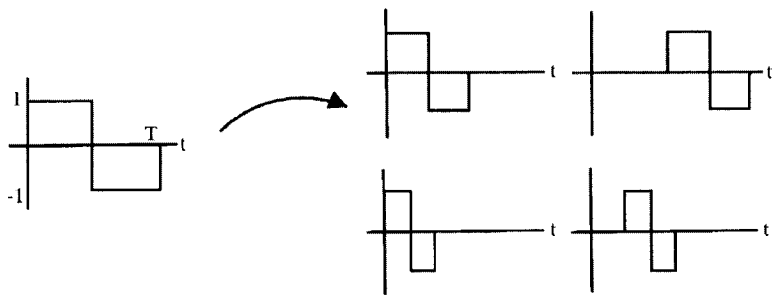


FIGURE 3.4. Haar wavelet with translation and scaling.

but rather for enforcing large-scale constraints on the allowable sequences of points in the smaller scale. (The term large scale refers to the coarser representation and the term small scale refers to the finer representation.) In fact, the multi-scale curve representation always explicitly stores the original curve sample points and the sample points in the higher-scale are only used for computing probabilities. Second, even if the points were not stored explicitly and reconstruction was used, in a discrete representation, there can be loss of information due to quantization errors and, unless compression is an important factor, redundancy is sometimes preferred. The potential for such redundancy can then reduce quantization artifacts that may occur when reconstructing from a finite set of rectangular functions, thus a smoother wavelet is not necessarily required.

6.4. Sub-band Coding. Sub-band coding is a method that can be used for performing a discrete wavelet transform. The main idea is to decompose a discrete signal using a filter-bank with filter kernels that are developed based on the desired wavelet basis function. A signal is repeatedly filtered and subsampled using both a low-pass filter and a high-pass filter. For the Haar basis, the decomposition kernels are:

$$L = \left[\frac{1}{\sqrt{2}} \quad \frac{1}{\sqrt{2}} \right] \tag{3.8}$$

$$H = \left[-\frac{1}{\sqrt{2}} \quad \frac{1}{\sqrt{2}} \right] \tag{3.9}$$

where L is the low-pass filter kernel and H is the high-pass filter kernel. The result is a pyramid of signals that are filtered at multiple frequency bands. Half of these signals maintain only the high-frequency components (the details) and the other half maintain only the low-frequency components(the coarse variations).

The multi-scale representation for the training set and input curves is produced using the sub-band coding approach with two slight variations. First, the original sample points are always maintained (along with the low-pass filtered sample points) and the High-pass filter is not applied (it is mainly useful for reconstruction purposes). Second, since the translation step τ is allowed to be any user-defined value (i.e. training set mixing can potentially occur on a sample point basis), subsampling may not always be performed. The resulting redundancy has the benefit of having a one-to-one correlation between samples of the original signal and samples of the filtered signal, simplifying the registration of sample points at different scales. A signal reduction issue arises when dealing with curves that are not closed. In such cases, the endpoints are padded using the value of the neighboring sample point.

7. Refinement of Curves using Hidden Markov Model

The Markov process in combination with the wavelet representation provides an efficient mechanism to produce a random realization of a curve with a high-order Markov assumption. However, the user has no control in the way a curve is generated. In order to provide a method for the user to bias the synthesis, the framework must take into account the likelihood that a user would draw a particular curve given that their intent is to produce another. This is accomplished by 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. In this work, the states in the observation layer represent the samples that are expected to

be drawn by the user and the states in the hidden layer represent the samples that the user actually meant to draw. Formally, a Hidden Markov Model Λ is defined as follows:

$$\Lambda = \{M, B, \pi, \mathcal{H}, \mathcal{O}\} \tag{3.10}$$

where M is the transition matrix with transition probabilities for 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. The set $\mathcal{H} = \{h_0, ..., h_n\}$ is the set of all hidden states and the set $\mathcal{O} = \{o_0, ..., o_m\}$ is the set of all observation states.

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

- **Learning**: Given an observed set of examples, what model Λ best represents that observed set?
- **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?
- **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?

Solutions to the above three problems are key to this work. Learning provides an automated method for modeling various types of outputs or drawing habits by simply providing the examples. Decoding allows for the synthesis of a new curve (sequence of hidden states) based on a coarse user input (sequence of observation). Evaluation is used to detect the appropriate class of curves that an input stroke belongs to by computing the likelihood that the input curve would be generated by the model in question.

8. Two-Level Hierarchical Hidden Markov Model

At the first level of the hierarchy (curve-level), the characteristics of training sets are expressed probabilistically 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. 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 (Fig. 3.5). We construct the set $\mathcal{G}^0$ consisting of N HMMs where each HMM is trained using a particular training ensemble:

$$\mathcal{G}^0 = \{\Lambda_1^0, \Lambda_1^0, ..., \Lambda_N^0\} \tag{3.11}$$

For example, Λ_1^0 may represent the set for terrains and Λ_2^0 may represent the set for roof-tops. Each Λ_n^0 in $\mathcal{G}^0$ is the trainable part of the *curve refinement model* and is further augmented in Chapter 4 to include additional constraints.

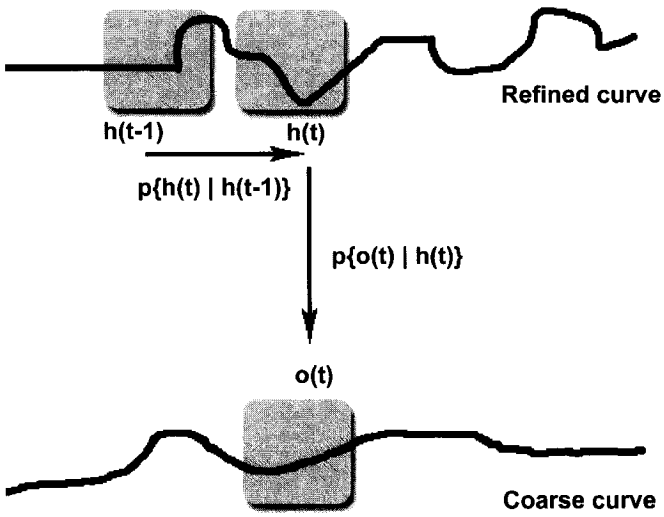


FIGURE 3.5. Local relationships of refined and coarse curves. For all curve segments (in gray), the transition matrix and confusion matrix store the above likelihoods. This is computed over every example in a given set.

At the second level (scene-level), another HMM is used to model high-level probabilistic constraints on the application of models at the first level. This is only used for the sketching application where relationships on the types of curve that can be

drawn are critical. The model allows us to represent restrictions on the types of refinements that are applicable on neighboring curves. For example, one can suggest that the cloud model in $\mathcal{G}^0$ 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 $\mathcal{G}^0$ and their relative positions. While in principle such constraints can be learned from labeled illustrations, for the purpose of this thesis, they are manually encoded in the form of a graph (Fig. 3.6).

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

$$\mathcal{G}^1 = \{\Lambda_1^1, \Lambda_2^1, ..., \Lambda_M^1\} \tag{3.12}$$

Each model in $\mathcal{G}^1$ 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 constraints on the order in which curves are drawn are not desired, a graph can suggest that every model can be followed by any other model, with only their relative positioning as a constraint. Each Λ_m^1 described thus far is the trainable part of the *scene refinement model* and is further augmented in Chapter 7 to include additional constraints.

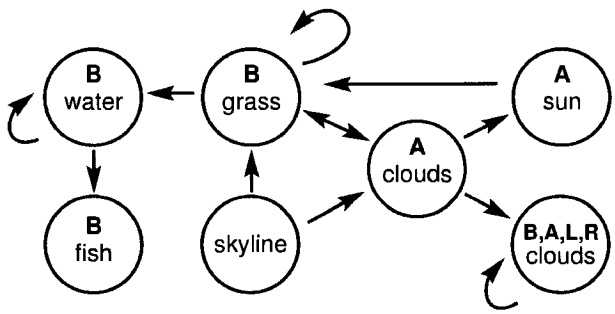


FIGURE 3.6. Example relationships in a scene. The labels correspond to HMMs at the curve-level and the letters above correspond to the allowable relative position (i.e. A: above, B: below, L: left R: right).

CHAPTER 4

Curve Refinements

Each curve-level HMM in the hierarchical HMM framework is a model that captures from data the constraints that are required to synthesize a refined curve from a coarse one. These HMMs, in combination with supplementary user-defined functions, are called *curve refinement models.* This chapter describes the methods for learning and applying these refinement models, including a description of the models' state space, transition and confusion matrices and the decoding algorithm. Issues regarding the practical and efficient implementation of these models are also addressed. Once the formalism for learning and decoding is established, a regularization framework is presented that provides for the seamless integration of supplementary user-defined analytical biases. The pseudo-code for the algorithms presented can be found in Appendix A.

1. Refinement Model Overview

A curve refinement model Λ^0 is an augmented HMM that includes supplementary analytical biases and parameters. These additional components further define the way in which the model is trained and applied. More formally, a refinement model is defined as follows:

$$\Lambda^0 = \{\Lambda, R, \mathcal{Q}, \mathcal{S}, \mathcal{A}, \mathcal{A}_o, \mathcal{X}, \mathcal{X}_o, \tau, \omega, \Upsilon, \Upsilon_o, \mathcal{T}, \mathcal{T}_o\} \tag{4.1}$$

The first parameter is the HMM $\Lambda = \{M, B, \pi, \mathcal{H}, \mathcal{O}\}$ as described in Eq. 3.10. The parameter $R = \{R_1, R_2, ..., R_n, \lambda_1, \lambda_2, ..., \lambda_n\}$ is a set of regularization functions and weights that embed additional biases into the model. The parameter $\mathcal{Q} = \{Q, \mathcal{H}', \mathcal{O}'\}$ consists of a quantization function Q and a set of labels $\mathcal{H}'$ and $\mathcal{O}'$ used to label the states in $\{\mathcal{H}, \mathcal{O}\}$ such that likelihoods can be expressed in the finite matrices M and B. The set $\mathcal{S} = \{s \mid s \in \mathcal{Z}^*\}$ is the set of scales that the states must encode. Each element identifies the number of times the Haar filter must be applied on the training curves. The sets $\mathcal{A}$ and $\mathcal{A}_o$ identify the curve attributes encoded in the hidden states and observation states respectively. The sets $\mathcal{X}$ and $\mathcal{X}_o$ are the set of auxiliary curve attributes that must be represented in the hidden and observation states respectively. The value τ is the translation step for the wavelet representation (this value is also used to determine the number of samples a state can encode). The value ω is the *stationarity window*, used to determine if the model is stationary or non-stationary. The parameters Υ and Υ_o are the distance metrics and associated parameters used for determining the similarity of hidden and observation states respectively. Finally, $\mathcal{T}$ and $\mathcal{T}_o$ is the mutli-scale representation of the refined and coarse curves from training where $\mathcal{T}_s$ and $\mathcal{T}_{o,s}$ are the set of training curves at scale s. The remaining sections of this chapter describe these parameters in more detail.

2. Learning Refinement Models

A refinement model Λ^0 is trained over a family of examples consisting of a set of *refined curves* $\mathcal{T}_0 = \{\alpha_1, \alpha_2, ...\alpha_n\}$ associated with a set of *control curves* $\mathcal{T}_{o,0} = \{\beta_1, \beta_2, ..., \beta_n\}$ (Fig. 4.1) where each curve is represented by T sample points. A refined curve depicts the desired solution that should be produced if a user sketches its associated control curve. If a user sketches a curve unlike any single one of the control curves, then the system must infer the desired output using segments from various refined curves. The control curve can be any user defined curve (for custom drawing habits) or it can be automatically generated by filtering the associated refined curve. It may be the case that segments from different control curves are identical,

though it is undesirable to have control curves that are everywhere similar. Control curves without any distinguishing features result in an ambiguous training set which increases the likelihood of non-unique solutions. It is assumed that each example $\{\alpha_i, \beta_i\}$ is a suitably normalized tuple such that the associated sample-points on the two curves are already in correspondence.

When only the observations are available, learning can be performed by applying algorithms such as the Baum-Welch algorithm or other Expectation-Maximization methods [**69**], using criteria such as maximum likelihood (ML) or maximum mutual information (MMI). In this work, a supervised learning paradigm is taken where the data for both the observation and hidden layers is explicitly provided by the user (during training, it is assumed that the values corresponding to the hidden states are directly available while during synthesis, they are not). The parameters of a HMM can therefore be estimated using the statistics of the training data, calculating probabilities of successive sample points along the refined curves and the probabilities that they generate the corresponding sample points along the control curves (discussed more in detail below). These points are represented by the states in the model. In the remainder of this thesis, the term *successive states* is used to refer to a sequence of states that represent a sequence of sample points from a curve.

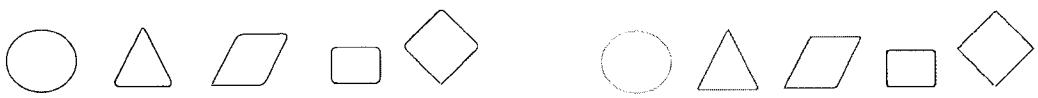


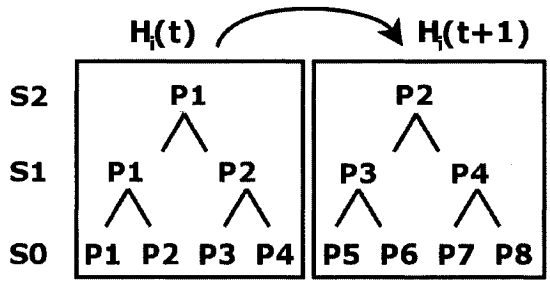
FIGURE 4.1. Examples from a training set. Curves on the left show the control curves while curves on the right show the associated refined ones that include color. Typically, the shape of the control curves are filtered versions of the refined ones, in this case the filtered ones are very similar to the originals. The set is sampled uniformly with 128 samples per example for a maximum of 1024 unique points.

2.1. Hidden States. The HMMs operate over a multi-scale curve description in order to capture long-range constraints on curves. Each hidden state encodes a

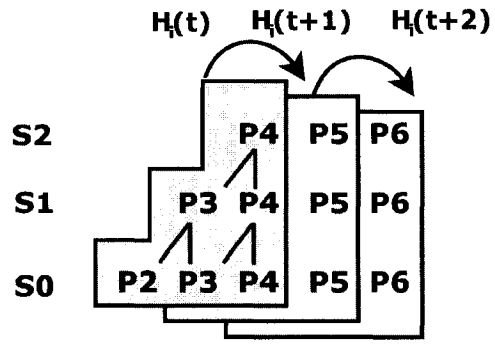
curve segment at multiple scales. A multi-dimensional state space $\mathcal{H}$ is used for the hidden states $\mathcal{H} = \{h_i \mid h_i \in \mathcal{R}^{|\mathcal{A}\times\mathcal{S}|}\}$ where $\mathcal{S} = \{s_0, ..., s_n\}$ is the set of scales and $\mathcal{A} = \{a_0, ..., a_m\}$ is the set of curve attributes. Let the function $H_i(t)$ correspond to a state h_i at position t in the sequence and let the function $H_i(t)_{a,s}$ correspond to the value of the state indexed by a particular attribute a at a particular scale s. In general, the function $H_i(t)_{\mathcal{W}}$ is the projection of a state onto a subspace that excludes the dimensions corresponding to elements in the set $\mathcal{W}$ by fixing their values.

Depending on the application, different scales and curve attributes may be required (e.g. for the sketching application, the attributes can include a curve's shape, color, thickness, etc.). While in principle all of the selected curve attributes can be represented at multiple scales, for the applications of interest, it is typically sufficient to only encode the shape at multiple scales ($\theta(s,t)$ where $s \in \mathcal{S}$) as this provides adequate multi-scale constraints for the desired output. (The likelihood of having similar segments decays exponentially with the segment length, hence one attribute can be sufficient to distinguish long segments.) As discussed in Section 6 of Chapter 3, a wavelet representation is used for the multi-scale curve descriptor.

2.2. Translation Step. The translation step defines the step size that is taken along the curves when encoding successive states and hence it specifies the size of curve segments (the granularity of the states). The size of the state space is then increased according to the number of samples in the segment. It is also used when constructing the multi-scale representation (as described in section 6). The translation step has a significant impact on the speed of the system, the ability to steer the synthesis procedure and the redundancy of the multi-scale representation (Figure 4.2). Setting a small translation step results in a slower and more redundant system but has the advantage of finer control resolution (prediction can be performed on a sample-point basis rather than on a segment basis). Setting a large translation step results in a faster system that produces sequences of large segment blocks at the cost of losing the ability to mix sample points within those segments.



(a) A state representation using three scales (with down-sampling) and a translation step of four sample points.



(b) A state representation using three scales (no down-sampling) and a translation step of one sample point.

FIGURE 4.2. Comparing the states for two different translation steps. In figure 4.2(a), the translation step at scale S0 is set to four sample points. Prediction and multi-scale decomposition is thus performed on a four-point segment basis. In figure 4.2(b), the translation step at scale S0 is set to one sample points. Prediction and multi-scale decomposition is thus performed on a sample point basis. (For opened curves, sample point padding is applied at the curve's end-points.) The shaded boxes show the information that each state encodes. Note that even when the translation step is set to one (the segment length is thus one), the state encodes information beyond the segment.

When computing the multi-scale representation for the curves (Chapter 3, Section 6), given the translation step for one scale, the translation step for the next scale is computed as follows:

$$\tau_{s+1} = max(1, \lfloor \tau_s / |K| \rfloor) \tag{4.2}$$

where $\lfloor \cdot \rfloor$ is the integer part of the argument (the *floor* function) and $|K|$ is the size of the filter kernel. The specified parameter τ is used to set the lowest scale translation step τ_0:

$$\tau_0 = min(|K|, \tau) \tag{4.3}$$

The degree of overlap between successive states is determined by both the set of scales used and the translation step. It is easy to see from Equation 4.2 and 4.3 that when the largest scale in the representation is sufficiently small (when $s_{max} \leq log_{|K|}(\tau_0)$) then all the information contained in a state pertains to that state only and there are no redundancies over neighboring states. (A point at scale s represents $|K|^s$ raw samples, hence the transition step must be at least $|K|^s$ in order to have mutually exclusive neighbors.) Otherwise, the state contains information that is already encoded in previous states, a typical (and acceptable) artifact of an N^{th}-order Markov assumption, where the transition probabilities applied to the current point have references to the $N-1$ points that have already been referred to in the transition probability applied to the previous point (this redundancy is illustrated in Fig. 4.2(b)).

In conjunction with the translation step, the set $\mathcal{S}$ determines the degree that a state encodes the curve's history. The elements of the set can span a contiguous set of scales (e.g. scales $\{0, 1, 2, 3, ..., s_{max}\}$) or a non-contiguous set (e.g. scales $\{0, 3\}$). The set must always include the zeroth scale such that all states encode the original sample points. (While points in the zeroth scale might not actually be used in probability estimation, they are required to realize the end result.) In Figure 4.2(a), the point $P1$ at scale $S2$ represents a summary for the four points $\{P1, P2, P3, P4\}$ at scale $S0$. In figure 4.2(b), the point $P4$ at scale $S2$ represents a summary for the three points $\{P2, P3, P4\}$ at scale $S0$. While the largest-scale sample point alone can be used to represent the entire curve segment, in general, points at any combination of scales can be used for computing the probabilities. A set of weights are assigned to each scale to emphasize its relative importance (used in Eq. 4.4 as part of the parameter Υ and discussed further below).

2.3. Observations. The state space $\mathcal{O}$ for the observations consist of a multi-dimensional space for capturing multiple control modalities at various scales; $\mathcal{O} = \{o_i \mid o_i \in \mathcal{R}^{|A_o \times S|}\}$ where A_o is the set of control attributes and S is the set of scales. Let the function $O_i(t)$ correspond to a state o_i at position t in the sequence and let the function $O_i(t)_{a,s}$ correspond to the value of the state indexed by particular attribute a at a particular scale s. In general, the function $O_i(t)_{\mathcal{W}}$ is the projection of a state onto a subspace that excludes the dimensions corresponding to elements in the set $\mathcal{W}$ by fixing their values.

The control attributes define the types of inputs that are expected from the user to steer the synthesis procedure. They are selected based on the application (e.g. for the robot path planning application, the desired path and robot facing direction can both be controlling components). By default, the value for the translation step and the set of scales for the observation states are set to be the same as those in the hidden states. The scales may be reconfigured empirically to customize the importance of history in the control layer, which can differ from the hidden layer. The translation step however must always have same value for both the hidden and observation states as this synchronizes the input segment length with the output segment length.

2.4. Auxiliary Attributes. In addition to the attributes in $\mathcal{A}$ and $\mathcal{A}_o$, the state space is augmented to accommodate for *auxiliary attributes* $\mathcal{X}$ and $\mathcal{X}_o$. The dimensionality of the state space the becomes $|\mathcal{A} \times \mathcal{S} + \mathcal{X}|$ for the hidden states and $|\mathcal{A}_o \times \mathcal{S} + \mathcal{X}_o|$ for the observation states. Auxiliary attributes are used to maintain supplementary information that may be required for additional processing of the states. There are two types of auxiliary attributes: *training set auxiliary attributes* and *decoding auxiliary attributes.* Training set auxiliary attributes are curves attributes that capture meta-information regarding the training data. Decoding auxiliary attributes are curve attributes that are not available during training but are captured as required when solving for the best hidden state sequence. These attributes can exist in both the observation states and the hidden states. In the hidden states, the auxiliary attributes are not bound to local consistency requirements and hence do not

enforce sequential constraints on successive sample points. In the observation states, the auxiliary attributes are not included as part of the generative model and hence do not influence the confusion matrix.

One example of an auxiliary attribute is the translation step. Since knowledge of the translation step is required for the eventual realization of the output curve and it cannot be assumed that every state has the same translation step (τ may not be an integral deviser of the number of sample points T), then the translation step is stored in the states as an auxiliary attribute. Additionally, the sample points at the zeroth scale can also be considered as auxiliary attributes when their importance weights are set to zero. (While these points are not explicitly used in constraining the sequence, they are required later to realize the output curve.) Other auxiliary attributes are discussed further throughout the remaining sections of this thesis as they pertain to the application.

2.5. Transition Probabilities. The transition probabilities $p\{H_i(t) \mid H_j(t-1)\}$ for all states in the training set are estimated from the statistics of successive sample points, counting the occurrence of successive states for each matching previous state. Rather than searching for exact matches, the transition probabilities are estimated by evaluating the *goodness* of a match. The probability that two states match is determined using a proximity function that evaluates the distance between states. A Gaussian distance metric G is used and the parameter Υ is defined by $\{G, \Delta^2, w, \sigma\}$ and used as follows:

$$p\{H_i(t)|H_j(t)\} = e^{-\Delta^2\left(H_i(t),H_j(t)\right)} \tag{4.4}$$

where

$$\Delta^2\big(H_i(t), H_j(t)\big) = \frac{\sum_{a\in\mathcal{A}}\sum_{s\in\mathcal{S}}\sum_{\tau'=0..\tau_s} w(s,a)(H_i(t)_{s,a,\tau'} - H_j(t)_{s,a,\tau'})^2}{\sum_{a\in\mathcal{A}}\sum_{s\in\mathcal{S}} w(s,a)\sigma^2(a)\tau_s} \tag{4.5}$$

and where $w(s,a)$ is the associated importance weight and $\sigma^2(a)$ is the attribute's mixing variance. This Gaussian blur is applied on the difference of two curve segments encoded in the states as a weighted sum over the specified scales and constraining

attributes, excluding all auxiliary attributes. (When dealing with the angles, the difference is always computed over two the interval $[-\pi, \pi]$). This effectively blurs the non-zero elements in the transition matrix, avoiding synthesis issues that can occur due to quantization errors or disjoint communication classes while also providing some degree of control and flexibility over the *mixing tendency*. The tendency to mix curve segments is determined by the value of the variance $\sigma^2(a)$. A small variance reduces the mixing tendency such that the output will be more similar to exact instances of the training set while a large variance allows the synthesis process to transition between states more easily at the cost of losing some local consistency with the training set.

For most of the experiments, a large weight is used for the shape attribute at a high scale and the variance is set empirically for each training set. For efficient implementation, to avoid iterating over zero values weights, only non-zero valued weights are stored in a vector where each element in the vector indexes the state location of the target scale and attribute that it applies on.

2.6. Transition Matrix. In a finite state space HMM, the transition probabilities for all states are stored in the matrix M where the $(i, j)^{th}$ element corresponds to the transition between state i and state j. This becomes problematic in continuous state spaces where there is an uncountably infinite number of states. One approach to address this issue is to quantize the continuous space using well defined boundaries, though this alone can lead to implementation issues regarding the size of the matrix. Even with the compact multi-scale representation, the size of the transition matrix grows exponentially with the number of attributes, irrespective of the number of non-zero probability transition, which may be relatively few in number. (Note that while M may become large, the total amount of training data having been provided by the user is probably of limited size.) The approach taken to avoid this issue is to dynamically label regions of the state space that contain states that exist in the training set and then use this label to index the matrix. This compacts what is otherwise a large and sparse matrix.

The estimated transition probabilities are stored in a matrix M that is indexed by a predefined state identifier. Let the finite set $\mathcal{H}' = \{L \mid L \in \mathcal{Z}^*\}$ be the quantized state space where each L in $\mathcal{H}'$ is a label for states in $\mathcal{H}$. Let the mapping $Q : \mathcal{H} \rightarrow \mathcal{H}'$ be the quantization function that assigns labels in $\mathcal{H}'$ to states in $\mathcal{H}$ (i.e. $\mathcal{R}^{|A \times S|} \rightarrow \mathcal{Z}^*$). Each state h_i that exists in the training set is labeled dynamically using Q and the transition matrix M is indexed using the labels in $\mathcal{H}'$. The value of each matrix element M_{ij} is then $p\{Q^{-1}(i) \mid Q^{-1}(j)\}$ where Q^{-1} is the inverse mapping of Q, computing a state in $\mathcal{H}$ that is representative of the label in $\mathcal{H}'$ (i.e. $Q^{-1}(k) = \overline{h}_k$). In this fashion, M has only non-zero entries and the size of M is proportional to square of the number of uniquely labeled hidden states in the training set; M is thus expressed using only an $|\mathcal{H}'| \times |\mathcal{H}'|$ matrix. In order to have random access to a state's label L, all candidate states include the label as an auxiliary parameter.

The quantization function Q is computed by evaluating Eq. 4.5 on each attribute separately. Those states that have all attribute similarity measures falling below the specified attribute thresholds are labeled with the same value L; i.e. if $p\{H_i(t)_a = H_j(t)_a\} < a_{tresh}$ for all attributes a then $Q(h_i) = Q(h_j)$. Unlike in the computation for the transition probabilities, this computation also includes the auxiliary attributes as they contain state information that must be preserved and hence take part in distinguishing states: $a \in \mathcal{A} \cup \mathcal{X}$. (To avoid over-blurring, the value of a_{thresh} should be relatively small.)

A new *centroid state*, representative of the label, is computed by averaging the attribute values over all states that have the same label: $Q^{-1}(i) = \overline{h}_i$ where $\overline{h}_i$ is the centroid state. The average is computed by a uniformly weighted sum of the attributes (angles are handled as a special case when their values span the first and fourth Cartesian quadrant). Computing the centroid state in this fashion can result in a drift where the distance of some samples to the centroid can increase beyond the error bound. However, this effect is minimal when the error bound is small, as the maximum drift caused by a sample point is $\frac{a_{thresh}}{n}$ where n is the number of samples associated with the label.

2.7. Stationarity Window. A stationary model is one where the likelihoods in the transition matrix (and confusion matrix) are not a function of position; $M(0) = M(1) = ... = M(T) = M$. Any locally consistent mixture of curve segments can occur at any point along the curve, with only the input and local history as constraining factors. In contrast, a non-stationary model may have a distribution that is a function of the position. In cases where a stationary model is not suitable (when the features of curves do not repeat consistently over the entirety of the curves), the transition matrix must be calculated over a predefined *stationarity window* ω. This window identifies the size of local regions that exhibit regular properties (which can be as small as one sample point).

Providing the option to specify a local stationarity window (hence global non-stationarity) accommodates sets that inherently exhibit position-dependent features. As an example, consider drawing the outline of a mountain. Initially, the edge should be colored brown or green and later it should be colored white or gray (simulating the snowy look in higher altitudes). Even though the underlying shape may not change, the output varies according to the position along the curve. Non-stationarity also helps preserve proportionality and sizing constraints over large scales by enforcing the sequential progression of sample points more strictly. The underlining premise in non-stationarity is that the characteristics of curves become functions of the position along the curve, hence the arc-length position itself becomes one of the constraining elements in the system.

The desired rate at which the transition matrix is permitted to vary is specified by the manually tuned stationarity window parameter ω. The stationarity window can be specified within the range $[1, T]$ as a multiple of the transition step τ. The transition matrix $M(t)$ is then computed using the statistics of the sample points in the training set that lie between a lower bound l and an upper bound u centered about t:

$$l = max(0, t - \lfloor \omega/2 \rfloor)$$
$$u = min(T, t + \lceil \omega/2 \rceil)$$

for open curves and

$$l = min^{+}(T - \lfloor \omega/2 \rfloor, t - \lfloor \omega/2 \rfloor)$$
$$u = (t + \lceil \omega/2 \rceil) \; MOD \; T$$

for closed curves, where ω is the stationarity window. The function $min^{+}(\cdot)$ returns the smallest positive number and $\lceil \cdot \rceil$ rounds up the argument (the *ceiling* function).

2.8. Confusion Matrix. The confusion matrix B stores the likelihood of observing a curve segment when the intent is to produce its associated refined segment. Given the set of tuples $\{\alpha_i, \beta_i\}$, the likelihoods $p\{O_j(t) \mid H_i(t)\}$ for all states in the training set can be estimated from the statistics of the set. The number of matching coupled states $\{h_i, o_j\}$ is computed by searching for exact matches over all examples within the stationarity window. Similar to the transition matrix, the confusion matrix is also index by precomputed state labels. The observation states are assigned labels from a finite state space $\mathcal{O}' = \{L \mid L \in \mathcal{Z}^*\}$ using the quantization function $Q : \mathcal{O} \rightarrow \mathcal{O}'$ from Eq. 4.5. The size of the matrix B is then $|\mathcal{H}'| \times |\mathcal{O}'|$.

Indeed this matrix only encodes the expected observations from a limited set of examples, which may or may not be exactly as drawn by the user. This poses a problem when the user draws a curve segment that has not been anticipated in the training set. One approach to address this issue is to provide a large confusion matrix with all possible inputs, train it under the training set and then and blur it to avoid non-zero likelihoods. However, attempting to anticipate in advance every possible curve segment that the a user can draw is inefficient and impractical with current memory limitations. Therefore, this issue must be addressed dynamically when decoding the model with the input, taking into account the possibility that the training set does not anticipate every input and must be used as an approximation to what the user is expected to draw.

2.9. Initial Distribution. To complete the configuration of the HMM, an initial distribution π for the hidden states must be specified. A uniform initial probability distribution is assumed. That is, before anything is drawn, all curve candidates have an equal likelihood of being synthesized.

3. Decoding Refinement Models

Once the model is trained, it can be used to refine a coarse input curve. Given an input curve and a HMM Λ^0 trained over a family of curves, a new refined curve is synthesized by solving for the maximum likelihood hidden state sequence:

$$\max_{H(0)\ldots H(T_o)} p\{H(0), H(1), \ldots, H(T_o) \mid O_{in}(0), O_{in}(1), \ldots, O_{in}(T_o), \Lambda^0\} \tag{4.6}$$

That is, the most likely sequence of refined curve segments (represented by the hidden states) is reconstructed using the sample points from the input curve (represented by the sequence of observation states). One approach to solve this problem consists of examining all possible sequences, computing the likelihood for each candidate sequence and then choosing the one with the maximum likelihood. This results in an overly redundant system where the same computations are repeatedly performed for the same candidate sub-sequence. Such an approach leads to a runtime complexity of $O(N^{T_o})$ where N is the number of hidden states ($|\mathcal{H}'|$) and T_o is the length of the observation sequence. Instead, a dynamic programming approach is taken where, for each observation in the input sequence, the best transitions between all successive states are maintained, avoiding redundant computations for the same transition likelihoods in different candidate sequences. The *Viterbi* algorithm [**96**] is used to solve this problem with a run-time complexity of $O(N^2 T_o)$.

The approach consists of iterating over the sequence of observations, updating the likelihood of candidate sequences by computing their compatibility with the input sequence up to the current observation. At each iteration, the maximum likelihood estimate for a partial observation sequence and hidden state sequence given that the current hidden state is h_i is computed (i.e. the likelihood of the best path passing

through state h_i at sequence point t):

$$\psi\big(H_i(t)\big) = \max_{H(0),...,H(t-1)} p\{H(0),...,H(t-1), H(t) = h_i, O_{in}(0),...,O_{in}(t) \mid \Lambda^0\} \quad (4.7)$$

This likelihood is computed for all states h_i by the following two steps:

$$\psi'\big(H_i(t)\big) = \max_{H_j(t)} \Big(p\{H_i(t) \mid H_j(t-1)\}\psi\big(H_j(t-1)\big)\Big)$$
$$\psi\big(H_i(t)\big) = p\{O_{in}(t) \mid H_i(t)\}\psi'\big(H_i(t)\big) \quad (4.8)$$

This two step iteration consists of a *propagation* step followed by a *conditioning* step. First, the distribution $\Psi(t)$ over all states h_i, $i = (1...N)$, is computed by propagating the previous distribution using the transition probabilities in $M(t-1)$ for $t > 0$. (At $t = 0$ the distribution π is used directly for initialization and propagation is skipped.) Then, using the probabilities in $B(t)$, the input is used to bias the propagated distribution. The resulting distribution is normalized.

In a typical Markov chain propagation, the probability for the current state is calculated by accumulating the likelihoods over all previous states that the current is dependent on. In contrast, the goal of the decoding algorithm is to eventually realize a sequence and thus only the most likely previous state that generates the current is considered. In order to maintain this partial sequencing information, for each candidate state $H_i(t)$ a back-pointer is maintained that points to the most likely previous state $H_j(t-1)$ that generates the current. This information is stored in each candidate state $H_i(t)$ as an auxiliary attribute.

At the end of the input sequence, the state with the largest likelihood in $\Psi(T_o)$ is selected and used as the root of a backtracking procedure that traverses the back-pointers and realizes the entire sequence of states (Fig. 4.3). Backtracking is essential for generating a curve as not only does it select the best transitions between successive states but also implicitly propagates information from future observations back to earlier points in the sequence. This avoids local maxima pitfalls that can occur in greedy strategies where the locally best point may not contribute to the best overall

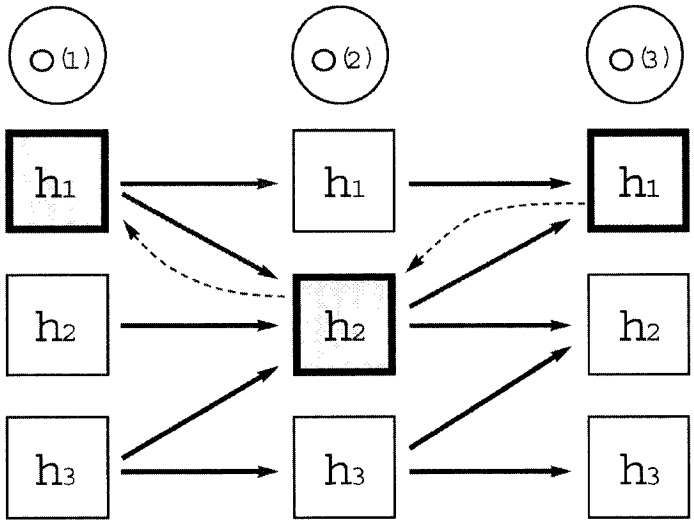


FIGURE 4.3. Synthesis diagram for three states $\{h_1, h_2, h_3\}$ and three input points $\{o(1), o(2), o(3)\}$. Solid arrows indicate all possible transitions, the ones shown in red indicate the best transitions. At the last point, the state with the greatest likelihood (h_1) is chosen, followed by a backtracking procedure that uses the back-pointers to traverse and extract the most likely previous states (shown by dashed arrows).

sequence. In this fashion, the maximum likelihood hidden state sequence that best describes the observation sequence is synthesized.

3.1. Thresholding. The synthesis procedure can be further accelerated by using heuristic pruning. Thresholding the state distribution increases the efficiency of the system by removing candidate states that are not expected to be part of the final solution. Although the solution space may not be convex over the sequence, it can be assumed that for low valued candidate solutions the likelihood varies smoothly. It is easy to see from Eq. 4.8 that the likelihoods are compounded over the sequence and hence only a drastic change in the observation sequence that is not anticipated in the training set can significantly change the rank of low probability sequences. That is, at each iteration, candidate solutions that have very low probability will likely remain low with respect to the top candidates unless all of the top candidates are drastically demoted. Given this assumption, it then becomes feasible to remove low ranking candidate states and provide a more efficient system. Only the most probable

m candidate states need to be maintained and the vector Ψ is normalized accordingly. The runtime for each iteration then becomes $O(2mN)$, $O(mN)$ to extract the top m candidates and $O(mN)$ for propagation, hence improvements occur when $m < N/2$.

3.2. Starting Point Invariance. With a stationary model, the synthesis procedure is inherently invariant to the starting point of the input curve. This is not the case with a non-stationary model where the starting position must first be synchronized to the training set. This causes an undesirable restriction, particularly when dealing with closed curves where users may wish start drawing the shape at any point along the curve. For example, consider the shapes in Fig. 4.1, having to force the users to always start the curves from a fixed position can undermine the simplicity and usability of the system. Therefore, in order to provide invariance to the starting point, the initial matrices $M(0)$ and $B(0)$ are computed over the entire arc-length (the stationary matrix). This allows the synthesis procedure to be bootstrapped to any state at any position along the curve. Based on this bootstrap, different transition and confusion matrices can be applied for different candidate states in the same iteration.

In order to keep track of the bootstrap, two auxiliary parameters are used. First, the state space is augmented to include a training set auxiliary attribute t_{train} that identifies the sequence position of the candidate state from training. This is used to uniquely label states that are found at different positions in the training curves and thereafter help identify the transition and confusion matrices that should be used when processing the candidate state. (Note the use of an auxiliary attribute rather than a constraining attribute, maintaining the ability to transition to states at different positions within the stationarity window.) Second, a decoding auxiliary attribute t^0_{train} is used to maintain the initial starting sequence point leading up to the current state. Its value is only set once, at $t = 0$, using the auxiliary attribute t_{train} from the candidate states $H_i(0)$. The value is passed along the sequence during propagation using the most likely previous state that generates the current (i.e. the back-pointer). Then, the state associated matrices are referenced by $M(t')$ and $B(t')$ where $t' = \left(t + H_i(t) \cdot \hat{t}^0_{train}\right)$ MOD T_M and T_M is the number of matrices. This shifts

the sequence point t by the bootstrap value of the candidate sequence and applies the time-shifted matrices.

3.3. Input Handling. As discussed in Section 1.2, it cannot be expected that all sample points from the user-drawn curves will have an exact match to some point in the control curves of the training set. In cases where exact matches do not exist, the input will cause the distribution to be zeroed as the system does not anticipate every possible input ($p\{O_{in}(t) \mid H_i(t)\}$ may not exist in B). This is further exaggerated when multiple input attributes are used, exponentially reducing the likelihood that a exact match can occur. To address this issue, a sigmoid function is used in order to blur the input bias. This function can be thought of as a soft threshold function, reducing the sensitivity to noise when applying the input conditioning step from Eq. 4.8. The sigmoid is suitable for modeling noisy user inputs as it can be assumed that the intent of the user within a given error range is equally distributed over the neighbors and sharply decays at points further away. The state similarity parameter Υ_o is then $\{G, \Delta, k, c, w\}$ where G is the sigmoid function, k and c are the sigmoid parameters, w is a function that returns the importance weights of scales and attributes and Δ is defined below. The probability that the input attribute $O_{in}(t)_a$ corresponds to the learned observation state attribute $O_j(t)_a$ is then computed by the following:

$$p\{O_j(t)_a \mid O_{in}(t)_a\} = \frac{1}{1 + e^{k(a)\left(\Delta(O_{in}(t)_a, O_j(t)_a) + c(a)\right)}} \tag{4.9}$$

where

$$\Delta(O_{in}(t)_a, O_j(t)_a) = \frac{\sum_{s \in \mathcal{S}} \sum_{\tau'=0..\tau_s} w(s) |O_{in}(t)_{a,s,\tau'} - O_i(t)_{a,s,\tau'}|}{\sum_{s \in \mathcal{S}} w(s)\tau_s} \tag{4.10}$$

and

$$k(a) = \frac{4.3944}{p_{10}(a) - p_{90}(a)}$$

$$c(a) = \frac{-(p_{90}(a) + p_{10}(a))}{2}$$

The sigmoid function can be considered as a blurred step function with blurring parameter k and a shift parameter c. These parameters identify the center of the sigmoid and the sharpness of the cutoff. A simple variable transformation allows for the the sigmoid shape to be conveniently specified by the 90^{th} and 10^{th} percentile thresholds (p_{90} and p_{10}). These parameters are used to control the degree that the input curve biases the synthesis procedure. Setting the 90^{th} and 10^{th} percentile to large values results in similar likelihoods over all observation states, reducing the importance of the input. Setting the 90^{th} and 10^{th} percentile to small values results in an increased sensitivity to the user input and hence provides an increase in the steering power. Figure 4.4 shows a plot of the sigmoid function with labels for the 90^{th} and 10^{th} percentiles for a curves tangent angle attribute.

The total likelihood over all available input attributes is computed by taking the product of the individual attribute likelihoods:

$$p\{O_j(t) \mid O_{in}(t)\} = \prod_{a \in \mathcal{A}_{in}} p\{O_j(t)_a \mid O_{in}(t)_a\} \tag{4.11}$$

where $\mathcal{A}_{in}$ is the set of control attributes used in the input. This product implicitly disregards any learned control attributes that are not provided by the input, allowing the users to freely select what control attributes they wish to use and where along the sequence they wish to use them.

When applying the sigmoid blur, an input sample point no longer acts as a unique conditional, but rather produces a distribution over the observation states. The conditioning step of Eq. 4.8 then becomes:

$$\psi\big(H_i(t)\big) = \max_{O_j(t)} \big(p\{O_j(t) \mid O_{in}(t), H_i(t)\}\big)\psi'\big(H_i(t)\big) \tag{4.12}$$

where

$$p\{O_j(t) \mid O_{in}(t), H_i(t)\} = p\{O_j(t) \mid O_{in}(t)\}p\{O_j(t) \mid H_i(t)\}$$

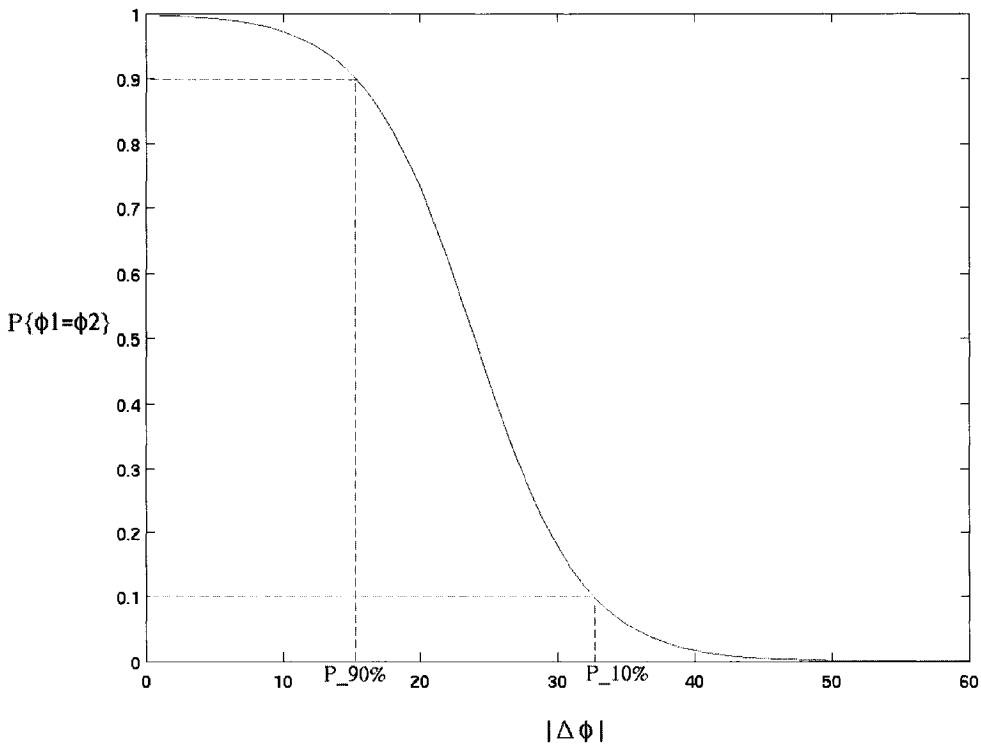


FIGURE 4.4. A plot of the sigmoid function for the tangent angle attribute ϕ.

That is, the observation that results in the best input bias is used to steer the synthesis procedure. The runtime of the system then increases from (N^2T) to $O(N^2T + N|\mathcal{O}'|T)$.

4. Adding Preferences using a Regularization Framework

Regularization [**90**] is used to solve for a data interpolating function that satisfies some supplementary preference, such as smoothness. The technique is typically applied in cases where noisy data or interpolating function ambiguities result in non-unique solutions. Such under-constrained problems require additional constraints in order to further restrict the set of admissible solutions. Given a desired regularization function, the goal is to minimize the error of a functional consisting of a weighted sum

of the interpolating function's residuals in conjunction with the *regularizing term*:

$$F(f) = \sum_{i=1}^{n} \frac{\left(f(x_i) - y_i\right)^2}{\sigma^2} + \lambda \int_0^1 \left(f''\right)^2 dx \tag{4.13}$$

The function $f(\cdot)$ that minimizes the above equation is the solution for the desired data interpolating function.

A probabilistic approach to regularization [**86, 47**] consists of maximizing the posterior of a Bayesian model:

$$\max_{f \epsilon \mathcal{M}} p\{f \mid D\} \propto \max_{f \epsilon \mathcal{M}} p\{D \mid f\} p\{f \mid \mathcal{M}\} \tag{4.14}$$

where f is an element of a set of functions $\mathcal{M}$ and D is the input data. Typically, the *data model* $P\{D \mid f\}$ assumes a Gaussian noise model and the *fidelity model* $P\{f \mid \mathcal{M}\}$ is the regularization bias commonly defined as an exponent of the smoothness term:

$$p\{D \mid f\} = e^{-\sum_{i=1}^{n} \frac{(f(x_i)-y_i)^2}{\sigma^2}} \tag{4.15}$$

$$p\{f \mid \mathcal{M}\} = e^{-\lambda \int_0^1 \left(f''\right)^2 dx} \tag{4.16}$$

The HMM framework can be formulated using Eq. 4.14. Rather than having a fixed noise model and smoothness constraint, priors for these components are learned from examples. That is, using the HMMs, the maximum likelihood hidden state sequence can be considered as the desired interpolation function, the confusion matrix can be considered to represents a learned data term $p\{D \mid f\}$ and the transition matrix can be considered to represent a learned regularization term $p\{f \mid \mathcal{M}\}$ (evaluating the goodness of the function using the previous neighbors).

Using this framework, users can embed supplementary *ad-hoc* biases to the system. These biases can be in the form of analytical functions that define a prior preference on the types of solutions that can be produced. Further, since our state space is discrete, we do not need to rely on traditional gradient-descent based variational calculus techniques but rather, we can apply the *Viterbi* algorithm to find the maximum. At each iteration of the decoding algorithm (Eq. 4.8), the distribution

vector $\Psi(t)$ is biased by augmenting the energy of each candidate state as follows:

$$E\big(H_i(t)\big) = -log\Big(\psi\big(H_i(t)\big)\Big) + \sum_k \lambda_k R_k\big(H_i(t)\big) \tag{4.17}$$

where $R_k(\cdot)$ is a regularization constraint (such as smoothness) and λ_k is the associated weight. This must be computed before the propagation step of Eq. 4.8 such that the regularization terms are taken into account when selecting the maximum likelihood previous state.

In general, there are few restrictions on the type of regularization constraints that are applied. First, they must be causal such that the distribution can be propagated forward. Second, when state thresholding is applied, it must also be assumed that the function is smooth. If the regularization terms require information from previous points (such as in a smoothness constraint), the history is readily accessible using the back-pointer of each state. It may be the case that the regularization terms require information from the input conditionals, which is also readily available. Any additional parameter required by the regularization term can be stored as an auxiliary attribute in the candidate states. For example, when the input conditional only consists of the tangent angles of an input curve (as discussed in Chapter 3, Section 1), a regularization function may instead require the Cartesian co-ordinates and hence must recompute the position for each sample point (assuming a bootstrap is available at $t = 0$). To avoid such redundant computations, this information can be stored and updated conveniently in auxiliary attributes, providing direct access to the desired forms of data.

5. Summary

This chapter described a framework that consists of learning and applying curve refinement models. The refinement process consists of solving for the maximum likelihood mixture of segments from the training set that best explain the input. The degree of mixing and the scales at which to mix in are specified by the parameters of the model. The presented framework is general and in order to apply it to real world

applications it must be customized accordingly. This entails defining the following components of the model: $\{R, \mathcal{S}, \mathcal{A}, \mathcal{A}_o, \mathcal{X}, \mathcal{X}_o, \tau, \omega, \mathcal{T}, \mathcal{T}_o\}$. That is, the regularization terms, the set of scales and attributes to use and the training data are all application specific components. Furthermore, the mixture variance, sigmoid parameters and importance weights in Υ and Υ_o must also be specified. These are discussed in more details in the following two chapters where the framework is customized for two applications: sketch beautification and robot path planning.

CHAPTER 5

Sketching Application

This chapter deals with the sketching application and presents the required framework customization, the experimental setup and results. Customizing the framework for this application consists of identifying and integrating the desired curve attributes, control schemes and regularization constraints into the model as pertained to sketching. A supplementary texture filling process that further enhances the sketch refinement process is also presented. The experiments consist of testing the system over a variety of training sets, input curves and parameters, examining the behavior of the system by subjectively evaluating the results.

1. Curve Attributes for the Hidden Layer

The hidden layer of the HMM encodes constraints on the types of sketches that can be synthesized. The states corresponding to this layer encode multi-attributed sketching elements consisting of the following components:

- Shape of the curve
- Color of the curve
- Thickness of the curve
- Fill-color
- Fill-transparency
- Fill-direction

The set of desired curve attributes is thus defined as follows:

$$\mathcal{A} = \{\theta(t), \Delta\theta(t), c(t), k(t), u(t), d(t)\}$$

The shape of the curve is represented using either a first-order representation, the tangent angle $\theta(t)$, or a second-order representation, the change in tangent angle $\Delta\theta(t)$. Both of these are evaluated experimentally with and discussed in more detail later. The color of the curve $c(t)$ is a 24 bit RGB value (eight bits per color channel) and the thickness $k(t)$ specifies the radius (in pixels) that should be used when drawing a curve's scan converted pixels. The filling parameters are used to "color in" the interior of the curve. The fill-color and transparency $u(t)$ are encoded using a 32 bit RGBA value (eight bits for each of the three color channels and eight bits for the alpha channel). The fill-direction $d(t)$ is used to identify whether the seed pixels for filling should be generated along the curve's normal or opposite to it; it is set to 0 for no fill, $+1$ for filling along the normal and -1 for filling opposite to the normal.

1.1. Seed Pixels. The fill-color, transparency and direction are used to generate *seed* pixels; pixels adjacent to the curve that initiate a texture filling process. If one desires, more than one seed pixel can be colored along each sample point of the curve, providing a larger-scale bootstrap for the texture synthesis process. The fill-direction must be adjusted in correspondence to a curve's sequence orientation (clockwise or counterclockwise) in order to fill the interior of a curve. Indeed, there may be cases where the synthesis of curve segments result in fill-directions aimed at the the exterior of a closed curve. This is particularly prevalent when drawing self-intersecting curves. For such segments, the user must either manually flip the fill-direction or completely remove the seed pixels at those points. Developing an editing tool to accomplish this is straightforward, setting $d(t) = -d(t)$ (or zero) for all curve sample points that fall within the user's boundary selection.

2. Curve Attributes for the Observation Layer

The observation layer of the HMM encodes constraints on the effect an input curve segment has on the resulting sketch. The states corresponding to this layer represent the types of inputs the system expects and includes the following attribute:

- Shape of curve

hence $\mathcal{A}_o = \{\phi(t),\ \Delta\phi(t)\}$. That is, the shape of the input curve is represented by either a first-order representation $\phi(t)$ or second-order representation $\Delta\phi(t)$. In principle, other input components can also be used to control the synthesis procedure, such as the pen-pressure and speed, though these are not experimented with in this thesis.

3. Normalization and Sampling

Recall that Curves in $\mathcal{T}$ and $\mathcal{T}_o$ are normalized over their arc-lengths and uniformly sampled. The values used to accomplish this are determined based on the type of model used; stationary or non-stationary. One of the key criteria that must be satisfied is to preserve the appropriate correspondence between the sample points from the control curves and sample points from the associated refined curves. In particular, to synchronize B and M, the number of samples used to represent the control curves must be the same as the number of samples used to represent the refined curves, despite the fact that their arc-lengths may differ.

3.1. Normalizing for a Stationary Model. When a stationary model is chosen, all curves in the training set are normalized using the arc-length of the longest curve in the set. This preserves the relative size of features in curves of different lengths. The curves are then sampled using a piecewise linear approximation; $p = (1-l)p_i + lp_{i+1}$ where $0 \leq l \leq 1$. First, each refined curve is sampled uniformly with a fixed, predefined sampling resolution. Then, each control curve is sampled uniformly using a sampling resolution that is determined dynamically; the sampling resolution is computed such that the resulting representation produces the same number of samples

for both a refined curve and it's associated control curve. This can be approximated by dividing the total arc-length of the control curve by the total number of sample points used to represent the associated refined curve.

Although all of the training curves are normalized, the input curve is not normalized. This allows for the possibility of applying the decoding algorithm in real-time, providing immediate feedback to the user (subject to a lag spanning the larger of the filter window or translation step).

3.2. Normalizing for a Non-Stationary Model. In a non-stationary model, the input curve and all training curves are normalized using their own arc-length. As such, all curve have an arc-length of 1 and their relative size is not preserved. This is required in order to synchronize the arc-length positions of the input curve with the non-stationary transition and confusion matrices. Once the curves are normalized, they are sampled uniformly with a fixed sampling resolution using a piecewise linear approximation. During synthesis, processing is delayed until the completion of the input curve; once the input curve is completely drawn, it is normalized over its arc-length, re-sampled and then used to synthesize the output.

4. Supplementary Sketch Refinement Preferences

There are three regularization constraints that are used to improve the results for the sketching application:

- Sequence coherence
- Example coherence
- Magnetic attraction to input

The set R is then $\{R_1, R_2, R_3, \lambda_1, \lambda_2, \lambda_3\}$ where each element is further described in this section.

The sequence coherence constraint is used to reduce the likelihood of generating out of sequence transitions, promoting solutions that are more consistent with the sequencing in the training set. That is, if two different candidates have a similar likelihood, the solution will be biased toward the candidate that is in sequence. This

constraint has also been used in earlier work by Hertzmann *et al.* [**38**] where the authors exemplify how coherence provides better progression over the sequence and hence better reflects the features (or styles) of a training set. In order to embed this bias in the model, the training set auxiliary attribute t_{train} is used to identify the position of candidate states in the training set. Using the back-pointer, the position of the last state that generates the current is identified and the likelihood of the current state is penalize if out of sequence:

$$R_1\big(H_i(t)\big) = \begin{cases} 1 & \text{if } H_i(t) \cdot \hat{t}_{train} - H_j(t-1) \cdot \hat{t}_{train} = 1 \\ c_1 & \text{otherwise} \end{cases} \tag{5.1}$$

where $c_1 > 1$ is the penalty factor.

The example coherence constraint is used to reduce the number of transitions between different examples in the training set. This biases the system to generate larger curve segments from a individual examples and avoid unnecessary transitions that may occur over similar examples. (Note that the translation step strictly enforces this constraint by limiting the minimum segment length.) To integrate this bias into the model, another training set auxiliary attribute is included to help identify the training example that the candidate state belongs to. State sequences with different example identifiers are then penalized:

$$R_2\big(H_i(t)\big) = \begin{cases} 1 & \text{if } H_i(t) \cdot \hat{id} = H_j(t-1) \cdot \hat{id} \\ c_2 & \text{otherwise} \end{cases} \tag{5.2}$$

where $c_2 > 1$ is the penalty factor and id is the example identifier.

Finally, an additional constraint is used to promote solutions that are closer to the input. This constraint is referred to as the *magnetic* regularization term as its biasing effect is similar to that of applying a magnetic attraction force between the input and output curves. This constraint helps avoid drift due to quantization errors. Moreover, even in ideal cases, because the state spaces for both the observation and hidden states encode the shape of a curve using its angles, there are no constraints learned explicitly on the positions of curves. This regularization term provides the

added advantage of enforcing position based constraints while also maintaining the flexibility of a first or second-order representation of the training set. In order to determine the distance between the input and candidate solutions, the Cartesian co-ordinates of the sequence leading up to the states must be determined.

Assuming that the Cartesian co-ordinates are available in the form of auxiliary attributes of the candidate states, the regularization term for the state can easily be computed as a function of the distance between the candidate points $\{x, y\}$ and the input points $\{x_{in}, y_{in}\}$. This is evaluated by averaging the square distance between all points in the candidate state and the input:

$$R_3\big(H_i(t)\big) = \sum_{l=0\ldots\tau} \big(x_{in}(t\tau + l) - H_i(t) \cdot \hat{x(l)}\big)^2 + \big(y_{in}(t\tau + l) - H_i(t) \cdot \hat{y(l)}\big)^2 \quad (5.3)$$

where τ is the transition step (the number of points in the segment). The method to compute the Cartesian co-ordinates is described below.

4.1. Cartesian Co-ordinates. To include position based constraints in the model, additional decoding auxiliary parameters $\{x, y\}$ are added to the states. These parameters identify the Cartesian co-ordinates for all sample points in the current state by traversing the most likely hidden state sequence up-to and including the state. Their values are computed by extrapolating the $\{x, y\}$ co-ordinates from either the previous point within a segment (within the same state) or the most recent point in previous state (identified by the back-pointer). Using the tangent angels at the lowest scale, the Cartesian point $\{x(l), y(l)\}$ is then calculated by $\{x(l-1) + \delta t(l)\cos(\theta(l)), y(l-1) + \delta t(l)\sin(\theta(l))\}$ where l is the sequence position of the input sample point and δt is the sampling resolution. (In this formulation, the parameter l is used to index the respective position of the auxiliary parameter in state $H(t)$ by taking the modulus remainder with the transition step τ). At $l = 0$, the initial point $\{x(0), y(0)\}$ is bootstrapped to the first input point drawn $\{x_{in}(0), y_{in}(0)\}$.

In order to properly match the resolution of the auxiliary co-ordinates with that of the control and refined curves from training, the sampling resolution $\delta t(l)$ is computed using two additional training set auxiliary attributes. One attribute stores the sampling resolution of the control curves δt_o and the other stores the sampling resolution of the refined curves δt_h:

$$\delta t(l) = \frac{\delta t_{in}(l)}{H_i(t) \cdot \hat{\delta t_o}(l)} \times H_i(t) \cdot \hat{\delta t_h}(l) \tag{5.4}$$

where δt_{in} is the sampling resolution of the input curve. According to the described normalization scheme, in the non-stationary case $\delta t_o = \delta t_h$, hence the resolution used degenerates to the input sampling resolution; $\delta t(l) = \delta t_{in}(l)$. In the stationary case, Eq. 5.4 compensates for the resolution difference between the control curves and refined curves. In both cases, the learned styles will contract or dilate according to the input sampling resolution.

When the second-order curve representation $\Delta\theta$ is used, the tangent angles then become a decoding auxiliary attribute. The Cartesian co-ordinates are computed using a second-order reconstruction $\{x(l-1) + \delta t(l)\cos(\theta(l-1) + \Delta\theta(l)), y(l-i) + \delta t(l)\sin(\theta(l-1) + \Delta\theta(l))\}$. The synthesized curve is then bootstrapped using the first two input points $\{x_{in}(0), y_{in}(0)\}$ and $\{x_{in}(1), y_{in}(1)\}$

4.2. Curve Closure. In some cases, it may be desirable to vary the degree of influence for a regularization term along the sequence. This is easily accomplished by defining the regularization weight as a function of the sequence position; $\lambda(t)$. To exemplify this, a sequence varying regularization weight is used to enforce curve closure. Maintaining closure when it is desired is an important component of the sketching system. If the user draws a closed curve, then the system must enforce this criterion as failing to satisfy it results in an output curve that is topologically different from the input, resulting in a significantly noticeable discrepancy between the input and output.

The approach taken to synchronize the closure of an output curve with that of the input curve consists of applying a large bias to increase the likelihood that the

output curve remains near the input curve. (If the output is always near the input, then when the input curve is closed the output curve will also be closed, and vice versa.) However, it is not desirable to apply this bias equally along the entire curve; the system must also provide the flexibility for the output to diverge from the input in order to express the learned styles. It is however desirable to restrict the output curve when approaching the the curve's endpoints, where closure is not guaranteed. Thus, to allow the interior of the output curve to divert from the input curve while converging at the endpoints, the magnetic regularization term is weighed by a function of the arc-length as follows:

$$\lambda_3(t) = ke^{|1-\frac{2t}{T_o}|} \tag{5.5}$$

where t is the current state sequence position and T_o is the length of the observation sequence. This smoothly increases the magnetic term at the curve's end-points where it is needed while reducing it when away from the end-points.

Since the sequence length T_o of the observation sequence in not known until the user has finished drawing the curve, this constraint is only applicable once the entire curve has been drawn and cannot be used in real time executions. To achieve this in a real-time drawing environment, it is first assumed that the drawn curve is not closed and synthesis is preformed concurrently while the curve is being drawn. Once the input curve is complete, the system evaluates if the input curve's endpoints are within a predefined distance. If the endpoints are close enough, then the system assumes that the curve is closed and regenerates the entire curve using the decay function from Eq. 5.5.

4.3. Additional Parameters. The remaining parameters of the model are determined empirically. Some are universally applied for all training sets while others are adjusted according the the nature of the set. The importance weights, scales, transition step and sigmoid parameters are set once. The shape of the curve is set to be the primary constraining attribute and all the other attributes have their importance weights set to smaller values (i.e. $w(\theta) >> w(a)$ for all other curve attributes a).

There are 4 scales used for the multi-scale representation and the highest scale is set to bear the most weight (twice the weight of the other scales). The scales include the following: the original sample points and 2, 4, 6 fold Haar filtered versions of those points (i.e. the points resulting from repeatedly applying the filter 2, 4, 6 times on the raw sample points). The sigmoid parameters are empirically determined; it is found that a 90 percentile threshold at 15 degrees and 10 percentile threshold at 33 degrees can provide a good degree of control with limited sensitivity to noise. The mixture variance, stationarity windows and regularization weights are empirically determined for each training set.

5. Drawing Output Curves

A curve is drawn by instantiating the decoded maximum likelihood hidden state sequence. The curve's Cartesian co-ordinates are captured from the auxiliary parameters described in section 4 while the other attributes are captured from their corresponding dimensions in the state. The samples from the states are then used as control-points in an integer scan-conversion algorithm [**66**] that uses a linear interpolant to produce the raw pixel values and locations. In cases where the input curve is closed, the end-points are also linearly interpolated.

5.1. Drawing Texture Fill Seeds. The texture fill seeds are points that are drawn adjacent to the scan-converted pixels to indicate how a curve should be shaded. They are rendered using the corresponding colors and directions extracted from the states. The exact locations of the seed pixels are computed by applying predefined placement rules that use the location of the current and previous pixels in the scan-converted curve. For example, if the fill direction is positive, and the previous pixel co-ordinates, $\{x_p(l-1), y_p(l-1)\}$, are both less than the current $\{x_p(l), y_p(l)\}$, then the location for the seed pixel is $\{x_p(l) + k(l), y_p(l)\}$. This is repeatedly applied for each fill-color dimension (there can be more than one seed pixel for each sample point), incrementing the co-ordinates accordingly. The fill update for the current pixel stops when either the seed pixels are exhausted or the placement location refers to a

point that is already filled. An assumption taken in this procedure is that the scan-converted curves do not superimpose each other as this can result in pixels emanating outside the interior of the curve.

5.2. Overlaying Curves. When a new curve is drawn, a new image layer is created for that curve. The size of the layer is adjusted using the maximum and minimum co-ordinate values (i.e. the bounding box of the curve). It can also be specified that the layer must span horizontally or vertically up to the edge of the image (e.g. for filling in the sky or the terrain). The layer order is determined by the order in which curves are drawn. The final RGB color value of a pixel is then computed by compositing the layers as follows:

$$C = \frac{A_1C_1 + (1 - A_1)\big(A_2C_2 + (1 - A_2)(A_3C_3 + ...)\big)}{A_{norm}} \tag{5.6}$$

where C is the composite color for the pixel, A_i is the alpha value for the pixel at layer i and C_i is the color of the pixel at layer i. The normalized composition is computed by dividing the unnormalized colors by a normalization constant computed as follows:

$$A_{norm} = A_1 + (1 - A_1)\big(A_2 + (1 - A_2)(A_3 + ...)\big) \tag{5.7}$$

6. Texture Filling

Specifying the interior colors along each curve as a supplementary attribute allows the system to initiate a post-processing texture synthesis procedure. This procedure is used to color inside the empty areas of a closed curve (or an opened curve that is bounded by the edges of the image). A statistical texture filling process, also based on a Markov Model of image properties, is applied to synthesize a new texture that *looks* similar to a sample texture in the training set. (In regards to the two level hierarchy of HMMs, the curve-level and the scene-level, this procedure can be considered as a third level processing phase; the pixel-level.)

The texture filling process is initiated from cues attached to the synthesized curves, acting as the "seeds" for an incremental stochastic pixel inference procedure.

The process consists of first identifying the empty pixels that need to be colored and then determining the color by searching the sample texture for a similar regions.

The search for empty pixels is performed over four orthonormal scan-lines; from left to right, right to left, top to bottom and bottom to top. When searching along a scan line, the first empty pixel that 1) has at least one filled neighbor that is not a boundary and 2) is found after an odd number of boundary crossings is added to the list of pixels to be filled and the next scan line is processed. (A boundary is identified by a unique color.) When all scan-lines are exhausted, the set of pixels to be filled is sorted in order of the number of filled neighbors each one has. For efficiency, sorting is performed at every n^{th} iteration. To avoid starvation of low-ranking pixels and provide a more uniform synthesis from all directions, at every m^{th} iteration the pixel rank is perturbed randomly and if the top pixel on the list belongs to the same scan-line as the top pixel from the last iteration, it is penalized. The highest ranking unfilled pixel is selected and its color is drawn as the maximum likelihood value of the probability of the color as described in [**98**].

The pixel inference procedure is bound by the curve's edge, its bounding box and the edges of the image. It is applied to each layer independently using the example texture from the corresponding training set. If the example texture does not have an alpha map, the alpha value is copied-over from the synthesized seed pixels. Any unfilled pixels remaining in the image take on the value of a specified background texture. That is, once post-processing of all layers is complete, every pixel that is left unprocessed is set with an alpha value of zero (completely transparent). The layers are then merged together with a predefined background texture that has its alpha value set to one (completely opaque).

7. Experimental Setup

The experiments described here were performed using a sketching application that implements the HMM framework. The application allows a user to draw interactively while performing curve synthesis in real-time. The user selects the class of curves used

for synthesis and controls various parameters of the synthesis process. Figure 5.1 shows a screen shot of the application's graphical user interface. The main window consists of three panes, the left pane is used for drawing, the right pane is used to present the results and the bottom pane provides buttons for common actions. In addition to the graphical user interface, a command line interface is used to provide direct access to all parameters and functions.

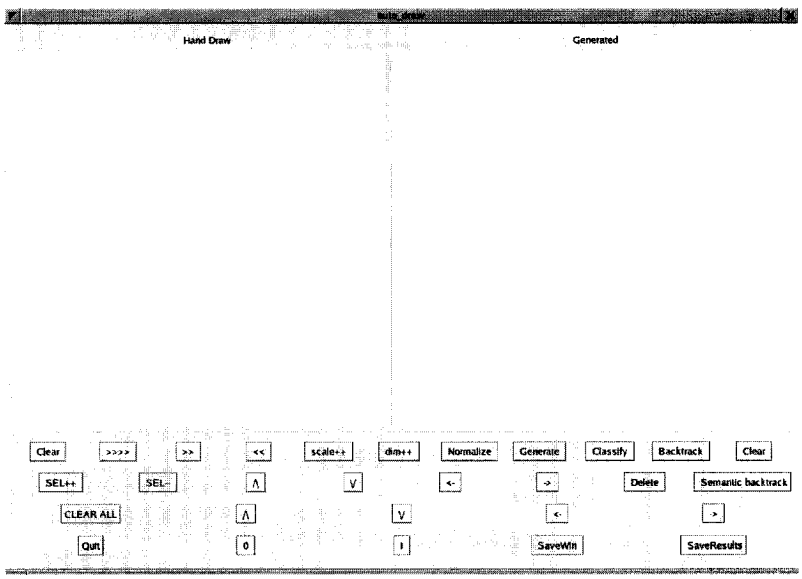


FIGURE 5.1. A screen-shot of the graphical user interface for the sketching application. Left pane is used for drawing, right pane is used to display the results and the bottom pane is used to provide quick access to common commands.

The training sets used in the experiments are carefully drawn by hand with an electronic pen and tablet. Where applicable, texture seeds are manually extracted from sample images and, using linear interpolation, are registered with the curves' sample points. The parameters for each training set are empirically determined based on a subjective evaluation of the results. These parameters are stored together with the training set and reloaded whenever the set is used. All experiments are executed in real time using a 3 Ghz Pentium 4 with 1 Gigabyte of RAM.

8. Experimental Results

This section presents the synthesis results for various training sets and input curves. The results are examined and evaluated subjectively under various parameter settings and curve attributes (such as color and fill-color with the texture filling process).

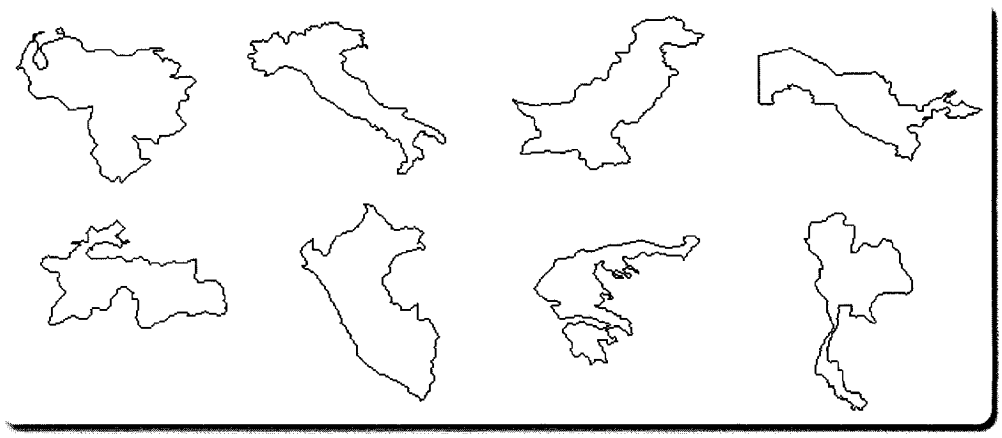


FIGURE 5.2. Refined curves from a training set used to draw coastlines. The entire set consists of 25 examples. The control curves are generated by applying a low-pass filter on the refined curves, removing the fine details that are too difficult to draw.

8.1. Coastlines. Figure 5.2 shows several examples from a training set used to produce coastlines. These refined curves exhibit low-level details that are too difficult or cumbersome to manually draw. For each curve, its associated control curve is automatically generated by blurring the refined one using a low-pass filter (removing the elements of the curve that are difficult to draw). This set is used to train a stationary model.

Figure 5.3 shows an example sketch and the results of the synthesis procedure when no regularization terms are used. It is easy to see that, although the synthesis exhibits the coastline features, the output is not an acceptable solution. One noticeable artifact is that the topology of the input curve is different than that of the output curve; the gap between the endpoints is too large for linear interpolation to

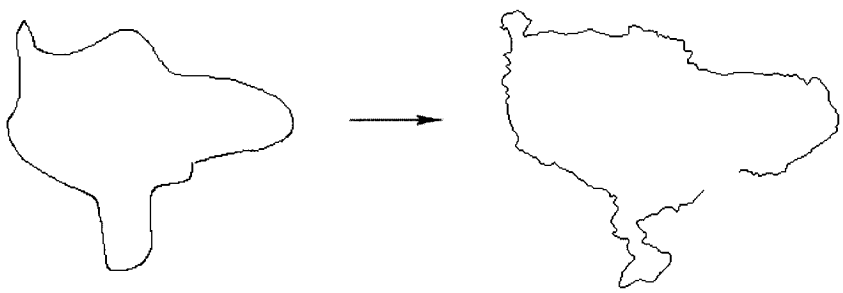


FIGURE 5.3. Example synthesis of a hand-drawn curve using the *coastlines* training set (Fig. 5.2).

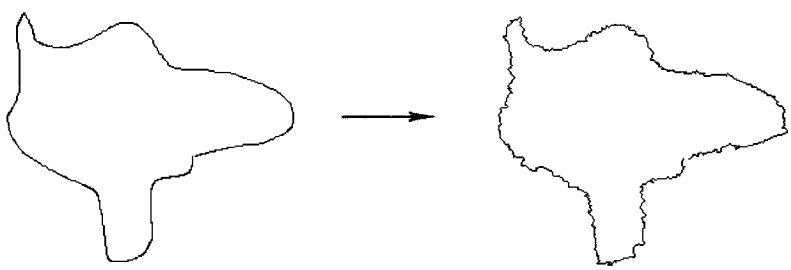


FIGURE 5.4. Example synthesis with a large magnetic bias.

take place without degrading the desired style. (For display purposes, the output curve has been left open, though the system normally performs a linear interpolation due to the fact that the endpoint of the input curve are close enough such that its considered a closed curve.) Figure 5.4 shows the result when the magnetic regularization term (Eq. 5.3) is applied. In this case, it is easy to see that the output curve is close enough to the input curve to observe to the closure constraint, though the learned coastline features are no longer as prevalent. When the magnetic term is too large, the candidate solutions cannot diverge far enough from the input curve in order to express the learned style.

Figure 5.5 shows the result when applying the sequence dependent decay function on the regularization weight (from Eq. 5.5). The magnetic attraction constraint is

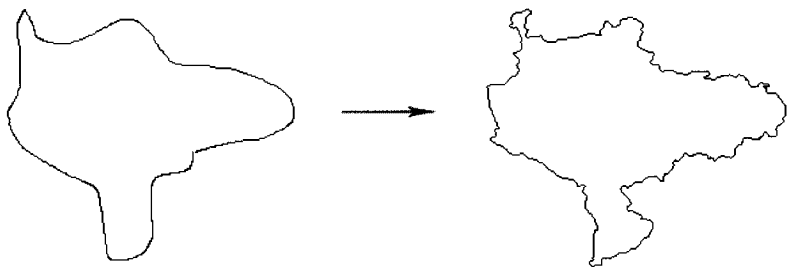


FIGURE 5.5. Example synthesis when applying the decay function on the magnetic bias.

relaxed throughout the inner parts of the curve, allowing for the coastline features to be expressed, while bearing more weight at the endpoints, resulting in an output that is topologically similar to the input.

Figure 5.6 shows the results when changing the mixture variance. It can be seen that the larger the variance, the more influence the input curve and magnetic terms have on the output. In the top left example, because the synthesis uses a very low mixture variance ($\approx$ 1 square degree), the resulting output consists of segments from only one of the training examples (almost an exact instance from training). The following curve on the right shows the output when increasing the variance by 20 degrees. The resulting curve is a mixture of segments from the training set, though there is insufficient blurring for the input and magnetic term to steer the process such that the output exhibits the same overall shape as the input. When increasing the variance further, the output begins to converge to the overall shape of the input. The bottom right curve shows the result when using a very large variance (in the order of 10000 square degrees). In this case, the learned constraint for sequential consistency have minimal influence and the shape of the output curve lacks the desired style.

Figure 5.7 shows the results for the same experiment used to produce the results shown in Fig. 5.6, but without the regularization term. This isolates the effects of the input conditional and demonstrates its influence when changing the mixture variance. It can be seen that the first few output curves are similar to those in

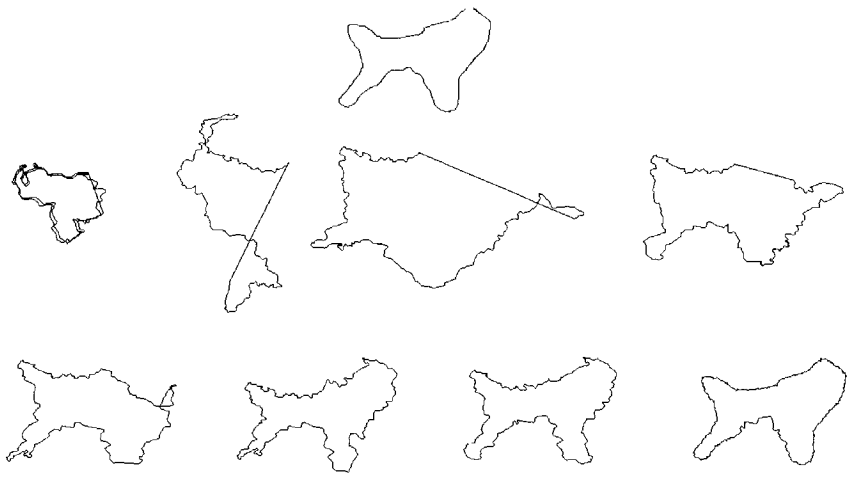


FIGURE 5.6. Example outputs demonstrating the effect of the mixture variance. Top center shows the input curve. From the top-left to bottom-right, the results are shown when increasing mixture variances.

the previous figure, demonstrating that the learned sequential constraints are the dominating biases in both experiments. When increasing the variance further, the input begins to play a bigger role, though it fails steer the process such that the output converges to the overall shape of the input, even when the sequential constraints are at a minimum. This further demonstrates the importance of the magnetic regularization term.

8.2. Leaves. Figure 5.8 shows a training set used for drawing leaves. It can be seen that for some examples in this set, the low-level details are a uniquely associated to the overall shape (e.g. the maple leaf has a unique overall shape and defining details), while for other examples, leaves with similar overall shapes exhibit distinct features (there are ambiguities in the set). Because the examples have well-localized features, this set is used to train a non-stationary model. In Fig. 5.9(a), the synthesis results are illustrated when using this set. It can be seen how the generated mixtures exhibit the desired leaf-like styles.

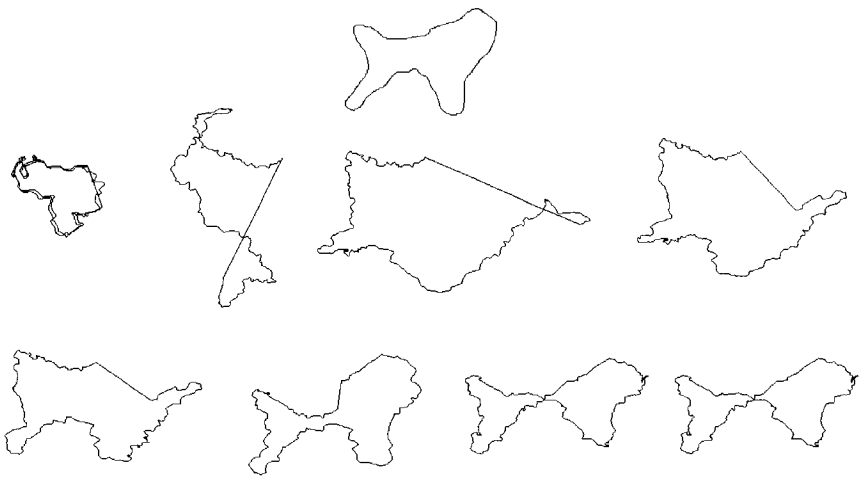
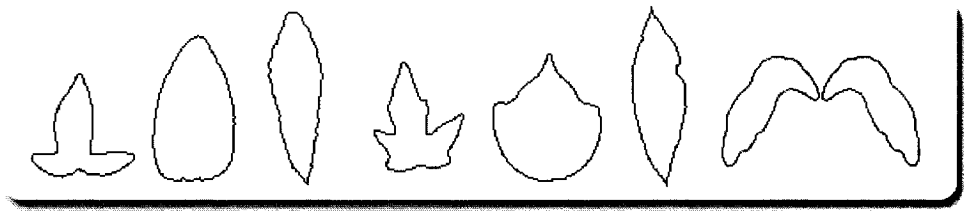


FIGURE 5.7. Examples demonstrating the results when increasing the mixture variance and excluding the magnetism term. Top center shows the input curve. From the top-left to bottom-right, the results are shown when increasing mixture variances. Because the input curve is closed, the long line segments are produced by linear interpolation to close the output curve.

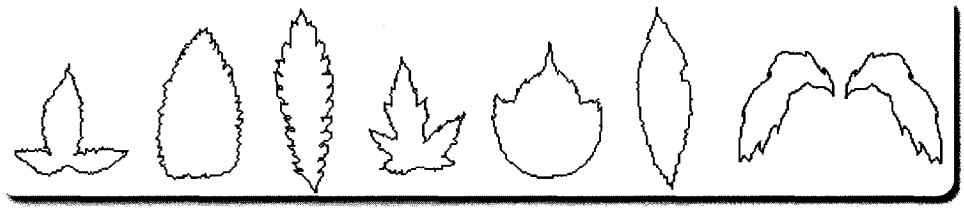
However, the resulting shapes are not symmetrical, a property often seen in real leaves but can sometimes be ignored in the realm of imaginative illustrations. Supplementary global constraints are required in order to enforce symmetry and hence it remains an open problem. Figure 5.9(b) shows the successive results of the ongoing synthesis process. It can be seen how input points later in the sequence affect the solution at earlier points.

8.3. Skyline. A set of training curves consisting of two primitive roof-top shapes are used to produce skylines (Fig. 5.10). Note the difference between the set shown in Fig. 5.10(b) and the set shown in Fig. 3.1(a); the connecting line segments are no longer required as the Gaussian blur allows the user to force a segment mixture despite the fact that such transitions are not seen in the set. In this example set, because the desired output consists of repetitive mixtures of these examples, the set is used to train a stationary model.

Figures 5.11 and 5.12 show the results when using this set. It is easy to see that the outputs consists of a locally consistent mixture of the training set that are



(a) Control curves produced by filtering the refined curves.

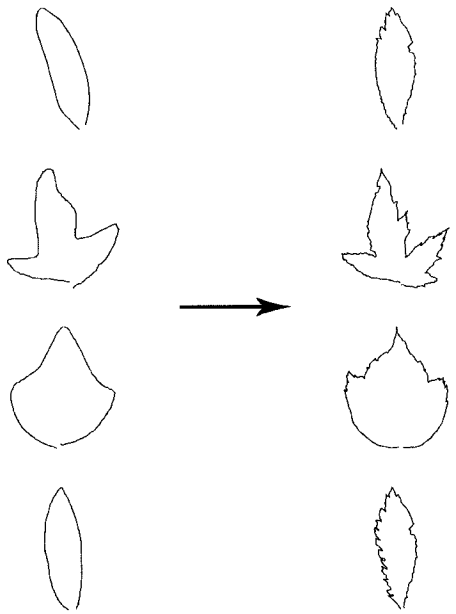


(b) Refined curves from a training set used to capture a style for leaves.

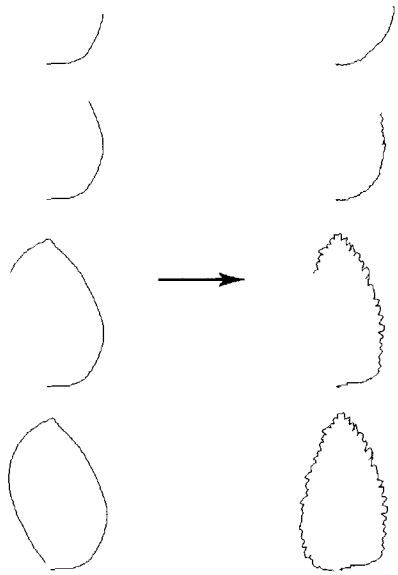
FIGURE 5.8. A training set used for producing outlines that look like leaves. The control curves are filtered versions of the refined ones.

guided by the input curve. Desired transitions that do not exist in the training set can also be seen, such as that from a vertical line to a sloping roof-top. To demonstrate the importance of the backtracking procedure, Fig. 5.12 shows the results when synthesizing a curve using a greedy approach and the Viterbi algorithm. It can be seen that when using a greedy strategy (middle curve), the input curve in its entirety is not considered and each segment is treated independently, resulting in the stair-case effect. Backtracking corrects this problem by considering the entire sequence leading to a satisfactory result.

8.4. Basic Shapes. Figure 5.13 shows a training set with various polygonal shapes and a round shape (used to train a stationary model) and Fig. 5.14 shows the results when using this set. It can be seen how the appropriate segments from the training set are synthesized in order to maintain the overall shape of the input curve. Figure 5.15 shows the same training set with color added. This set is used to produce the results shows in Fig. 5.16, 5.17 and 5.18. In Fig. 5.16, the effect of applying the example selection coherence bias (Eq. 5.2) is demonstrated. It can be seen from the

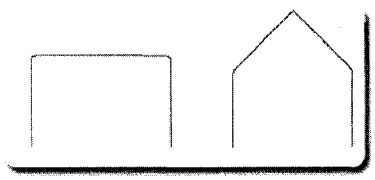


(a) Example synthesis using the leaf training set.

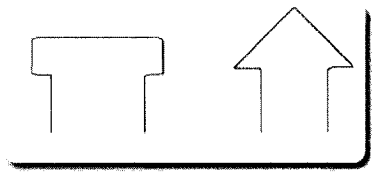


(b) Results of the synthesis procedure while accepting additional input (from top to bottom).

FIGURE 5.9. Curve synthesis using the leaves training set.



(a) Control curves.



(b) Refined curves.

FIGURE 5.10. A simple training set used to draw skylines.

colors along the output curves that when this regularization term is used there are less transitions between examples and the output is more uniformly colored.

8.5. Fish. Figure 5.20 shows the results when using a training set consisting of fish shapes (Fig. 5.19). It can be seen that 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 degree of mixing by changing the variance parameter. For this training set, a non-stationary models was used.

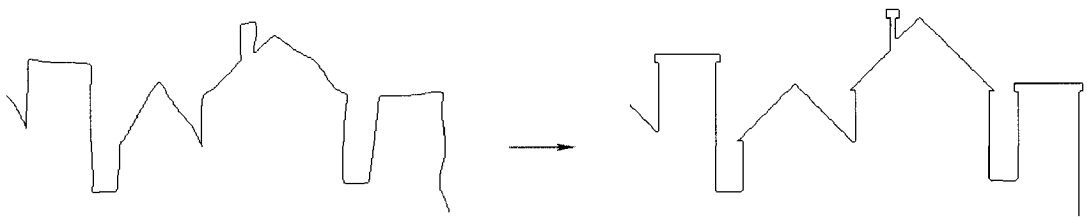


FIGURE 5.11. Example synthesis using the skyline training set.

8.6. First Order versus Second Order Representation. Figure 5.21 shows a training set consisting of only one example (a curl-like pattern) associated with a simple control curve (a straight line). This set is used to generate the results show in Fig. 5.22. Instead of using the tangent angles $\theta(t)$, the shape attribute of the curve is encoded using the second-order representation ($\Delta\theta(t)$). One of the difficulties often seen when using a second oder representation is that errors are accumulated over the entire curve. Figure 5.22(a) shows an example synthesis where, due to accumulated errors, the output curve drifts away from the input. Further, since the control curve in this training set consists of only a straight line, the system is not trained to respond any differently to different input curvatures. This lack of control is demonstrated in Fig. 5.22(b). Applying the magnetic regularization constraint helps avoid these issues by biasing the distribution such that the output curve is more likely to remain close to the input (Fig. 5.22(c)). Note that because an orientation invariant representation is used, despite the fact that the system is trained using only one example oriented in one direction, the pattern is repeatable along any arbitrary direction. Such orientation invariance is sometimes desired, but not always (i.e. trees are always vertical, text horizontal etc.). Using the same training example, Fig. 5.23 shows the results when changing the input sampling resolution. It can be seen how the size of the curl pattern dilates when reducing the sampling resolution.

Figure 5.24 shows the results when using both the first and second-order representation. For the first-order representation, the training sets consists of the patterns

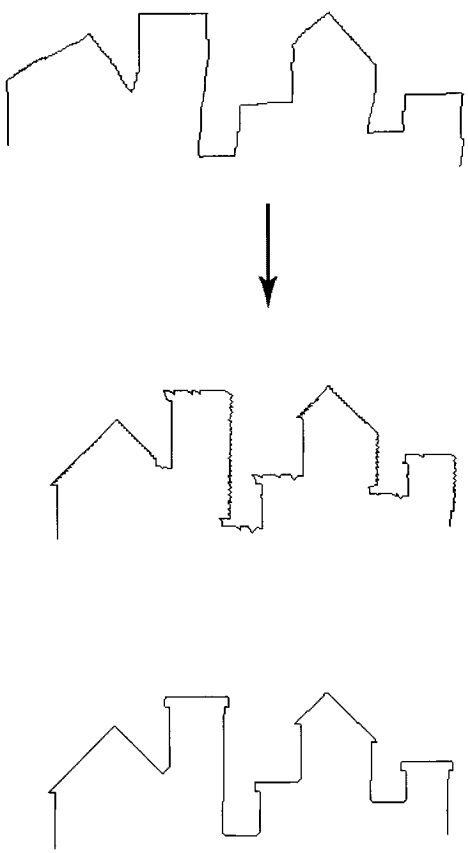


FIGURE 5.12. Example demonstrating the effect of backtracking. The top curve shows the input, the middle curve in shows the result when using a greedy approach, the bottom curve shows the result when backtracking.

rotated at four principle directions, up, down, left, right. (The actual examples consists of vertical and horizontal patterns that are traversed along both directions.) Using the second-order representation results in outputs are similar to that seen in texture maps, warping the pattern over the input curve. Using the first-order representation, the outputs preserve the rectilinear shape of segments from the training sets.

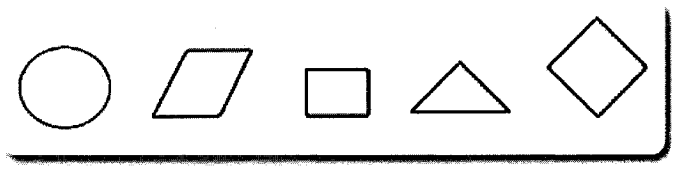


FIGURE 5.13. A training set consisting of basic shapes.

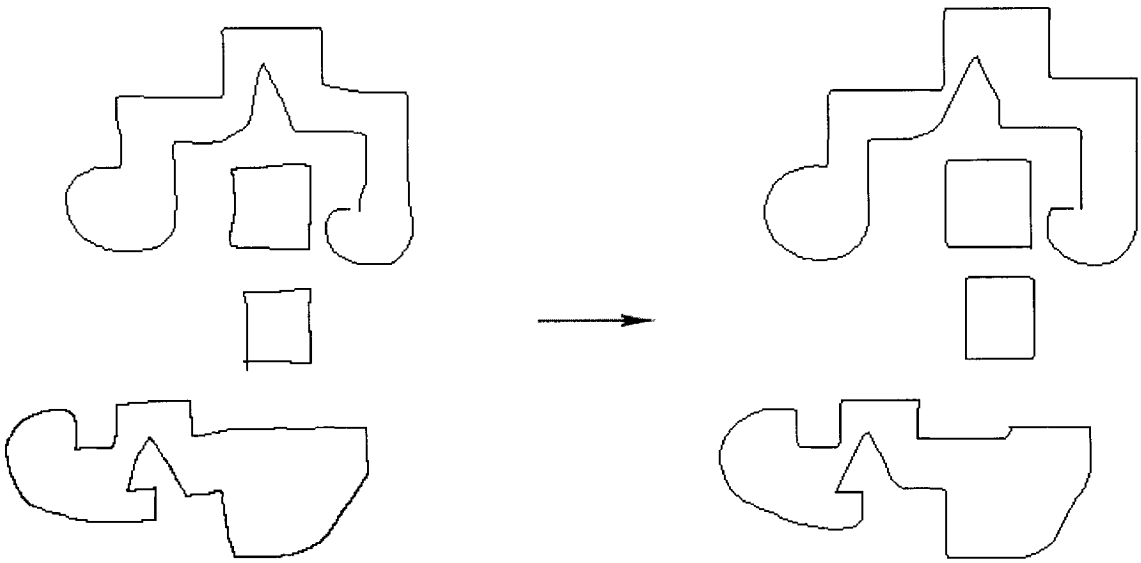


FIGURE 5.14. Synthesis results using the basic shapes training set.

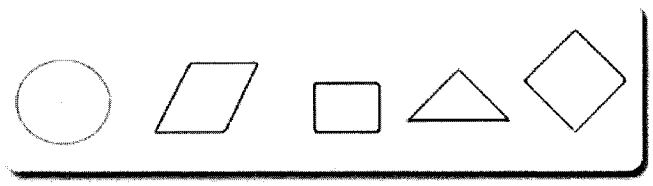
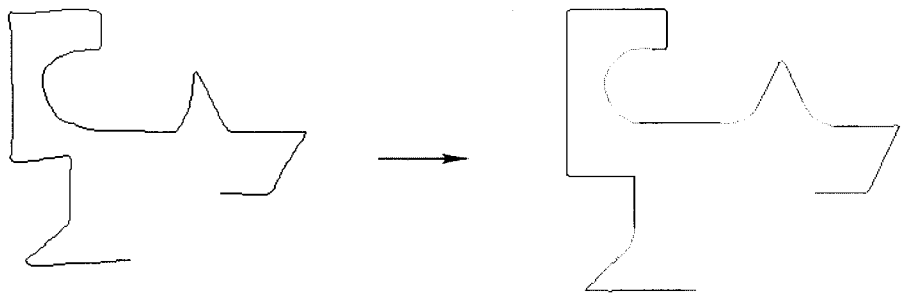
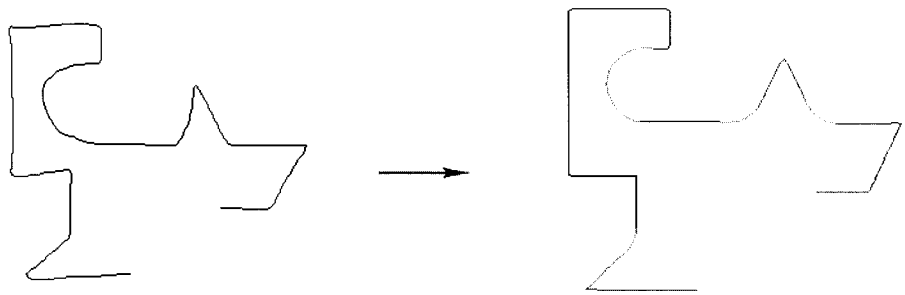


FIGURE 5.15. Training set with the color attribute.

Figure 5.25 shows a simple training set used to further demonstrate the degree of control when using the second-order representation (a stationary model is used). The results are show in Fig. 5.26. It can be seen in Fig. 5.26(a) that, starting from



(a) Without the example coherence regularization term.



(b) With the example coherence regularization term.

FIGURE 5.16. Results displaying the effect of the example coherence regularization term. When using the term, there are fewer transitions between training examples.

the top-left, the process is controlled to execute the desired turn, but when the input curvature is too low, the output curve follows a straight trajectory while the input slowly drifts away, an example of *input drift*. Figures 5.26(b) and 5.26(c) show the results when increasing the magnetism term. It can be seen how the output curve follows the input curve by having to perform a few turns that in the short term are divergent from the input. This training example is further used to learn the shapes for both a left turn and a right turn (the model is trained by traversing the curve from both end-points). Figure 5.27 shows the results with various setting for the magnetism term.

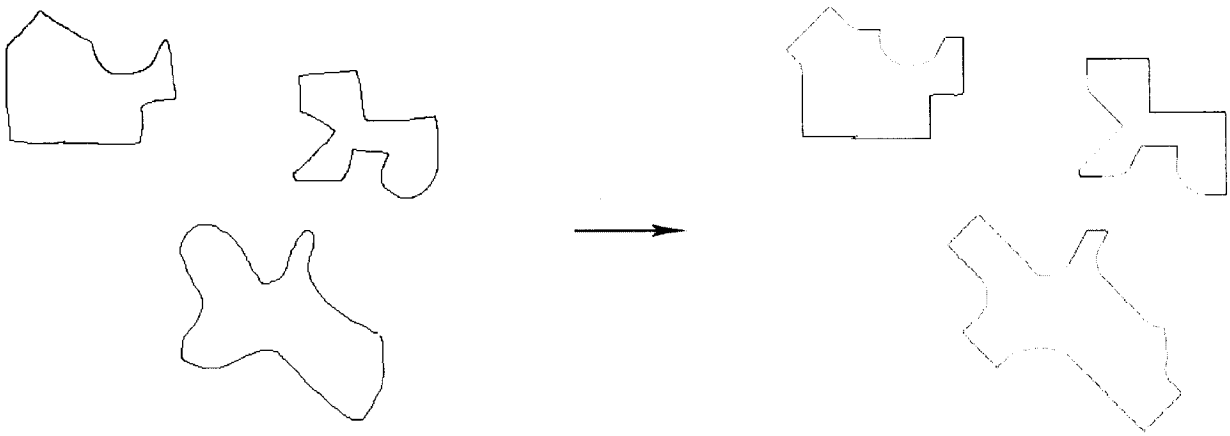


FIGURE 5.17. Example synthesis using the basic shapes training set.

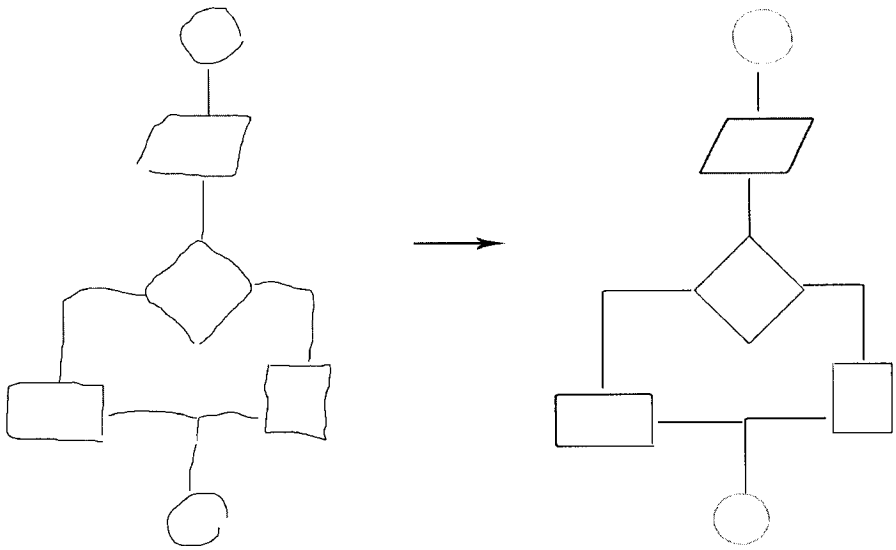


FIGURE 5.18. Example synthesis using the basic shapes training set.

8.7. Texture Fill. Figure 5.28 illustrates the results when including the fill-color attribute and Fig. 5.29 shows the texture that was used for the texture filling process illustrated in Fig. 5.30. It is easy to see that the texture is extrapolated from the contours to produce the desired full-color illustration.

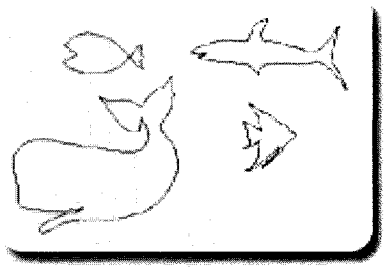


FIGURE 5.19. Training set with examples of fish. Control curves (not shown) are blurred versions of the refined ones.

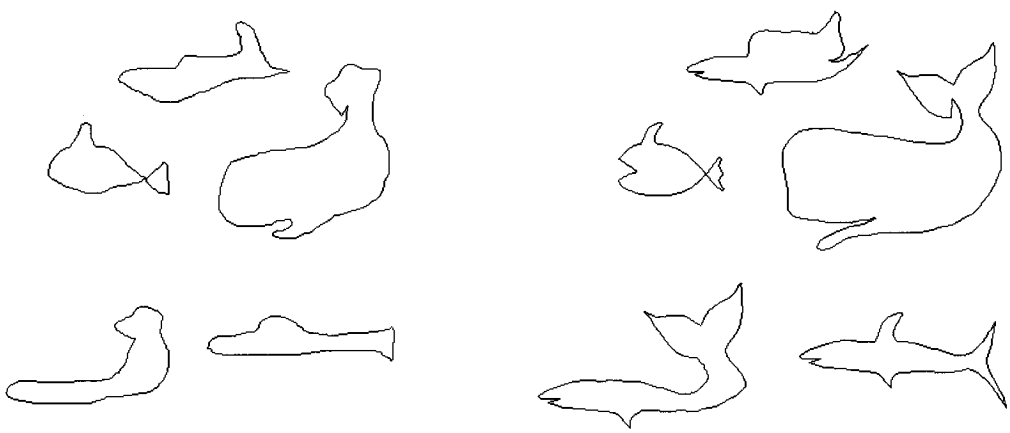


FIGURE 5.20. Example synthesis of fish shapes. The left shows the inputs and the right shows the results. Some results are exact instances from the training while others are segment mixtures.

8.8. Additional Examples. Figure 5.31 shows example synthesis with and without the multi-scale representation. It can be seen that the large scale features are not captured without using the multi-scale representation. In Fig. 5.32, the refinement of an entire sketch is illustrated. Each curve is refined by first manually selecting the desired training set and then applying the synthesis procedure using that set. Figures 5.33 and 5.34 show screen-shots of the application's user interface with more synthesis examples.

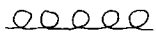
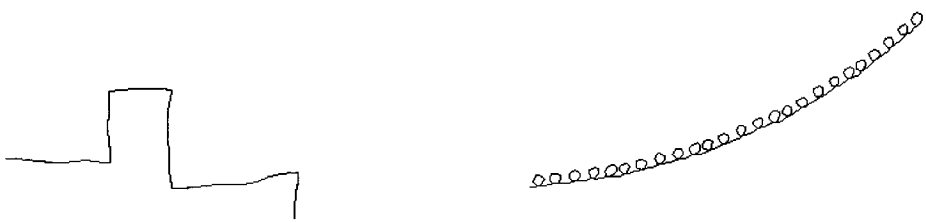


FIGURE 5.21. Training set consisting of a curl-like pattern associated to a simple control curve.

(a) Drift due to accumulated errors in orientation.



(b) Lack of control in position.

(c) Control asserted due to the magnetic term.

FIGURE 5.22. Synthesis of a *curl* pattern using the second-order shape descriptor. Left shows the input and right shows the synthesis results.

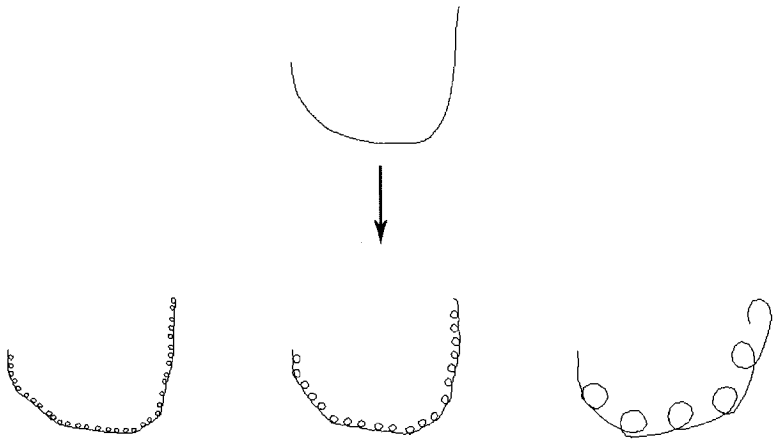


FIGURE 5.23. Synthesis results for three different sampling resolutions. Top shows the input, bottom left to right show the results when reducing the resolution (fewer input samples).

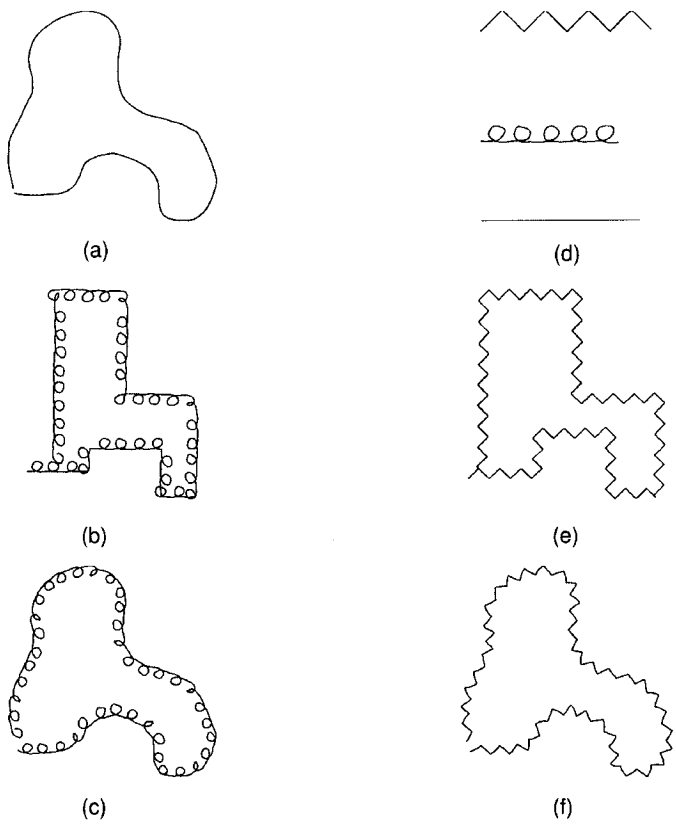
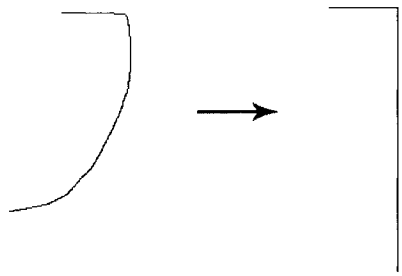
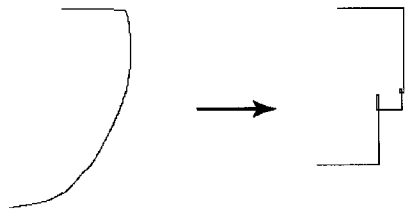


FIGURE 5.24. Examples demonstrating the difference between a first and second-order representation. Figure (a) shows the input curve and Fig. (d) shows the two patterns and the control curve (straight line segment). Figures (b) and (e) show the results using the first-order representation and figures (c) and (f) show the results using the second-order representation.

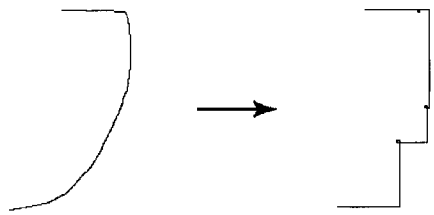
FIGURE 5.25. A training set consisting of a right turn (traversed from top-left to bottom-right). It is also used to learn the shape of a left turn (traversed from bottom-right to top-left).



(a) Input orientation drift due to low curvatures that are not found in training.

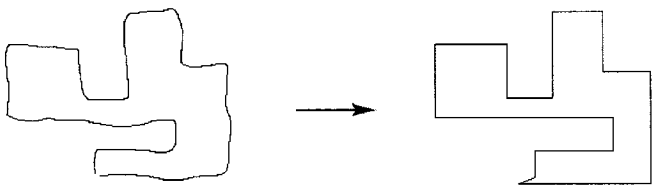


(b) Low magnetism weight. Note how the curve's position diverges from the input in order to make the right turns such that, in the long term, the output is closer to the input. A behavior not apparent when using a greedy strategy.

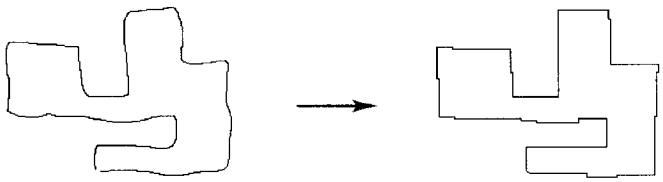


(c) Large magnetism weight.

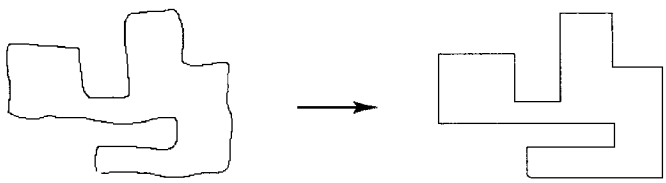
FIGURE 5.26. Effects that result when using a training set consisting of only a right turn.



(a) Synthesis without the magnetic term. The output curve fails to close properly.



(b) Synthesis with the magnetic term. The output remains near the input, but is too noisy.



(c) Synthesis with the magnetic term and the decay function applied to its weight. This preserves both longer line segments and the closure condition.

FIGURE 5.27. Results when using a training set consisting of left and right turns.

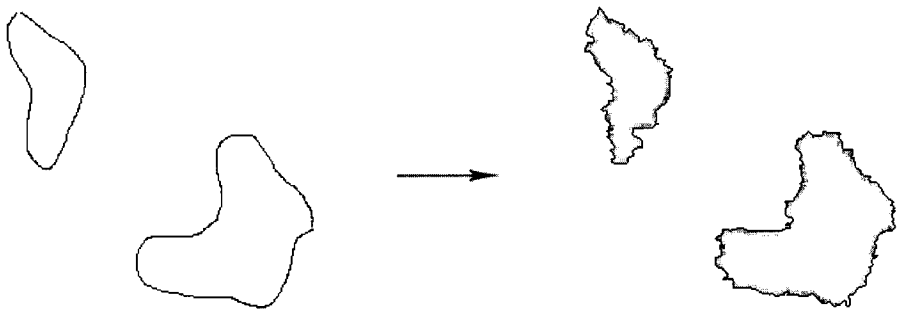


FIGURE 5.28. Synthesis of coastlines with texture seeds.

FIGURE 5.29. Texture image used for coastlines.

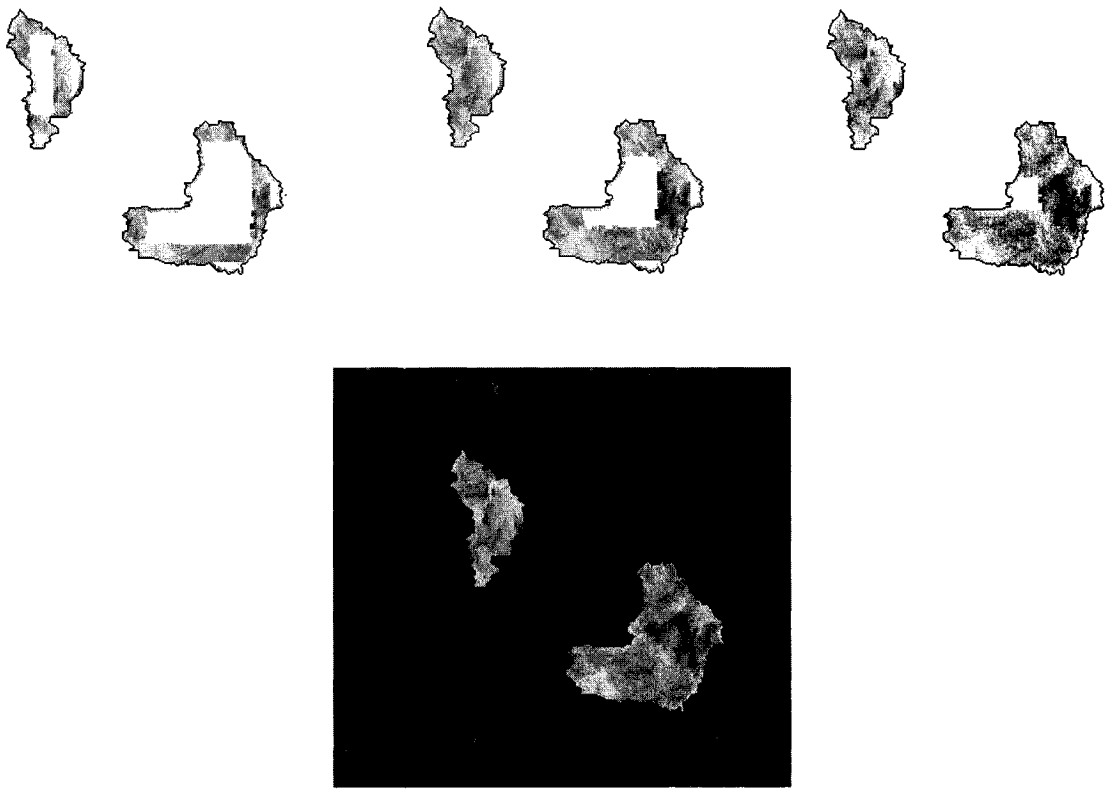


FIGURE 5.30. Texture filling process. The progression of the texture fill process is shown at the top, from left to right and the final result is shown at the bottom.

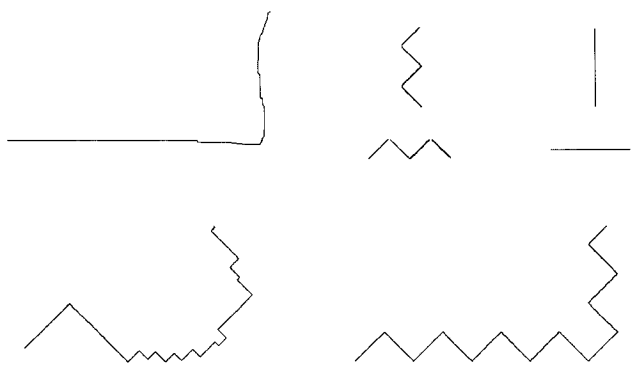


FIGURE 5.31. Example synthesis with and without the multi-scale representations. Top left shows the input, top right shows the training set, bottom left shows the result when the wavelet representation is omitted and the bottom right shows the result when the wavelet representation is included.

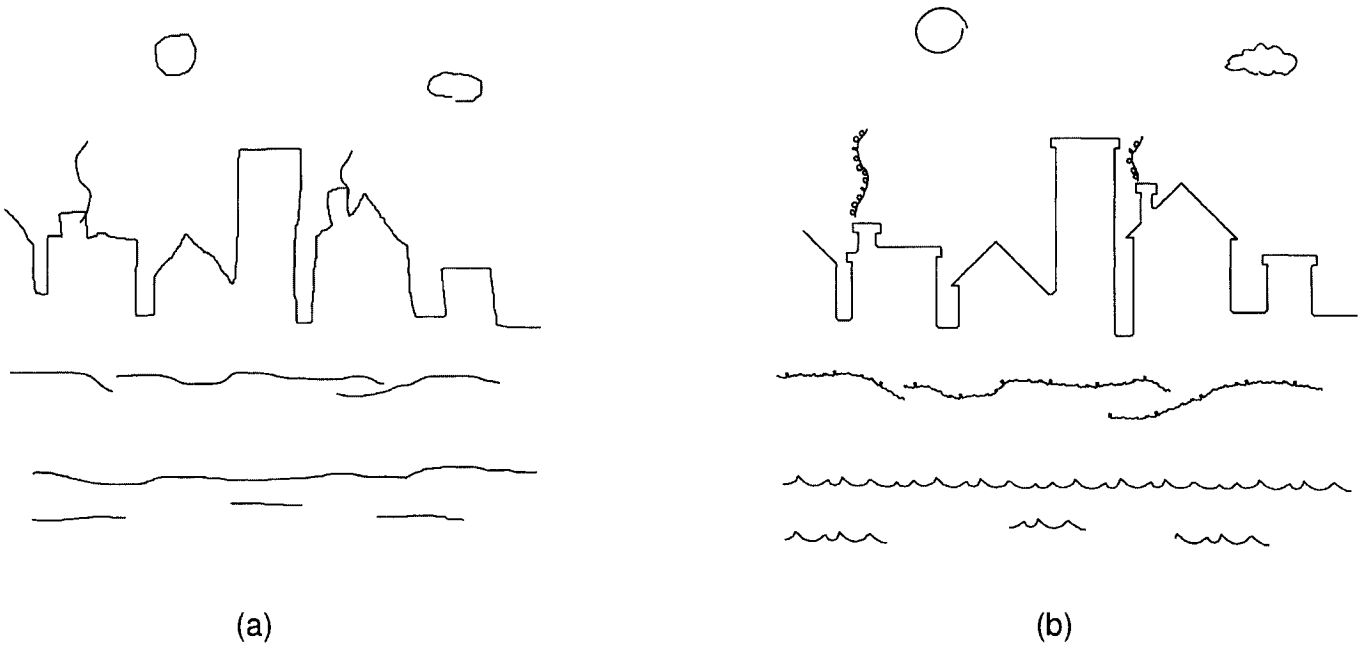


FIGURE 5.32. Sketch refinement using several different training sets (assigned manually). The left shows the input and the right shows the results.

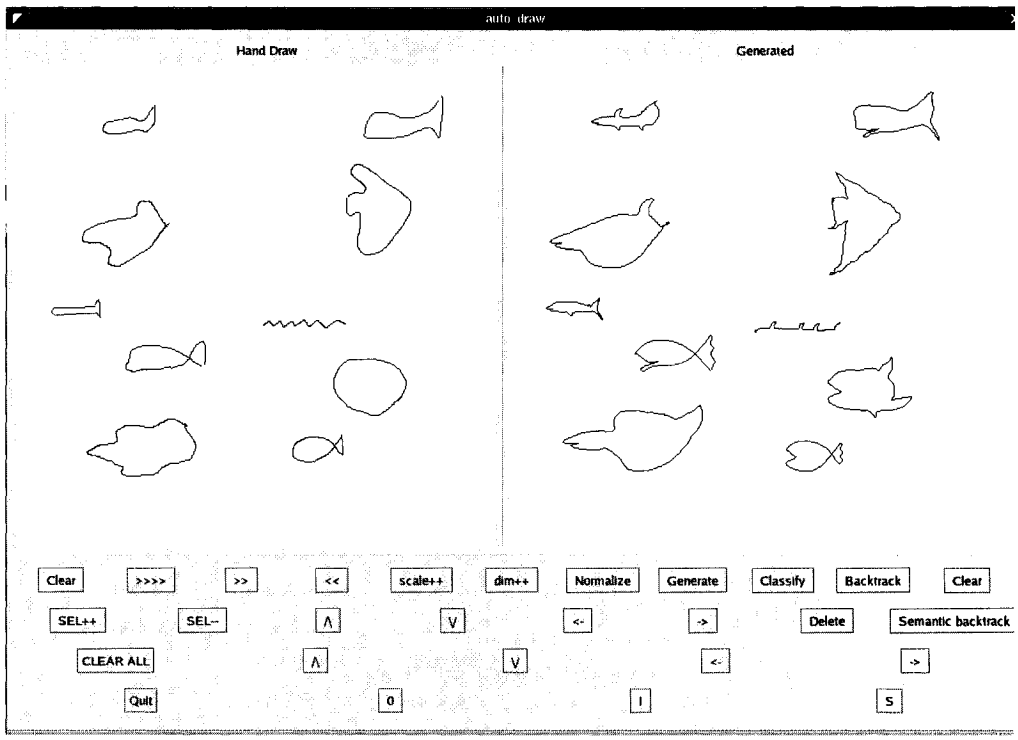


FIGURE 5.33. Screen-shot: more examples using the fish-shapes training set. Note that when the training set is not rich, input curves that do not resemble the limited set of segments produce odd results. Such results may be interesting in the realm of fiction!

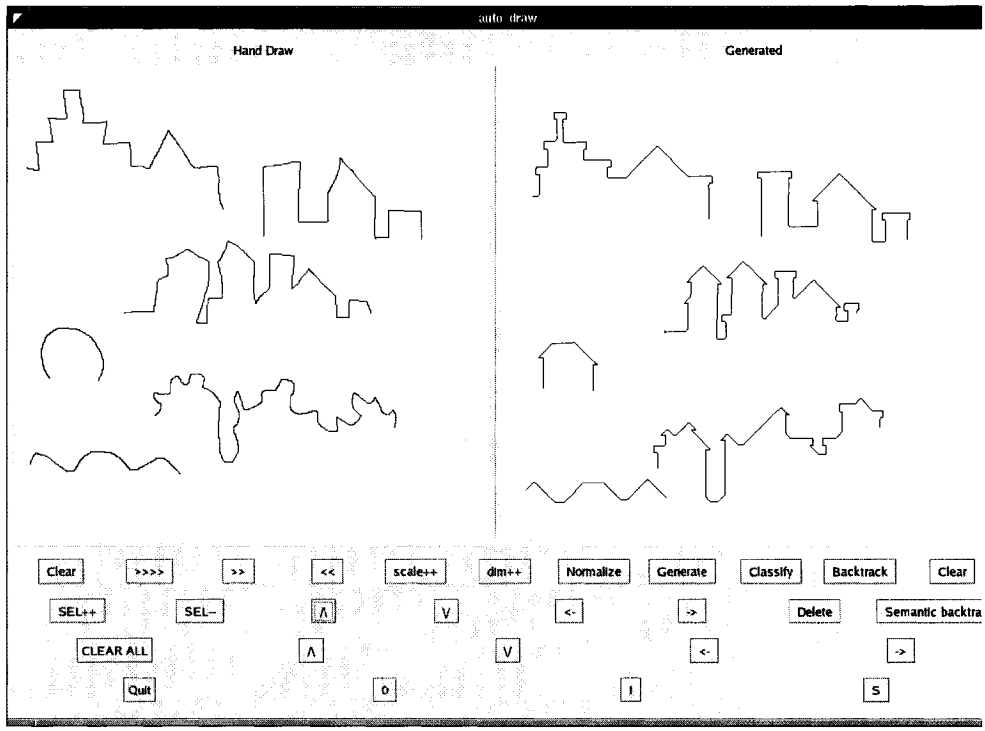


FIGURE 5.34. Screen-shot: synthesis using the roof-top training set shown in figure 5.10.

CHAPTER 6

Path Planning Application

A robot trajectory can be represented by a signal that identifies the positions a robot must sequentially follow in order to reach the desired destination. This chapter describes the application-specific framework customization required for producing such signals. While there are various new components introduced into the system, most of the methods presented for the sketching application are common to this domain. This chapter also presents the path synthesis results and exemplifies how the system attempts to predict *permissible* robot trajectories; paths, guided by an input *goal* trajectory, that avoid obstacles while maintain the learned constraints. The training set used in the experiments consist of example trajectories for non-holonomic motions.

1. Path Attributes and Parameters

The hidden layer of the HMM encodes constraints on the allowable sequence of positions a robot can traverse (the allowable trajectories). Using either a first-order or second-order representation, the states corresponding to this layer encode the shapes of segments from the allowable trajectories ($\mathcal{A} = \{\theta(t),\ \Delta\theta(t)\}$). As in the sketching application, there are 4 scales used for the multi-scale representation and the highest scale is set to have the most weight (twice the weight of the other scales). Unlike in the sketching application, where it is not always necessary to have exact sequential

consistency with the training set to achieve the desired visual effect, the sequential constraints learned for producing robot trajectories must be strictly enforced. The mixture variance is thus set to a small value (≈ 5 square degree), just large enough to allow for small discrepancies that may occur from quantization errors or minor large-scale inconsistencies. Further, in the path planning application, the absolute location of features in a trajectory does not impose a constraint on the desired output (at any point, the robot should have the potential to perform any maneuver), hence a stationary model is used. (Though it is conceivable to have a training set where stationarity is desired, for example, there may be a maneuver that is desired only at the beginning or end of the trajectory.) The translation step τ is set to one sample point, allowing for transitions to take place on a point-by-point basis.

The observation layer of the HMM encodes the expected control mechanism used to guide the robot in order to follow a goal trajectory. The states corresponding to this layer encode segments of the goal trajectory and include the following components:

- The shape of the goal trajectory
- The direction of the robot's axis

The goal trajectories are paths that the user (or high-level planner) produces to control the robot (typically excluding the complexities incumbent by the mechanical constraints of the robot). Each goal trajectory is associated with the more complicated and allowable trajectory (i.e. the control/refined curve coupling). Their shapes are represented using the same order and multi-scale representation as those used for the allowable trajectories. The direction of the robot's axis is an absolute orientation that identifies where the robot must face along the goal trajectory (i.e. the forward direction). It is represented by a first-order representation without the multi-scale components. The set of control attribute is thus defined as follows: $\mathcal{A}_o = \{\phi(t),\ \Delta\phi(t), \theta(t)\}$. The sigmoid blur parameters are set to the same values as those used in the sketching application as it is expected that a user will steer the process.

2. Regularization Terms

There are several components that contribute to what constitutes a valid path. First, the HMM must be taken into account such that the output is consistent with the training examples. Second, it is preferred that the generated paths stay near the input trajectory. Finally, the output curve should not go through or approach too close to obstacles in the environment. This combination of hard and soft constraints can result in complex paths that are otherwise difficult to determine efficiently using traditional analytical models. Further, very few existing planners are example-based. In summary, the components are:

- The learned constraints of the HMM
  - local shape consistency with training examples
  - control by the goal trajectory
- Distance to the goal trajectory
- Obstacles avoidance

Using the regularization framework describe in Section 4, these components are combined together to provide the desired control scheme.

To reduce the average distance between the goal trajectory and the synthesized one, the magnetic regularization term, describe in Eq. 5.3, is used. The regularization weight is empirically specified and is fixed over the entire curve. (One may suggest ways to set λ_3 based on the divergence exhibited between the control and refined curves in the training set.)

To avoid obstacles in the environment, an energy field is generated over free space and its values are used to bias the likelihood of candidate paths. The field is generated by applying a distance transform over the obstacles in the environment. A suitable function must result in high energies at regions near the obstacles and low energies at regions far form obstacles. As such, the energy of a state is updated by the following:

$$R_4\big(H_i(t)\big) = \max_{obs} \frac{1}{\sum_{l=0\ldots\tau} \big(x_{obs} - H_i(t)_{x(l)}\big)^2 + \big(y_{obs} - H_i(t)_{y(l)}\big)^2} \tag{6.1}$$

The energy of the state is augmented by the value of the field, calculated as the inverse of the average square distance between the position of points in the current candidate state and the position of the nearest obstacle (the obstacle that results in the maximum field value). At positions close to or on the obstacles, the energy approaches infinity while at areas further away the energy decays to zero. The regularization weight λ_4 controls the degree of influence the obstacles have on the solution. Large values will coerce the robot maintain a large distance from the obstacles while small values will allow the robot to reach closer to the obstacles, allowing it to traverse through narrower free-space regions.

For efficiency, the environment is preprocessed by generating the field in advance over a grid. To evaluate the energy for a state, the auxiliary parameters that identify the state's Cartesian co-ordinates are used to index the grid, providing random access to the gird values. Figure 6.1 shows a field generated for a sample environment.

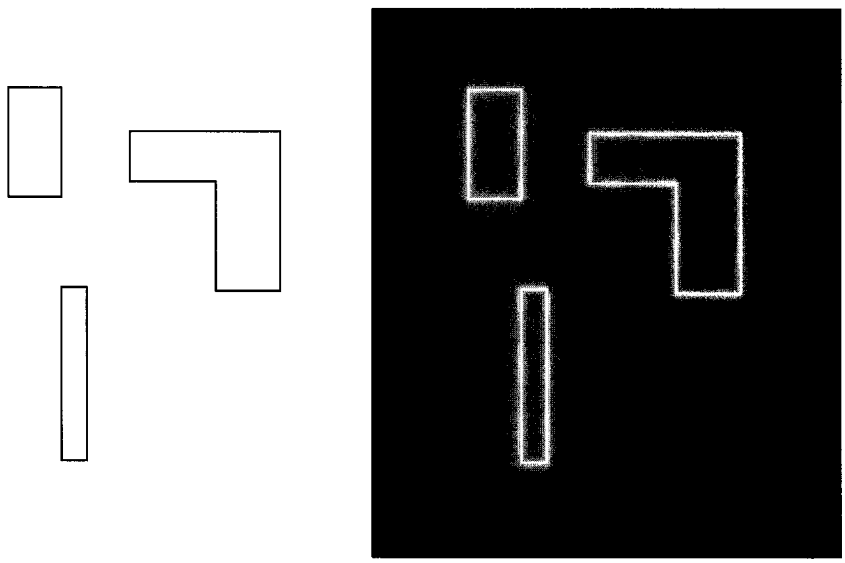


FIGURE 6.1. Example energy field. The left image shows the environment and the right image is a plot of the energy field.

3. Dynamic Sampling

The obstacle avoidance and magnetic regularization terms often result in competing factors. In order to avoid obstacles, the synthesized path may divert from the input more than any divergence exhibited between the control and refined curves in the training set. This results in a discrepancy between the arc-length that is required to reach the goal and the arc-length of the input trajectory. (Eq. 5.4 only compensates for arc-length discrepancies that exist in the training set.) In such cases, the synthesized path may not converge to the goal trajectory.

To address this issue, during the propagation and conditioning steps, instead of using the fixed sampling rate to determine the current sequence position, each candidate state is synchronized with the input dynamically. This is accomplished by computing the appropriate sequence position of the input trajectory for each candidate state independently. As such, at any given iteration, a different input conditional can be applied to different candidate states, depending on where on the input path the resulting candidate state is synchronized to.

The sequence position of the input trajectory is identified by the projection of candidate path onto the input trajectory (Fig. 6.2). This is performed by computing the dot product of the vector v_1 from the candidate path's starting point to the current point with the vector v_2 from the input's starting point to the the last input sample point used. The length of the projection $|q|$ is computed as follows:

$$|q| = \frac{v_1 \cdot v_2}{|v_1|} \tag{6.2}$$

If the dot product is negative or if the length of the projection is smaller than $|v_1|$, then the same sequence position l is used for the next iteration ($l_{i+1} = l_i$), otherwise the proceeding sequence position $l + 1$ is used ($l_{i+1} = l_i + 1$). This sampling scheme results in a sequence progression that either waits for the output to catch up to the input (if ahead) or attempts to catch up with the output (if behind) and, to help ensure progress and avoid cycles, it never back-steps to earlier points.

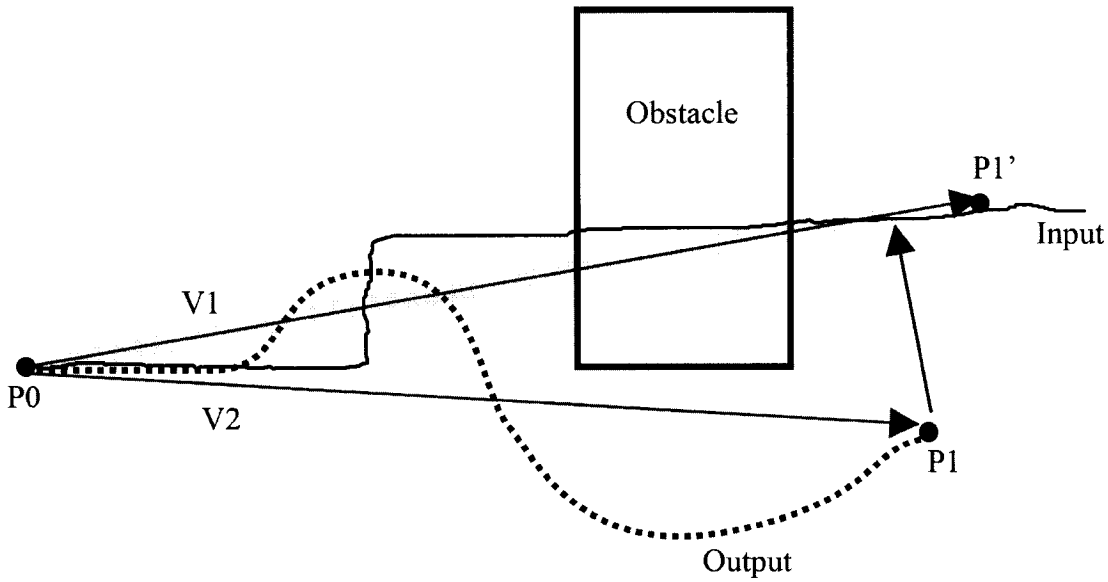


FIGURE 6.2. Projection of the vector $V2$ (from the output trajectory) onto the vector $V1$ (from the input trajectory). The projection is used to determine which sample point along the input curve should be used for applying the input conditional.

The system then iterates over the propagation and conditioning steps until either all candidate paths are within some acceptable distance to the last point of the input trajectory or a maximum number of iterations is reached (the default maximum is twice the number of samples from the input). This maximum iteration limit is required to avoid infinite propagations (there is no guarantee that all, or any, candidates paths will converge). While decoding the model, there will likely be cases where some candidate states result in paths that have converged while others have not. In such cases, only the candidates that have not reached the goal must be propagated further. An auxiliary parameter is used to identify if a state has reached the goal. The candidate states that have reached the goal are labeled as leaf states and excluded from the next propagation step (Eq. 4.8 is modified to exclude all leaf states).

Selecting the best trajectory then consists of choosing the leaf state with maximum likelihood and backtracking. Because likelihood comparisons are made between

candidate states from different iterations (resulting from longer or shorter sequences), the probability normalization constants must be considered in the likelihood computation. The normalization constants used at each iteration are divided by the sum of all normalization constants and then the likelihoods are multiplied by their corresponding value.

4. Experimental Setup

Experiments have been performed using the sketching application's Graphical User Interface as described in chapter 5. A modification to the interface was performed in order to provide a method for drawing simulated obstacles in the environment and computing the energy field. The goal trajectories are hand-drawn and the desired robot forward directions are manually entered by specifying the corresponding vectors at the desired locations. The lack of forward directions in some or all sample points does not have an adverse effect on the output as only the input components that are actually entered in the system are applied (Eq 4.11). A training set that simulates the actions of a robot subject to a bounded turning radius constraint is used and the results are evaluated subjectively.

5. Results

Figure 6.3 shows a manually constructed training set used to learn non-holonomic motion constraints. The refined trajectories are shown on the left and the expected goal trajectories are shown on the right. For the forward directions, the sequence of tangent angles along the refined trajectories are used (the refined trajectories are used for both the hidden states and the secondary dimension of the observation states). This set is used to train a stationary model using a first-order representation of the curves and all of the regularization terms described in this chapter and the previous chapter are included.

Figure 6.4 shows an input trajectory that is refined using several training sets; the non-holonomic training set and the sweep-like and curl-like training sets from

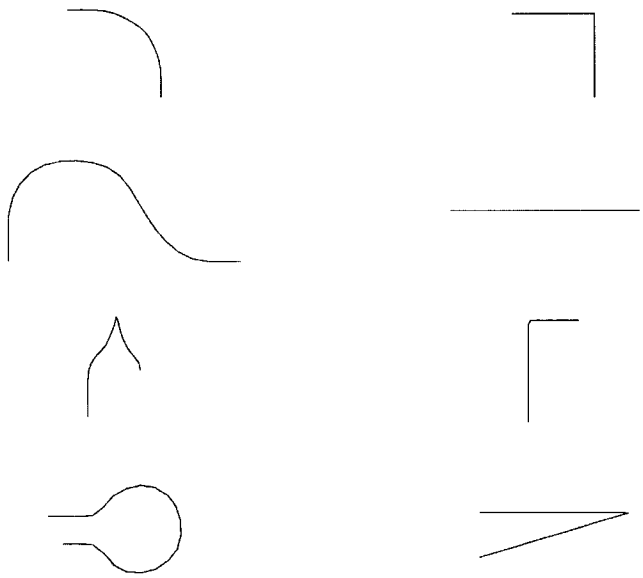


FIGURE 6.3. A training set with example paths for non-holonomic motions. Paths on the left display the constrained motions while paths on the right display the associated unconstrained goal path. The forward directions used for the secondary control attribute consist of the tangent angles along the constrained motions (left). The full set consists of the above set at four orientations to form a rectilinear set.

previous chapter. It can be seen how the resulting paths form analogies to the input path and learned styles. They follow the overall goal trajectory and remain locally consistent with the training set.

Figure 6.5 shows two more examples that demonstrate the results when using the non-holonomic training set. It is easy to see that the generated paths follow a smooth trajectory while preserving the desired overall trajectory. It can bee seen in the left example that the bottom right turn was synthesized as an extended loop about that corner rather than the typical smooth turn (as shown in training). The system takes into account the fact that the proceeding segment consists of a second turn, immediately after the first, limiting the space available for performing the standard smooth turn and thus the tighter loop maneuver is required.

In Fig. 6.6, the input consists of a goal trajectory with forward directions specified at various points along the goal trajectory (indicated by arrows). The results show how the predicted output paths attempt to follow the overall goal trajectory

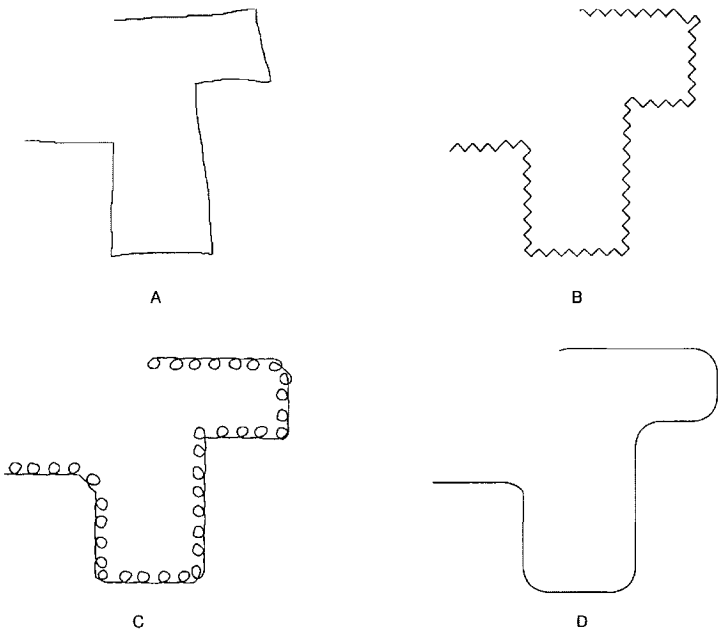


FIGURE 6.4. The examples above show the input path (A) and the synthesized paths (B,C,D) using three training sets. The three training sets consist of a zig-zag pattern for a sweep motion, a curl-like pattern for a narrow-beam sensor scan and the bounded turning radius pattern.

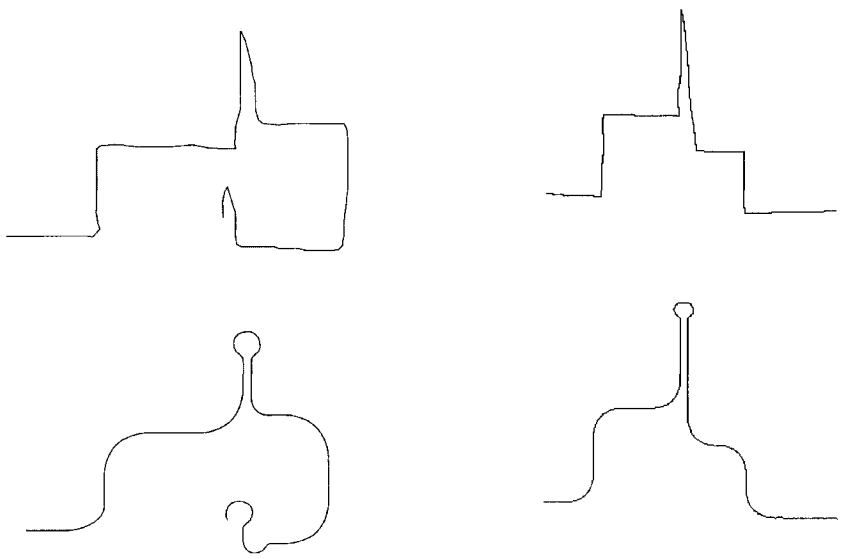


FIGURE 6.5. Example path synthesis using the non-holonomic training set. Top shows the input and bottom shows the resulting path

while performing the required maneuvers that align the robot to the desired forward directions. Figure 6.7 shows an example that demonstrates the importance of the magnetic regularization term. It can be seen how the output remains close to the input only when the magnetic regularization term is used.

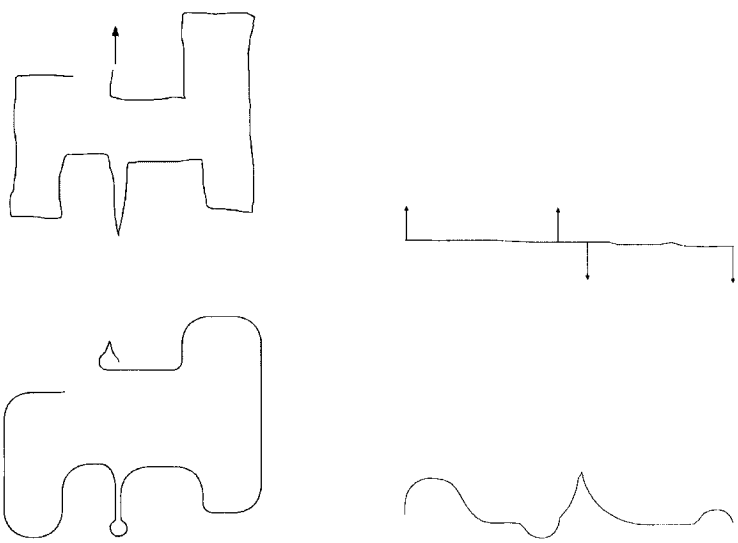


FIGURE 6.6. Results when including the forward direction as an input condition (indicated by arrows). Top shows the input and bottom shows the generated paths. The cusps indicate a direction reversal.

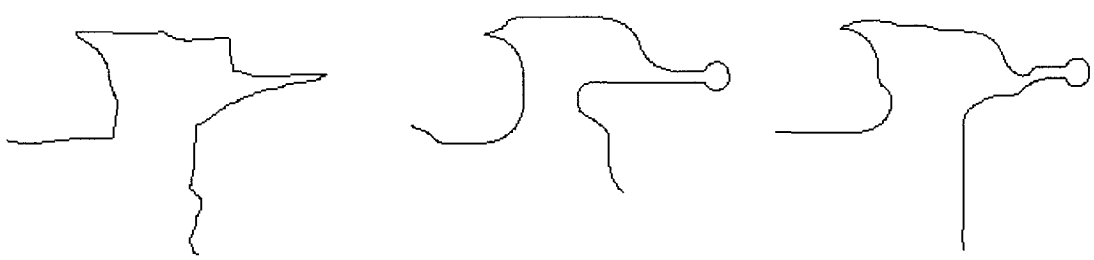


FIGURE 6.7. Results with and without the magnetic regularization term. The left path shows the goal trajectory, the middle shows the resulting output without the magnetic regularization term and the right path shows the output with the magnetic regularization term. The cusps indicate a direction reversal.

In Fig. 6.8, the results of several example syntheses with the obstacle avoidance term are illustrated. It is easy to see how the generated paths avoid the obstacles

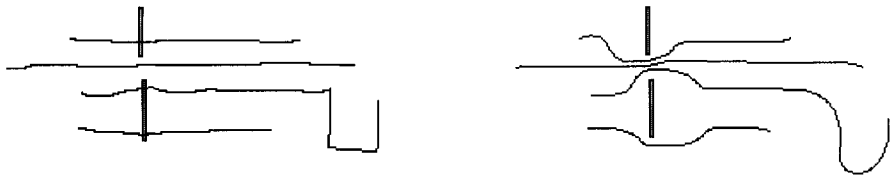


FIGURE 6.8. Example path synthesis going through a narrow region. Left shows the input, right shows the output and the shaded areas show the obstacles.

while roughly following the goal trajectory. Figures 6.9 and 6.10 show the path planning results in two simulated environments. The goal trajectory directs the robot to traverse either too close to or through obstacles but the resulting paths avoid the obstacles and preserve the learned non-holonomic constraints.

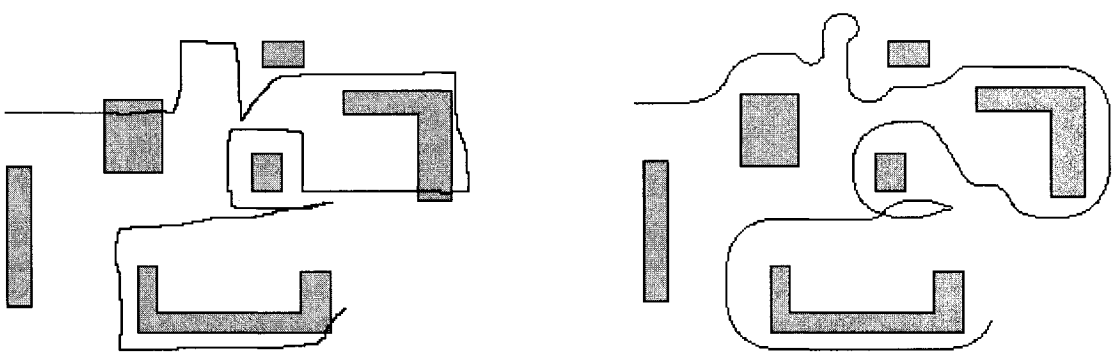


FIGURE 6.9. Example path synthesis with obstacle avoidance. Left shows the input and right shows the output.

Figure 6.11 demonstrates the effect when modifying the weight of the obstacle avoidance regularization term and applying the dynamic sampling technique. It can be seen that with a large regularization weight, the paths stay further away from the obstacles while a small weight allows the paths to go through narrow regions.

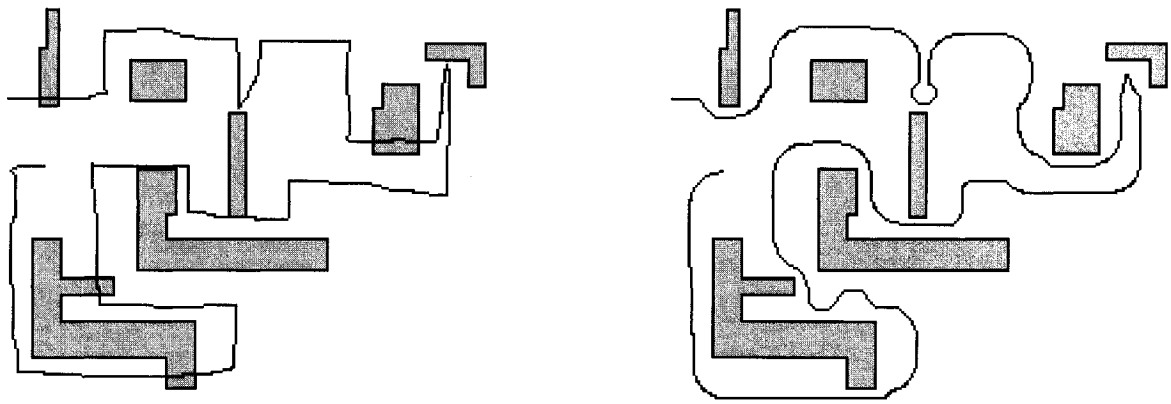
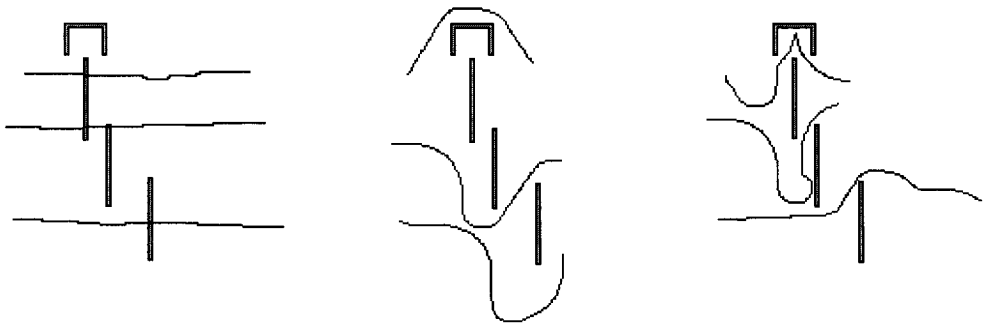


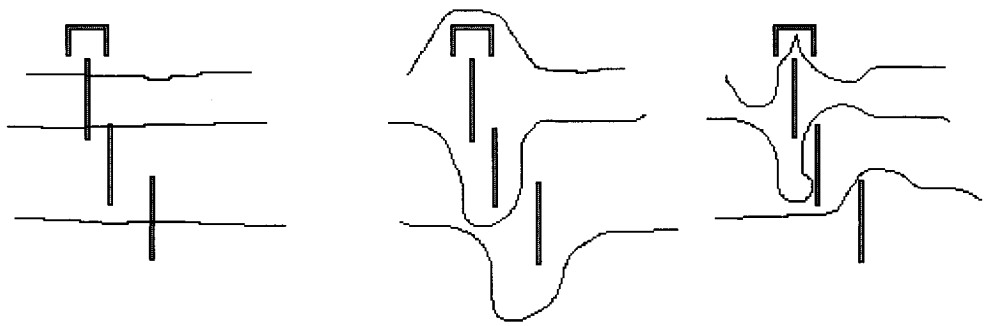
FIGURE 6.10. Example path synthesis with obstacle avoidance. Left shows the input and right shows the output.

Furthermore, because the output path diverts far from the input path, there are not enough sample points to reach the destination. The dynamic sampling technique overcomes this issue by compensating for the missing sample points.

Figure 6.12 illustrates an example where the system does not find a solution that converges to the final goal. The large obstacle obstructs a direct path and in order to reach the goal the path must divert far from the input, requiring a number of samples that is beyond the specified maximum. The results when using the second-order representation are shown in Fig. 6.13. It can bee seen that the path can be generated along arbitrary directions. To achieve this results, the parameters of the sigmoid are adjusted to impose the input conditional more heavily. When using the absolute angles, the bias from the input is accumulate over successive iterations while when using the second-order representation, the input variation is narrowly localized (i.e. when performing a turn, the input conditional is distinct only at the corner) and a single match must sufficiently bias the distribution. Though when the match is not sufficient, the magnetism term helps maintain the overall shape. This can be seen when comparing the first (top-left) and fourth (bottom-right) u-turns to the other



(a) Synthesis with a fixed sampling rate.



(b) Synthesis with dynamic sampling.

FIGURE 6.11. Example path synthesis going through a narrow region. Left shows the input. The middle shows the output using a large value for λ_4 and right shows the output with a small value for λ_4 (the obstacle avoidance term). It can be seen that due to the large divergence, the output trajectory lags behind the goal trajectory. This is compensated for by the dynamic sampling technique.

ones. Those u-turn segments are not synthesized in direct accordance to the training data but rather some other maneuver is used to follow the trajectory.

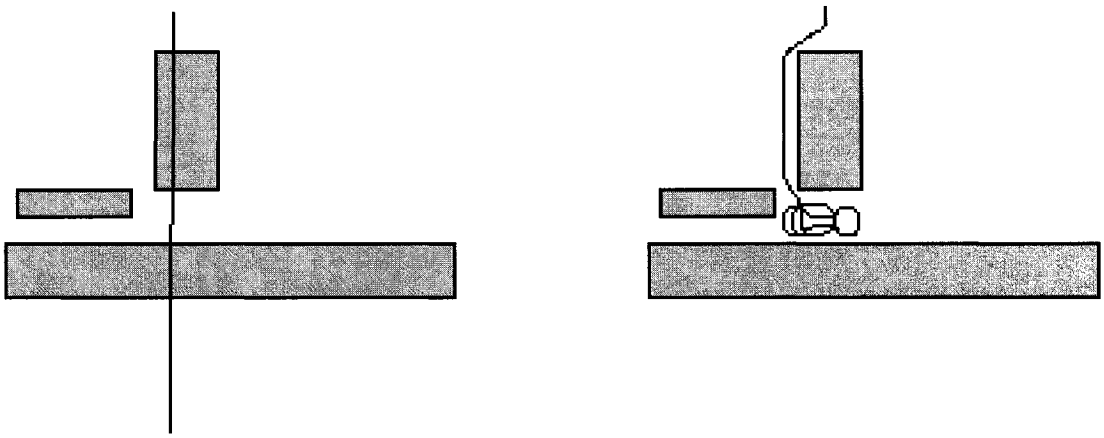


FIGURE 6.12. An exmaple where the robot does not reach its goal. Left shows the input and right shows the output.

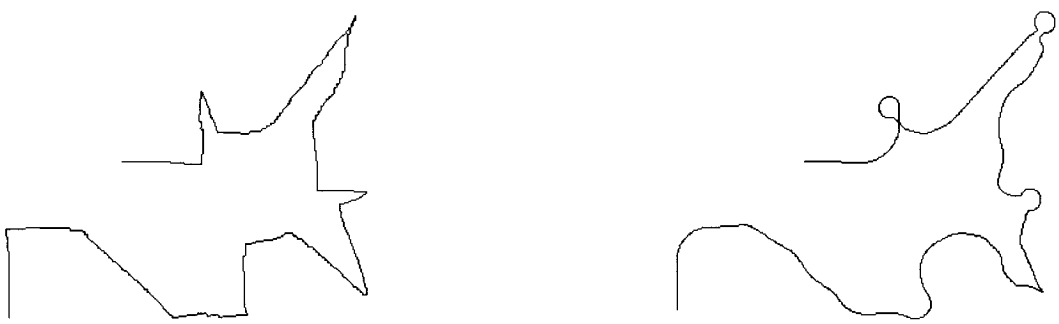


FIGURE 6.13. Example synthesis using the second-order representation. Note that when using this representation, the curves have few distinguishing values. In some cases, the input matches well and the desired features are generated (the learned u-turn maneuver) while in other cases, the match is not sufficient but the magnetism term helps steer the process (in cases where the learned u-turn should have been generate another maneuver was used).

CHAPTER 7

Classification of Curves

In previous chapters, it is assumed that the validity of applying a HMM on an input curve is subjectively determined by a human operator. This chapter describes a framework for automatically classifying those curves, identifying which set of training examples the curves best belong to. This allows the system to objectively determine, in a maximum likelihood sense, the compatibility between models and curves.

The applicability of the classification framework is exemplified by making two extensions to the sketching application: one for automated model selection and the other for automated curve extraction. For the automated model selection, when a user draws a curve, the system attempts to determine the most likely HMM that should be applied to refine the curve (and thus implicitly recognize it). This is accomplished by taking into account both the shape of the input and its context in the sketch (its relationship to other curves). The approach consists of iteratively evaluating and decoding the Hierarchical Hidden Markov Model described in chapter 4. For the automated curve extraction, when an input curve is rendered in the form of an image, the system attempts to automatically extract from the image the most likely curve that belongs to a particular model. This is accomplished by evaluating all possible curves in the image with respect to the HMM and selecting the one with maximum likelihood. Results and discussions are presented for each of these applications.

1. Scene Refinement Model

A *scene* consists of a set of curves drawn in accordance with a well defined set of rules that constrain the types of curves based on their spacial position and sequential ordering. The sequential constraints in a scene can suggest, for example, that backgrounds must be drawn first, followed by other objects which in turn can be followed by other objects, each drawn over the previous. They can represent typical drawing habits, such as when users draw the profile of a cartoon face, the forehead will most likely be followed by a nose, then a mouth and a chin. Conversely, the scene can be sequentially unconstrained, such that every curve can follow any other type of curve. The spacial constraints are applied to further restrict the types of curves that can be drawn based on their relative locations. They can suggest, for example, that some types of curves can be drawn above or below other types. High-level quantifiers are used, such as *above, below, left, right, in and out*, determined relative to the edges and center of the bounding box of each curve.

A scene-level HMM encodes such constraints by restricting the types of refinements that can be applied on curves in a sketch (the curve-level HMMs). Recall from Eq. 3.11 that the set $\mathcal{G}^0$ is the set of all HMMs applicable to a scene. A *scene refinement model* Λ^1 is then an augmented HMM (Eq. 4.1) where the states correspond to curve-level models in $\mathcal{G}^0$. In this fashion, all of the methods described in Chapter 4 are applicable to the scene-level models. This section further describes the approach take to learn and apply scene-level refinements.

1.1. Learning Scene Constraints. A scene-level HMM Λ^1 is trained using a graph $\mathcal{Y} = \{\mathcal{D}, \mathcal{E}\}$ that defines the high-level scene constraints (Fig. 3.6). The nodes $\mathcal{D}$ of the graph refer to the HMMs in the curve-level and an associated position (above, below, left, right, in or out). For example, if there are three models, one to produce grass, one to produce trees and one to produce clouds, there could be as many as eighteen nodes. The edges $\mathcal{E}$ of the graph include weights that identify 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 is drawn above a cloud is very small.

1.2. Hidden States and Transition Matrix. The hidden states of the scene-level HMM represent nodes from the graph $\mathcal{Y}$ and the multi-dimensional state space can easily accommodate for both attributes (model and position). Each curve-level HMM in $\mathcal{G}^1$ is manually assigned a unique *model label* L that is encoded in the state's first dimension. For efficiency, a state's second dimension is used to encode all of the allowable positions using a six bit number where each bit refers to a location identifier (i.e. the first bit can represent *up*, the second *down* etc., using this value as a bit-mask, simply applying a bitwise OR operation on the position of the curve can determine the curve's positional validity). The likelihoods for the transition matrix M^1 are captured directly from the edges of the graph.

1.3. Observation States and Confusion Matrix. The observation states of the scene-level HMM encode the observed positions of curves relative to one another (an observation consists of an input curve drawn at an observable position relative to the previously drawn curve). The confusion matrix B^1 is the Identity matrix such that the observation states correspond directly to the hidden states. Recall from Equation 4.11 that although the observation states refer directly to the hidden states (and hence must be two dimensional), the first dimension is implicitly disregarded as only the curve position is used in the input observation (the type of curve being drawn is not an observable).

1.4. Additional Model Parameters. All scene-level HMMs assume a stationary model as enforcing a constraint on the absolute sequential position of curves can become overly restrictive, excessively reducing the number of scenes that can be produced. There is one regularization term embedded in the scene-level model and it is used to take into account the shape of the curve (discussed further below). The translation step is set to one (iterating on a curve by curve basis) and the multi-scale

representation in not used (it is assumed that it is sufficient to only consider the immediate neighbor in the sequence of drawn curves).

It is expected that the exact position of curves are given, thus there is no input blurring applied (the sigmoid parameters are set to very small values). Further, the similarity function in Υ_o is modified such that a bitwise OR operator is used to determine if two states are similar (recall that the position identifier is represented by a six bit number). Finlay, note that the curve-level HMMs are labeled arbitrarily and the labels are not meant to imply a distance metric between models (identifying the similarity between different types of models is an open problem). Therefore, state blurring is also not applicable on the hidden states (the mixture variance is set to a low value).

1.5. Scene Refinements. The scene refinement process consists of first determining the most likely sequence of curve-level HMM models that should apply to each curve, then refining the individual curves using the associated models. Given a sequence of K curves $\Phi(0), \Phi(1), \ldots, \Phi(K)$, a set $\mathcal{G}^0$ with N curve refinement models $\mathcal{G}^0 = \{\Lambda_1^0, \Lambda_2^0, \ldots \Lambda_N^0\}$ and a scene-level HMM Λ^1 trained under a particular scene graph $\mathcal{Y}$, the most likely sequence of curve refinement models that apply on each curve must be determined:

$$\max_{\Lambda_1^0 \ldots \Lambda_N^0} p\{\Lambda^0(0), \ldots, \Lambda^0(K) \mid \Phi(0), \ldots, \Phi(K), \Lambda^1\} \tag{7.1}$$

There are two criteria that must be considered when solving for this maximum:

- The high-level scene constraints.
- The similarity between an input curve $\Phi(k)$ and the models' training curves.

These two criteria are combined using the regularization framework such that the scene-level HMM Λ^1 is decoded with a regularization term that measures the compatibility of the candidate models with the curve's shape.

1.6. Regularization Term for Evaluating Model Compatibility. The scene-level regularization term R^1 is developed to bias the distribution toward models

that are more compatible with the input curve. The value is computed over all curve-level models as a function of the input curves. For each of the K input curves, a vector is used to represent the log-likelihoods of all models in $\mathcal{G}^0$ where in the resulting vector sequence $R^1(0), R^1(1), ..., R^1(K)$, the value $R^1_n(k)$ is the log-likelihood that the model Λ^0_n can generate the observation sequence corresponding to the input curve $\Phi(k)$. This likelihood is computed by applying the Evaluation algorithm over all HMMs in $\mathcal{G}^0$ using the corresponding input curve. For the k^{th} curve, a refinement model Λ^0_n is evaluated by iterating over the curve's arc-length and computing the following:

$$\psi'_{\Lambda^0_n}\big(H_i(t)\big) = \sum_{H_j} \Big(p\{H_i(t) \mid H_j(t-1), \Lambda^0_n\} \psi_{\Lambda^0_n}\big(H_j(t-1)\big) \Big)$$
$$\psi_{\Lambda^0_n}\big(H_i(t)\big) = p\{O_{in,k}(t) \mid H_i(t), \Lambda^0_n\} \psi'_{\Lambda^0_n}\big(H_i(t)\big) \tag{7.2}$$

where $O_{in,k}(t)$ corresponds to the observation for the k^th input curve at sequence position t (i.e. $\Phi(k,t)$) and $H_i(t)$ is the candidate hidden state for the curve-level model Λ^0_n. The likelihood that a model Λ^0_n can generate the observation sequence corresponding to the curve $\Phi(k)$ is computed by taking the the sum over all states in $\Psi_{\Lambda^0_n}(T_k)$. The value for the regularization term is computed as the logarithm of this sum:

$$R_n(k) = \log\Big(\sum_{H_i} \psi_{\Lambda^0_n}\big(H_i(T_k)\big) \Big) \tag{7.3}$$

where T_k is the sequence length of the k^{th} curve. This procedure is almost identical to the steps for decoding the HMM (Eq. 4.8). Instead of choosing the maximum previous state at each step of the process, the sum the probabilities of all matching states is used. That is, all of the possible ways that the model can be used to synthesize the curve are considered and the accumulation of the individual likelihoods is used as a measure of its total likelihood.

1.7. Probability Normalization. While the probability vectors are normalized at each iteration of the decoding algorithm, in the evaluation algorithm the compound probabilities over the entire curve are required (Eq. 7.2). However, for long

curve segments, the probabilities may reach very small values and become difficult to store. Thus, at each iteration, the probability vector $\Psi_{\Lambda_n^0}(t)$ is normalized and the normalization constant $c_{\Lambda_n^0}(t)$ is stored. Once all models have been evaluated, the normalization constants at each iteration are themselves normalized over all models then compounded over the entire sequence as follows:

$$c'_{\Lambda_n^0}(t) = \frac{c_{\Lambda_n^0}(t)}{\sum_i c_{\Lambda_i^0}(t)} \tag{7.4}$$

and

$$C_{\Lambda_n^0} = \prod_t c'_{\Lambda_n^0}(t) \tag{7.5}$$

where $C_{\Lambda_n^0}$ is the compound normalization term used to determine the candidate model's likelihood. Further, since each curve-level model is customized with potentially different parameter settings, their values must also be taken into account. The mixture variance must be included as a normalization constant for the probability vector (divide by σ). The state's energy is also normalized using the sum of the regularization constants. It is assumed that the same sigmoid parameters are used for all training sets (the sigmoid parameters change as a function to the user) and that the translation step is the same for all models in G^0 (the sequence length is the same for each model).

1.8. Decoding the Applicable Curve Refinement Models. Once the vectors $R^1(0), R^1(1), ..., R^1(K)$ are computed, they are used in decoding the scene-level HMM Λ^1. This is accomplished using the same approach taken for decoding the curve-level HMM (Chapter 4). First, the initial distribution over the hidden states $\Psi_{\Lambda^1}(0)$ is assumed uniform and the first regularization vector $R^1(0)$ is used to bias $\Psi_{\Lambda^1}(0)$ as follows: $log\big(\psi_{\Lambda^1}(H_i^1(0))\big) + R^1_{H_i^1(0)_L}(0)$ where $H_i^1(0)$ is a hidden state in the scene-level model and $H_i^1(0)_L$ is the label that corresponds to a curve-level model. The distribution is then propagated using the transition matrix M^1, biased using the input observation (the location of the next drawn curve relative to the previous) and regularized again using the next regularization term. Once this is performed over all

input curves, the curve-level model that has maximum likelihood is selected and then the backtracking procedure is executed. The result is a solution for Eq. 7.1, selecting the most likely sequence of refinement models that apply on the drawn curves. Each curve is then decoded using the associated refinement model.

2. Scene Refinement Results

Figure 7.2 shows the results for generating cartoon facial profiles. In this example, the input scenes consist of sequences of curve segments corresponding to the cartoon components (forehead, nose, mouth, chin and hair). It is assumed that the user has prior knowledge on the way in which a cartoon face is segmented and draws the curves accordingly. There are five curve-level HMMs used where each HMM is trained using six examples of each segment. Each example also includes a supplementary attribute for curve thickness. Figure 7.1 shows the graph used to train the scene-level HMM (there are no positional constraints imposed on this model). It is easy to see from Fig. 7.2 that the curve segments draws by the user are refined using the appropriate model.

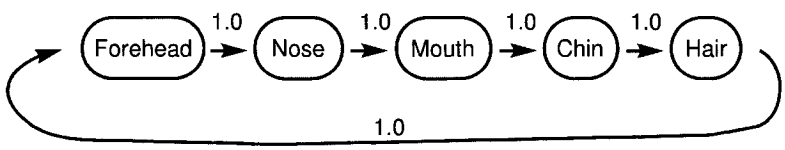


FIGURE 7.1. A graph used to train a scene-level HMM for cartoon facial profiles.

Figures 7.3, 7.6, 7.5, 7.7 illustrate additional examples of scene refinements. The results are produced using the training sets shown in Fig. 7.8 and Fig. 7.9. Figure 7.5 shows the results when applying both a greedy strategy and the Viterbi algorithm when decoding the scene-level HMM. With the greedy approach, the system only considers the current likelihood vector when selecting the maximum likelihood state, hence curves drawn later in the sequence do not affect the selection of the refinement models applied to previously drawn curves. In the example, all horizontal curves below the skyline are rendered using the *grass* model, despite the fact that the system

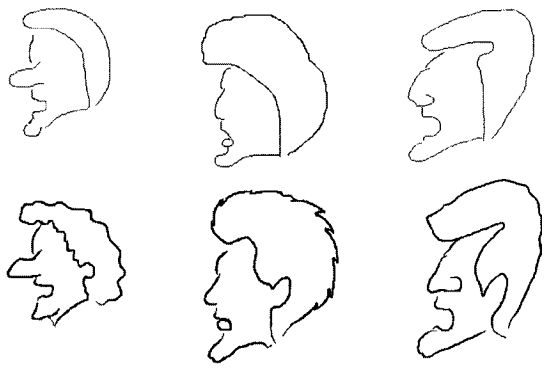


FIGURE 7.2. Generating profiles of cartoon faces. The top sketches show the input and the bottom sketches show the results.

recognizes that the shape below the bottom horizontal line is most similar to a fish. Using Viterbi, classification is performed by taking into account the probabilistic dependencies that arise from all of the curves in the scene. Since a curve can only be refined by a *fish* model when its preceded by a curve that has been refined by a *water* model (or another fish model as show in Fig. 7.9), the refinement applied to the bottom horizontal line is then updated and the curve is re-rendered using the water model.

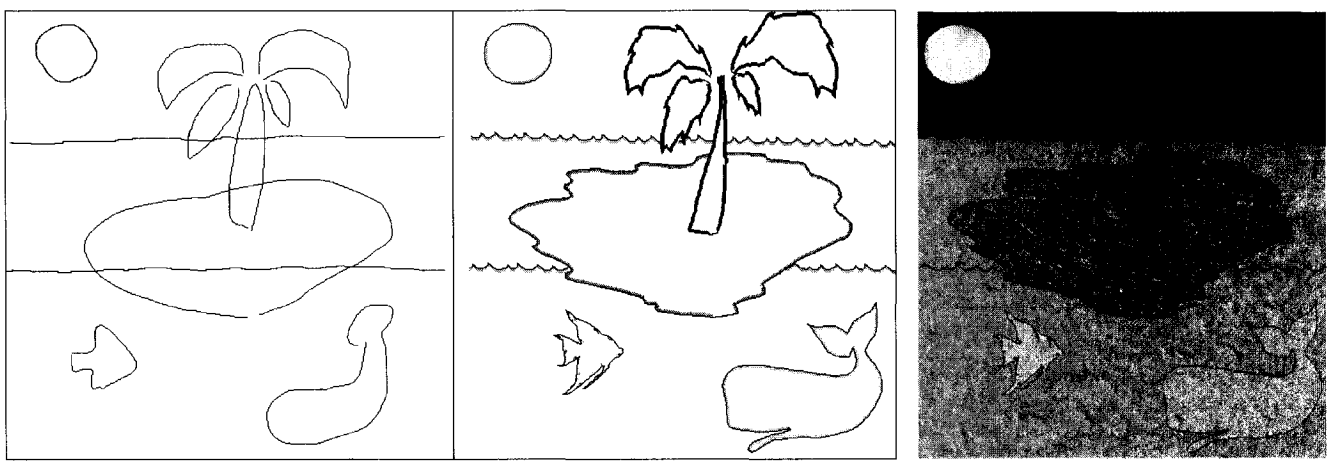


FIGURE 7.3. Synthesis of an island scene. Left shows the input, middle shows the generated scene, including seeds for texture fill, right shows the resulting texture filled scene using the texture sample shown in Fig. 7.4.

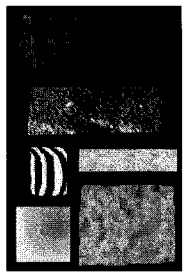


FIGURE 7.4. Training texture used to generate the texture fill in Fig. 7.3.

3. Curve Extraction

The sketching application is extended such that users can automatically extract curves from an image of a sketch by simply clicking with a pointing device on the desired curves. Once a curve is extracted, it is available for the user to edit it using traditional curve transformations (split, merge, move, scale, rotate, filter, etc.) or to apply the automated curve refinement procedure. The key issue that must be addressed is how to identify what curve the user intends to extract? Figure 7.10 shows some of the possible candidate curves that can be extracted from an image. If the system can automatically identify the curve that *stands out* from the rest, in some desired context, then it can include it as a potential candidate for the user's selection. The approach taken to address this issue consists of first identifying all possible curves in an image and then applying the Evaluation algorithm described in the previous section in order to rank them with respect to the model. As such, rather than using a traditional constraint, such as curvature continuity, this approach provides a system that can extract curves that are consistent with a wide range of learned constraints.

3.1. Generating Candidate Curves. It is assumed that an image of a sketch is given as input and the image consists of thin edges that are two pixels thick with a well defined foreground color. (In practice, there are well established methods that can extract foreground and thin edges [**14, 73**].) It is also assumed that the curves are not adjacent to any other curves and they do not superimpose on each

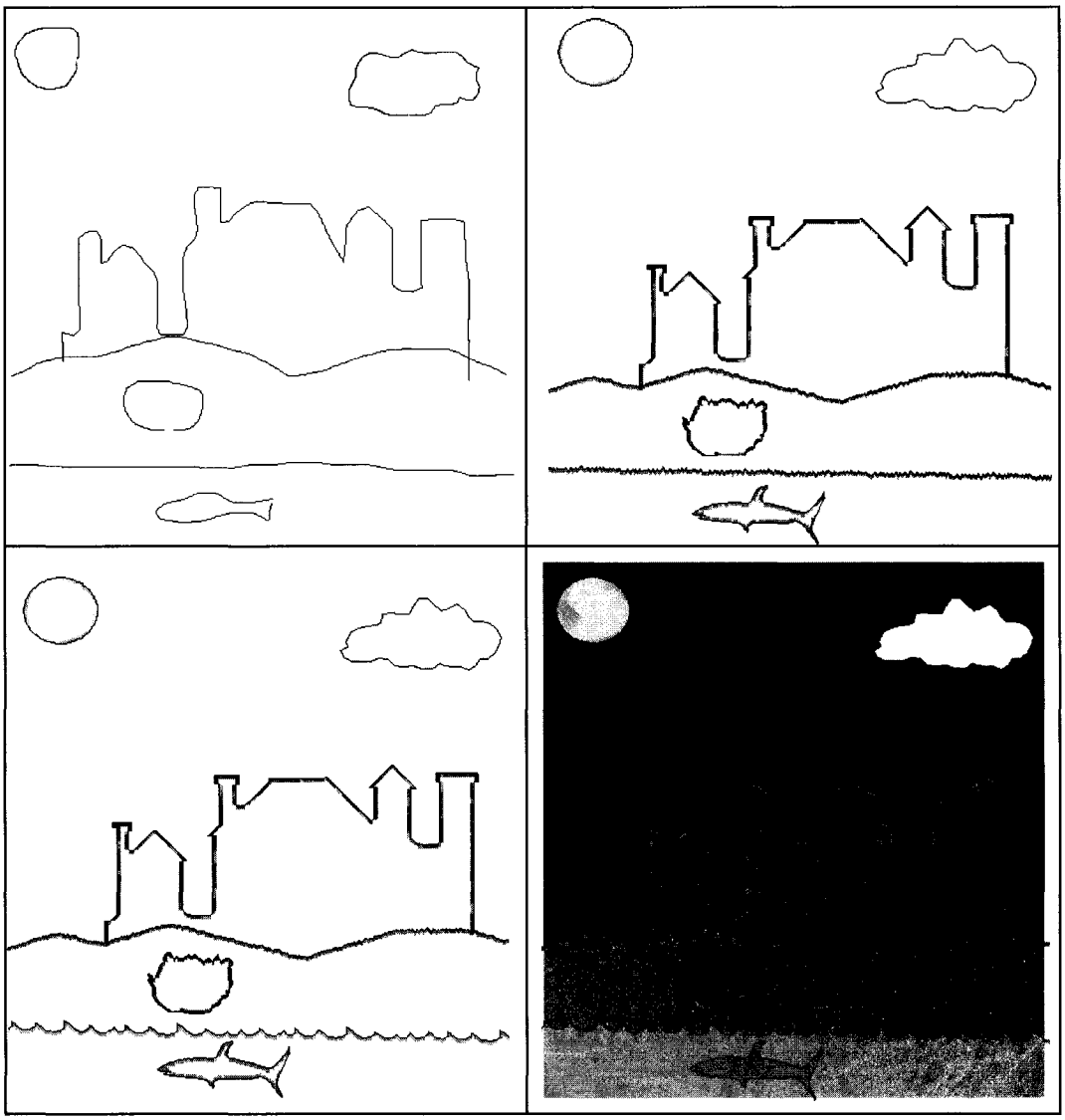


FIGURE 7.5. Top left shows the input sketch, top right shows the output using a greedy method, bottom left shows the output using Viterbi and bottom right shows the result when applying the Markovian texture filler.

other but can intersect (superposition can only occur over two pixels). When the user clicks on the given image near the desired curve, all possible curves that begin from the nearest edge must be identified. The approach to this problem consists of first finding a starting point on the curve and then recursively iterating over the neighbors of pixels to create a *curve segment tree*. The curve segment tree encodes all

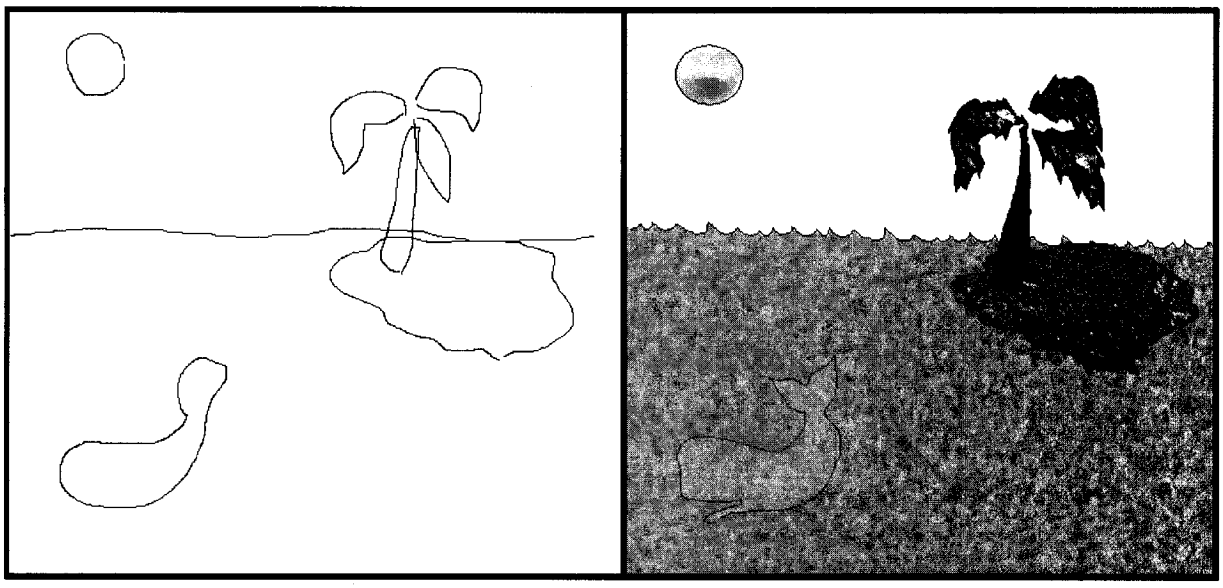


FIGURE 7.6. Scene refinement example using the island training set (Fig. 7.8).

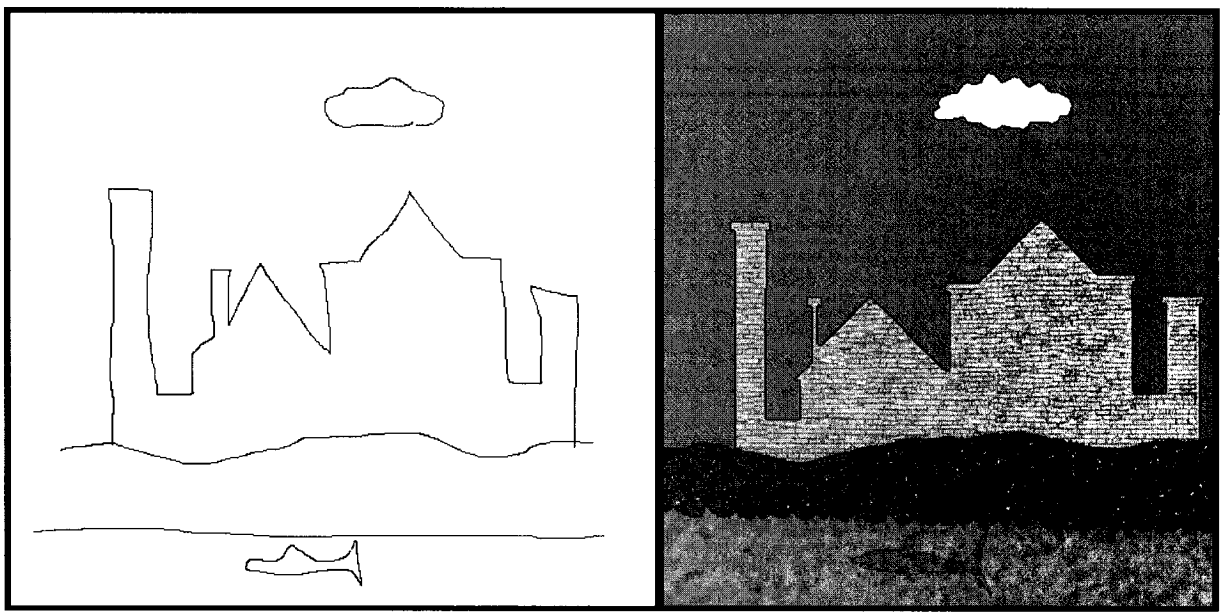


FIGURE 7.7. Scene refinement example using the city skyline training set (Fig. 7.9).

possible curves that begin from the the starting point. The nodes of the tree represent curve segments and the edges represent their junctions (Fig. 7.12).

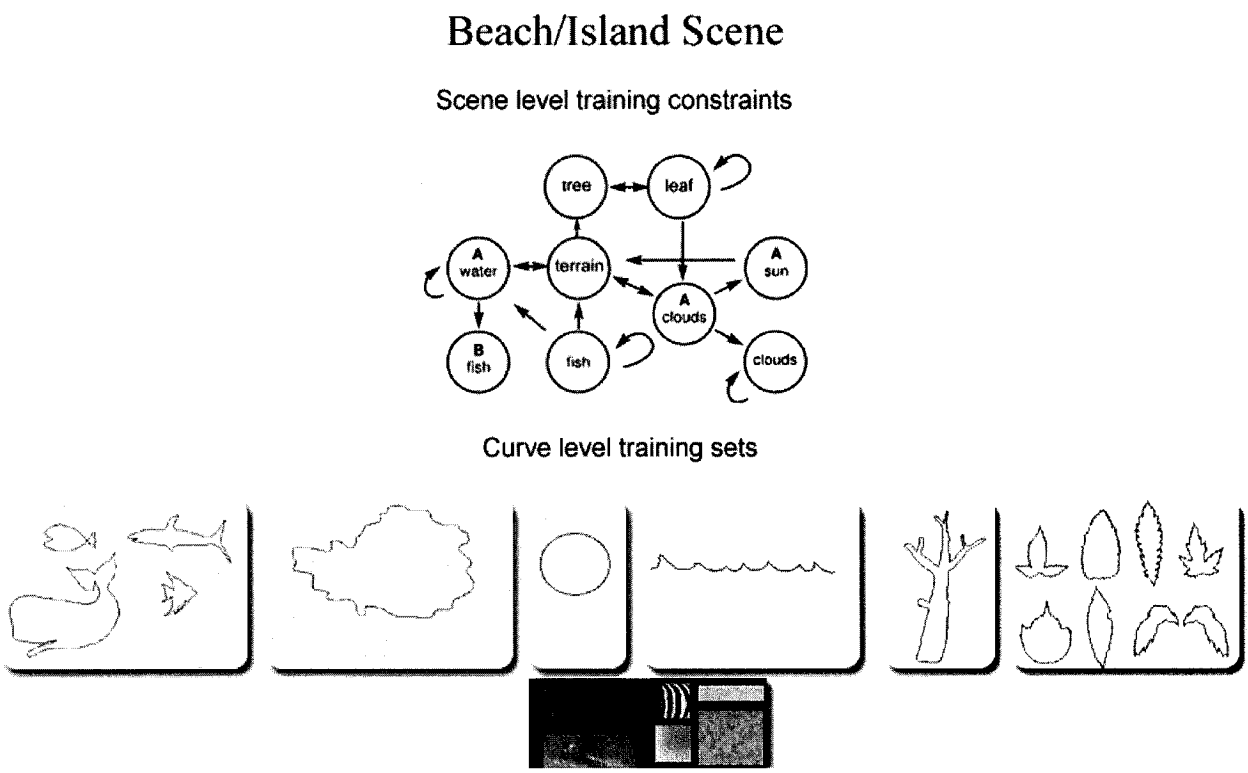


FIGURE 7.8. Training set used to generate *tropical island* scenes. The top shows the scene-level constrains, the middle shows the curve-level training sets and the bottom shows an example texture.

3.2. Starting Point. The starting point is determined based on the number of admissible neighbors a pixel has (the admissibility of a neighbor is described further below). To find the starting point s, the system first searches for the nearest pixel, within some distance d to the mouse click, that matches the foreground color (Fig. 7.11). This distance is set to provide a margin of error such that a user does not need to deal with the accuracy required for clicking exactly on the curve. Once this pixel p is found, the four neighboring pixels are examined (above, below, left and right) and if their color matches the foreground color, they are labeled as admissible neighbors. If there is only one admissible neighbor, then the current pixel is considered as the starting point, otherwise, the system recursively examines each neighbor. On successive iterations, a neighboring pixel is only considered admissible if it is not

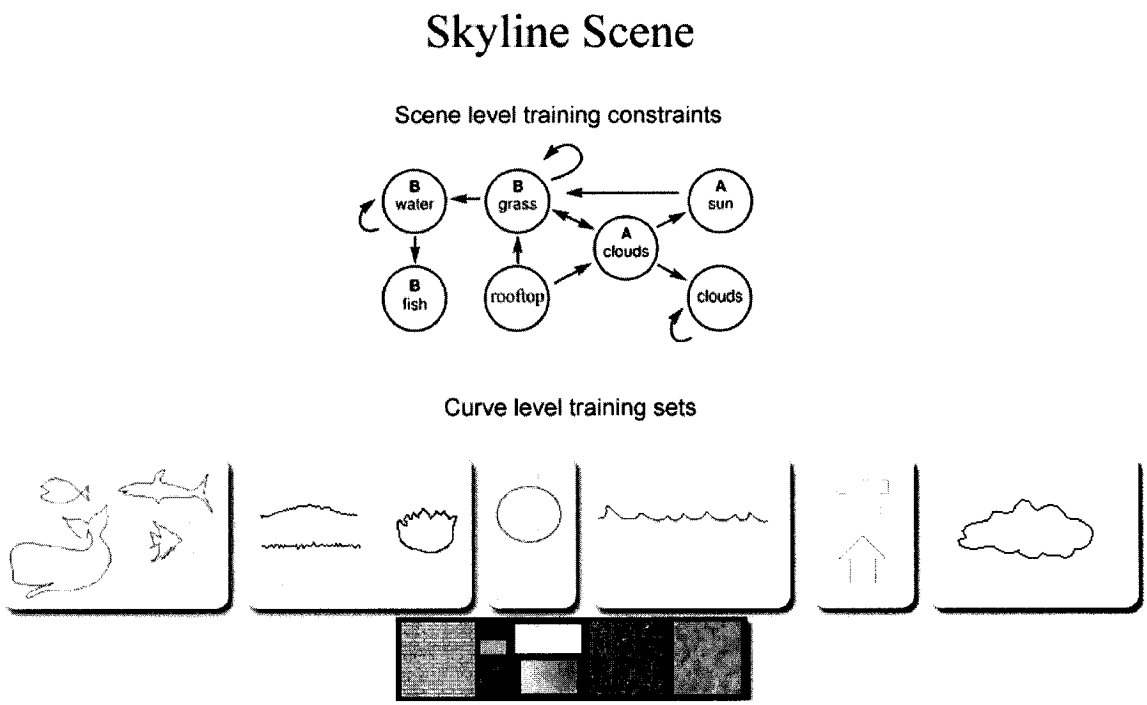


FIGURE 7.9. Training set used to generate city skyline scenes. The top shows the scene-level constrains, the middle shows the curve-level training sets and the bottom shows an example texture.

the same pixel from the previous iteration (avoids revisiting the same pixel). This is performed for l steps and if an end-point is not found within this number of steps then the original starting point is chosen.

3.3. Path Segment Tree. Once the starting point is found, a similar recursion is performed to construct the curve segment tree. The tree is initialized at the starting point p_0 and the root node encodes the curve segment c_0^0 containing one point $(p_0(x), p_0(y))$. The neighbors are then examined recursively to determine if there is a junction. If there is only one admissible neighbor p_1, there is no junction and the point is added to the current segment c_j^k, otherwise, for each admissible neighbor p_i, a new curve segment c_i^{k+1} is created and the point p_i is included in that segment. The result is a hierarchy of curve segments (Fig. 7.12) corresponding to all junctions in the sketch.

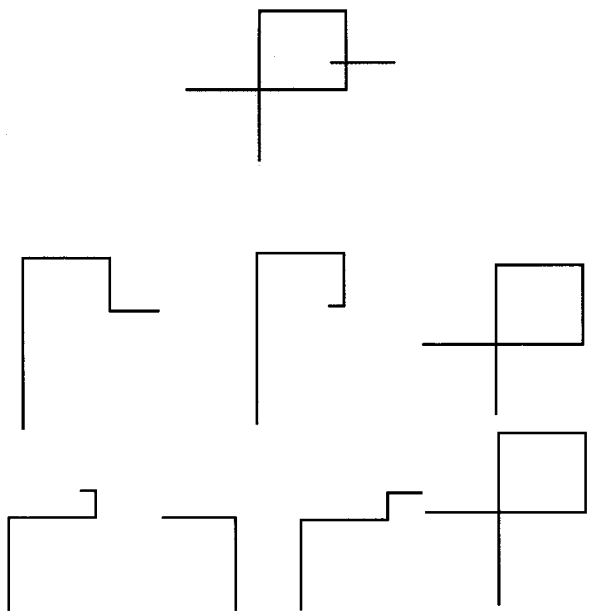


FIGURE 7.10. A set of candidate curves that can be extracted form an image. The top figure shows the original image and the figures below show the candidate curves.

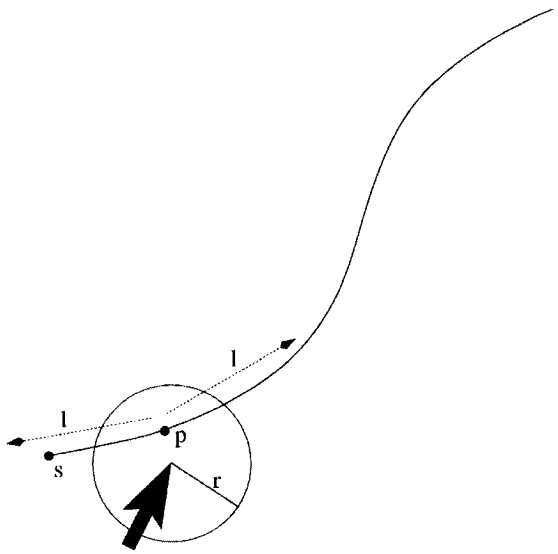


FIGURE 7.11. Finding the starting position s. First, the system searches for the nearest pixel p matching the foreground color, then it recurses up to l steps to find the starting point.

3.4. Pruning. When constructing the segment tree, to avoid issues that can occur with loops, a pixel is only considered as an admissible neighbor if that pixel does not already exist in the current segment or any of the parent segments up to the root of the tree (i.e. is it has not been visited yet). For each node in the tree, a lookup table is maintained in order to provide random access to this information. This lookup table consists of the pixels that have been visited when traversing the tree and are considered inadmissible. However, in order to allow for self-intersecting curves, pixels that have three admissible neighbors are never included in the lookup table, irrespective of whether they have been traversed. This allows the system to retrace those pixels twice (the second time they are labeled as inadmissible as they no longer have three admissible neighbors).

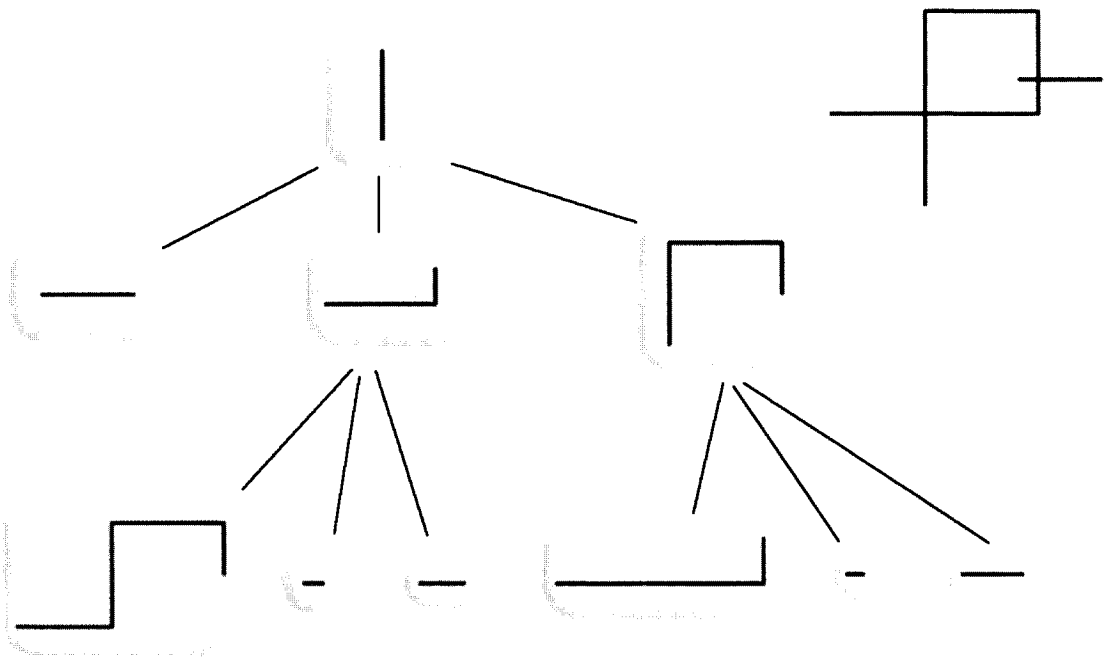


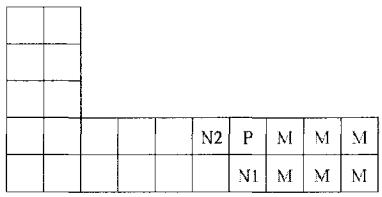
FIGURE 7.12. Curve segment tree. Top right shows the original image with the starting point highlighted

In both the search for the starting point and construction of the segment tree, a first-order look-ahead is performed in order to ignore neighbors that satisfy certain criteria. In particular, because it is assumed that the width of a curve is two pixels,

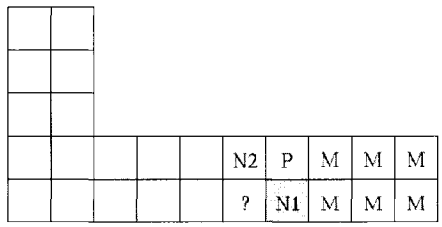
the neighbors of neighbors are pruned as follows: if p_0, p_1 and p_2 are neighbors of p and if all of p_0's neighbors are neighbors of either p_1 or p_2, then p_0 in not considered. That is, when a neighbor p_0 does not lead to a junction, all of its neighbors are reachable directly from the neighbors of the source pixel and hence p_0 can be ignored. For consistency, the same order is followed when pruning neighbors.

3.5. Ranking Candidate Curves. The curve segment tree provides the system with a list of candidate curves that must be ranked. The tree is first traversed to construct all candidate curves. The curves are thereafter sampled using the same sampling rate used in training, filtered to reduce aliasing effects and normalized if the training examples are also normalized. The tangent angles along the curves are also computed from the Cartesian points and the multi-scale representation is applied.

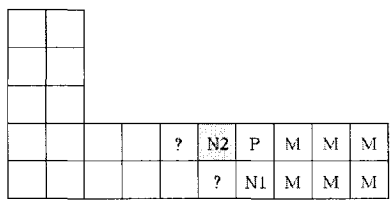
Each curve is ranked based on the likelihood that the model can produce that curve. The likelihood is determined by applying the Evaluation algorithm (Eq. 7.2). Once all candidate curves are ranked, they are sorted from best to worst and the user is presented the top candidate. If desired, the user can further scroll through the list to examine other solutions. Figure 7.14 shows an example curve extraction using the zig-zag patterns training set (Fig. 5.31) and figure 7.15 shows the the extraction using the leaf training set (Fig. 5.8). Figure 7.16 shows an example extraction and refinement using the leaf training set and Fig. 7.17 shows another extraction example using the basic shapes training set (Fig. 5.13). In each example, the curve that is most similar to the training set is extracted.



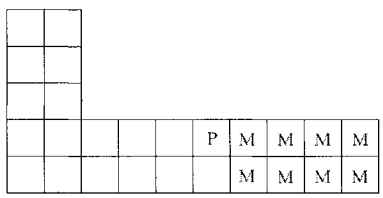
(a) Pixel P has two candidate neighbors $N1$ and $N2$.



(b) Examine the admissibility of $N1$. Its inadmissible as its only neighbor is adjacent to $N2$.



(c) Examine the admissibility of $N2$. Its admissible as there exists a neighbor that is not adjacent to any of the neighbors of P.



(d) Update the lookup table and recurse over the next pixel.

FIGURE 7.13. First-order pruning of pixels from a two pixel thick image. Pixels labeled with M are inadmissible.

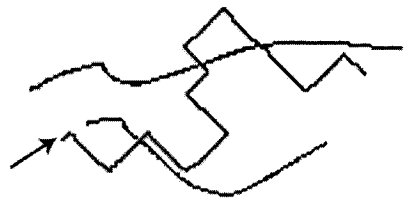


FIGURE 7.14. Extraction of a zig-zag pattern (shown in red).

FIGURE 7.15. Example extraction using the leaves training set. Note how the extraction algorithm can make the right selection even when there are junctions where the curvatures at different branches are locally similar.

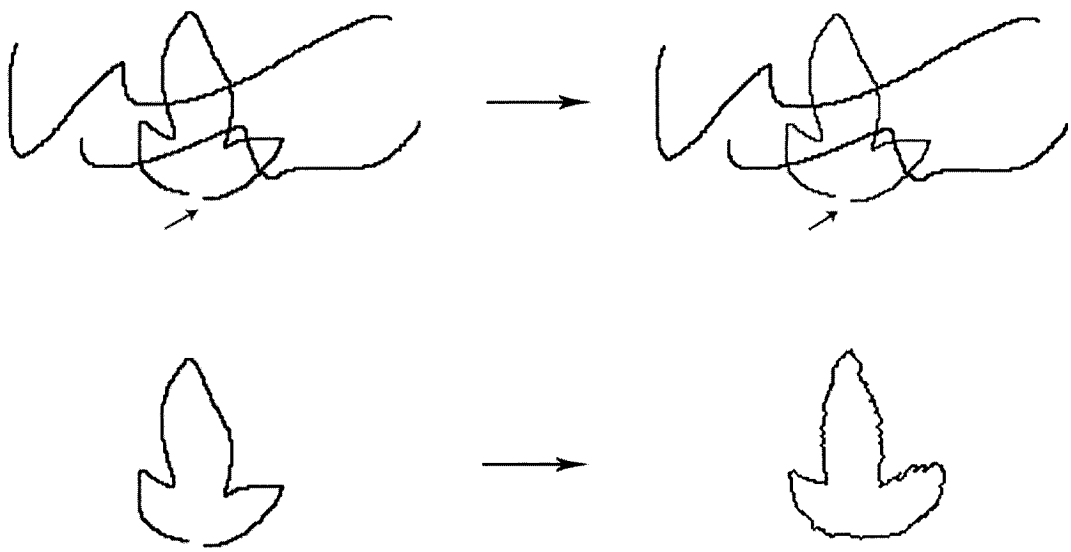


FIGURE 7.16. Example extraction and refinement. Top left shows original image and the user pointer, top right shows the automatically extracted curve (in red). Bottom left shows the curve isolated by dragging it and bottom right shows result of the automated refinement process.

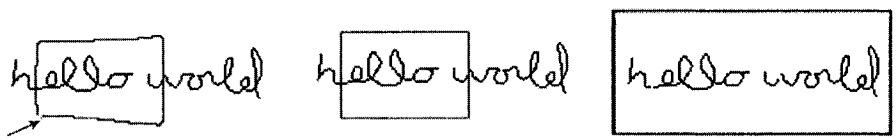


FIGURE 7.17. Left shows extraction (in red), middle shows refinement, right shows a resize.

CHAPTER 8

Conclusion

This thesis presented a machine learning framework for automatically classifying and refining hand-drawn curves. The underlying model consisted of a Hidden Markov Model that encodes constraints on the types of output curves that can be synthesized, the types of input curves that are expected and the effect the input has on the synthesis. Using a regularization framework, these Hidden Markov Models were combined with supplementary user-defined constraints that further restrict the types of outputs that can be produced. In addition, by extending the dimensionality of the models' state space, constraints on multiple curve attributes at multiple scales were encoded without exponentially increasing the computational complexity of the system. Decoding these models resulted in the synthesis of novel curves that exhibit a similar look to examples in the training set while also adhering to the the user-defined constraints. Finally, a Hierarchical Hidden Markov Model was developed in order to model high-level constraints on the types of refinements that can be applied. Evaluating the likelihood that a curve can be generated by a curve-level model in conjunction with evaluating the applicability of the model based on the high-level constrains resulted in a classification scheme that takes into account both the shape of the curve and its context.

The applicability of the described framework was exemplified by two applications; a sketching application and a robot path planning application. For the sketching

application, Hidden Markov Models were trained using several training sets. Each sets consisted of control curves that exemplify the types of inputs a user would draw and refined curves that exemplify the desired look. These models were used to extract curves from an image, identify what class of examples the drawn curves best belong to and then augment those curves using the appropriate model.

Based on a subjective evaluation of the results, it was shown that novel full-color 2D illustrations that exhibit the desired look can be generated from the coarse sketches. This was demonstrated using both the Viterbi algorithm and a greedy algorithm. It was shown that because the greedy approach does not take into account the entire sketch when synthesizing the individual elements, the results did not properly reflect the learned curve styles while the Viterbi algorithm generated satisfactory results as the entire sequence of inputs was considered before selecting the final solution. The synthesis results were further examined under different parameter settings and curve attribute. It was demonstrated how the mixture variance, regularization terms and weights, stationarity window, sampling resolution and the curve representation affect the output. A texture synthesis procedure was also developed that further enhanced the output. It was shown how the synthesized texture seeds can be used to initiate a texture synthesis procedure that fills the interior of the synthesized curves.

For the path planning application, it was demonstrated that the same learning framework can be used to learn constraints on robot trajectories from examples. Using the regularization framework, a bias to avoid obstacles was embedded into the system. It was demonstrated that the generated paths followed the desired input trajectory while satisfying the learned constraints and avoiding the obstacles.

1. Future Work

One open problem that remains to be addressed is that of automatically finding good values for the parameters that control the synthesis process. For example, the synthesis of novel illustrations 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. This is complicated by the need to account for the regularization terms and input bias. At present, the mixing fractions are fixed and predetermined manually but their automatic determination remains an open problem. A possible direction for work can be to examine how to set these values based on some initial conditions, such as a maximum divergence form the input or a minimum distance to the obstacles. This can lead to an in depth theoretical analysis of the system to attempt to prove that certain conditions are guaranteed to occur when using the training set under the specified parameter settings. For example, in the path-planning application, it may be possible to determine that the system is resolution complete with respect to the training data and selected parameters. Another approach can be to attempt to infer the parameter settings of one set from the parameter settings of another. A potential approach to this can consist of attempting to equalize the likelihood of mixing neighboring segments or to apply cross-validation techniques and determine if one set can reconstruct its members as well as the another can for its own members. (Such an approach can also be used to evaluate the richness of a set).

Another interesting problem that can be examined is that of automatically classifying the initial training sets. This allows the user to simply provide a bulk of examples and the system would automatically group them. Indeed this is the clustering problem, an extensively studied research problem, though the problem of measuring the distance between clusters of curves remains an open problem. Building on the presented framework, a potential approach to this problem can consist of using the evaluation algorithm to identify similarities between examples or groups of examples.

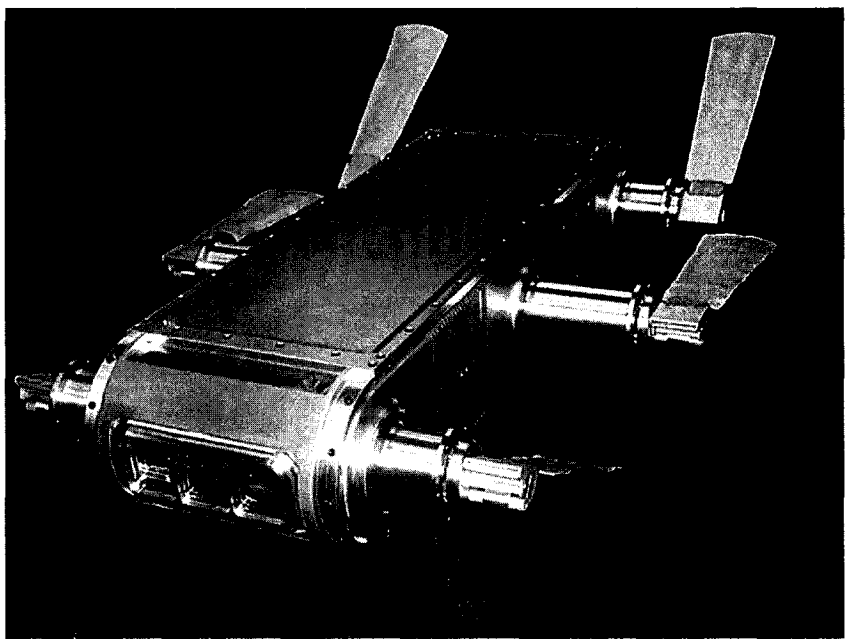
One significant open issue concerns the application of global constraints to the curves being synthesized. For example, in the sketching application, the synthesis results using the leaves and skyline training set did not exhibit symmetry, a property often found in such examples. As another example, in the texture filling process,

pixels that are produced from different parts of the contour may not join in a desirable fashion. For example, texture filling for leaves would require some specialized constraints in order to have the veins of the leaves meet at the right location. Developing an approach that can learn such constrains or developing a smart interface that allows users to interactively accomplish this task remains an open problem.

In the curve extraction algorithm, one of the difficulties that can occur is where there is an excessive number of candidate paths in the path-segment tree. This is especially apparent when the contours are more than two pixels thick. A direction for future work can be to develop methods that can dynamically prune the graph in parallel to the evaluation process. While its always best to evaluate a path in its entirety, one can suggest that an iteratively deepening look-ahead can help prune candidate paths that are not expected to rank high enough to be part of the final solution.

In general, the problem of providing natural and convenient interfaces can be found in many domains. The extent in which machine learning is used in developing smart interfaces is limited and the potential opportunities for research are vast. There are a number of applications and domains that the presented learning system can be extended to. In particular preliminary work is taking place for the control of an underwater robot with eighteen degrees of freedom (Fig. 8.1). By customizing the framework, the system can simplify the control of a complex robot by automatically generating the appropriate sequence of gaits or motor commands that lead the robot to follow the desired goal trajectory (Fig. 8.1). Another domain of application lies in animation editing. Preliminary result show that when using the refinement system, a pen-stroke can conveniently control the motion of an articulated figure (Fig. 8.2). Pen speed and pressure can further assist as supplementary cues for controlling the motion. These are just a few examples that illustrate the extensibility of the framework. The novel ideas presented in this thesis form a foundation with great potential for developing new techniques that can suitably represent, learn and express properties

of examples in order to help address a variety of problems in Computer Graphics and Robotics.



(a) An underwater robot developed at McGill University. The robot has six legs, each parametrized by three parameters (fin frequency, amplitude and shift) for a total of 18 DOF.

(b) The predicted path for the robot. Left shows input and the right shows the predicted paths and gaits (color coded).

FIGURE 8.1. Preliminary results for controlling an underwater robot. Training set consisted of several simulated motions.

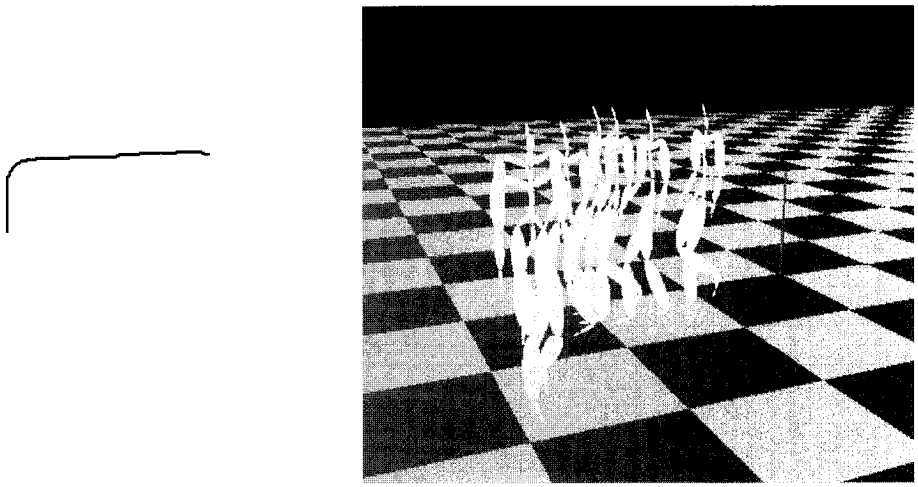


FIGURE 8.2. Preliminary results for motions synthesis. Training set consisted of several example motions. The left shows the input curve and the right shows the resulting motion.

APPENDIX A

Pseudo-Code for Learning and Decoding the Refinement Models

In this appendix the pseudo-code for the learning and decoding algorithm is present. The fist section presents the state labeling algorithm, followed by the algorithm for computing the transition and confusion matrices. Finally, the decoding algorithm is presented.

1. State Labeling Algorithm

The original states, the labels and their representative states are easily combined together by using augmented data structures. Algorithm 1.1 shows the pseudo-code for labeling states. The procedure first takes in as input a set of multi-attributed curves and generates the multi-scale representation for each curve. Following this, the procedure creates a table called $Tset$, where each (i, j) element in the table stores the multi-dimensional state H for the j^{th} segment of the i^{th} curve ($Tset$ is the complete state representation of the training curves). A label is then assigned to each unique state in $Tset$ (the function *SameState(·)* evaluates if Equation 4.5 is below all attribute thresholds), the set $\mathcal{H}'$ is augmented to store all new labels and Q^{-1} is computed by averaging the states having the same label.

Algorithm 1.1: MAKELABELEDTSET(*multiAttributeCurves* α)

global $\mathcal{S}, \tau$

$maxLabels \leftarrow 0$

$C \leftarrow$ MAKEMULTISCALECURVES$(\alpha, \mathcal{S}, \tau)$

for $i \leftarrow 0$ **to** $C.numberOfExamples$

  for $j \leftarrow 0$ **to** $C[i].numberOfSegments$

    do $\begin{cases} segment \leftarrow C[i][j] \\ H \leftarrow \text{MAKEMULTISCALESTATE}(segment) \\ Tset[i][j] \leftarrow H \\ Tset[i][j].label \leftarrow -1 \\ maxLabel++ \end{cases}$

$label \leftarrow 0$

for $i \leftarrow 0$ **to** $maxLabels$

  $Q^{-1}[i] \leftarrow new$ QUEUE()

$\mathcal{H}' \leftarrow new$ QUEUE()

for $i \leftarrow 0$ **to** $Tset.numberOfExamples$

  for $j \leftarrow 0$ **to** $Tset[i].numberOfSegments$

    if $Tset[i][j].label = -1$

      then $\begin{cases} Tset[i][j].label \leftarrow label \\ Q^{-1}[label].push(Tset[i][j]) \\ \overline{H} \leftarrow \text{AVERAGE}(Q^{-1}[label]) \\ \textbf{for } k \leftarrow i \textbf{ to } Tset.numberOfExamples \\ \quad \textbf{for } l \leftarrow 0 \textbf{ to } Tset[k].numberOfSegments \\ \quad\quad \textbf{if } Tset[k][l].label = -1 \textbf{ and } \text{SAMESTATE}(Tset[k][l], \overline{H}) \\ \quad\quad\quad \textbf{then } \begin{cases} Tset[k][l].label = label \\ Q^{-1}[label].push(Tset[k][l]) \\ \overline{H} \leftarrow \text{AVERAGE}(Q^{-1}[label]) \end{cases} \\ \mathcal{H}'.push(label) \\ label++ \end{cases}$

for each $i \in \mathcal{H}'$

  do $\begin{cases} \overline{H} \leftarrow \text{AVERAGE}(Q^{-1}[i]) \\ Q^{-1}[i].clear() \\ Q^{-1}[i].push(\overline{H}) \end{cases}$

return $(Tset, Q^{-1}, \mathcal{H}')$

1.1. Transition Matrix Algorithm. Algorithm 1.2 shows the pseudo-code for computing the transition matrix over the stationarity window range $[l, u]$. In the stationary case, the procedure is only called once with $l = 0$ and $u = T$ for closed curves and $u = T - \tau$ for open curves. In the non-stationary case, the procedure is called for every segment, adjusting the lower and upper bounds respectively. The first step is to call the function *MakeLabeledTset(·)* (Algorithm 1.1) to generate the labeled state representation for the refined curves. Because the training set representation is indexes the position along the curve by the segment number ($H(t)$ refers to the t^{th} segment), the input parameters l and u must first be divided by the transition step τ. The procedure then initializes an $|\mathcal{H}'| \times |\mathcal{H}|'$ transition matrix with all probabilities set to zero. The main likelihood computation is performed at line (i), evaluating the Gaussian similarity measure (Equation 4.4 over the constraining attributes. This similarity is then added to the likelihood of generating the following state.

This procedure is performed as a preprocessing step and is repeated each time the translation step, stationarity window, the variance or weights of Equation 4.5 are modified. For systems with limited memory (such as a video card), this preprocessing step must be computed during runtime. In such cases, instead of computing the entire matrix for each $i \in \mathcal{H}'$, the desired state is passed as an argument and the procedure returns the column of the matrix corresponding to the potential transitions for that state.

Algorithm 1.2: MAKETRANSITIONMATRIX(*multiAttributeCurves* α, *pos* l, *pos* u)

global $\mathcal{S}, \mathcal{A}, \tau$
$(Hset, Q^{-1}, \mathcal{H}') = \text{MAKELABELEDTSET}(\alpha)$
$l \leftarrow \lfloor \frac{l}{\tau} \rfloor$
$u \leftarrow \lceil \frac{u}{\tau} \rceil$
$\text{INITMATRIX}(M, \mathcal{H}' \times \mathcal{H}', 0)$
for each $i \in \mathcal{H}'$
  for $j \leftarrow 0$ **to** $Hset.numberOfExamples$
    for $t \leftarrow l$ **to** u
      do
        $similarity = \text{GAUSSIANSIMILARITY}(Q^{-1}[i], Hset[j][t], \mathcal{S}, \mathcal{A})$ (i)
        $nextState \leftarrow (t+1)$ MOD $Hset[j].numberOfSegments$
        $nextStateLabel \leftarrow Hset[j][nextState].label$
        $M[i][nextStateLabel] += similarity$
$\text{NORMALIZE}(M)$
return (M)

1.2. Confusion Matrix Algorithm. The procedure for estimating B is outlined in Algorithm 1.3. It bears similarities to Algorithm 1.2 with the primary difference of searching for exact matches of the tuples representing the associated hidden and observation states. In order to disregard the hidden states' auxiliary attributes, the set of auxiliary attributes is temporarily disabled before attempting to check for a match. The function *SameState(·)* then ignores the auxiliary attributes. This results in a redundancy of likelihoods similar to that in the transition matrix. (Note that while the procedure only uses an exact matching criterion, because the auxiliary attributes are ignored in the source state, a single pass over the training set is not sufficient to produce the desired matrix.)

Algorithm 1.3: MAKECONFUSIONMATRIX(*multiAttributeCurves* $\alpha, \beta, pos\ l, pos\ u$)

global $\mathcal{A}_{aux}l \leftarrow \lfloor \frac{l}{\tau} \rfloor$
$u \leftarrow \lceil \frac{u}{\tau} \rceil$
$(Hset, Q_H^{-1}, \mathcal{H}') = \text{MAKELABELEDTSET}(\alpha)$
$(Oset, Q_O^{-1}, \mathcal{O}') = \text{MAKELABELEDTSET}(\beta)$
$\text{INITMATRIX}(B, \mathcal{H}' \times \mathcal{O}', 0)$
$temp \leftarrow \mathcal{A}_{aux}$
$\mathcal{A}_{aux} \leftarrow NULL$
for each $i \in \mathcal{H}'$
  for $j \leftarrow 0$ **to** $Hset.numberOfExamples$
    for $t \leftarrow l$ **to** u
      do
        if $\text{SAMESTATE}(Q_H^{-1}[i], Hset[j][t])$
          then $similarity \leftarrow 1$
          else $similarity \leftarrow 0$
        $obs \leftarrow Oset[i][t].label$
        $B[i][obs] += similarity$
$\text{NORMALIZE}(B)$
$\mathcal{A}_{aux} \leftarrow temp$
return (B)

1.3. Decoding Algorithm. Algorithm 1.4 shows the pseudo-code for point-based decoding of the HMM with regularization priors. The main body consist of two parts, the first part, starting at line (i), implements the propagation step and the second part, starting at line (ii), implements the input conditional step. In the first part, the procedure begins by iterating over all states in the previous distribution $\Psi(t-1)$. (The vector $\Psi(t)$ stores both the likelihood and the state information for all candidate states and is indexed by the state labels; i.e. $\Psi(t,i).label = i$.) For each of the previous states, the likelihood of generating the next state, for all possible next states, is computed. This includes evaluation of the regularization biases as applied to the candidate state sequence, preceded by a procedure call *SetAuxParams(·)* that

updates additional decoding auxiliary attributes required for regularization. The state that has the highest likelihood is then stored in the next distribution with a back-pointer to the previous. In the second part, the input conditional is applied using the sigmoid function and the likelihoods in the confusion matrix. (The function *SigmoidProd(·)* computes the product of the sigmoid blur over all the input attributes as shown in Equation 4.11.) The best input conditional that matches the input observation is used.

For illustrative purposes, this algorithm does not implement thresholding, hence the candidate state vector Ψ can be indexed by the state labels (providing random access when checking for the best succession of states). When thresholding, the state vector itself must be sorted by the highest ranking states and cannot be indexed by the labels. The thresholding implementation can achieve similar runtime efficiencies by using a supplementary data vector that indexes the state labels to their locations in Ψ.

Algorithm 1.4: DECODE($pos\ t, input\ O_{in}$)

global $\Psi, M, B, Q_H^{-1}, Q_O^{-1}, \mathcal{H}', \pi$

$\Psi_{next} \leftarrow new$ VECTOR()

if $t = 0$

then $\Psi_next = \pi$

else for each $S \in \Psi[t-1]$

for each $l \in \mathcal{H}'$

do (i)

$S_{next} = Q_H^{-1}[l]$

$S_{next}.lable = l$

$S_{next}.likelihood = S.likelihood * M[t-1][S.label][S_{next}.label]$

$S_{next}.backPtr = S.label$

SETAUXPARAMS(S_{next})

REGULARIZE(S_{next}, O_{in})

if $\Psi_{next}[S_{next}.label] = NULL$

then $\Psi_{next}[S_{next}.label] = S_{next}$

else if $\Psi_{next}[S_{next}.label].likelihood < S_{next}.likelihood$

then $\Psi_{next}[S_{next}.label] = S_{next}$

for $i \leftarrow 0$ **to** $\Psi_{next}.size$

do (ii)

$bestConditional \leftarrow 0$

for each $O \in Q_O^{-1}$

do

$conditional =$ SIGMOIDPROD(O, O_{in})

$conditional = conditional * B[t][i][O.label]$

if $conditional > bestConditional$

then $bestConditional = conditional$

$\Psi_{next}[i].likelihood = \Psi_{next}[i].likelihood * bestConditional$

$\Psi[t] = \Psi_{next}$

REFERENCES

[1] C. Alvarado and R. Davis, *Resolving ambiguities to create a natural computer-based sketching environment*, International Joint Conference on Artificial Intelligence, 2001, pp. 1365–1374.

[2] C. Alvarado, M. Oltmans, and R. Davis, *A framework for multi-domain sketch recognition*, Proceedingsof AAAI Spring Symposium on Sketch Understanding, March 2002.

[3] O. Arikan, D. A. Forsyth, and J. O'Brien, *Motion synthesis from annotations*, ACM Transactions on Graphics **33** (2003), no. 3, 402–408.

[4] R. Arkin and R. Murphy, *Autonomous navigation in a manufacturing environment*, IEEE Transaction on Robotics and Automation **6** (1990), no. 4, 445–454.

[5] J. Arvo and K. Novins, *Fluid sketches: Continuous recognition and morphing of simple hand-drawn shapes*, Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology, 2000.

[6] E. W. Aslaksen and J. R. Klauder, *Continuous representation theory using the affine group*, Journal of Mathematical Physics **10** (1969), no. 1, 2267–2275.

[7] Jonas August and Steven W. Zucker, *Sketches with curvature: The curve indicator random field and markov processes*, IEEE Transactions on Pattern Analysis and Machine Intelligence **25** (2003), no. 4, 387–400.

[8] M.l Banks and E. Cohen, *Real time spline curves from interactively sketched data*, SI3D '90: Proceedings of the 1990 symposium on Interactive 3D graphics, ACM Press, 1990, pp. 99–107.

[9] Z. Bar-Joseph, R. El-Yaniv, D. Lischinski, and W. Werman, *Texture mixing and texture movie synthesis using statistical learning*, IEEE Transactions on Visualization and Computer Graphics **7** (2001), no. 2, 120–135.

[10] R. I. Brafman, *A heuristic variable grid solution method for pomdps*, Proceedings Fourteenth National Conference on Artificial Intelligence (AAAI), 1997, pp. 727–733.

[11] M. Brand and A. Hertzmann, *Style machines*, Proceedings of ACM SIGGRAPH, 2000, pp. 183–192.

[12] R. A. Brooks, *A robost layered control system for a mobile robot*, IEEE Journal of Robotics and Automation **2** (1986), no. 1, 14–23.

[13] A. Bruderlin and L. Williams, *Motion signal processing*, Proceedings of ACM SIGGRAPH, August 1995.

[14] J. Canny, *A computational approach to edge detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence **8** (1986), no. 6.

[15] E. Catmull and J. Clark, *Recursively generated b-spline surfaces on arbitrary topological meshes*, Computer-Aided Design **10** (1978), no. 6, 350–355.

[16] G. C.-H. Chuang and C.-C. J. Kuo, *Wavelet descriptor of planar curves: Theory and application*, IEEE Transanctions on Image Processing **5** (1996), no. 1, 56–70.

[17] H. H. Clarck, *Using languages*, Cambridge University Press, 1996.

[18] K. Craik, *The nature of explanation*, Cambridge University Press, 1943.

[19] D. Doo and M. Sabin, *Behaviour of recursive division surfaces near extraordinary points*, Computer-Aided Design **10** (1978), no. 6, 356–360.

[20] L. E. Dubins, *On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents*, American Journal of Mathematics, vol. 79, 1957, pp. 497–517.

[21] G. Dudek and J. K. Tsotsos, *Shape representation and recognition from multi-scale curvature*, Computer Vision and Image Understanding **68** (1997), no. 2, 170–189.

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

[23] F. J. Estrada and A. D. Jepson, *Controlling the search for convex groups*, Technical Report CSRG-482, January 2004.

[24] G. Farin, *Curves and surfaces for computer aided geometric design*, Academic Press, 1992.

[25] S. Fine, Y. Singer, and N. Tishby, *The hierarchical hidden markov model: Analysis and applications*, Machine Learning **32** (1998), no. 1, 41–62.

[26] A. Finkelstein and D. H. Salesin, *Multiresolution curves*, Proceedings of ACM SIGGRAPH, July 1994, pp. 261–268.

[27] A. R. Forrest, *The twisted cubic curve: A computer-aided geometric design approach*, Computer Aided Design **12** (1980), no. 4, 165–172.

[28] D. R. Forsey and R. H. Bartels, *Hierarchical b-spline refinement*, Computer Graphics **22** (1988), no. 4, 205–212.

[29] W. T. Freeman, J. B. Tenenbaum, and E. Pasztor, *Learning style translation for the lines of a drawing*, ACM Transactions on Graphics **22** (2003), no. 1, 33–46.

[30] D. Gabor, *Theory of communication*, Journal of the Institution of Electrical Engineers **93** (1946), no. 26, 429–457.

[31] S. Geman and D. Geman, *Stochastic relaxation, gibbs distribution and the bayesian restoration of images*, IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 6, 1984, pp. 721–741.

[32] E. Giunchiglia and T. Walsh, *A theory of abstraction*, Artificial Intelligence **26** (1992), no. 2-3, 323–390.

[33] M. Gleicher, *Motion editing with space-time constraints*, Proceedings of the 1997 Symposium on Interactive 3D Graphics, April 1997, pp. 139–148.

[34] A. Grossmann and J. Morlet, *Decomposition of hardy functions into square integrable wavelets of constant shape*, Society for Industrial and Applied Mathematics, J. Mathematics **15** (1984), no. 1, 723–736.

[35] A. Haar, *Zur theorie der orthogonalen funktionensysteme*, 1909.

[36] M. Harada, A. Witkin, and D. Baraff, *Interactive physically-based manipulation of discrete/continuous models*, Proceedings of ACM SIGGRAPH, August 1995.

[37] D. J. Heeger and J. R. Bergen, *Pyramid-based texture analysis/synthesis*, Proceedings of ACM SIGGRAPH, 1995, pp. 229–238.

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

[39] J. K. Hodgins, W. L. Wooten, D. C. Borgan, and J. F. O'Brien, *Animating human athletics*, Robotics Research: The Eighth International Symposium, no. Springer-Verlag:Berlin, August 1995, pp. 356–367.

[40] T. Igarashi and J. F. Hughes, *Smooth meshes for sketch-based freeform modeling*, ACM Symposium on Interactive 3D Graphics, 2003, pp. 139–142.

[41] T. Igarashi, S. Matsuoka, and H. Tanaka, *Teddy: A sketching interface for 3d freeform design*, Proceedings of ACM SIGGRAPH, 1999, pp. 409–416.

[42] T. Igarashi, S. Kawachiya S. Matsuok and, and H. Tanaka, *Interactive beautification: A technique for rapid geometric design*, ACM Symposium on User Interface Software and Technology (UIST), 1997, pp. 105–114.

[43] D. W. Jacobs, *Robust and efficient detection of convex groups*, Computer Vision and Pattern Recognition, 1993, pp. 770–771.

[44] G. Johansson, *Visual perception of biological motion and a model for its analysis*, Perception and Psychophysics **14** (1973), no. 2.

[45] P. N. Johnson-Laird, *Mental models*, Foundations of Cognitive Science, Cambridge University Press, 1983, pp. 469–493.

[46] R. D. Kalnins, L. Markosian, B. J. Meier, Michael A. Kowalski, Joseph C. Lee, Philip L. Davidson, Matthew Webb, John F. Hughes, and Adam Finkelstein, *WYSIWYG NPR: Drawing Strokes Directly on 3D Models*, ACM Transactions on Graphics **21** (2002), no. 3, 755–762.

[47] D. Keren and M. Werman, *A bayesian framework for regularization*, IEEE International Conference on Pattern Recognition, 1994, pp. 72–76.

[48] K. Koffka, *Perception: and introduction to the gestalt-theory*, Psychological Bulletin **19** (1922), 531–585.

[49] L. Kovar, M. Gleicher, and F. Pighin, *Motion graphs*, Proceedings of ACM SIGGRAPH, 2002.

[50] T. Kurtoglu and T. F. Stahovich, *Interpreting schematic sketches using physical reasoning*, AAAI Spring Symposium on Sketch Undestanding, 2002.

[51] Y. L., T. Wang, and H.-Y. Shum, *Motion texturing: A two-level statistical model for character motion synthesis*, Proceedings of ACM SIGGRAPH, 2002.

[52] J. A. Landay and B. A. Myers, *Interactive sketching for the early stages of user interface design*, Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 1995, pp. 43–50.

[53] J. C. Latombe, *Robot motion planning*, Kluwer Academic Publishers, 1991.

[54] L. Kavraki J-C. Latombe, *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*, IEEE Transactions on Robotics and Automation, vol. 12, August 1996.

[55] J.-P. Laumond, P. E. Jacobs, M. Taix, and R. M. Murray, *A motion planner for nonholonomic mobile robots*, IEEE Transactions on Robotics and Automation, vol. 10, 1994, pp. 577–593.

[56] J. Lee and S. Y. Shin, *A hierarchical approach to interactive motion editing for human-like figures*, Proceedings of ACM SIGGRAPH, 1999, pp. 39–48.

[57] H. Lipson and M. Shpitalni, *Correlation-based reconstruction of a 3d object from a single freehand sketch*, AAAI Spring Symposium on Sketch Understanding, 2002, pp. 99–104.

[58] D. G. Lowe, *Perceptual organization and visual recognition*, Kluwer Academic Publisher, 1985.

[59] T. Lyche and K. Morken, *Knot removal for parametric b-spline curves and surfaces*, Comput. Aided Geom. Des. **4** (1987), no. 3, 217–230.

[60] S. Mallat, *A theory for multiresolution signal decomposition : the wavelet representation*, IEEE Transaction on Pattern Analysis and Machine Intelligence **11** (1989), no. 1, 674–693.

[61] D. Marr and E. C. Hildreth, *Theory of edge detection*, Proceedings of the Royal Society of London, 1980, pp. 187–217.

[62] Y. Meyer, *Wavelets: Algorithms and applications*, Society for Industrial and Applied Mathematics, J. Mathematics (1993), xii+133.

[63] T. Mitchell, *Machine learning*, McGraw Hill, 1997.

[64] J. Miura and Y. Shirai, *Hierarchical vision-motion planning with uncertainty: Local path planning and global route selection*, IEEE/RSJ International Conference on Intelligent Robots and Systems, 1992, pp. 1847–1854.

[65] J. Pineau, G. J. Gordon, and S. Thrun, *Applying metric-trees to belief-point pomdps*, Advances in Neural Information Processing Systems 16 (Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, eds.), MIT Press, Cambridge, MA, 2004.

[66] M. L. V. Pitteway, *Algorithm for drawing ellipses or hyperbolae with a digital plotter*, The Computer Journal **10** (1967), no. 3, 282–289.

[67] Z. Popovic and A. Witkin, *Physically based motion transformation*, Proceedings of ACM SIGGRAPH, August 1999.

[68] K. Pullen and C. Bregler, *Motion capture assisted animation: Texturing and synthesis*, Proceedings of ACM SIGGRAPH, 2002.

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

[70] J. Reeds and L. Shepp, *Optimal paths for a car that goes both forwards and backwards*, Pacific Journal of Mathematics, vol. 145(2), 1990, pp. 367–393.

[71] D. Rubine, *Specifying gestures by example*, Proceedings of ACM SIGGRAPH, 1991, pp. 329–337.

[72] E. Saund, *Finding perceptually closed paths in sketches and drawings*, IEEE Transactions on Pattern Analysis and Machine Intelligence **25** (2003), no. 4, 475–491.

[73] E. Saund, D. Fleet, D. Larner, and J. Mahoney, *Perceptually-supported image editing of text and graphics*, ACM Symposium on User Interface Software and Technology (UIST), 2003, pp. 183–192.

[74] T. W. Sederberg, *Algebraic piecewise algebraic surface patches*, Computer Aided Geometric Design **2** (1985), no. 1, 53–59.

[75] T. Sezgin, T. Stahovich, and R. Davis, *Sketch based interfaces: Early processing for sketch understanding*, Perceptive User Interfaces Workshop, 2001.

[76] M. Shilman and P. Viola, *Spatial recognition and grouping of text and graphics*, Eurographics Workshop on Sketch-Based Interfaces and Modeling, 2004.

[77] S. Simhon and G. Dudek, *Path planning using learned constraints and preferences*, IEEE Internation Conference on Robotics and Automation (Taipei, Taiwan), May 2003, pp. 2907–2913.

[78] ———, *Analogical path planning*, AAAI National Conference on Artificial Intelligence Conference, July 2004, pp. 537–543.

[79] ———, *Pen stroke extraction and refinement using learned models*, Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBM'04), August 2004, pp. 62–69.

[80] ———, *Sketch interpretation and refinement using statistical models*, Eurographics Symposium on Rendering, June 2004, pp. 23–32.

[81] R. Simmons and S. Koenig, *Probabilistic robot navigation in partially observable environments*, Proceedings of the International Joint Conference on Artificial Intelligence, 1995, pp. 1080–1087.

[82] S. Singh and M. C. Leu, *Optimal trajectory generation for robotic manipulators using dynamic programming*, ASME Journal of Dynamic Systems, Measurement and Control, vol. 109, 1989.

[83] J. Smith, A. Witkin, and D. Baraff, *Fast and controllable simulation of the shattering of brittle objects*, Graphics Interface 2000, no. Montreal, May 2000, pp. 27–34.

[84] D. Sturman, *Interactive keyframe animation of 3-d articulated models*, Graphics Interface '86, Tutorial on Computer Animation, 1986.

[85] R. S. Sutton, D. Precup, and S. Singh, *Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning*, Artificial Intelligence **112** (1999), no. 1, 181–211.

[86] R. Szeliski, *Bayesian modeling of uncertainty in low level vision*, Kluwer, 1989.

[87] D. Terzopoulos and K. Fleischer, *Modeling inelastic deformation: Viscoelasticity, plasticity, fracture*, Computer Graphics **22** (1988), no. 4, 269–278.

[88] M. Thorne, D. Burke, and M. van de Panne, *Motion doodles: An interface for sketching character motion*, ACM Transactions on Graphics **23** (2004), no. 3, 424–431.

[89] S. Thrun, *Probabilistic algorithms in robotics*, AI Magazine **21** (2000), no. 4, 93–109.

[90] A. N. Tikhonov and V. Y. Arsenin, *Solution of ill-posed problems*, Winston and Sons, 1977.

[91] X. Tu and D. Terzopolous, *Artificial fishes: Physics, locomotion, perception, behavior*, SIGGRAPH '94 Proceedings, July 1994.

[92] G. Turk and J. O'Brien, *Shape transformation using variational implicit functions*, Proceedings of ACM SIGGRAPH, August 1999, pp. 335–342.

[93] D. G. Ullman, S. Wood, and D. Craig, *The importance of drawing in mechanical design process*, Computers and Graphics **14** (1990), no. 2, 263–274.

[94] S. Ullman and A. Sha'ashua, *Structural saliency: The detection of globally salient structures using a locally connected network*, International Conference on Computer Vision, 1998, pp. 321–327.

[95] M. Unuma and R. Takeuchi, *Generation of human motion with emotion*, Computer Animation '93 Proceedings, 1993, pp. 77–88.

[96] A. J. Viterbi, *Error bounds for convolutional codes and an asymptotically optimum decoding algorithm*, IEEE Transactions on Information Theory **IT-13** (1967), no. 2, 260–269.

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

[98] L.-Y. Wei and M. Levoy, *Fast texture synthesis using tree-structured vector quantization*, Proceedings of ACM SIGGRAPH, 2000, pp. 479–488.

[99] W. Welch and A. Witkin, *Free-form shape design using triangulated surfaces*, Proceedings of ACM SIGGRAPH, July 1994.

[100] L.R. Williams and D.W. Jacobs, *Stochastic completion fields: A neural model of illusory contour shape and salience.*, Neural Computation **9** (1997), no. 4, 837–858.

[101] A. Witkin and M. Kass, *Space-time constraints*, Computer Graphics **22** (1988), 159–168.

[102] A. Witkin and Z. Popovic, *Motion warping*, Proceedings of ACM SIGGRAPH, August 1995.

[103] R. Zeleznik, K. Herndon, and J. F. Hughes, *Sketch: An interface for sketching 3d scenes*, Proceedings of ACM SIGGRAPH, 1996.