THE UNIVERSITY OF CALGARY

Curve Synthesis by Example

by

Meru Brunn

A THESIS
SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

August, 2006

THE UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled "Curve Synthesis by Example" submitted by Meru Brunn in partial fulfillment of the requirements for the degree of MASTER OF SCIENCE.

Dr. Mario Costa Sousa
Supervisor
Department of Computer Science

Dr. Faramarz Samavati
Co-Supervisor
Department of Computer Science

Dr. Brian Wyvill
Department of Computer Science

Dr. Gerald Hushlak
Department of Art

Date

Abstract

In this thesis, we present a method for constructing curves based on a nearly exact reproduction of a given set of silhouettes and contour-based line drawn work. We capture these line drawings with multiresolution analysis based on reverse subdivision, which extracts the characteristics of the drawn stroke. These captured styles are saved in a library and can then be applied to newly drawn strokes, letting a user create new drawings with the look of the original. We also support the use of styles with gaps or discontinuities, and can repeat a short style section seamlessly over a longer curve. We demonstrate results from both scanned images and interactive sketches, showing how we can capture these types of complex curves and let users without particular drawing skills easily reproduce them.

Acknowledgments

Here is where all the people that made this come together need to be mentioned. First and foremost, thanks to my supervisors, Mario Costa Sousa and Faramarz Samavati, for guiding this work through its various stages on the road to completion. Special thanks also to my NPR partner-in-crime, Kevin Foster, for never allowing a dull moment, and to the many others in the lab that made it such a great place to be. And of course, the people that made being in and around Calgary so much fun, during and after school hours: Reg Sawilla, Tony Grimes, Kristian McComb, Julia Taylor-Hell and Mike Gill. Finally, a big thank you to my mother, Margot, and sister, Ariel, for all the encouragement along the way!

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

For the past decades, the driving force behind most computer graphics research on rendering has been to generate realistic images, generally by creating results indistinguishable from photographs through simulating or approximating the physical interaction of light on surfaces.

More recently, the area of non-photorealistic rendering (NPR) has examined a separate problem: creating images that convey expressive or interpretive value [GG01, SS02]. Many of these methods try to reproduce stylized images, for example by mimicking the effects achieved by artists working with traditional tools and media. Stippling drawings [WS94, DHvOS00, Sec02], impressionist paintings [Hae90, Lit97, KGC00] and watercolour illustrations [Coc91, CAS*97] have all been successfully approximated, using a variety of methods including visual and physical rendering and observational modelling of the creative process that artists use.

In this research we investigate the NPR approach of *rendering-by-example* for creating artistic computer-generated reproductions of images and 3D models. The goal of rendering-by-example is to capture artistic styles and re-use them in an automatic or semi-automatic manner. The basic idea is to determine reproducible patterns that can be detached from an existing work and seamlessly transferred to a new one. Rather than trying to simulate the process of generating the image or finding an algorithm to produce it, characteristic elements are extracted from the original and applied to a new image that assumes that style. This central idea has found

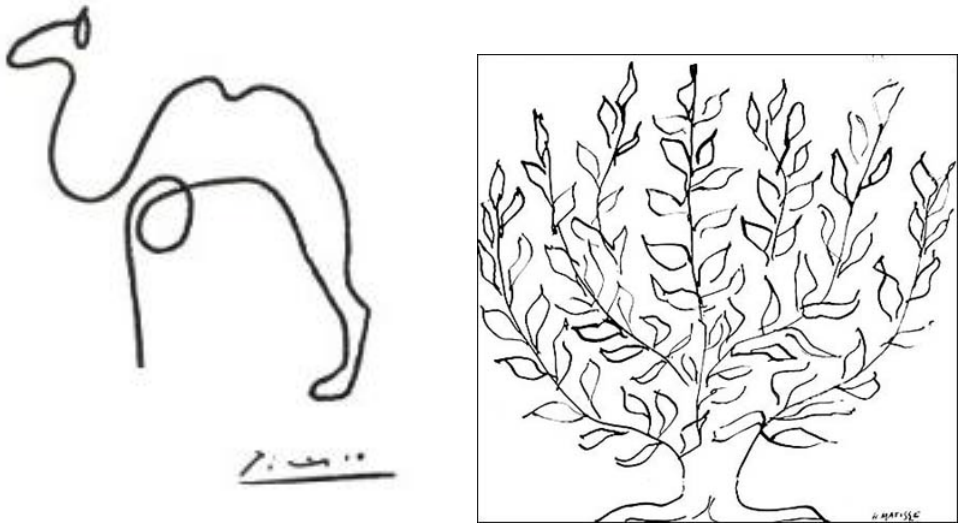
numerous applications in computer graphics. For example, Sloan *et al.* [SMGG01] use 2D image samples from paintings to generate spherical shading maps. These can then be used for texturing 3D objects in the colour and lighting style of the sampled artwork. And beyond rendering, Bregler *et al.* [BLCD02] re-use the stylized motion of traditionally animated cartoons to animate new characters. Their work makes the primary benefit of the technique clear: adopting it as an artistic aid or tool can result in considerable savings of time and effort and enable results unachievable by non-expert users.

1.1 The Problem

Line drawings are perhaps the earliest form of artwork, found at prehistoric sites in caves and in rock art throughout the world. In the modern era, however, the aesthetic attraction and prominence of painted works limited artistic drawings to the role of studies or pre-sketches. During the Italian Renaissance, Leonardo da Vinci's elaborate use of light and shade introduced the thought that drawings themselves could be art. This, in turn, led to the study and mastery of line styles and marks [Spe72, Act97]. We can trace from this the modern-day appreciation and collection of finished works that consist of drawings, for example by artists such as Picasso and Matisse. Figure 1.1 shows examples of drawings from prehistory through modern times, from the line illustrations of the earliest days of depiction to modern artworks.

This progression is also evident in the *contour drawings* we have used as input in this work. The natural objects of clouds, trees, and bushes drawn in this technique,

(a) (b)



(c) (d)

Figure 1.1: Historical examples of line drawing styles. (a) Prehistoric petroglyph from Writing-on-Stone Provincial Park, Alberta. (b) Sedge—da Vinci, circa 1510. (c) The Camel—Picasso, early 20th century. (d) The Bush—Matisse, 1951.

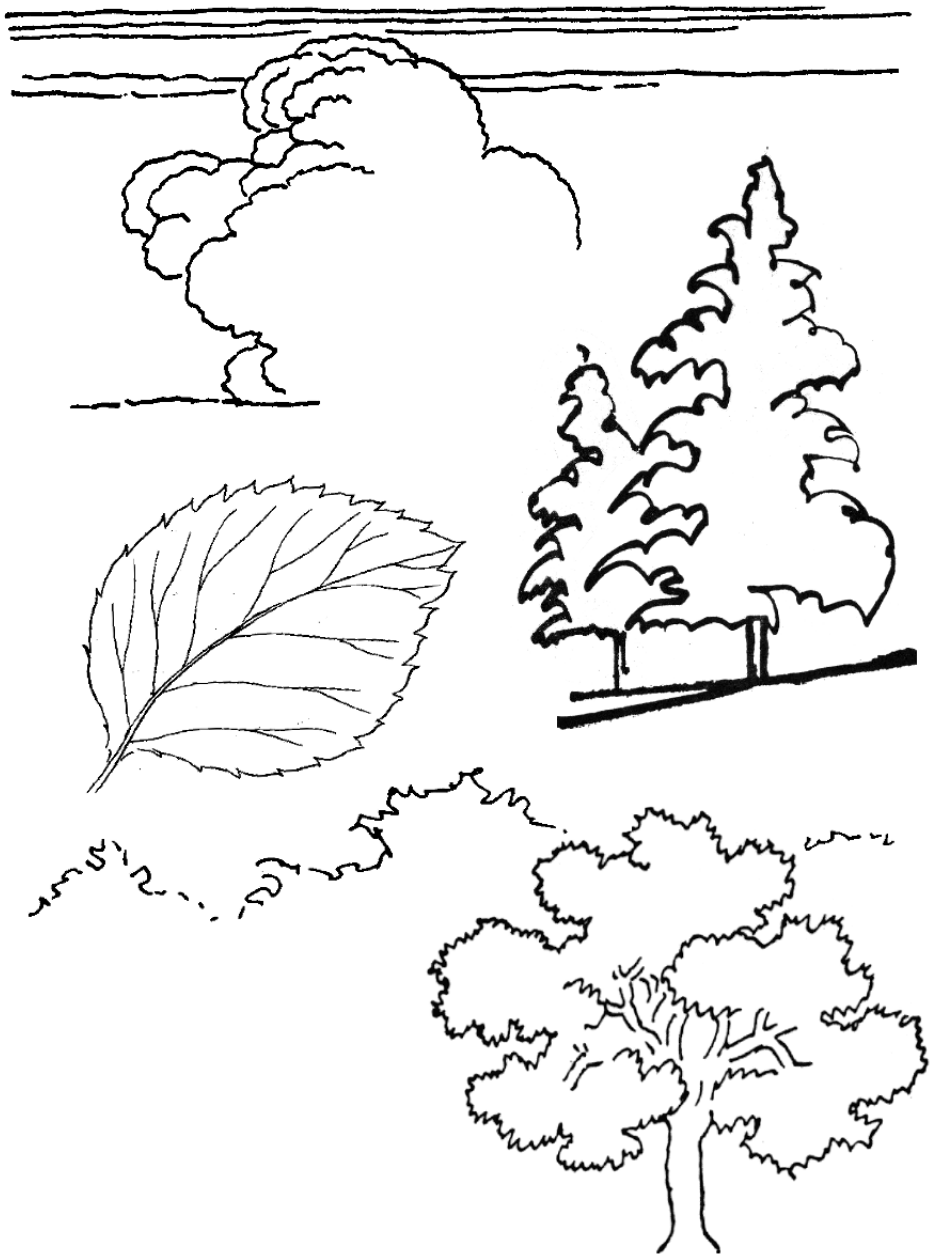


Figure 1.2: Artistic silhouettes and line styles used in traditional illustrations of landscape elements. These contour drawings emphasize object silhouettes and use strokes and indication to show shape and form without shading or texturing. From [Rei87]

shown in Figure 1.2, are typical of landscape elements in traditional architectural illustrations. Their stylized silhouettes emphasize important surface features, showing shape and form without using shading or texturing. This allows their curves to be expressively rendered, even in simple pen-and-ink or pencil sketches with monochromatic and fixed-width lines [Sul97, Cra00, Whi94, Nic41]. The detail in these types of drawings—where the physical outline of the object is more important than the line quality—requires skilled artists to create. And it is these complexities we would like to capture for easy reproduction by non-expert users.

Since these illustrations by their nature do not contain attributes such as width, colour, and texture, capture of these line styles must accurately reproduce all the fine details of the pen's stroke. The capture must contain both the micro-level variations that imply surface type, and the larger-scale, macro-level variations that imply shape, in order to reproduce the look of the drawings. In the examples in Figure 1.2, micro-level detail defines the fluffy characteristic of the clouds and the serrated edges of the leaf. In contrast, the macro level details are evident in the branch and foliage variations of the centre-right and bottom-right tree figures that let the viewer distinguish between deciduous and coniferous tree types.

The by-example techniques used to achieve reproductions of these have included (1) statistical techniques that approximate the curve's look by reproducing local features [KMM*02, SD04], and (2) multiresolution techniques that explicitly capture all the variations of a curve's shape along a path [FS94, HOCS02]. Our approach, which uses a multiresolution technique to acquire all the levels of detail, is outlined below.

1.2 Our Approach

Our aim is to capture a library of artistic silhouettes and stroke styles using a multiresolution technique for rendering-by-example. The key to our method lies in using multiresolution analysis to decompose a stroke into two elements: (1) the *path*, or basic direction of the stroke, and (2) the hand gesture *style*, or detail variations along the path that make up its character. The advantage to this technique lies in capturing the multiple levels of details in a form that can be reconstructed on an arbitrary new path.

Our system, shown in Figure 1.3, consists of an interactive application for acquiring new stroke styles and an illustration system for applying them. (1) We use existing artwork from sketchbooks, instructional materials, architectural landscape illustrations, and also works that are hand drawn by the user. (2) Original drawings are scanned at high resolution and used as input, from which line and silhouette curves are extracted, either automatically or with user direction. (3) These are passed to a style extraction routine that isolates the characteristic look of the curve, which is then saved in a re-usable library of styles (4). Our interactive illustration system then allows the user to create a sketched image and transfer stored styles to its strokes. (5): within that system, the user may sketch a drawing with a tablet or mouse and (6), (7) interactively apply captured styles to its strokes. We refer to these strokes as *base curves* for the new styles. Attributes such as style re-orientation, repetition, and level-of-detail selection are provided to fine-tune the final appearance of the strokes.

Our approach uses the local multiresolution filters of Bartels and Samavati [BS00]

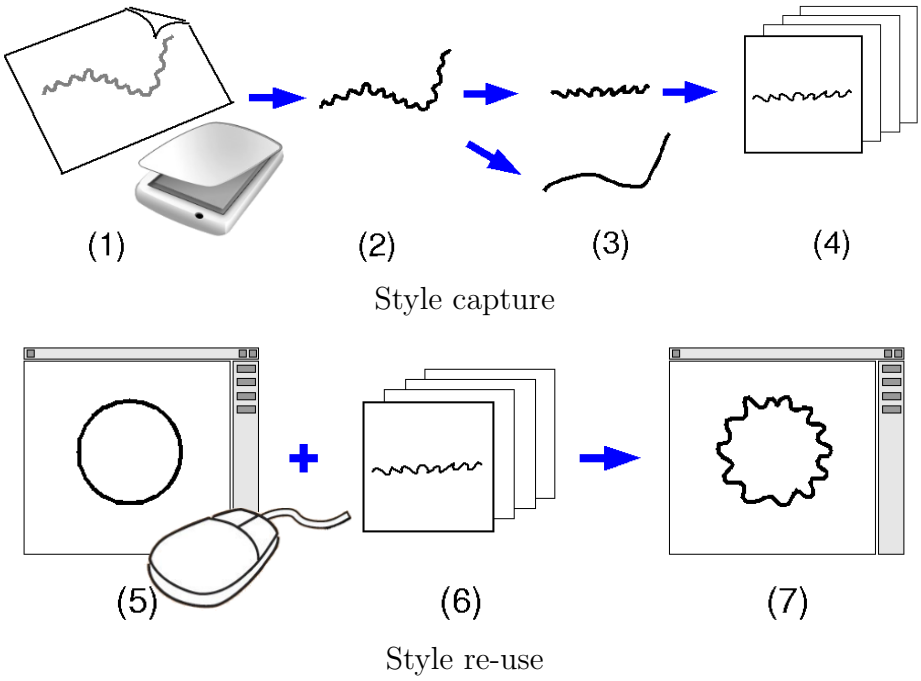
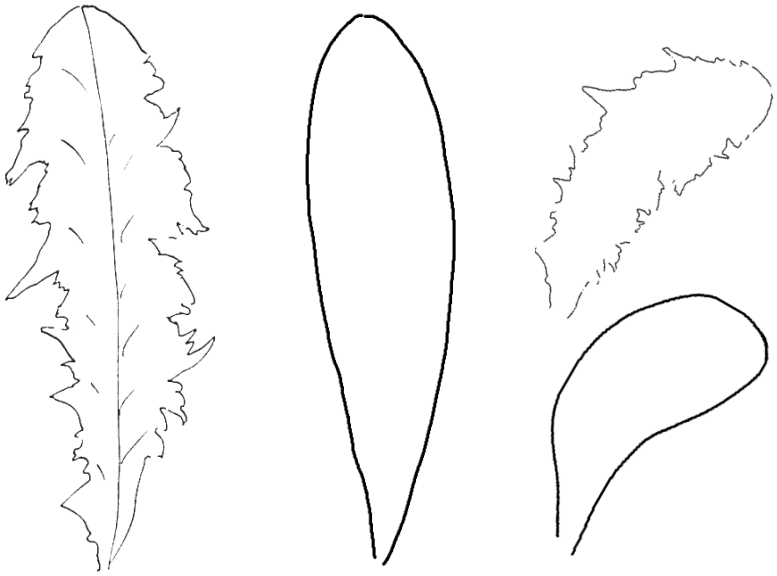


Figure 1.3: Overview of the interactive style capturing and re-use system. Top: Capture of a specific style from an original drawing by (1), (2) acquiring a scanned image and selecting a stroke from it; (3) separating the stroke style and path using multiresolution decomposition; (4) storing the style in the library. Bottom: Re-use of a stored style by (5) creating a new illustration; (6) selecting and applying a stored style using multiresolution reconstruction; (7) resulting in an illustration in the style of the captured and stored original.

and global-filter producing method of Samavati and Bartels [SB99] to perform this decomposition. To re-use the style we perform the multiresolution synthesis operation, reversing the decomposition and reconstructing the stored style on a new path, and generating a new curve, shown in Figure 1.4. As part of that reconstruction, our algorithms align the extracted style with the orientation of the new path, and allow the style to be seamlessly repeated along the curve. We also capture and re-use styles that contains gaps or holes, and can automatically extract these from existing



(a) (b) (c)

Figure 1.4: The dandelion sketch (a) consists of gesture strokes which represent its characteristic serrated leaf pattern. Abstracting the basic form of the figure reveals the path (b), which follows the basic direction and orientation of the drawing device. Multiresolution decomposition extracts the gesture strokes from the path, in a way that allows them to be re-applied to a new path (c, bottom) to create a new illustration (c, top).

illustrations. We also note that the multiresolution approach offers the considerable benefit of built-in level-of-detail optimization, reducing the space and time needed to store and generate the curves.

1.3 Contributions

The main contribution of our work is to provide a system for high-quality style extraction and re-application using multiresolution analysis and synthesis. As we discuss in Chapter 2, unlike previous work in this area, our method avoids tedious

construction of large sets of example strokes and does not require computationally expensive statistical processing to generate new curves with the styles. We achieve nearly exact reproduction of styles from single input samples. Our multiresolution approach is easy to implement and provides very efficient style application using what is essentially a simple filtering of the input curve points. Compared to previous multiresolution approaches, we have (1) multiple high quality global filters, (2) very fast local filters, (3) a method for repetition of a short style segment along a curve, (4) a method for using styles consisting of multiple discontinuous strokes, such as hatching styles.

1.4 Overview

The rest of this thesis is arranged as follows: Chapters 2 and 3 provide background with a review of the related literature in line and curve rendering-by-example and a description of the multiresolution operations and filters behind our work.

The next three chapters describe the major stages of the pipeline of our multiresolution stroke style capture and re-use system. Capturing input curves from scanned illustrations, Chapter 4; extraction of the characteristic gesture styles of curves using multiresolution analysis, Chapter 5; and re-use of the extracted styles on new curves to generate illustrations with the same look, Chapter 6.

Results are presented in Chapter 7, including a more in-depth comparison of our method with wavelet-based approaches. Conclusions and directions for future research are described in Chapter 8.

Chapter 2

Line Rendering-by-Example

The papers we review here focus on rendering-by-example techniques for curve and line styles, a relatively recent line of NPR research. We classify the major works in this relatively recent line of non-photorealistic computer graphics research below, categorizing them into two groups: statistical and matching methods, (Section 2.1), and multiresolution methods, (Section 2.2).

These groups can be broadly characterized as "top-down" and "bottom-up" approaches, respectively. The top-down approach of statistical analysis creates output with variations that match that of the input, determined through probabilistic matching algorithms. This typically regenerates a discovered pattern from the input onto new data with controllable, randomized variations that effect a similar, but not exactly copied look. The bottom-up approach of multiresolution analysis captures levels of details from the input and rebuilds them directly to create the output. Generally, this results in a more exact reproduction of the original as the variations are simply transferred, rather than regenerated through statistical processes.

For this work, we have used a multiresolution technique to achieve these types of nearly exact reproduction. However, we also have some control over the reproduced look of captured styles. By manipulating reconstruction parameters such as style repetition we can generate different output using the same input style, as shown in Chapter 6.

Table 2.1 summarizes the works that have focused on stroke style capture. These

Statistical and matching methods

Authors	Year	Main technique
Jodoin *et al.* [JEGPO02]	2002	Gibbs probability
Kalnins *et al.* [KMM*02]	2002	Markov model
Freeman *et al.* [FTP03]	2003	Nearest neighbour matching†
Simhon and Dudek [SD04]	2004	Markov model‡

Multiresolution methods

Authors	Year	Main technique
Finkelstein and Salesin [FS94]	1994	Wavelets
Hertzmann *et al.* [HOCS02]	2002	Curve analogies

†Does not use probabilistic or statistical methods
‡Uses a hierarchical approach

Table 2.1: Categorization of previous work in capturing and re-using artistic styles. The statistical techniques are the most prominent area of research in recent years.

can also be separated as using either approximations to exact reproduction (statistical), or exact reproductions (multiresolution).

2.1 Statistical and Matching Methods

These approaches use statistical analysis of the input to construct a model that probabilistically produces output that has the same characteristics or "structure." Matching techniques use non-statistical functions to assign styles from an input to an output curve. We group both methods together on the basis of their common input and output format requirements: For accurate results, both require multiple inputs for a large "seed" space from which to either initialize probabilistic generators

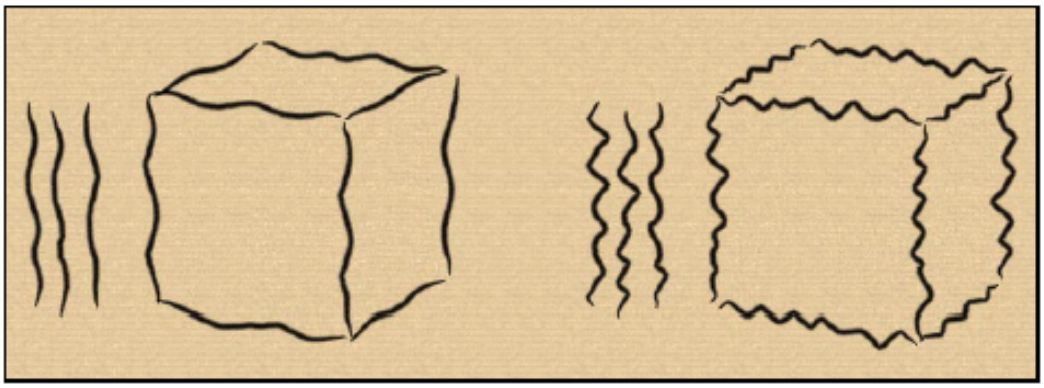


Figure 2.1: Example results achieved by the multistroke input matching technique of Kalnins *et al.*'s [KMM*02] interactive NPR system.

or build matching libraries.

2.1.1 WYSIWYG NPR: Drawing Strokes Directly on 3D models

In the first category, Kalnins *et al.*'s [KMM*02] interactive NPR system uses stroke rendering-by-example for quickly repeating a style, or "crease stroke" around an object's silhouette. Their method uses several example strokes, which provide input for automatic synthesis of "similar," statistically generated, strokes. They store strokes as a spline *base path* and *offset list* of vectors perpendicular to the path representing small "wiggles" relative to the base path. Stroke generation occurs either by repeating the stroke to cover a new, longer path, or with synthesis-by-example of new strokes. To synthesize a new stroke, a Markov random field is created. It links the similarity of an example curve segment, represented as a pair of offsets, with the probability of adding the segment that consists of the offset pair. Then, to create the new stroke, an initial offset point is selected and the field

traversed, adding high-probability segments to the new curve.

A limitation of their synthesis-by-example method lies in the capture and storage of stroke offset information. As stated in Appendix A of [KMM*02], self-intersecting strokes (backtracking along a stroke path) and discontinuous strokes (breaks along a stroke path) are not supported in their system. As we describe in Chapter 5, our system supports these more complex stroke styles. Figure 2.1 shows an example of strokes synthesized by example from several input samples in their system.

2.1.2 Sketch Interpretation and Refinement using Statistical Models

Simhon and Dudek [SD04] use Hidden Markov models to statistically re-generate curve styles onto new illustrations. Their system is targeted toward enabling non-expert illustrators to create 2D drawings from captured curve outlines and attributes such as thickness, colour, and texture. Their approach infers a detailed, generated curve onto a coarse path by finding correspondence between undetailed "control" curves and detailed example drawings. The control curves and detailed curves are provided as input to train the statistical model. It computes probabilities for both the likelihood of individual sample points occurring along the curve path and, at the scene level, for probabilities of entire curve sections being followed by others. This allows for scene level constraints that restrict the chances of the system generating, for example, a tree shape above a cloud shape.

Like the other statistical approaches, in order to produce reliable output, their system requires a training set of at least several examples to seed the generator. In addition, each example stylized sketch requires corresponding control curves to extract the learned style. In contrast, we require only a single example and can

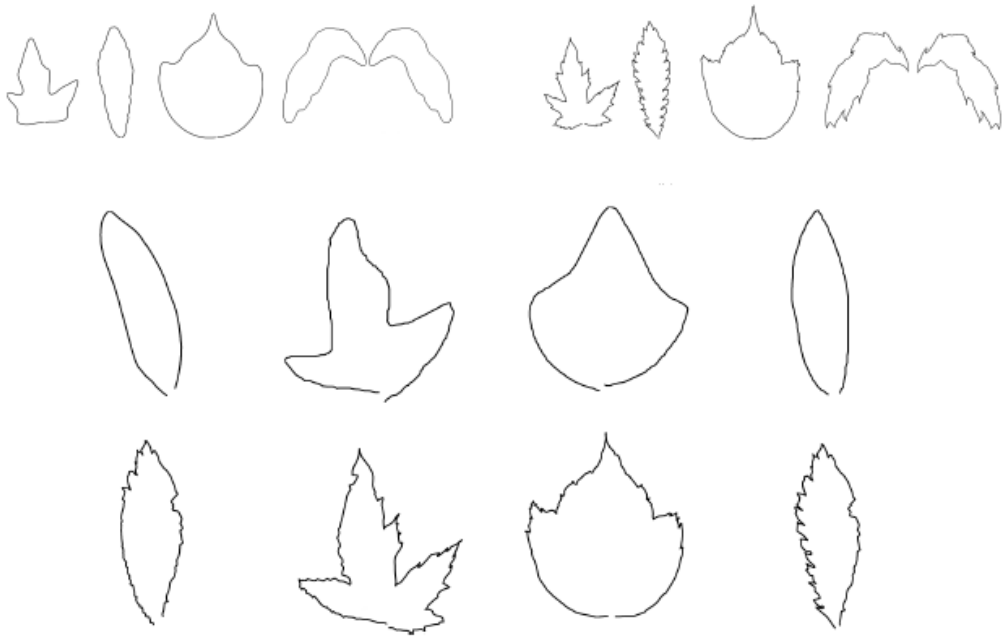


Figure 2.2: Example of input and results from the statistical system of Simhon and Dudek [SD04]. The top row shows a control and corresponding detailed curve set, which are used to train the Hidden Markov model. The center row is a set of base paths that an end-user would input to the system, which generates the stylized versions, bottom row.

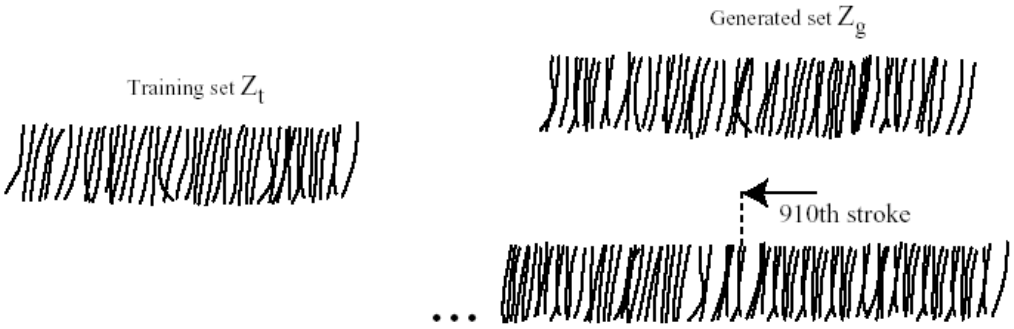


Figure 2.3: An example of hatching styles synthesized by the system of Jodoin *et al.* [JEGPO02] from the input pattern sketched at left. The top right shows a short, randomized segment in the style of the original, while the bottom shows the result of using a too small input sample on a longer segment, resulting in an obvious repetition of the pattern.

determine the curve style directly, without specific associated control curves. An example input and output set from their system is shown in Figure 2.2.

2.1.3 Hatching by Example: a Statistical Approach

Another statistical synthesis technique, by Jodoin *et al.* [JEGPO02], synthesizes a set of hatching strokes that "looks" like a given sample. Their technique uses a statistical model that conditionally selects strokes from the sample hatching pattern to generate a similar output, rather than generating entirely new strokes. Their statistical approach uses a re-application of Shannon's *N-gram entropy*, a statistical measure from information theory of the conditional probability of the next letter of a word when the preceding ones are known. This measure is used for a generation model for synthesizing strokes by repeatedly selecting from a pool of candidate hatching strokes taken from the example input. The candidates picked are determined by

their Gibbs probability, which varies inversely with the exponential of the Euclidean distance between the most recent strokes and those previously selected. This results in a new generated hatching stroke set that approximates the sample. A method of computationally validating the generated sequence is also presented. It consists of a statistical measure for comparing stroke sets using the relative distance between previous strokes and the current stroke of the generated sequence to measure the fit of the approximation.

Due to the design of the system, new strokes are not generated, they are always drawn from the sample set. To avoid a uniform appearance and evident repeated patterns, a correspondingly larger library of input examples must be provided. Figure 2.3 shows a sketched training set and both long and short generated patterns derived from it. This issue is common to all approaches that reproduce short samples over a longer curve. One limitation is that the complex processing and formulas required to compute the input probabilities also hinder efficient processing of large or complicated input patterns. In contrast, as shown in Chapter 3, we only use simple, fast arithmetic operations while processing hatching styles, for example in Figure 7.5. We support these line styles as an additional feature of our curve synthesis system.

2.1.4 Learning Style Translation for the Lines of a Drawing

Freeman *et al.*'s [FTP03] "style translation" system uses stroke similarity to coerce a line-drawn illustration into a new style. They manage this translation by matching the strokes of the input to a style in a multi-style library of samples. The matched strokes are then translated by replacing them with a corresponding stroke in an alternate style from the library.

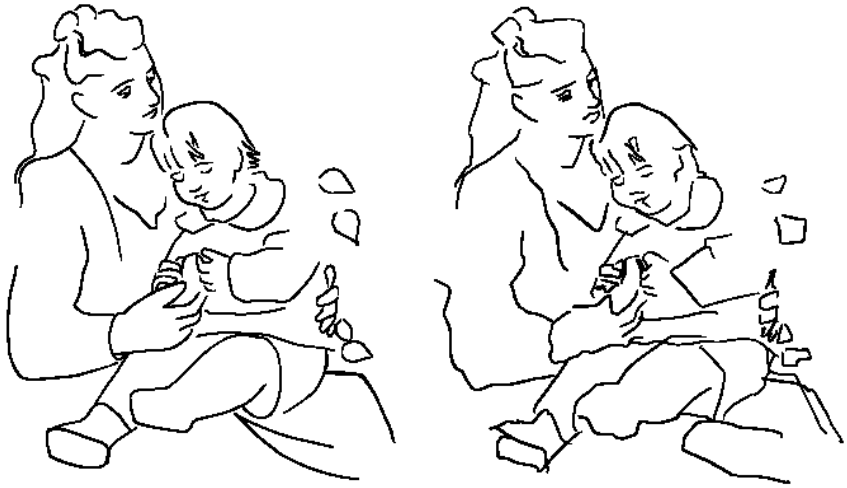


Figure 2.4: Freeman *et al.* [FTP03] introduce the idea of "style translation," where an input image's strokes are matched to a library of generic curves that can then be exchanged with similar curves in different styles. Left: an input sketch and (right) its corresponding matched and translated output from the style library.

The system proceeds by first matching the strokes of the input illustration to a generic style in the library. Since there is no guarantee that the illustration's strokes will be a good match with the generic library style, a combination of two approaches is used. The set of k strokes from the generic style that most closely match the input stroke is found. The matching criteria is a nearest neighbour algorithm, measuring the simple Euclidean distance between strokes. This set is then linearly-combined to generate weighting coefficients that describe the translation from the input stroke to the library strokes. The linear combination is as follows: A matrix, A_1^y, is formed with k columns consisting of the nearest strokes. Given an input stroke $\vec{y}$, the least-squares solution $\vec{x}$ to $\vec{y} = A_1^y \vec{x}$ is the linear combination that describes the input stroke in terms of its nearest matches from the style library. Translation is then a simple matter of retrieving the alternate styles of all the illustration's strokes and

replacing the columns of A_1^y with the corresponding strokes of the new style. This new matrix, A_2^y, describes the translation of the input line $\vec{y}$ to the new style.

The primary drawback of the system is that translation may only occur between style sets that have been entered into the library. If the library stroke set is too dissimilar to the input, the matching step will not produce acceptable output in the translation stage. Ultimately, gaining generality of input in their system is only possible by maintaining a several-thousand element table of possible strokes in various styles. However, even with the large number of example strokes, it is possible that the translated style will be a poor fit as there is no user control over the assignment process. As shown in our results, we are capable of performing stylization with only a single example style. This style can be repeated over all strokes of the image, or restricted to certain strokes, or combined with arbitrary other styles to create a final composition. The styles may also include discontinuities or overlapping parts, and can be automatically extracted from scanned images. Figure 2.4 shows an example input and stylized result from their style translation system.

2.2 Multiresolution Methods

Multiresolution methods for stroke style rendering-by-example operate on the insight that a curve's style can be extracted as a layer of detail from the curve's basic path. The details are represented as deviations perturbing the smoothly varying path into a more complex shape. For this approach, the principal consideration is how to identify, represent, and re-apply these deviations. Hertzmann *et al.* [HOCS02] view the details as a set of simple local transformations that can align the points of two

curves while Finkelstein and Salesin [FS94] use wavelet decomposition to "peel" off the high resolution details of a curve, iteratively separating the style and resulting in a final low-resolution path curve.

2.2.1 Curve Analogies

Hertzmann *et al.* [HOCS02] synthesize strokes from examples by creating a complement curve that is analogous to a given pair. Given related curves A and A', and new path B, the style of A' is extracted in a form that allows it to be applied to B to generate the new curve B' that is analogous to A'. In contrast to Jodoin *et al.*'s [JEGPO02] stroke-level synthesis, Hertzmann *et al.* view a curve as a collection of local features that can be individually extracted and re-assembled into new curves.

First, the relationship between the related curves A (base) and A' (style) is captured by computing an equivalent parametrization for A and A'. Offsets of features along the curve are then computed using Euclidean distances of centers-of-mass (the weighted midpoint of curve samples around a neighbourhood). A rigid transformation between A and B is also found, which gives the transformation needed to move the offsets from the local area of the A' curve to the new B'curve. The similarity of B' to A' is determined by evaluating a cost function, $E(B')$, which minimizes the distance between $B'$and A' in three ways:

1. The shape of B' as compared to A'. This is measured by the neighbourhood distance metric which minimizes the distance between a set of K local samples about two (parametrically) related areas of B' and A'.

2. The relationship of B's shape to B' as A's shape is to A'. Measured by the

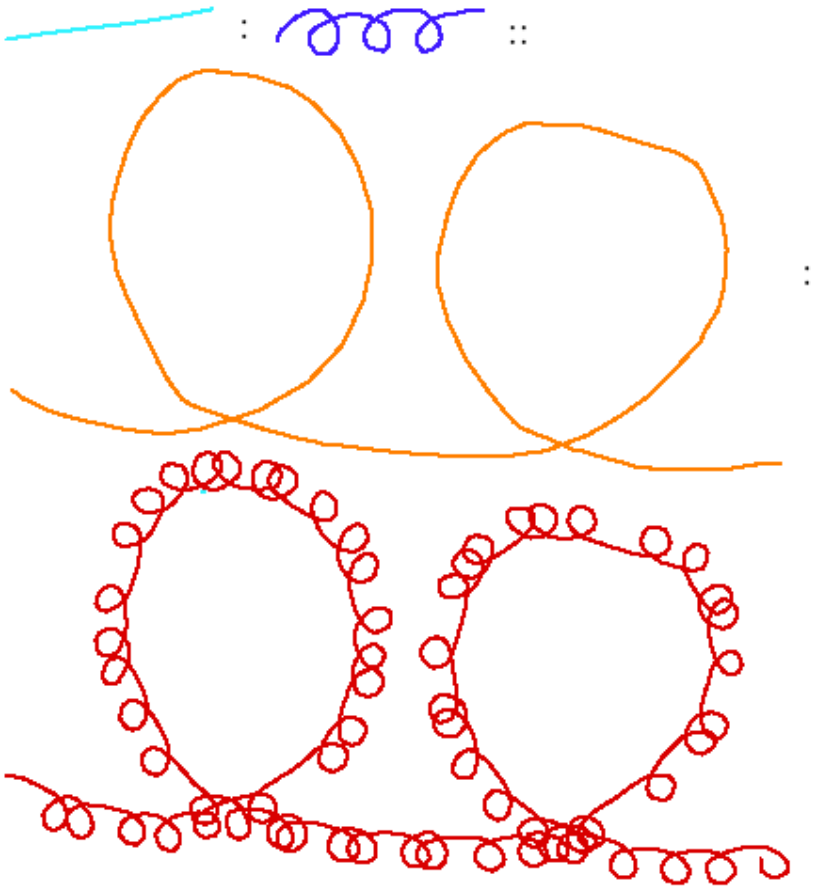


Figure 2.5: Hertzmann *et al.* [HOCS02] define a style synthesis method using analogies, where a style is deduced from the relationship of two curves, one simple and the other complex. The analogy step then attempts to modify a new simple curve into a more complex one such that its relationship mirrors that of the original two. Top: an analogy example pair. Center: the input curve. Bottom: the generated output.

same distance metric as (1), using instead the curves B and A.

3. The relationship of B's position and orientation to B' as A's is to A. Measured by the distance between the centers-of-mass of a set of local samples after applying the rigid transformation found previously.

Ideally, all points in a region of B' (the synthesized curve) should match some region in A', though the regions are mixed randomly to create the similarity in the new curve. The cost function takes into account the offset of the region's points from the curve segment's center-of-mass. If the chosen curve region is a good fit, it is added to the generated curve.

A major drawback of the system is the complexity of computing the similarity between the curves and synthesizing new curves. At the publication date in 2002, the computations required to generate the analogous curves required several seconds for a single curve, precluding its use for real-time synthesis. While that probably is no longer the case today, we find that our similar results can be achieved with simpler computations, allowing for interactive speeds with larger and more complex input. Our method offers the advantage of completely separating the style from the input without the requirement of the analogy step. In our approach, the multiresolution technique automatically determines it by reducing the curve down to its basic path with repeated decomposition (Chapter 3). This allows us to create a library of styles that are completely independent of the original curve from which they were extracted. We also provide a method for using curve images as input (Chapter 4) and we support broken, or non-continuous curves and styles (Section 5.3). Finally, the multiresolution decomposition that we use only requires that the total number

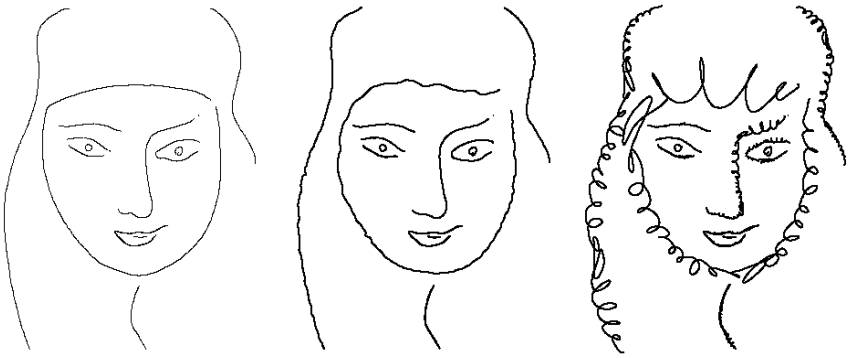


Figure 2.6: Finkelstein and Salesin [FS94] introduce the notion of curve editing on two levels: affecting the "sweep," or path of the curve without changing the details, or style, and changing the style without affecting the sweep. This image shows their results with the latter process, performing a style change of a fixed illustration. We provide an interactive editor for both processes, including a re-usable style library and applier for changing the details, and an interactive multiresolution curve editor for playing with curve sweep.

of details stored in the hierarchy be equal to the original input size, rather than the multiple levels of sampled curves required in the pyramid method. Figure 2.5 shows a result of the curve analogy system. Our analogous result, using only a short style sketch and new input curves, is in Figure 7.1.

2.2.2 Multiresolution Curves

In this seminal work, Finkelstein and Salesin [FS94] describe a wavelet-based multiresolution method that can capture and re-use a curve style. The multiresolution representation allows convenient curve control for editing the path or details of the curve, switching between detail levels, and re-applying details to a modified or edited curve. They also create a "curve character library" of extracted details for applica-

tion to completely new curves. Figure 2.6 shows their system in action, changing the look of an illustration without affecting its underlying curve paths.

Part of our approach, the global multiresolution construction (described in Chapter 3), corresponds to their method, so we simply note similarities and differences to their approach here. The method of Samavati and Bartels [SB99], provides a general construction based on reversing subdivision, which generates an equivalent multiresolution representation to that of Finkelstein and Salesin [FS94]. Conceptually, they differ primarily in the creation of the multiresolution filtering matrices. Though [FS94] construct endpoint-interpolating cubic B-spline wavelets, their construction also appears to be a special case, limited to cubic B-splines. In contrast, [SB99] avoid explicit construction of the wavelet basis function and the associated evaluations thereof by using multiresolution methods from reversed subdivision. They also allow use of discrete curve point data directly.

Compared to [FS94], our multiresolution based rendering-by-example system provides the following additional features:

- Two different global multiresolution filters for extracting curve styles, important for getting good results when re-applying (Chapter 3)
- Alternatively, two local multiresolution filters, which achieve good quality extractions but much more efficiently (Chapter 3)
- A method for using styles consisting of multiple discontinuous strokes, such as hatching styles, which generates similar results to the hatching-pattern specific system of Jodoin *et al.* [JEGPO02] (Section 5.3)

- A simple and efficient method for orienting the style to the curve tangent using the extracted discrete details directly (Section 6.2)

- Methods for repeating a short style segment along a curve, by simple repetition and by multiresolution blending, achieving results like those of Hertzmann *et al.* [HOCS02] (Section 6.3)

Chapter 3

Multiresolution Methods

In this research, we have used multiresolution methods to extract and apply curve styles. The primary advantage of this approach is the ability to separate the style as *details* from the curve's coarse level approximation, or *base path*, in a simple and efficient manner. For our style-by-example purposes, the important observation is that these captured details can be used independent of the underlying base. In concrete terms, the style details generated by multiresolution decomposition represent perturbations of the curve path. This idea, first exploited for rendering-by-example by Finkelstein and Salesin [FS94], lets us define the capture of a style as extracting the set of details required to reconstruct the style curve. Re-use of the style then consists of reconstructing those details on a new curve. This representation of the artistic stroke as a low resolution path of coarse points and high resolution style details lets us transfer the captured style to an arbitrary new curve. Figure 3.1 shows how relative detail vectors are used to modify the style of a curve.

3.1 Analysis and Synthesis

In general terms, multiresolution analysis is used to create a hierarchical representation of a given function. It includes a lowest-level coarsest approximation to the function—the base path—and successive detail levels. Together, these form a *filter bank*, which encapsulates the multiresolution representation. The filter bank is

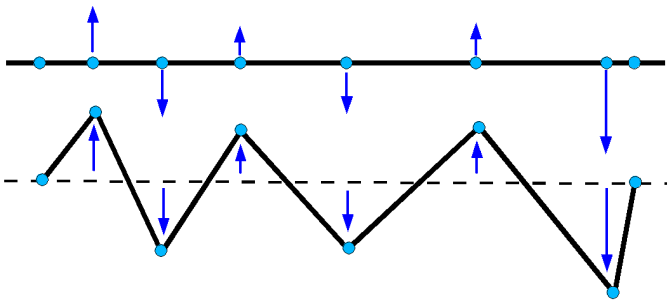


Figure 3.1: Using relative detail vectors to generate a complex curve from a simple curve. In this simplified characterization of the reconstruction process the vertices of the straight piecewise linear curve (top) are perturbed by the blue relative detail vectors to generate a jagged result (bottom).

generated via repeated decomposition of the original function using filters defined by a scheme. *Synthesis*, the reverse process, regenerates the original function by reconstructing it from the base path and detail levels of the filter bank [SDS96].

The decomposition steps are performed by multiplying the input with two multiresolution filtering matrices. The first, A, downsamples, generating a lower-resolution approximation, while the B matrix extracts the detail lost from the downsampling. Reversing decomposition uses two more matrices. P, the subdivision matrix, restores the resolution of the curve before decomposition. Q does the analogous operation for the captured detail vectors. These two combined reverse the effect of A and B. As a specific example, the make-up and elements of these filters for the local B-spline scheme used in this work are shown in Figures 3.2 and 3.3. Samavati and Bartels [SB04] provide an introduction to the use of the filters.

Mathematically, we can specify the multiresolution operations at a high level in terms of the four filter matrices A^n, B^n, P^n and Q^n of a specific scheme. The

$$\mathbf{A}^n = \begin{bmatrix} \mathbf{A}_s^n \\ \mathbf{A}_r^n \\ \mathbf{A}_e^n \end{bmatrix} \quad \mathbf{A}_s^n = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & \cdots \\ -\frac{1}{2} & 1 & \frac{3}{4} & -\frac{1}{4} & 0 & 0 & \cdots \end{bmatrix}$$

$$\mathbf{A}_r^n = \begin{bmatrix} 0 & 0 & -\frac{1}{4} & \frac{3}{4} & \frac{3}{4} & -\frac{1}{4} & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & -\frac{1}{4} & \frac{3}{4} & \frac{3}{4} & -\frac{1}{4} & \cdots \\ & & & & & & & & \ddots \end{bmatrix}$$

$$\mathbf{A}_e^n = \begin{bmatrix} \cdots & 0 & 0 & -\frac{1}{4} & \frac{3}{4} & 1 & -\frac{1}{2} \\ \cdots & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{B}^n = \begin{bmatrix} \mathbf{B}_s^n \\ \mathbf{B}_r^n \\ \mathbf{B}_e^n \end{bmatrix} \quad \mathbf{B}_s^n = \begin{bmatrix} -\frac{1}{2} & 1 & -\frac{3}{4} & \frac{1}{4} & 0 & 0 & 0\ldots \\ 0 & 0 & -\frac{1}{4} & \frac{3}{4} & -\frac{3}{4} & \frac{1}{4} & 0\ldots \end{bmatrix}$$

$$\mathbf{B}_r^n = \begin{bmatrix} 0 & 0 & 0 & 0 & \frac{1}{4} & -\frac{3}{4} & \frac{3}{4} & -\frac{1}{4} & 0 & 0\ldots \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & -\frac{3}{4} & \frac{3}{4} & -\frac{1}{4}\ldots \\ & & & & & & & & \ddots & \end{bmatrix}$$

$$\mathbf{B}_e^n = \begin{bmatrix} \cdots & 0 & 0 & \frac{1}{4} & -\frac{3}{4} & 1 & -\frac{1}{2} \end{bmatrix}$$

Figure 3.2: The filtering matrices for multiresolution decomposition using the local B-spline scheme. When multiplied with a column vector of points, A (top), coarsens the curve, resulting in a lower-resolution approximation with only half the points. B (bottom), captures the details lost during coarsening that are required to reconstruct the original. From [SB04].

$$\mathbf{P}^n = \begin{bmatrix} \mathbf{P}^n_s \\ \mathbf{P}^n_r \\ \mathbf{P}^n_e \end{bmatrix} \qquad \mathbf{Q}^n = \begin{bmatrix} \mathbf{Q}^n_s \\ \mathbf{Q}^n_r \\ \mathbf{Q}^n_e \end{bmatrix}$$

$$\mathbf{P}^n_s = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & \cdots \end{bmatrix} \qquad \mathbf{Q}^n_s = \begin{bmatrix} 0 & 0 & 0 & 0 & \cdots \\ \frac{1}{2} & 0 & 0 & 0 & \cdots \\ -\frac{3}{4} & \frac{1}{4} & 0 & 0 & \cdots \\ -\frac{1}{4} & \frac{3}{4} & 0 & 0 & \cdots \\ 0 & -\frac{3}{4} & -\frac{1}{4} & 0 & \cdots \\ 0 & -\frac{1}{4} & -\frac{3}{4} & 0 & \cdots \end{bmatrix}$$

$$\mathbf{P}^n_r = \begin{bmatrix} 0 & \frac{3}{4} & \frac{1}{4} & 0 & 0 & \cdots \\ 0 & \frac{1}{4} & \frac{3}{4} & 0 & 0 & \cdots \\ 0 & 0 & \frac{3}{4} & \frac{1}{4} & 0 & \cdots \\ 0 & 0 & \frac{1}{4} & \frac{3}{4} & 0 & \cdots \\ 0 & 0 & 0 & \frac{3}{4} & \frac{1}{4} & \cdots \\ 0 & 0 & 0 & \frac{1}{4} & \frac{3}{4} & \cdots \\ & & & & & \ddots \end{bmatrix} \qquad \mathbf{Q}^n_r = \begin{bmatrix} 0 & 0 & \frac{3}{4} & -\frac{1}{4} & 0 & 0 & \cdots \\ 0 & 0 & \frac{1}{4} & -\frac{3}{4} & 0 & 0 & \cdots \\ 0 & 0 & 0 & \frac{3}{4} & -\frac{1}{4} & 0 & \cdots \\ 0 & 0 & 0 & \frac{1}{4} & -\frac{3}{4} & 0 & \cdots \\ 0 & 0 & 0 & 0 & \frac{3}{4} & -\frac{1}{4} & \cdots \\ 0 & 0 & 0 & 0 & \frac{1}{4} & -\frac{3}{4} & \cdots \\ & & & & & & \ddots \end{bmatrix}$$

$$\mathbf{P}^n_e = \begin{bmatrix} \cdots & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \cdots & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{Q}^n_e = \begin{bmatrix} \cdots & 0 & 0 & 0 & \frac{1}{2} \\ \cdots & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 3.3: The filtering matrices for reconstruction using the local B-spline scheme. The P matrix subdivides coarsened points from downsampling with A, restoring the original resolution. Q (right), does the equivalent with the captured details. The curve points and details are then added together, restoring the original detailed, high resolution curve. From [SB04].

superscripts here refer to the decomposition level, where n is the original, highest resolution input. Given a column vector C^n of samples, a lower-resolution level C^{n-1} is created via decomposition by multiplying with the *analysis filters* A^n and B^n. The matrix A^n is used to downsample the input, giving a coarsened approximation to C^n:

$$C^{n-1} = A^n C^n. \tag{3.1}$$

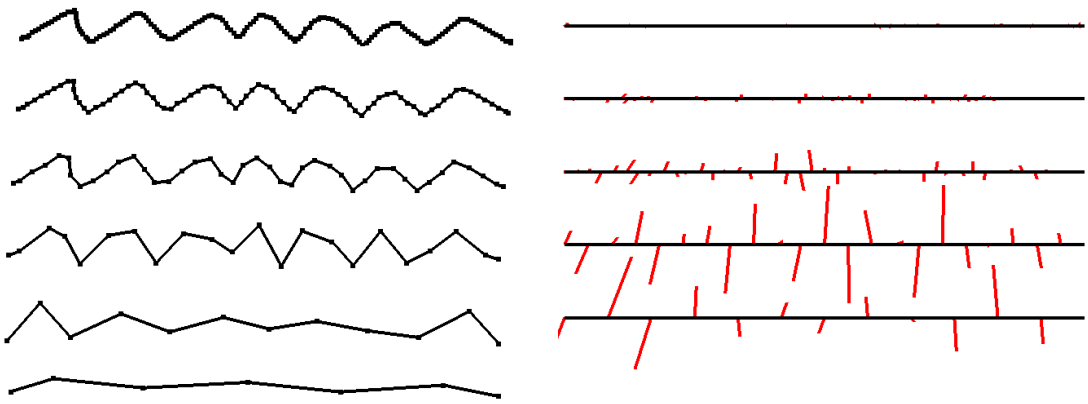
The *details* D^{n-1} lost through the downsampling are captured using B^n:

$$D^{n-1} = B^n C^n. \tag{3.2}$$

Recovering C^n, the previous level's points, is called *reconstruction*. It involves refinement of the coarsened points C^{n-1} and details D^{n-1} using the *synthesis filters* P^n and Q^n, which reverse the operations of A^n and B^n:

$$C^n = P^n C^{n-1} + Q^n D^{n-1}. \tag{3.3}$$

The multiresolution operations are successively applied to each generated level creating a hierarchy of details that gives us the original input when reconstructed. Although A^n, B^n, P^n, and Q^n are varied at different levels, they have a regular and repetitive sparse structure. In particular, P^n and Q^n are banded, with compact bands and with each subsequent column or row being a shifted version of the first, characteristic one, apart from the first and last few. This structure is guaranteed by the properties of the multiresolution analysis and synthesis matrices we use, described below in Section 3.3. It is also the primary factor in allowing the efficient



(a) (b)

Figure 3.4: Decomposition of a sketched style into a series of levels of (a) lower-resolution approximations, or C, and (b) their associated detail vectors, QD (scaled 3x for illustration). The first decomposition of the original curve generates the next curve below it and the details on the right, and so on. Note that at the highest level (top right), the details have almost no influence on the visible shape of the curve.

implementation of these operations, as described in Section 5.2. From here we will drop the superscripts from the matrices when it does not produce ambiguity.

Unlike $C^n..C^0$ and $D^{n-1}..D^0$, $Q^nD^{n-1}..Q^1D^0$ consist of vectors which perturb the curve into the higher-level path. Figure 3.4 shows the process of decomposition along with the generated levels. The hierarchy of decomposition starts from the top left (original curve). The right side then shows the extracted details after one step of decomposition and the new lower-resolution curve is on the next line. The successively lower-resolution approximations C^i to the original curve (top left) are shown down to the base path C^0 (bottom left). The right column shows the associated detail vectors D^i extracted at each level.

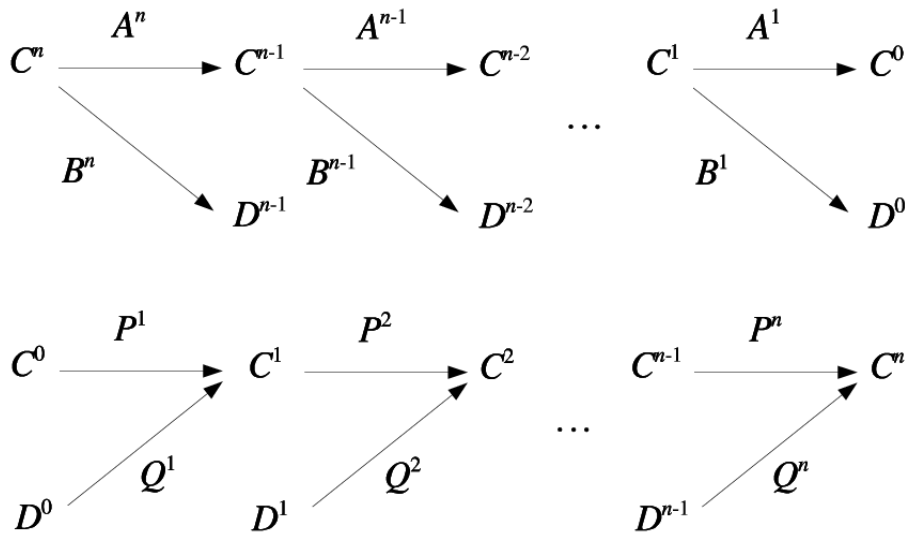


Figure 3.5: Filter bank view of decomposition and reconstruction. Decomposition (top) from an input curve C^n (top-left) into detail levels D^i and lower-resolution versions C^{n-i}, ending in a final base path, C^0. The A^n matrix downsamples and filters the points into the lower resolution curve, and the B^n matrix captures the detail lost in the process. Bottom: reconstruction of the filter bank from base path C^0 and detail levels D^i using the curve point subdivision matrix $P^{1..n}$ and the detail refinement matrix $Q^{1..n}$. A simple addition combines the result of the refined points and refined details into the exact, higher-resolution, original curve levels, resulting in the original input curve, C^n.

3.2 Filter Banks

In concrete terms, we use the filter bank [FS94] as a convenient multiresolution representation, storing the base points of our coarsest curve approximation and the details for each level of reconstruction.

Beginning from the original discretized curve C^n, we apply the analysis filter A to compute C^{n-1}, a coarsened approximation of it. The details are captured and

saved in the filter bank as detail level D^{n-1}. The procedure then iterates, using C^{n-1} as our high-resolution curve to be down-sampled with the final set of points forming the base level, C^0, that reconstruction will begin with. Figure 3.5 (top) shows the process. Reconstruction, or synthesis, of the high-resolution curve from the filter bank proceeds analogously. The base points C^0 are subdivided using the filter P and the details D^0 are refined by Q. These are then added together to reform C^1 the first reconstructed level. The steps repeat until all details levels have been applied, at which point our original curve has been recovered, as shown at the bottom of Figure 3.5.

3.3 Multiresolution Schemes

The matrices A, P, B and Q mentioned above form the core of the multiresolution approach, and the efficiency of the technique depends on their structure. Banded, compact matrices result in more efficient decomposition and reconstruction operations, allowing us to avoid ever working directly with matrix operations such as full matrix-matrix multiplications. In particular, linear systems generated by banded compact matrices are solvable in linear time, as opposed to cubic time operations for general matrices. This is of critical importance for the performance of multiresolution analysis.

The multiresolution approach that we have employed is based on reverse subdivision as described in Bartels and Samavati [BS00, SB99]. [BS00] provides structures that are *biorthogonal*, while [SB99] uses semi-orthogonal structures, similar to the wavelets used in the multiresolution by-example style capture by Finkelstein and

$$
\begin{array}{cc}
\vdots & \\
0 & \\
\frac{-1}{24264} & \\
\frac{31}{6066} & \vdots \\
\frac{-559}{8088} & 0 \\
\frac{988}{3033} & \frac{1}{8} \\
\frac{-9241}{12132} & \frac{-1}{2} \\
1 & \frac{3}{4} \\
\frac{-9241}{12132} & \frac{-1}{2} \\
\frac{988}{3033} & \frac{1}{8} \\
\frac{-559}{8088} & 0 \\
\frac{31}{6066} & \vdots \\
\frac{-1}{24264} & \\
0 & \\
\vdots &
\end{array}
$$

Figure 3.6: Comparison of filter coefficients for the cubic B-spline scheme. Left: Regular column of Q matrix from [FS94]. Right: the equivalent regular column from reverse subdivision [SB99] used in this work. Notice that the number of non-zero entries in the right column is five, versus 11 for wavelets. This difference in the number of entries means less than half the work is required to filter a set of input points for the repeated case. The structure of the coefficient entries, featuring powers-of-two and simpler rational numbers, favours our filters as well.

Salesin [FS94], but more compact and with a better structure.

The biorthogonality of [BS00] guarantees sparseness and very compactly banded matrices compared to conventional wavelets, and local support for A, B, P, and Q, whereas semi-orthogonality only provides for sparseness in P and Q and does not allow local support [SDS96]. The tradeoff is that these "local" matrices do not provide as high a quality as the "global" ones of [SB99]. The guaranteed sparseness lets us use a linear time solution, LU decomposition, for our systems.[1] And the local support ensures that changing a single point in C^k does not affect all the others, leading to more intuitive response when manipulating the curve points, as when using a multiresolution editor (Section 6.5).

The compactness in the reverse subdivision approach is evident in the cubic B-spline scheme: the repeated column of Q has only five non-zero entries, versus 11 with the conventional approach of Finkelstein and Salesin [FS94] and Stollnitz *et al.* [SDS96], shown in Figure 3.6. This leads to significantly reduced execution time for filtering the input points. We include detailed performance measurements in Section 7.1. In addition, since the entries of these matrices form filter values, their simplicity and type is important in minimizing round-off error and allowing a simple implementation. Consequently, simpler, exact rational numbers are preferred over complicated rational numbers, whose specification in a program introduces round-off error before filtering even begins. Additionally, powers-of-two allow the use of low-level optimization techniques for more efficient execution.

[1] Finkelstein and Salesin do provide a linear time implementation for analysis in Appendix B of [FS94]. It is based on a special case for their semi-orthogonal cubic B-spline wavelets for 2D curves. However, the reverse subdivision approach allows simple implementation using a banded linear solver for all four of our filtering schemes, in addition to supporting the 3D case.

For the same scheme, reverse subdivision produces better numbers than the conventional wavelet approach. A regular column of the Q matrix of the cubic B-spline scheme from both approaches is shown in Figure 3.6. The left column is the result of the wavelets based approach of Finkelstein and Salesin [FS94] and the right is the result of the reverse subdivision approach of Samavati and Bartels [SB99].

For this work, we have applied the filters generated from *locally*-optimal and *globally*-optimal solutions, using the coefficient matrices and methods described in Bartels and Samavati [BS00, SB99], respectively. We have implemented the filters for the global and local cubic B-spline and global and local Chaikin schemes. For comparison purposes, we have also implemented a simplified local cubic B-spline scheme and the wavelet filtering of Finkelstein and Salesin. Local filters provide the best possible approximation by minimizing the least-squares error around the width of the filter. The global method differs by implicitly constructing analysis filters that are optimal over the entire set of curve points, found by minimizing the least-squares error around the width of the filter. Despite this, we often find acceptable performance from the local filters. A recent result of Bartels *et al.* [BGS06] shows that the local least squares often provides a good estimation of the least squares solution, particularly for larger sized problems.

In the global method, Equation (3.1) (Section 3.1) is now computed as

$$P^{n^T} P^n C^{n-1} = P^{n^T} C^n, \tag{3.4}$$

which is a banded, linear system. Note that this is equivalent to

$$C^{n-1} = (P^{n^T} P^n)^{-1} P^{n^T} C^n, \tag{3.5}$$

where $(P^{n^T} P^n)^{-1} P^{n^T}$ is the implicitly constructed A, solved as a linear system. The details are extracted by first rearranging Equation (3.3):

$$Q^n D^{n-1} = C^n - P^n C^{n-1}. \tag{3.6}$$

Then following the derivation using normal equations in Samavati and Bartels [SB99], we reach

$$Q^{n^T} Q^n D^{n-1} = Q^{n^T} (C^n - P^n C^{n-1}). \tag{3.7}$$

Computing D^{n-1} then again involves solving a banded, linear system. Note that $Q^{n^T} Q^n$ and $P^{n^T} P^n$ can be pre-computed as they are constant for a given scheme. Reconstruction simply uses the scheme's subdivision filter P and its extension Q in Equation (3.3). The implementation of these filters is described in Section 5.2.

Chapter 4

Stroke Input

As discussed in Chapter 3, our multiresolution filters for by-example style capturing operate on discrete point data representing piecewise linear curves. These curves are brought into our capturing system from both scanned images of pre-existing illustrations, and interactively sketched examples. For the scans, we have the task of first extracting curves from the raster images. We refer to this general process as *vectorization*, and we provide two complementary techniques for it. The first, skeletonization, in Section 4.1, automatically finds a set of ordered points that lie within the drawn curves of the image of the stroke. The stroke may also consist of multiple disconnected segments, which are detected and preserved. Our second technique, contour tracing, in Section 4.2, results in an outline-following extraction of the stroke, and lets the user select a particular sub-area to use for subsequent style capture. These two techniques are compared on various examples in Section 4.3.

Our capturing system also provides an interface for directly sketched input curves. For these inputs, we skip the vectorization step by using the mouse or tablet point data directly. This interactive input allows us to use complex, self-intersecting styles that are not easily extracted from raster images. Examples of these types of styles are shown in Figure 4.1. We note here that the subsequent stroke style capture (Chapter 5) does not differ for our two input forms, as the vectorization step produces an ordered point list which is identical to that of sketched input.

The overall pipeline is shown in Figure 4.2. Both vectorization paths return a set

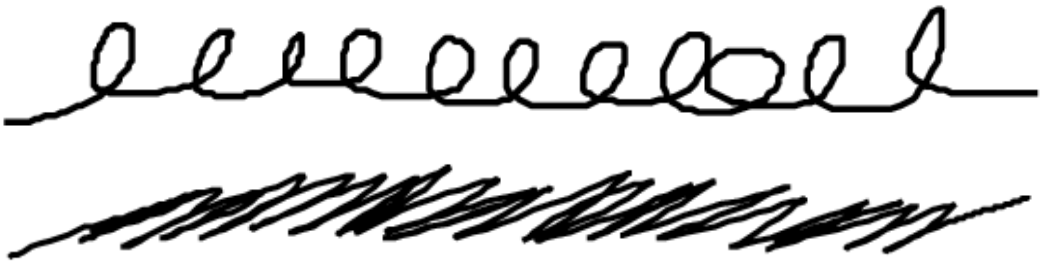


Figure 4.1: Examples of hand-sketched styles that can be directly input into our system for capturing and re-use as example line and curve styles. These sketched styles feature self-intersecting and overlapping curves that can not be easily automatically determined from raster images by a vectorization system.

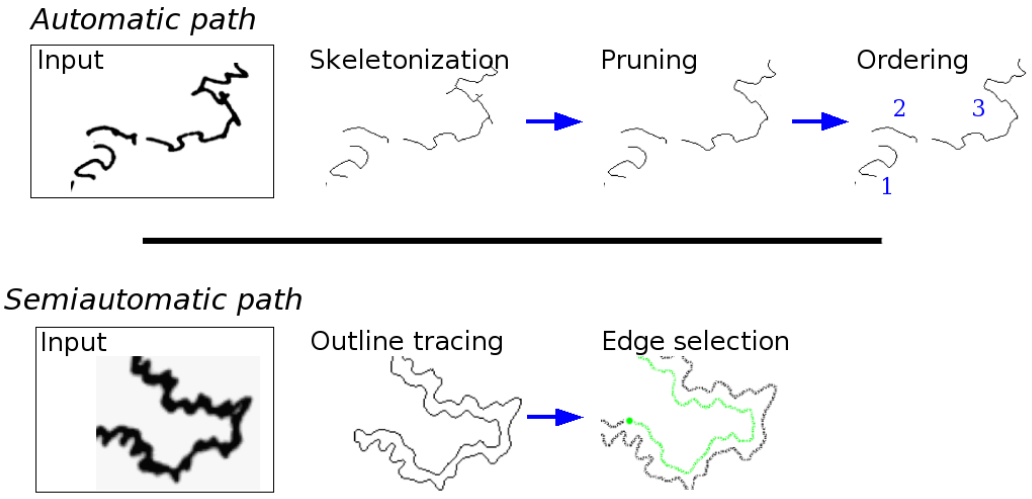


Figure 4.2: Overview of vectorization of an input raster image with either of the two paths provided in our system. The fully automatic method also handles identifying and ordering multi-section curves, while the semi-automatic path requires user interaction after tracing to specify which part of the closed curve to use.

of control points representing a polyline along the curve or curves of the drawing, given raster images representing one or more strokes of a line style that we wish to capture and re-use in our system. Since the primary source of input for this procedure are scanned drawings, for the greatest generality we assume that input images may be in colour or stored in artifact-introducing lossy compression formats such as JPEG. This necessitates a cleanup step before proceeding: we create a base image via a three-step process that results in a simple two-tone figure used as input to the vectorization proper.

The base is generated by first reducing the image to grayscale using a standard intensity mapping from colour to luminance [HB97]. Secondly, we filter the input image with a median-cut filter to remove high-frequency noise [GW87]. This filter replaces each input pixel with the median value of its neighbours. The result is that individual speckles, small isolated marks, or noise on the image are greatly reduced in value. We note that this simple filtering is sufficient at this stage, as the multiresolution decomposition used in style capturing also provides us with the noise removal and smoothing option of dropping the highest frequency detail levels. Finally, we threshold the image, discarding weaker valued and lighter boundary pixels, and reducing it to two tones for either skeletonization or outline tracing. These stages applied to a portion of a typical input image are illustrated in Figure 4.3.

4.1 Automatic Extraction with Skeletonization

Our automatic extraction algorithm consists of the image-processing operations of skeletonization and pruning, followed by ordering of the generated segments. We note

(a) (b)

Figure 4.3: Detail from the high-resolution (300+dpi) scan of an artistic tree silhouette drawing. The input image (a) shows typical artifacts introduced including colour, multiple gray levels and ill-defined borders, and compression artifacts from a lossy technique such as JPEG. The image after reduction to two tones by grayscaling, median-cut filtering, and automatic thresholding is shown in (b).

here that this vectorization path differs from the other in transparently supporting multi-segment curves, as well as single curve images.

4.1.1 Skeletonization

The *skeleton* of an area in a raster image is a representation of its connectivity and extent that consists solely of single-pixel lines through the "center" of the area [GW87]. It is produced by successively "thinning," or peeling, away pixels from around the area's boundary. This repeated process eventually reveals a simple frame for the surrounding area. By its nature, it generally does a good job of representing the path of a smooth, varying line. And since our emphasis is on capturing traditional pen-and-ink and pencil styles, this fits well for the curves of much of our input. We have adapted the freely available skeletonization code published by David Eberly of Magic Software [Ebe05]. Figure 4.4 shows a typical skeleton of an input image produced by thinning. One consequence of the technique is the production of *branches* wherever variation along the boundary occur. This can be from artifacts of the scanning process, minor variations due to pen or brush strokes, or simply due to curves with more than two distinct ending points – each end will produce a skeletal branch. Our next processing step removes all such branches to create simple curves for input to our style extraction system.

4.1.2 Pruning

As demonstrated by Figure 4.4, the result of skeletonizing a stroke image can result in unwanted branching artifacts that distract from the overall path of the stroke. It is necessary to remove these to produce a simple, non-branching stroke that can be

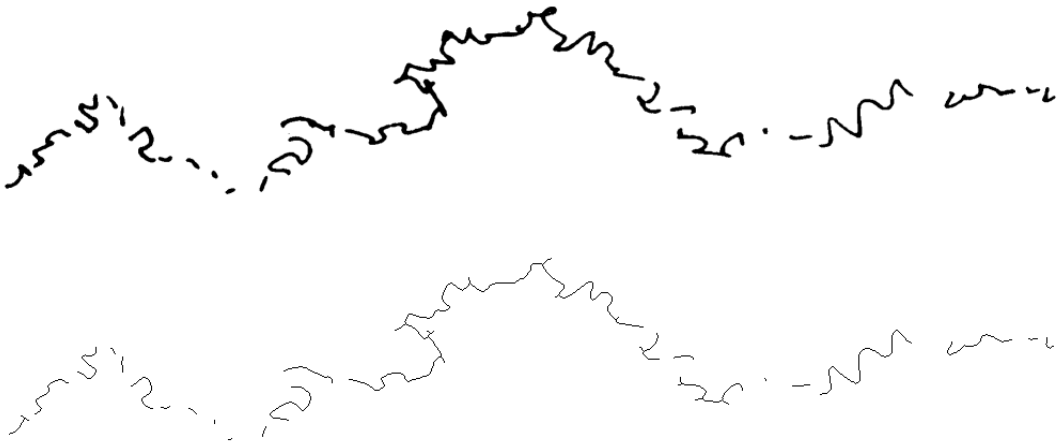


Figure 4.4: Skeletonizing an scanned input image (top) of abstracted, leafy, tree or bush foliage via repeated thinning. Notice how even a single stroke of a pen or pencil results in a multi-pixel width line when scanned at high resolution. Overlapping, looping, crossed, or simply thicker sections of the drawing all result in branches in the single-pixel width skeleton (bottom).

unambiguously vectorized. The procedure for performing this is *pruning*. Our pruning algorithm identifies *articulation*, or branching, points and selects which branch to remove to preserve the overall path of the stroke. The 4-connected output of the skeletonization process guarantees that these points have no more than 3 neighbours. Two neighbours indicates the midpoint of line, while three implies a separate branch, which we flag. For each of these points we remove the shortest open branch, where open branches are those that do not end in another articulation point. After each articulation point has been examined, we use the resulting image as input for another iteration of searching and pruning. The algorithm terminates when no further articulation points are identified in the image. Figure 4.5 shows several iterations of the pruning algorithm.

Our algorithm also removes branch loops, such as arise from circularly closed areas of input sketches. To identify these cases, we check for branches which share a common end point, and remove the shorter branch, as shown in Figure 4.6.

4.1.3 Ordering

After the process of skeletonization and pruning, the input image may consist of multiple simple strokes, representing a curve with a discontinuous sketched style. In this case, to complete the vectorization, we must extract these strokes and place both them and their constituent points into a consistent spatial order. In other words, after this stage, stepping from the extracted start point to the end point should follow the logical path of the pen that created the image. Our approach starts with the assumption that the sketch follows a consistent and discernible orientation across the page from a global minimum point. We take this to be the far lower left of the

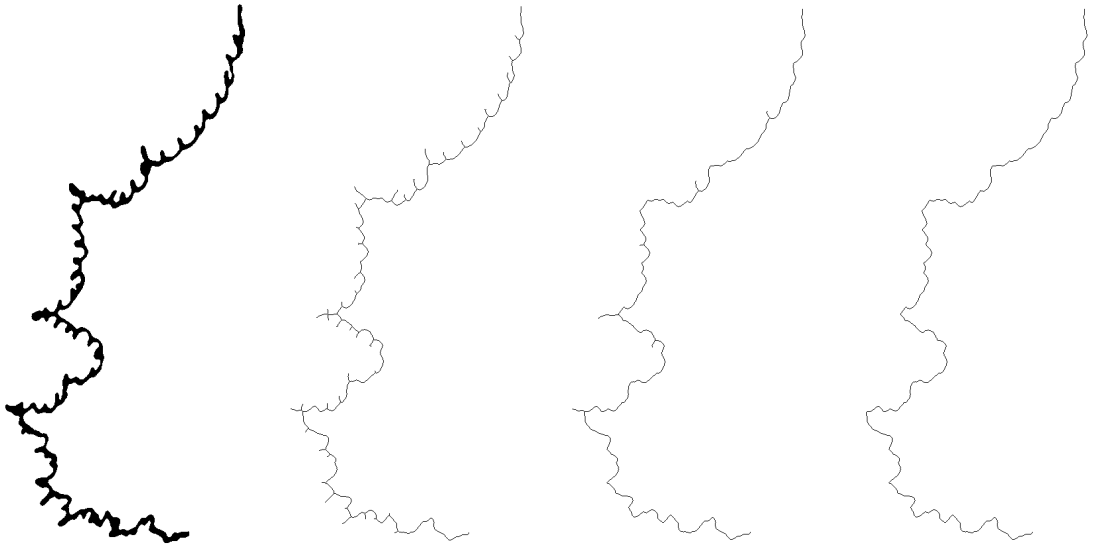


Figure 4.5: Pruning the skeleton of an input image via repeated branch removal. The input image (a) must be reduced to a single, simple curve for use in the style capture process. (b) Skeletonization reveals a complex outline with first and second-level branches. (c) After a single pruning iteration the first level branches have been removed. (d) After a second iteration the remaining branches are removed, leaving the simple curve outline with good representation of the original.

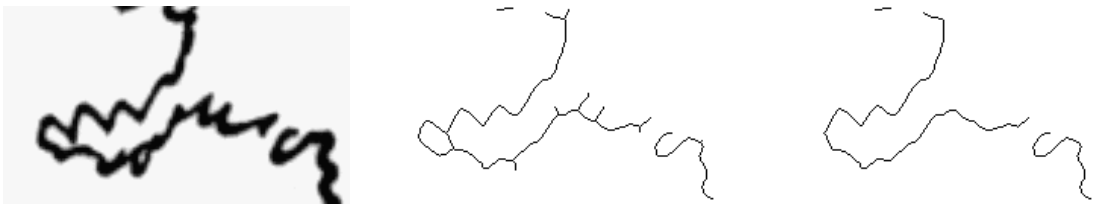


Figure 4.6: Removal of branch loop artifact. (a) Detail of input tree silhouette sketch. (b) Skeleton showing branch loop. (c) Pruned result without loop.

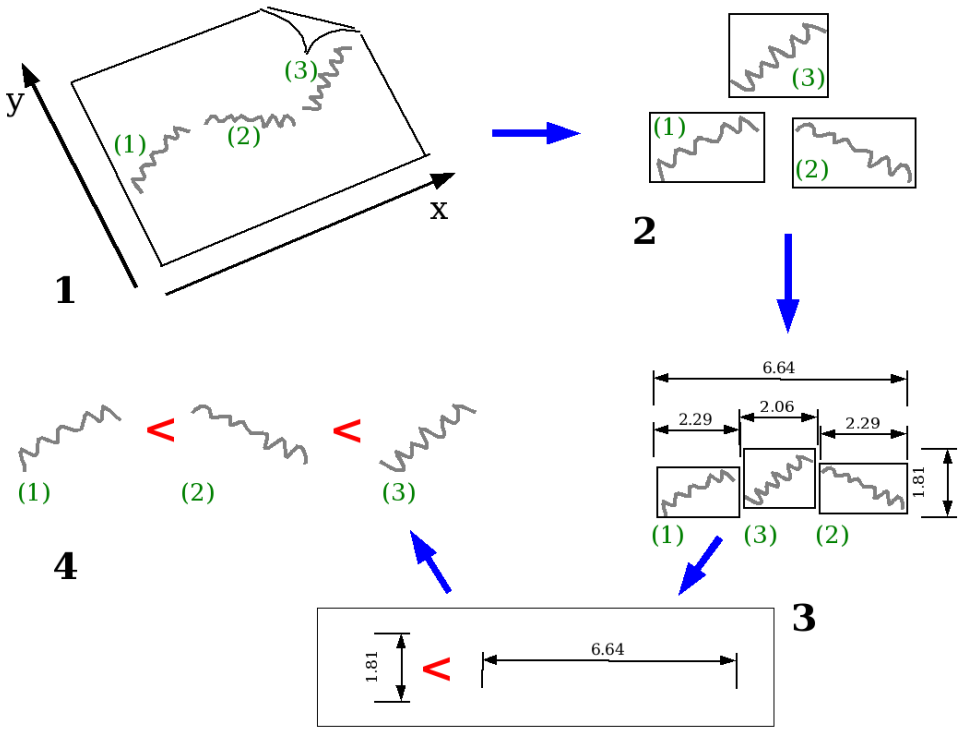


Figure 4.7: Determining an ordering for stroke segments within a sketched style. Clockwise from upper-left: (1) a scanned curve, consisting of three segments, is selected as input. (2) Because the separate segments of the curve are independently extracted, they must be re-ordered to match their layout in the original curve image. (3) The largest dimension of the curve's bounding box determines the overall orientation of the curve. For this illustration, it is the horizontal, or x, axis. (4) The individual segments are sorted to determine their ordering. The sort occurs on the coordinate of the bounding box's midpoint that corresponds to the image's major dimension. The stroke segments from the figure are therefore sorted on their x coordinate, restoring the original ordering.

input image. We then find the bounding box of the stroke image using the union of the bounding boxes of all extracted segments. Here we make an assumption about the illustration: whichever bounding-box dimension predominates (either horizontal or vertical) we assume indicates the overall orientation of the image and its stroke segments. We then order the individual segments by sorting them on the coordinate of the midpoint of their bounding boxes that corresponds to the predominant dimension of the image. While we find this simple compass-style North-South or East-West ordering for the segments works well, illustrations that do not follow a roughly straight path across the page will not extract correctly. For those cases, we can usually easily separate the image into pieces and extract them individually. Finally, to ensure the direction of points within each segment are consistent, we check that the distance from the end point of each segment to the start point of the following segment is a minimum. If not, we reverse the order of the points within the segment to achieve this. The process is illustrated in Figure 4.7, and Figures 4.4 and 7.6, (Chapter 7), show this method on typical examples. While we find this method to be suitable for all our example input, certain discontinuous stroke paths will fail. For example, those that are circular or self-intersecting will not be ordered correctly using our horizontal or vertical restriction. Those cases may be resolved by manually cropping the figure to smaller all-vertical or all-horizontal sections before vectorization.

4.2 Semi-Automatic Extraction with Outline Tracing

Our semi-automatic technique has two steps to find suitable curves for use in style capturing. (1) The input image is traced, capturing the silhouettes of the input image. (2) The user may then specify which section of the trace to extract as the desired or representative part of the image.

4.2.1 Outline Tracing

As a first step, our system traces the outline of the input image, using the standard *contour trace* algorithm [Pav82]. In general, this algorithm can be used to extract arbitrary contours from an image based on the pixel value. But to simply identify outlines or silhouettes, we restrict the input to two thresholded values, as in the skeletonization process.

The contour trace is based upon a local search of the eight surrounding pixels (the 8-neighbourhood) of an outline pixel. Once an initial boundary point is found, the search iterates from that pixel, scanning in a clockwise direction about its 8-neighbourhood. When a subsequent boundary pixel is encountered, it is marked as part of the outline. The tracing then continues as before by exiting the second pixel and searching clockwise about it. This algorithm will traverse the image outline until it reaches its starting point. It terminates there with a closed curve when the initial pixel is entered for a second time from the original direction of entry. Figure 4.8 shows the beginning, iterative step, and end of the process. An example input image and the resultant trace is shown in Figure 4.9.

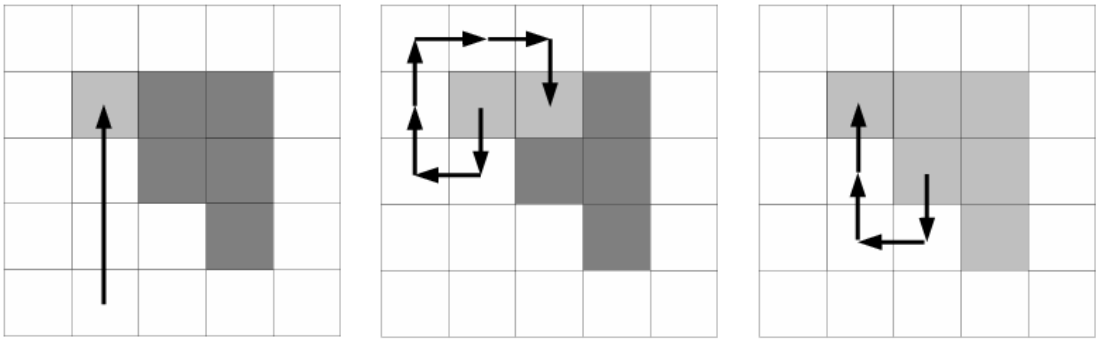


Figure 4.8: Visualization of the start, repeated step, and halting condition of the 8-neighbour trace algorithm. (a) After determining an initial pixel on the outline of the image we wish to trace, we note its coordinates and our direction of entry into it. (b) The repeated step of the trace exits the pixel in the direction of entry and searches about it in clockwise order until another outline pixel is found. (c) When we re-enter the initial pixel from the same direction as it was first found, we fulfill the stopping condition and have completed the tracing loop.

4.2.2 Edge Selection

Finally, we use an interactive tool to select an open segment of the closed output curve generated by the tracing algorithm. This allows the user the freedom to determine the length and starting and end points of the desired image section, as shown in Figure 4.9. The selected section is then extracted from the traced curve for use as input to the style capturing system.

4.3 Comparison of Skeletonization and Tracing

While fully automatic methods may be preferred for convenience, we have found that the two techniques are complementary. In general, skeletonization performs best on images where most detail is encapsulated in the broad variations of the curve's path.

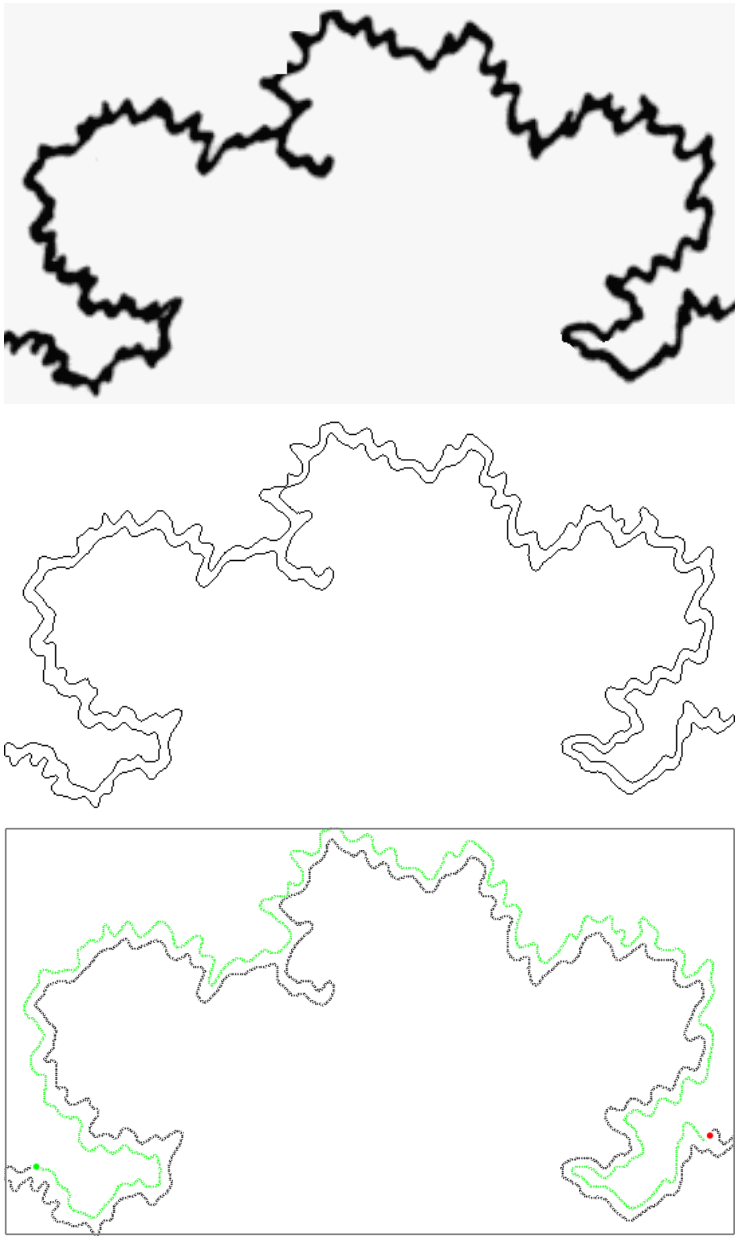


Figure 4.9: Example curve from a full contour trace of an input image. Top: Raster image input of detail from a pen-and-ink sketched tree silhouette. Centre: Closed contour curve found by tracing about the sketch's outline. Bottom: Selecting an open section of a traced image for use as a captured style.

This is because variations away from the path tend to emerge as branches, which are necessarily pruned in order to generate a single, simple, curve for input to the multiresolution style capture process.

In contrast, outline tracing preserves all variation of the input while generating the single, simple, curve in marching around the exact path of the figure. This allows all details of the image to be kept, with the restriction that a specific section of the closed result curve be selected as input for style capturing. Figure 4.10 shows an example that favours outline tracing, but where skeletonization also produces a reasonable result. Figure 4.11 shows an example where skeletonization produces an equivalent result to outline tracing and selection, but done automatically.

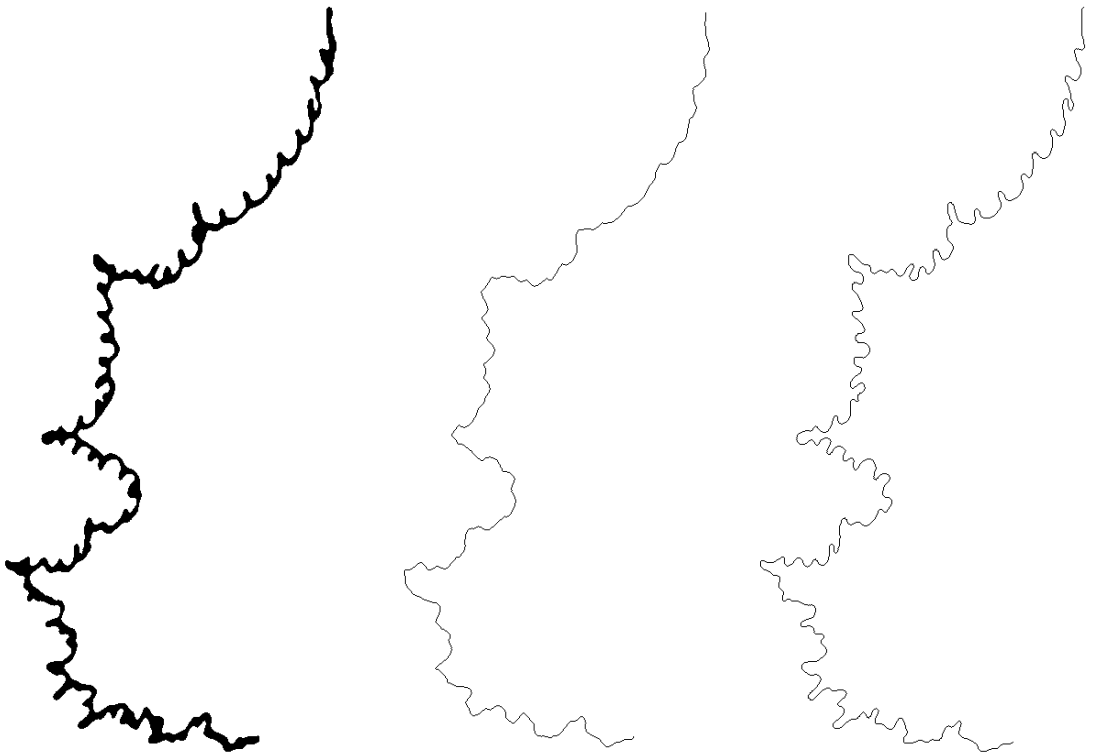


Figure 4.10: Comparison of input vectorization techniques. This input image favours outline tracing over skeletonization in keeping all the silhouette details intact. (a) Detailed tree silhouette sketch. (b) Curve extracted via automatic skeletonization and pruning. (c) Manually selected outer edge of curve generated by automatic tracing of the silhouette.

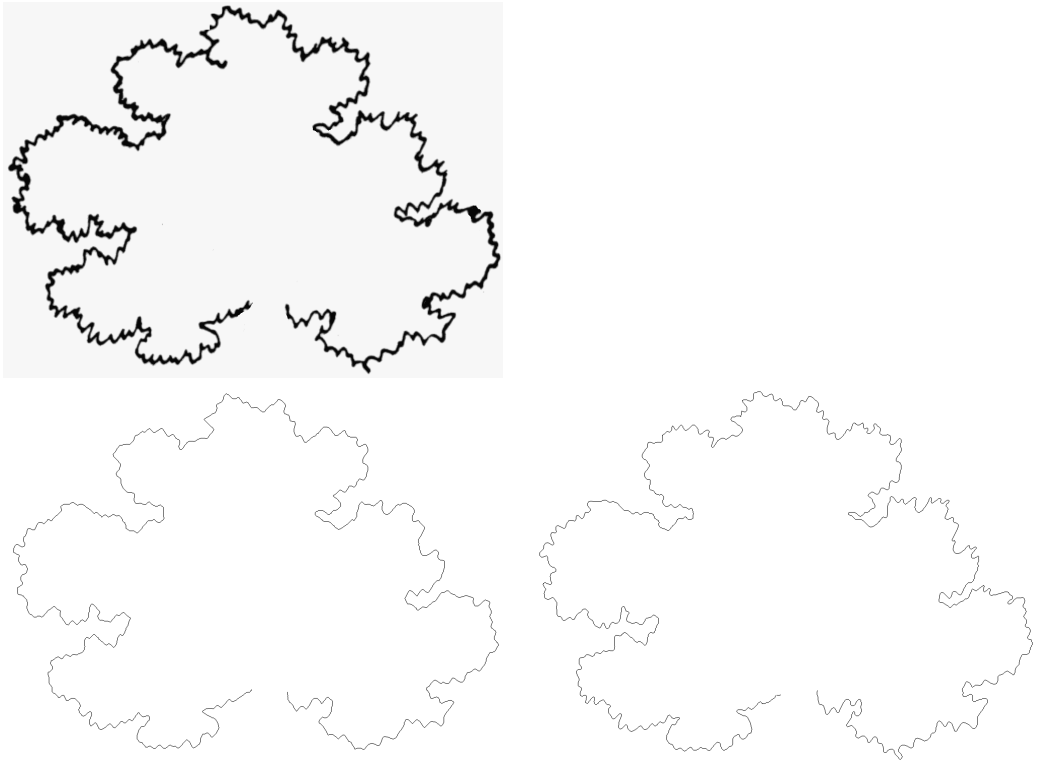


Figure 4.11: Another example comparing input vectorization techniques. This input image has almost all curve details contained in the basic path. Skeletonization thus gives essentially equivalent output to the outer edge traced and selected using contour tracing. (Top) Detailed tree foliage silhouette sketch. (Bottom Left) Curve extracted via automatic skeletonization and pruning. (Bottom Right) Manually selected outer edge of curve generated by automatic tracing of the silhouette.

Chapter 5

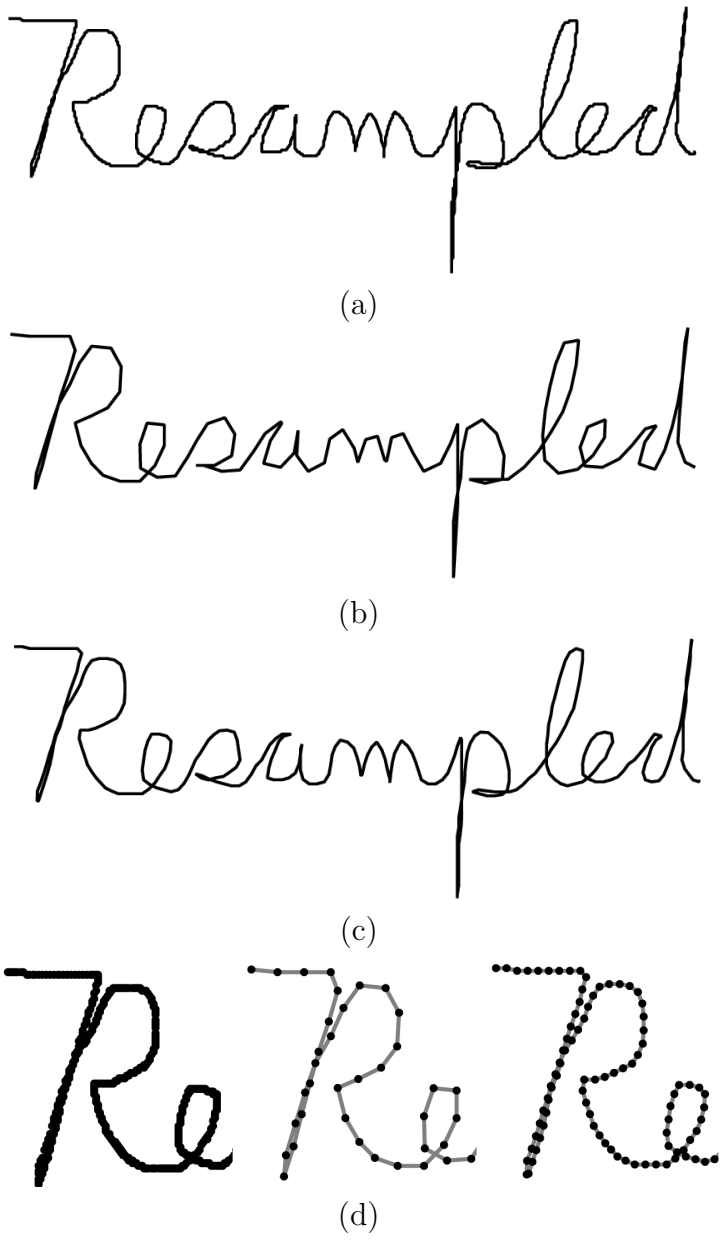
Capturing Artistic Styles

With the input acquired, we can proceed to capture the curve style by separating it from its path. We use our multiresolution analysis approach, described in Chapter 3. This involves two steps: First, we resample the input curve from its original resolution to fit the constraints of the filtering matrices (Section 5.1). Next, we create the multiresolution filter-bank structure that encapsulates the information needed to reconstruct the captured style on a new path (Section 5.2). The filter-bank's principal component is the set of details extracted from the successively filtered and downsampled input curve. To this, we add a corresponding set of multiresolution flags that we use to specify discontinuous curves (Section 5.3). These types of curves have blank or empty sections where the pen has been lifted off the page, and are commonly used to form hatching strokes or to achieve the artistic effect of *indication*. (Another extension stores the angles of the detail vectors with respect to the base path, allowing style reconstruction normal to the new base path. We discuss this in the context of style re-use in Section 6.2).

5.1 Curve Resampling

Our multiresolution decomposition filters place a constraint on the number points in the input curve. In general, we require the number, n, to be of the form $2^k + N$, for $k > 2$, where N is a small constant associated with the multiresolution scheme

used. For the local and global Chaikin filters $N = 2$ and for the local and global cubic B-spline filters $N = 3$. This power of two restriction is due to the iterative divide-and-conquer nature of the decomposition algorithm, similar to that imposed by the fast Fourier transform. To conform to this restriction, we first resample the input points from the vectorized or interactively captured curves to the closest value, higher or lower, that satisfies the constraint. We use a uniform resampling algorithm that generates the appropriate number of new points along the path of the input curve. The algorithm works by computing a stepping distance along the curve from its arc-length, and then linearly interpolating a new point position at each step. This places the new points at uniformly spaced intervals, and also smooths variation in the density of points along the curve, which may naturally arise from using a sampled device such as a mouse or pen and tablet. Figure 5.1 shows typical use in decimating a hand-sketched input curve to a fraction of its points while smoothing and preserving its shape. Our choice in using a uniform-resampling comes from the observation that our input is generally from scans whose resolution is much higher than that of the CRT or LCD displays we use to visualize our results. The implication is that no visible curve-specific details are lost, as the resampled curve still has less than a pixel's width spacing between points. And for sketched input, the low-resolution sampling of typical pen and mouse interfaces usually still provides us with a point per-pixel of the continuous parts of the drawn curve.



(a)

(b)

(c)

(d)

Figure 5.1: A hand-drawn curve, before and after resampling. (a) The user-drawn input with 1383 points. (b) The resampled curve at 10% of original (138 points). (c) And at 25% (345 points). (d) Detail of input, 10%, and 25% resampled curves showing smoothing as a result of uniform resampling.

Scheme	Minimum	Reference for Filter Values
Chaikin local	10	Bartels and Samavati [BS00]
Chaikin global	10	Samavati and Bartels [SB99]
Cubic B-spline local	11	Bartels and Samavati [BS00]
Cubic B-spline global	11	Samavati and Bartels [SB99]

Table 5.1: Multiresolution filtering schemes implemented in our stroke style capturing and re-using system. The filter's minimum defines the smallest number of points in the lower-resolution curve after all iterations of multiresolution decomposition on the input curve. We have found smaller values generate smoother base paths, leading to more accurate style reproduction.

5.2 Filter Bank Creation

Creation of the filter bank now proceeds using the curve points from the resampled input. Though we have used matrix notation to describe the process, we take advantage of the banded nature and the regularity of all the matrices to create and reconstruct the filter-bank in time linear to the number of input points. Table 5.1 summarizes the filtering schemes we have implemented in this work.

For the local schemes, the matrix entries are used directly to filter the input points. For the global schemes, the creation of the filter banks follows the basic steps described in Section 3.3. We take advantage of the sparse, banded nature of the P and Q matrices from Samavati and Bartels [SB99] to give us fast decomposition steps in our implementations of the Chaikin and cubic B-spline global schemes. For these schemes, Equations (3.4) and (3.7) are solved using a tri- and penta-diagonal banded linear system solver, respectively. Using the Chaikin scheme as an example, the right-hand-side of each equation is determined using simple linear-time filter application. Solution of the linear system requires only one pass of Gaussian elimination to reduce

the banded system into a bi-diagonal band, and is then trivially solvable using back-substitution. We have used the banded LU solver from Numerical Recipes in C, Press *et al.* [PTVF92] in our implementation.

The generated detail levels and final base path are then stored in a simple data structure, referred to as the filter bank. Reconstruction is performed with simple filtering applied directly to the detail levels and base points to regenerate the stored levels. The reconstruction steps, though also linear in time, are therefore significantly faster than decomposition.

Figure 5.2 compares the decomposition using local and global schemes. The input tree silhouette consisted of 4031 points vectorized from a scanned image. The Chaikin schemes both resulted in 11 levels of decomposition, while the global and local cubic B-spline schemes had 10 and nine.

As discussed in Section 3.3, local filters provide the best possible approximation over a small section of the curve equal to the width of the filters, while the global method is optimal (in minimizing least-squares error for the downsampled curve) over the entire set of points. The tradeoff for the increased accuracy of the global filters is speed of execution: while the global multiresolution method generally produces a smoother base path, it has the drawback of requiring the solution of banded linear systems, making it slower than the simple filtering of the local schemes. Some sample timings of the B-spline scheme's execution appear in Table 7.1.

Our goal in applying these filters has been to achieve the best possible style capture and re-use on a wide variety of input. As such, our interactive system relies on visual checking by the user to determine the quality of the extraction and subsequent re-application of the style. However, we have noted that the scheme

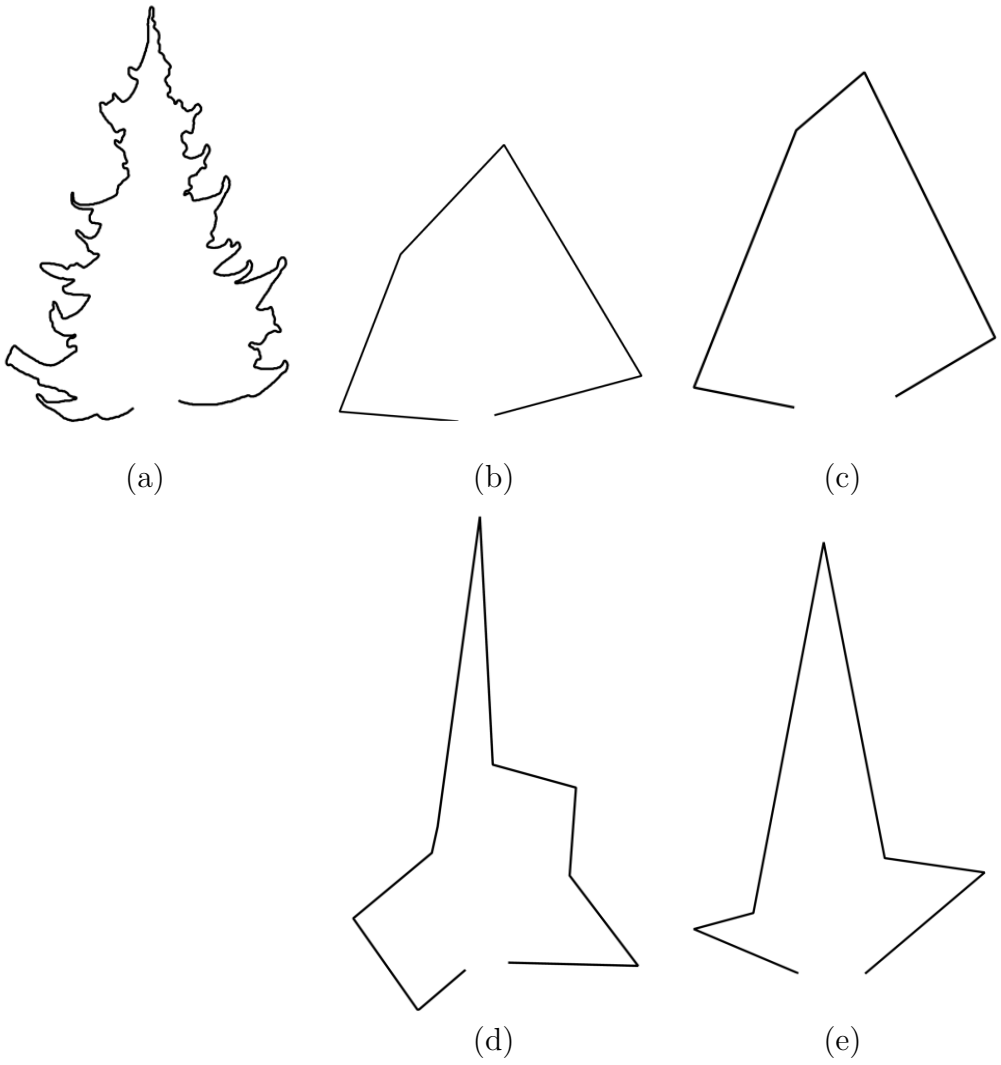
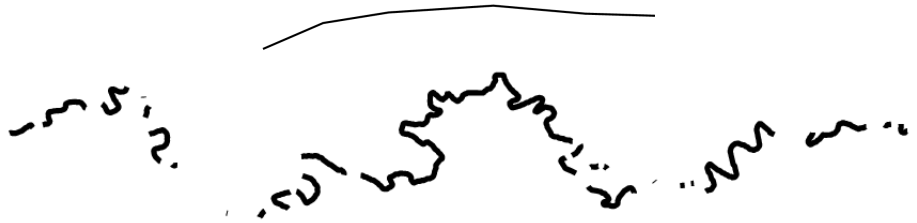


Figure 5.2: Base paths extracted from an input silhouette curve (a) using various schemes. The base paths generated by (b) local and (c) global Chaikin, and (d) local and (e) global cubic B-spline schemes are shown. For this example, the Chaikin results seem to generate smoother base paths. They also have a smaller number of final path points, at six each, than the local and global cubic B-spline paths, at 10 and seven, respectively. In general, the choice of scheme is specific to the input used, and can only be made after comparing the smoothness values or look of the extracted base paths. Our system provides an immediate, interactive visualization of the base paths for this process.



(a) Local Chaikin; curvature 21858

(b) Global Chaikin; curvature 92198

(c) Local cubic B-spline; curvature 176464

(d) Global cubic B-spline; curvature 378611

Figure 5.3: The link between quality of decomposition and the reconstruction is shown in this example. The top curve is the base path and below is the application of the captured style (from Figure 4.4) on a new base path. We have found that lower curvature values generally indicate a smoother base path, which implies a more accurate reconstruction.

with the smoothest base path generally gives the best possible reconstruction after application; we discuss the reason for this in Section 6.1. As an automatic measure of smoothness, we compute the curvature of the base path generated from the fully-decomposed input curve. Our method simply sums the angles between each consecutive line segment of the piecewise-linear base path, which is then given for each scheme. Our experiments have shown that the lowest usually indicates the smoothest path. This provides a heuristic for automatically selecting the likely best result. The user may also visually select the scheme to use from a display of all the base paths in our interactive system. The link between smoothness and the quality of reconstruction is shown in the example in Figure 5.3.

5.3 Discontinuous Styles

As supported by our vectorizer, we have also extended the style capturing technique to include curves whose style contains multiple strokes. This added flexibility allows us to re-create illustrations with hatching strokes, or silhouettes which feature the common artistic technique of "indication"—leaving broken sections of the profile to show patterns of light, connectedness, relative distance, or relative location with respect to the viewer. The examples in Figure 5.4 show variations of this effect. These discontinuous styles can be taken from either the multi-stroke vectorizer or the interactive style sketching interface.

For either input source, the basis of our method is to flag the visual start- and end-points of each stroke segment within the drawn curve. We use the flags to indicate the extent of the segments during rendering. This is complicated by the requirement

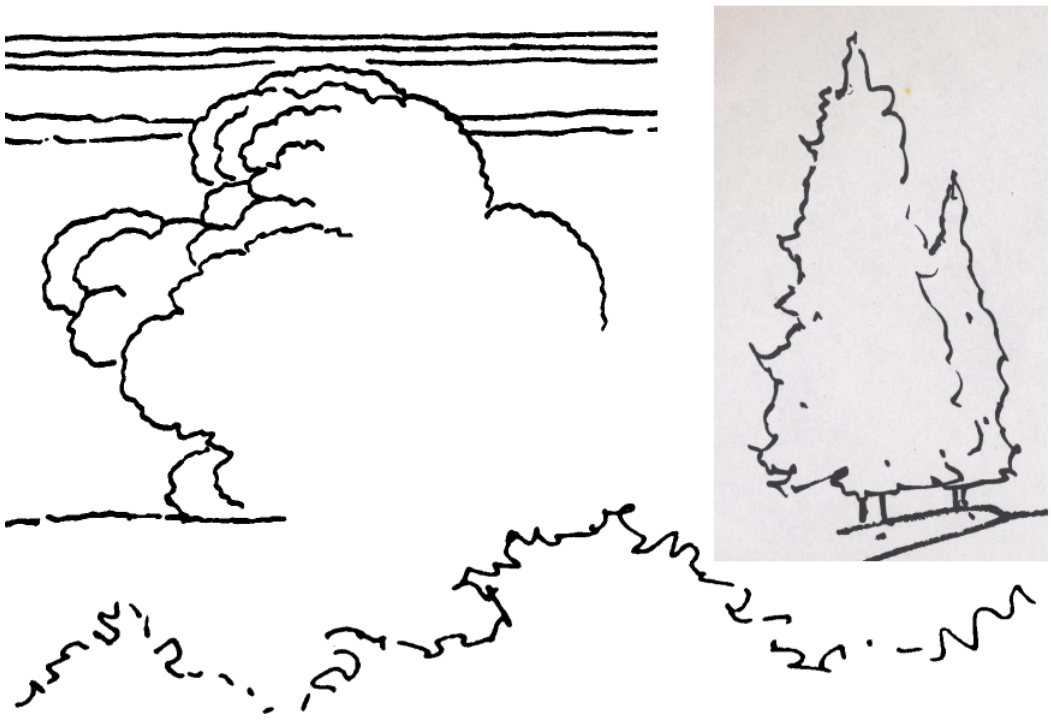


Figure 5.4: Top: using indication to help show the relative positions of multiple, obscured clouds (left) and trees (right). Bottom: foliage drawn using indication to imply a continuous silhouette made up of separate trees or bush tops.

that we also keep the flags in sync with the number of points in the reconstruction. The two reasons for this are the necessity of resampling the input curve to a possibly lower or higher number of points, and the user's option of choosing a lower level of reconstruction during style application (Section 6.4). Both of these operations will change the number of points in the final curve rendering from those in the original captured style. This means that any direct mapping between the flags and the curve points will be lost after reconstruction (unless a full reconstruction to the original curve resolution is done).

Our solution to maintain the correspondence between the flags and curve points for rendering is to process them through multiresolution decomposition and reconstruction. This generates and stores a set of "flag-detail" vectors alongside the curve details. At any level, the number of reconstructed flags will be equivalent to the elements in the reconstructed curve, preserving the mapping. The flag-details are constructed as points whose x-coordinate is the point's number in the sequence and whose y-coordinate is the flag value, either 1.0 or 0.0. For example $[0.0, 0.0], [1.0, 0.0], [2.0, 0.0], [3.0, 1.0], [4.0, 1.0], [5.0, 0.0]$ represents a flagged gap between the fourth and fifth points of the curve. These flag points represent a step function, which can be decomposed and reconstructed like our input curve. However, after reconstructing to a lower level, the flags will only be an approximation to the original flags, just as a partially reconstructed curve will only approximate the original. The means that the flags will have values between 0.0 and 1.0, unlike as we originally specified them. In our experiments and observation, we have found that using a threshold of 0.5 and above to indicate a 1.0 flag gives good results for higher detail levels. This value has been used for all the discontinuous style examples in

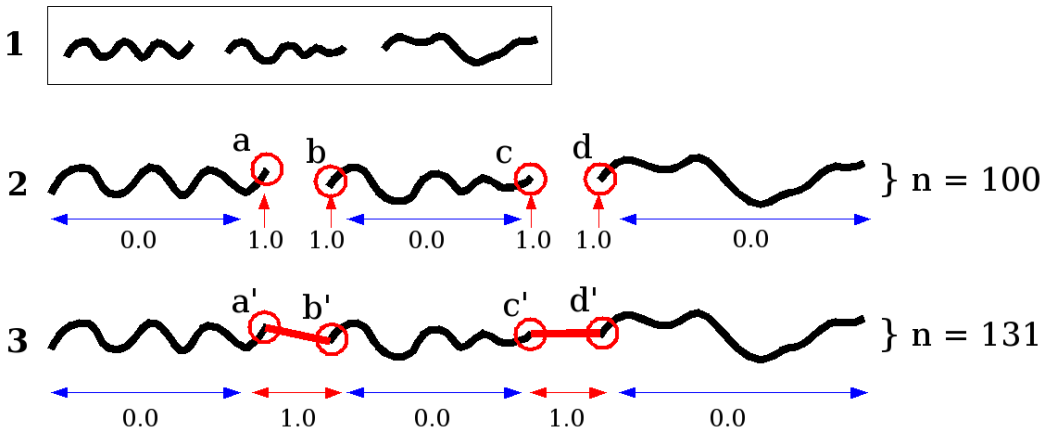


Figure 5.5: When capturing and re-creating curves consisting of multiple, independent stroke segments (1), we use a parallel set of flags for each detail vector. (2) The flags are set to 0.0 for each curve point that lies on a stroke segment or to 1.0 at the start and end points of gaps between segments, (a, b) and (c, d), respectively. After resampling the input curve and flags (to 131 points from 100), the gap start and end points positions have changed and so need to be adjusted to maintain their utility. (3) Resampling fills in the stroke segment gaps by interpolating new points between the gap start and end points (a, b) and (c, d). The resampled points (a', b') and (c', d') are found to be the resampled curve points closest to the original start and end pairs using simple Euclidean distance. The point flags within their range are set to 1.0 to indicate that those points have been resampled into the gaps and should not be drawn.

this work.

Our flag details construction proceeds as follows: First, the style sketch is assigned a flag array corresponding to the set of input points. The flag values are initialized by either the multi-stroke ordering algorithm in the vectorizer or the interactive sketcher. The start and end of gaps in the curve (where the pen has left the paper) are flagged with a value of 1.0, while all other point flags for the curve are set to 0.0. Algorithm 1 is then used to create the filter-bank for this discontinuous style.

Algorithm 1 Creating a discontinuous filter-bank.

FILTER-BANK CREATION:
Input: $Segments[m]$, $Flags[m]$
Output: $FilterBank$

1: Resample $Segments$ and $Flags$
2: Call ADJUST FLAGS
3: **while** $Segments$ can be downsampled **do**
4: Decompose $Segments$ and $Flags$
5: Save details from $Segments$ and $Flags$ into $FilterBank$
6: $Segments \leftarrow$ decomposed $Segments$
7: $Flags \leftarrow$ decomposed $Flags$
8: **end while**
9: Save final $Segments$ as base path into $FilterBank$

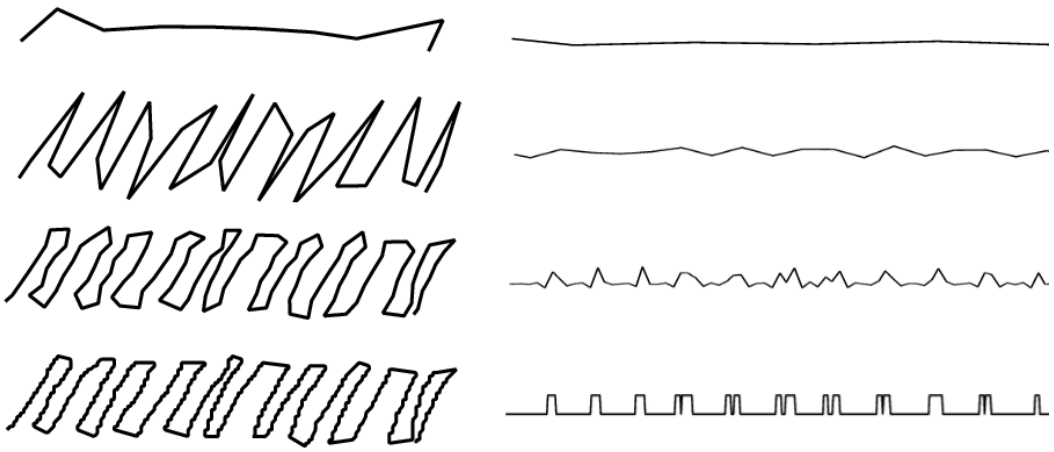
ADJUST FLAGS:
Input: $Flags[m]$, $ResampledFlags[m]$
Output: Updated $ResampledFlags[m]$

1: **for** Each pair $begin, end$ of flagged gap points in $Flags$ **do**
2: Find the closest (Euclidean distance) points $rbegin, rend$ in $ResampledFlags$
3: Set the flags values of the points in the range $[rbegin..rend]$ to 1.0
4: **end for**

(We note here that in our implementation continuous styles are merely a special case: the flags are maintained, but are all set to zero, thus leaving no gaps between curve segments). The inputs are $Segments[m]$ and $Flags[m]$, the curve points and corresponding initialized flags. (Line 1): As usual, we resample the arbitrary sized inputs to a size our filters expect. (Line 2): This resampling disrupts the flags array, so we use ADJUST FLAGS to update it to match the new points in the curve. It first finds the begin and end points of each gap in the original curve and matches those points to the resampled curve. This is done by finding the nearest (Euclidean) point in the resampled curve to each begin and end point from the original. Then the flags

of the resampled curve in that range are updated. Since those flags have been placed into the gaps, they are set to 1.0, as illustrated in Figure 5.5. During rendering of the final curve any points between the first and last one flagged in a sequence will be skipped, as they fall into the gap area. (Lines 4-7): Once the flags are updated, the creation continues by making a parallel set of flag details along with the standard curve details. This lets us reconstruct the curve with the original discontinuous areas flagged for non-rendering, which is sufficient to preserve the look of the style at any level.

Only one additional step is required during reconstruction: since the flags are not necessarily fully regenerated, they may be only partially reconstructed. We deduce the flag states with a thresholding value that selects the flags that should be enabled. We have found that a threshold value of 0.5 reliably selects the correct flags, down to all but the lowest levels of detail. At those levels, the reconstructed style has so few points that it is generally unrecognizable. Figure 5.6 shows a visualization of the flag values at several levels and the resulting curves when they are used for rendering a captured and re-applied hand-sketched hatching style. Figure 5.7 shows an example reconstruction of all the levels of another captured, discontinuous, hatching style.



(a)

(b)

Figure 5.6: Visualization of the use of flags for re-using a discontinuous style. Each figure shows levels one, three, five and eight (of eight). (a) Left: the captured style rendered at each level without discontinuity flags. Right: the captured flag point values rendered as a curve. Each flag point consists of its ordinal value and its reconstructed flag value. (b) The style applied to and reconstructed on a new base path using the flag values with a threshold of 0.5 to indicate gap starting and ending points.

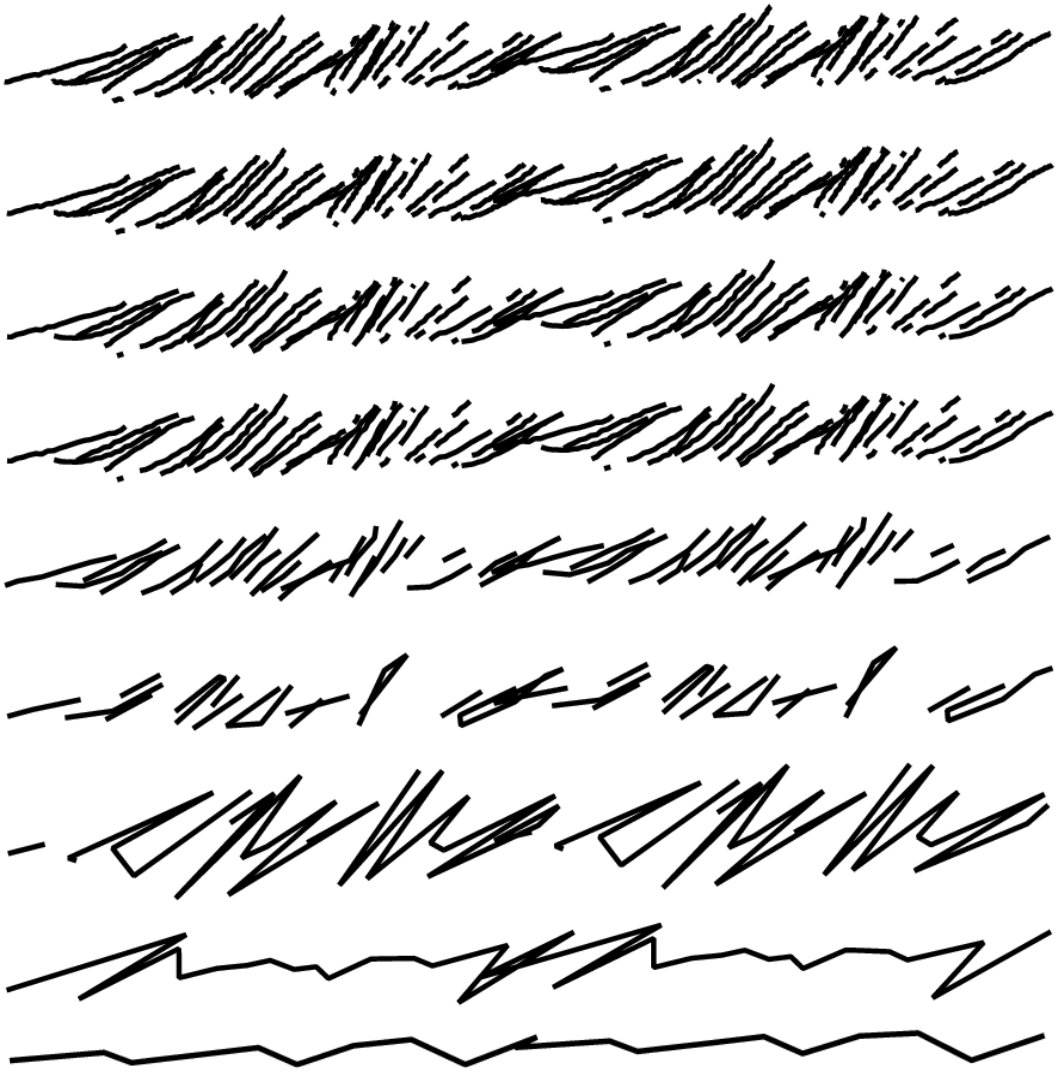


Figure 5.7: Using multi-stroke gap values during decomposition for curve rendering. Top to bottom: full decomposition from level nine to level one of a complex, hand-sketched hatching pattern repeated twice on a new base path. The style has been extracted with the global Chaikin scheme. Note how the look of the style is preserved at each of the first several lower-resolution levels. As the number of points drops from the 3268 in the top level down to the 71 in the third lowest level, the result predictably suffers and the flags are no longer reliable guides for the style rendering.

Chapter 6

Re-Using Artistic Styles

In its simplest form, applying a captured style can consist of a single step: reconstructing the captured details on a new base curve. As described in Chapter 3, this results in the style being "built" onto the curve during reconstruction. We describe the basic style application in Section 6.1. Then, to that basic idea, we add tools for manipulating the look of the result for both precise reproductions or stylized versions of the captured curves.

The first tool corrects a flaw revealed when rebuilding the captured details on the new curve. While the detail vectors are relative to the old base curve, the detail vector angles are not. Section 6.2 describes our approach to achieve this relativization, generating intuitively correct looking results. In Section 6.3 we provide methods for repeating a shorter style over a longer base curve. The look of the style is preserved by overlapping and blending multiple copies along it. And Section 6.4 shows how we use the multiresolution nature of the method to drop details of low visual importance to the final rendering, gaining time and space savings during processing.

We have implemented these procedures in our style applier, which is an interactive sketching system. This approach provides an artistic tool and has the advantage of combining the automatic and semi-automatic parts, allowing us to experiment with the multiresolution techniques that are the basis of this work. The use of the system involves sketching an illustration, and then interactively assigning styles from the library to its strokes. Another tool within the style applier is a multiresolution base

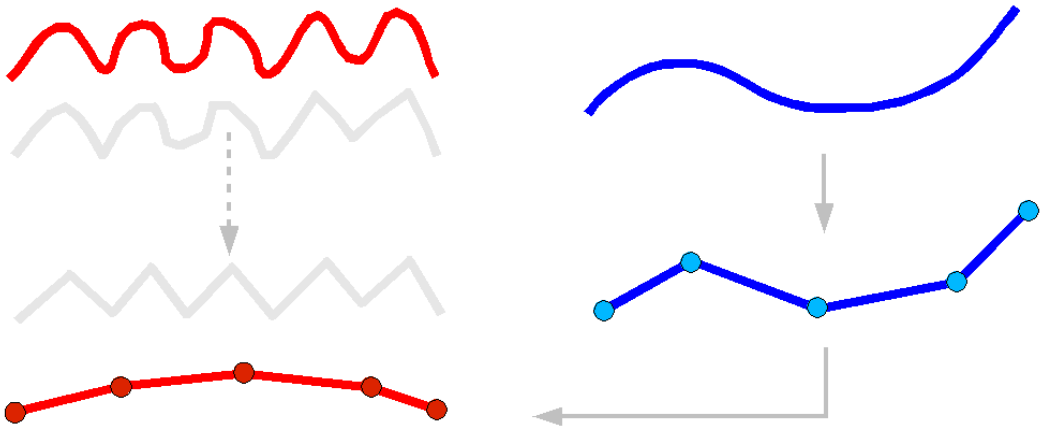


Figure 6.1: Point replacement for reconstructing a style on a new base path. The basis of the multiresolution style re-use technique is to rebuild the captured details upon a new base path. Left: A sketched style is decomposed down to its lowest level producing a base path with five points. Right: The new curve that will receive the style is resampled down to five points, which replace those of the base path of the decomposed style.

path editor. It is used to interactively manipulate the base curve at any level of detail, showing the reconstructed style and any of the above parameters immediately. We describe this in Section 6.5.

6.1 Point Replacement

Our starting point for reconstruction is *point replacement* of the captured style's base curve. This consists of replacing the base path points of an extracted style and reconstructing the details on it. The base path, as the coarsest approximation to the original curve, is made up of some small, fixed, number of points, n. We replace those points, the result of the final decomposition step, with points taken from the new base curve that the style will be applied to. The relationship between the two curves used

in this process is shown in Figure 6.1. The new base path is first resampled since it is from a user's hand-sketched curve and has an arbitrary number of points. We use the resampling algorithm of Section 5.1 to generate the required n points along it for the replacement, and then simply copy those generated points into the filter-bank, replacing the original ones. The filter-bank is then reconstructed on those points using the original detail levels, resulting in the application of the style to the new curve. We note here that the uniformly-spaced points generated by our resampling algorithm will be an accurate representation of the high-resolution original, given the generally consistent sampling of a mouse or tablet pen.

Smoothness of Base Points

A necessary outcome of this technique is that small variations in the base path which have not been extracted into a detail level will be discarded when those points are replaced. For this reason, selecting the smoothest result during style extraction minimizes any possible distortion that would be introduced by reconstructing on the new base path points. Figure 5.3 shows a typical example of the difference in bath path smoothness that results from using various filters. In that figure, the high frequency variations in the base paths of the cubic B-spline schemes are lost when points from a new, smooth, resampled, base path are copied in, which causes the "deflated" look of the reconstructed style. In general, the smoothness itself is a function of the frequencies of the curve detail and length of the sample, combined with the number of decomposition levels achievable by the multiresolution scheme selected. For example, a short curve of a few dozen points with very high frequency details, such as wildly varying spikes, will only allow a few levels of decomposition

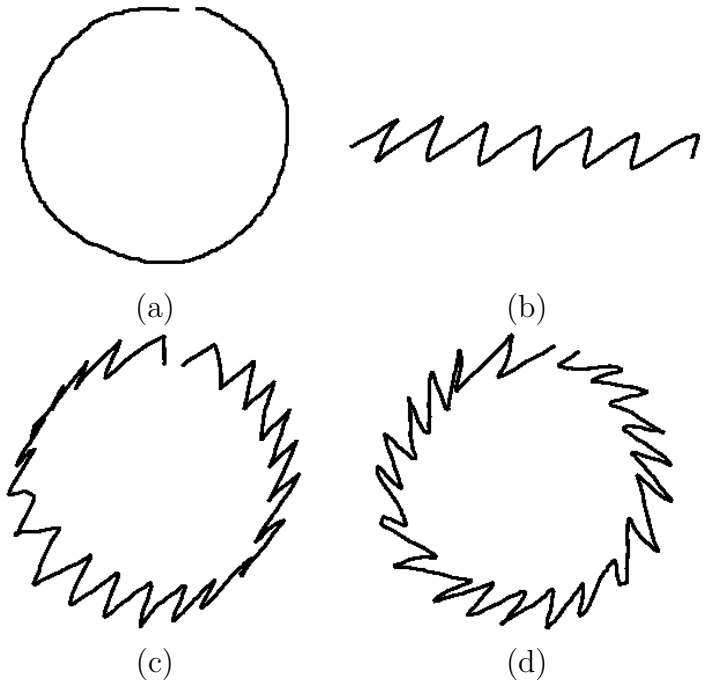


Figure 6.2: Effect of re-orientation of detail vectors when reconstructing a style on a new path. A new sketched base path (a), has a sketched, jagged style (b) applied to it without (c) and with (d) detail angle re-orientation of the style. The re-orientation process converts the detail vector angles into relative offsets, like the detail vectors themselves.

before the filter's minimum width is reached, leaving unextracted details in the base path. Experimentally, we have found that higher resolution input such as 160+dpi scans and sketches on the order of hundreds of points generate acceptably smooth base paths.

6.2 Detail Re-Orientation

A problem with the simple style application described above occurs when the orientation of the new base curve differs significantly from the original base curve. In

those situations the captured details will not have an intuitively "correct" look, as in Figure 6.2(c). This problem was also addressed by Finkelstein and Salesin [FS94], in the context of curve editing. The problem arises during reconstruction, but it is created by the decomposition. At each stage of the multiresolution analysis, the detail vectors are extracted relative to the coarsened curve approximation. When these details are reconstructed on the new base path, the relative offsets are "naively" added to the new base path segments, distorting the style.

Figure 6.3 illustrates the situation. In (a), the detail vectors d of a captured style are shown on their base path t. From there, a decomposition step extracts those vectors as relative offsets to t. The naive reconstruction on the new base path $\bar{t}$, shown in (b), makes it clear that the style cannot be preserved. The arbitrary variations of the new path force the detail vectors into different orientations with respect to it. To solve this, we consider a local frame at the curve point that is tangent to the curve and normal to the curve. We extract the angles between the detail vectors and the tangents for each multiresolution level during the decomposition steps. We then use them to re-orient the detail vectors on the new base path. Figure 6.3(c) shows the process. We compute the angle β_i that each unmodified detail vector d_i makes with the segment $\bar{t}_i$ of the new base. This is then used to determine $\phi_i = \alpha_i - \beta_i$, the angle that d_i must be rotated through to give it an orientation of α_i with $\bar{t}_i$. The rotated vector $\bar{d}_i$, now has the same orientation to $\bar{t}_i$ as d_i did to t_i. This is done for each detail level of the reconstruction, leaving any partial reconstruction also appropriately oriented.

The corrected reconstruction is shown in Figure 6.2(d): the captured style is now clearly visible at all orientations over the circle. We note that extraction through

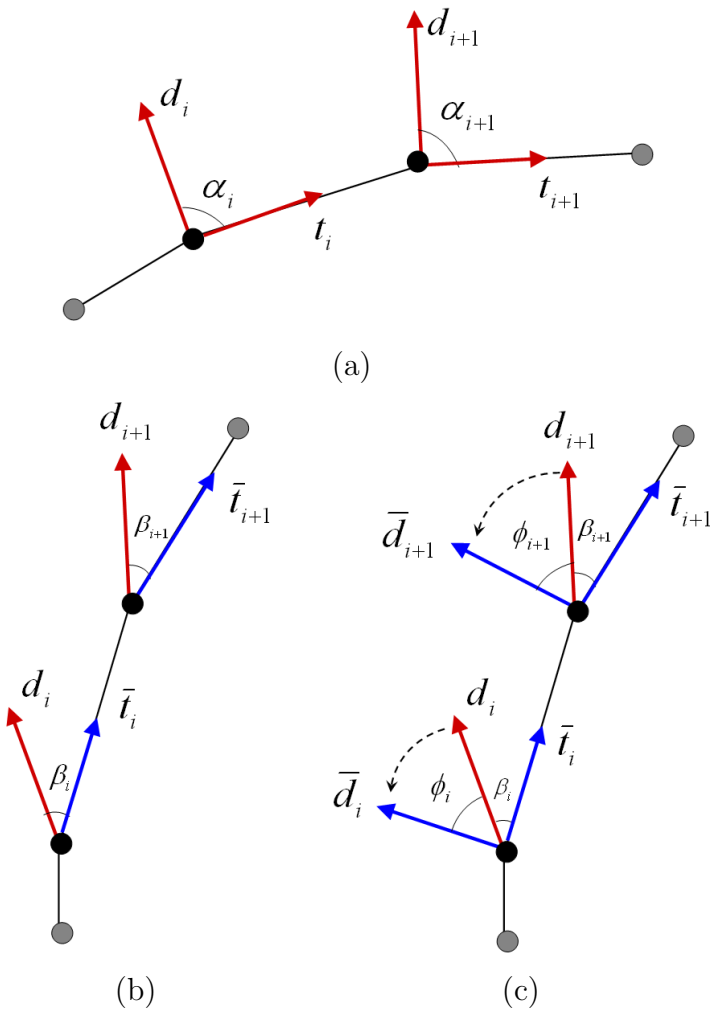


Figure 6.3: Detail vector re-orientation. (a) The detail vectors d are shown with their associated angles α to the path of their base curve t. (b) shows the result of the style application to the new base $\bar{t}$ with naive reconstruction. The detail vector angles α are lost and the new angle β distorts the style. Our solution is shown in (c): before reconstruction the detail vectors d are rotated by angle $\phi = \alpha - \beta$, re-orienting them to angle α with respect to the new base.

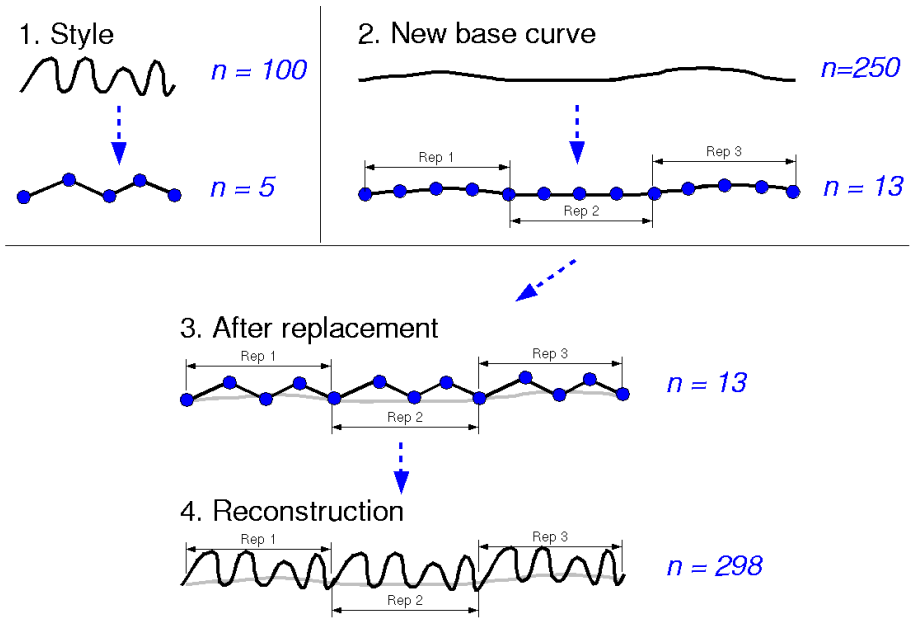


Figure 6.4: A short style segment (1) is decomposed into its base path of $n = 5$ points. (2) The new base curve is resampled to a compatible number of points $m(n-1)+1 = 3(4)+1 = 13$, with m being the number of repetitions desired. (3), (4) The replaced points are reconstructed using the resampled points.

decomposition coupled with this method gives us complete separation of the style from the original path of the stroke.

6.3 Style Repetition

We extend our point replacement technique to support repeating a short style segment multiple times over a single new base path. The biggest issue we face here is visually camouflaging the transition from the end of one style reconstruction and the start of the next.

In our initial solution, we divide the new base curve into multiple segments and

reconstruct the details upon them, using a single point of overlap between them. The effect then is of a single unbroken curve with a long, regular, style applied. In many cases, however, the captured sketched style is irregular at the ends, as in the top example of Figure 6.6. This leads to a distorted look when repeating the style. We can solve this interactively by dropping points at the ends of the style and re-creating the filter bank, described below.

Our second approach allows a more flexible arbitrary overlap factor for the style repetitions, and performs multiresolution blending of the detail vectors to merge the joining area of the two styles. This gives significantly better chances of finding a suitable, seamless, look.

6.3.1 Style Repetition By Point Dropping

Our method again uses the resampling algorithm to select the appropriate number of points from the new base path. For a style with an extracted base path of n points that we wish to apply m times, we resample the new base path to $m(n-1)+1$ points. We then reconstruct the style on points $0..(n-1)$, $(n-1)..2(n-1)$, etc., so each style segment overlaps at its joining point with the next segment. This technique allows us to apply a style to an arbitrarily shorter or longer base path than the style itself. Figure 6.4 shows the base path resampling and replacement that leads to reconstructing a style repeated over a longer new base path.

Figure 6.5 shows the result of several applications of a short sketched style to a new base curve using this technique. The thrice-repeated application is most similar to the original, but the effect of using fewer or more repetitions over the curve produces a family of related styles, any of which may be used instead of the

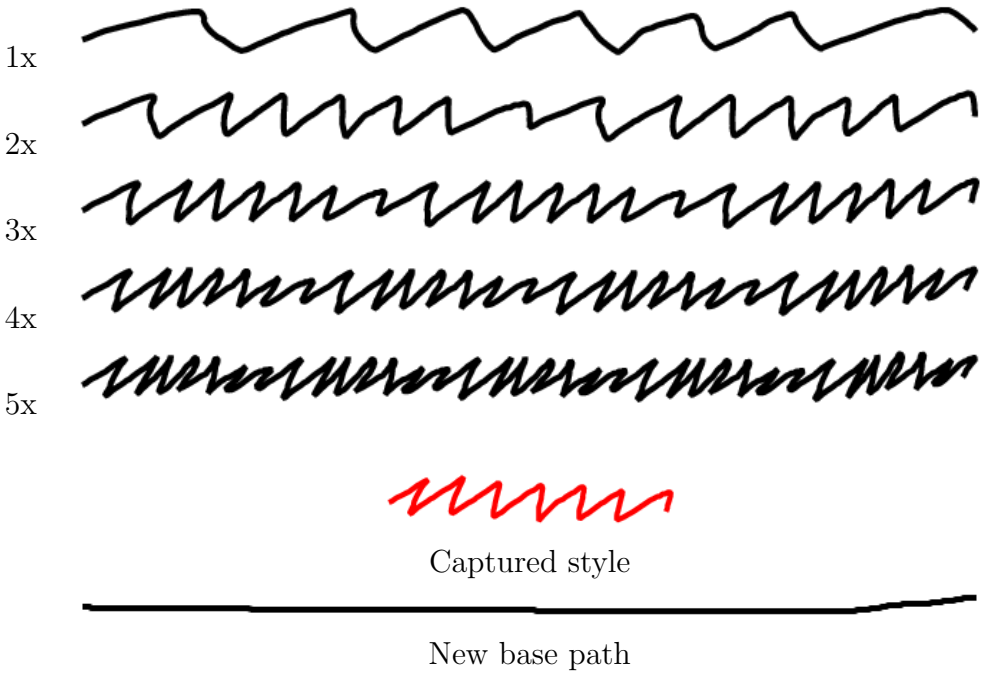


Figure 6.5: Style repetition over a curve. We apply a small captured sketched style with repetition to a single new base path. The result shows an intuitive stretched or squashed look when the style is either too narrow or too wide to fill the allocated space.

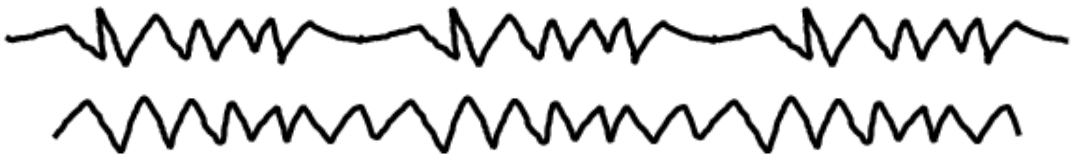


Figure 6.6: Fixing repetition of a style by dropping end points. The smooth areas between the three repetitions of the style (top) are eliminated by dropping start and end points of the extracted style. The result is a seamless repetition (bottom).

exact reproduction. In our implementation, an interactive preview of the effect of varying the number of repetitions is shown. The top row (1x) shows how the original captured style is stretched to cover the length of the new base path with one repetition. In contrast, the bottom (5x) shows how the style becomes squashed when it is reconstructed on path segments that are shorter than the original style.

One issue with this technique is that the style reproduction is not always exact at the end points. The problem is the existence of some ad hoc columns for end-point interpolation near the first and last columns of the multiresolution matrices. The top image of Figure 6.6 shows a style repetition with this problem. We currently resolve this interactively during the style extraction process. We provide an interface that allows the user to drop points from the ends of the style until the inner regular section is reached. The result is visualized in real-time by reconstructing the altered style on an example new base path. When the extraction generates a seamless repetition, we store the modified style for re-use. In practice, with a few trials we can find a good result from most styles, such as the bottom image of Figure 6.6. This method imparts some flexibility to the process, leaving the final visual appearance to the user's discretion.

Algorithm 2 Generating a seamless repetition of a style on a new base path using detail blending

DetailBlend
Input: $FilterBank$, $NewBaseLeft$, $NewBaseRight$, $Overlap$
Output: $OutCurve$

for each detail level l in $FilterBank$ **do**
  $NewLeft \leftarrow$ Reconstruct up to l of $FilterBank$ on $NewBaseLeft$
  $NewRight \leftarrow$ Reconstruct up to l of $FilterBank$ on $NewBaseRight$
  $n \leftarrow$ Number of detail vectors within $Overlap$ range of $NewLeft$, $NewRight$
  $NewLeftFB[l] \leftarrow$ Copy refined detail level l from $FilterBank$
  $NewRightFB[l] \leftarrow$ Copy refined detail level l from $FilterBank$
  $Blended[n] \leftarrow$ Blend n overlapped details from end of $NewLeftFB[l]$ and start of $NewRightFB[l]$
  Update end of $NewLeftFB[l]$ with $Blended[n]$
  Update start of $NewRightFB[l]$ with $Blended[n]$
end for
$tn \leftarrow n$ from top detail level
Set base path of $NewLeftFB$ to $NewBaseLeft$
Set base path of $NewRightFB$ to $NewBaseRight$
$NewLeft \leftarrow$ Reconstruction of $NewLeftFB$
$NewRight \leftarrow$ Reconstruction of $NewRightFB$
Append points of $NewLeft$ to $OutCurve$
Append points of $NewRight$ from [tn..length($NewRight$)] to $OutCurve$

6.3.2 Style Repetition By Detail Blending

In our alternate approach, we blend overlapped style sections to achieve a seamless repetition. This differs from dropping points, above, by allowing two adjacent style segments to be averaged and merged until a smooth transition is found. This merging of adjacent segments is enabled by dividing the new base path into independent pieces, rather than simply resampling it into the appropriate number of points.

The blending occurs in a multiresolution fashion, as described in Algorithm 2. Its inputs are $FilterBank$, representing the style to apply; $NewBaseLeft$, the left

base path segment; $NewBaseRight$, its adjoining right-hand segment, and $Overlap$, the amount of overlap between the end of the left and start of the right segments. It returns a curve in $OutCurve$ which is the result of reconstructing the blended $FilterBank$ on the left and right hand side of the overlap.

In the first part, lines 1-9, new filterbanks are created level-by-level with the averaged, overlapped details. For each level, we first find the overlapped vectors by reconstructing the style on the new base (lines 2 and 3), giving us two point curves $NewLeft$ and $NewRight$. The number of overlapped points n is determined by taking the minimum of the number of points that fall within the range defined by the overlap value, $Overlap$, for $NewLeft$ and $NewRight$. In our system $Overlap$ is represented by a screen-space parameter. In lines 5-7, the n overlapped details are copied from $FilterBank$, refined, and averaged into the temporary list of detail vectors, *Blended*. They are then copied into the appropriate level of the new filterbanks $NewLeftFB$ and $NewRightFB$ (lines 8 and 9). The blended values are copied to the end of the left segment and the start of the right segment.

At the end of the loop, each level's detail vectors have been updated with the result of the blending. The final steps use the blended filterbanks as though they were conventional styles in our system (lines 12-15): We reconstruct them on the base path segments $NewBaseLeft$ and $NewBaseRight$, which applies the blended style to each. The final single returned curve $OutCurve$ is created by copying the points created by the $NewBaseLeft$ reconstruction and the points of the $NewBaseRight$ reconstruction that are not duplicated (lines 16 and 17). The number of duplicates dropped from the $NewBaseRight$ reconstruction is determined by the number of overlapped points from the last, highest resolution, reconstruction of $FilterBank$

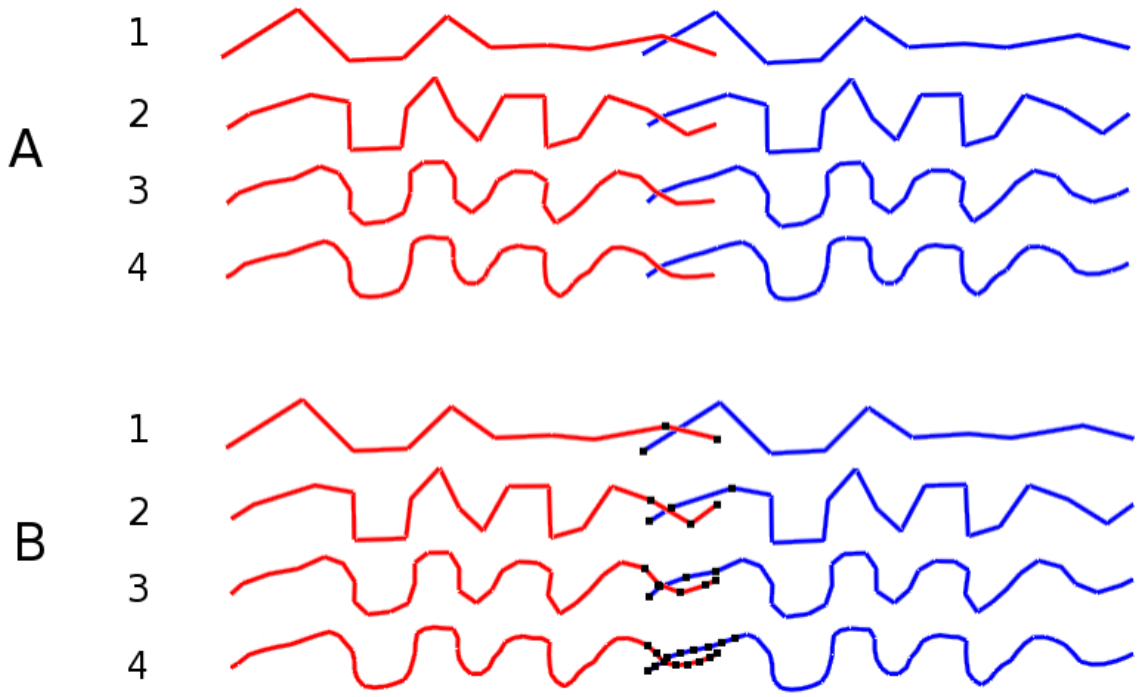


Figure 6.7: Constructing a blended filterbank. (A) Visualization of the left and right style segments after reconstruction with the blended detail vectors. The major discontinuities at the end and start of the style have been smoothed together, eliminating the obvious join seen in Figure 6.8. (B) Determination of overlapped detail vectors for each of the first four detail levels of the style filterbank. The black points represent the locations of the detail vector bases that are within the overlap range (approximately 12% in this example).

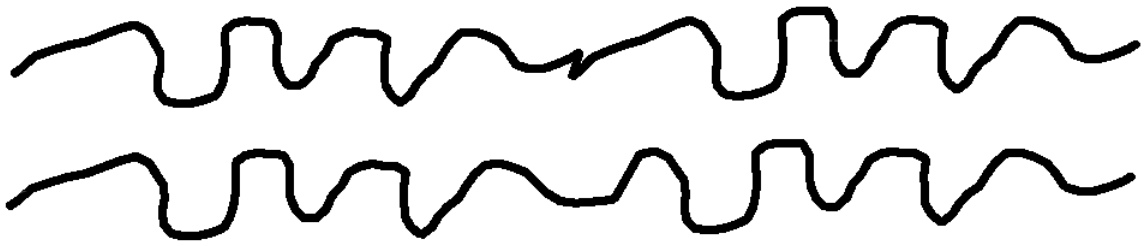


Figure 6.8: Fixing repetition of a style by blending overlapping detail vectors. The disjoint area between the two repetitions of the style (top) are eliminated by overlapping them and blending together the overlapped detail vectors using Algorithm 2.

(line 11).

Figure 6.8 shows a style repetition discontinuity fixed with Algorithm 2. The top figure shows a twice-repeated application, without point dropping. The bottom shows a blended overlapped repetition, with an overlap range of approximately 12%. The detail vectors identified for blending, and the subsequent reconstruction of the blended filterbanks on the new, separate, repeated base segments are shown in Figure 6.7. This technique, like the point dropping, is semi-automatic: the user must adjust the overlap until a suitable result is found. The difficulty in finding an automatic metric for the overlap parameter is mostly due to the display variations introduced by using user-drawn, arbitrary new base path curves. Given the basic premise of the technique as a method for generating aesthetic results, we feel this method is still a valuable enhancement to the basic point-dropping approach.

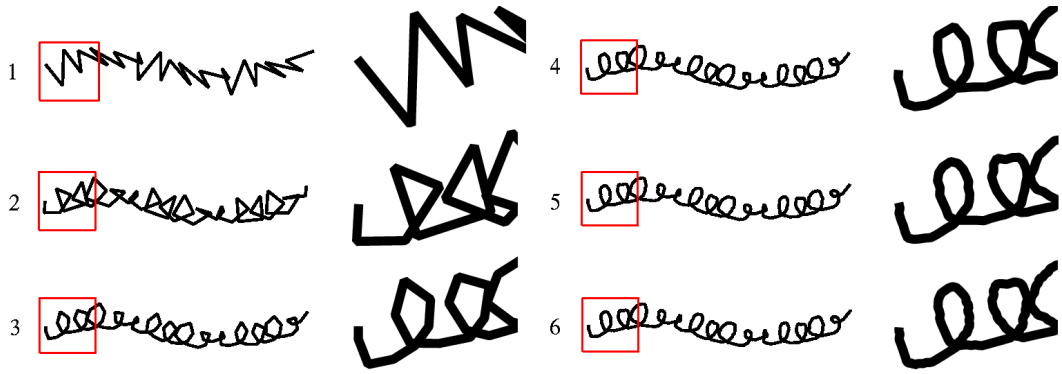


Figure 6.9: Steps 1 to 6 in a full reconstruction of a captured style. The zoomed area (right) shows how building to just level 4 gives a good approximation with only a fraction of the points in the full reconstruction.

6.4 Build Level Selection

As a final step for the style application, we can selectively discard some of the highest-resolution detail levels. We can do this without impacting the visual appearance of the final curve because of the nature of the multiresolution decomposition: the highest detail levels encode very small variations of the underlying curve. In our data, these details are often "noise" from the scan or the input device which are not required for a good reconstruction. Dropping the highest levels thus complements the simple image filtering by removing this noise from the input. As an added advantage, we save computation time by reducing the number of synthesis passes needed to apply the style.

Figure 6.9 shows the successive levels of a style reconstructed on a new base path. Note that the final three reconstructions to levels 4, 5, and 6 are virtually indistinguishable at the printed resolution. In general, we find that we can leave off

the final two or three levels without affecting the new curve's look, but with considerable savings in computation and space. For example, the level 4 reconstruction generates a curve with only 201 points versus 777 for the full reconstruction to level 6. To automatically select levels to discard, we average the magnitude of the detail vectors in the first few levels of decomposition and compare it relative to the final level, where the detail vectors typically have the greatest magnitude. A threshold of 2.5 to 5% will tend to remove only levels representing input noise, while preserving the style at typical print or screen resolutions.

6.5 Interactive Base Path Editing

Once the style has been assigned to a base curve and its various attributes selected, the user may want to alter the final look of the illustration. Rather than re-drawing the base paths, we have implemented a multiresolution curve editor for them. This allows the user to interactively change the paths while the fully-reconstructed style with attributes is updated instantly. The full advantages of multiresolution editing are available: being able to make both broad changes to the overall sweep of the curve at low resolutions, and small adjustments to fine details at high resolutions. This reduces trial-and-error and is in keeping with our goal of providing a convenient interactive artistic tool.

The implementation follows readily from our existing multiresolution framework. Rather than treating the sketched base path as a simple curve, we resample and construct a filter bank from it using the global cubic B-spline scheme. When a particular level is selected for editing, we reconstruct the curve to that level and

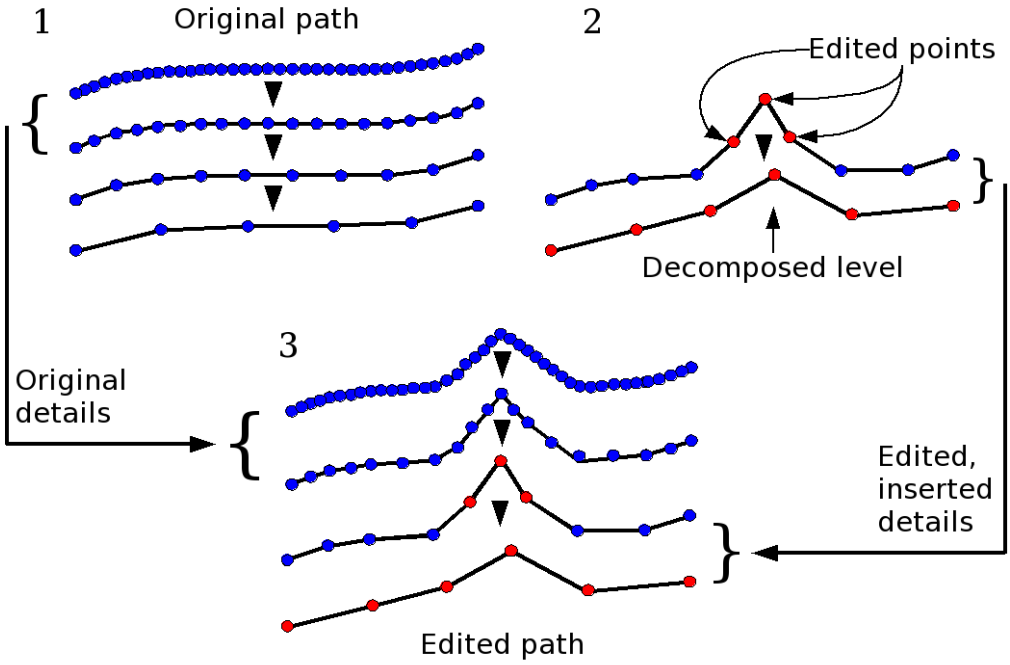


Figure 6.10: The base path editor in the style applier allows the user to interactively adjust the sweep of the curve while visualizing the style reconstruction in real-time. (1) The user sketches a high-resolution base path (top). A filter-bank is created from the path, resulting in several detail levels (below thick arrows). (2) The user selects to edit the base path at the second level (top) and adjusts several points (marked in red) into a new curve shape. A new base level is created by multiresolution decomposition of the edited curve level (below thick arrow). (3) A new filter-bank for the edited curve is created from the edited levels one and two, and the unedited levels three and four from the original filter-bank. When reconstructed (top), the new base curve displays the effects of the low-resolution path change.

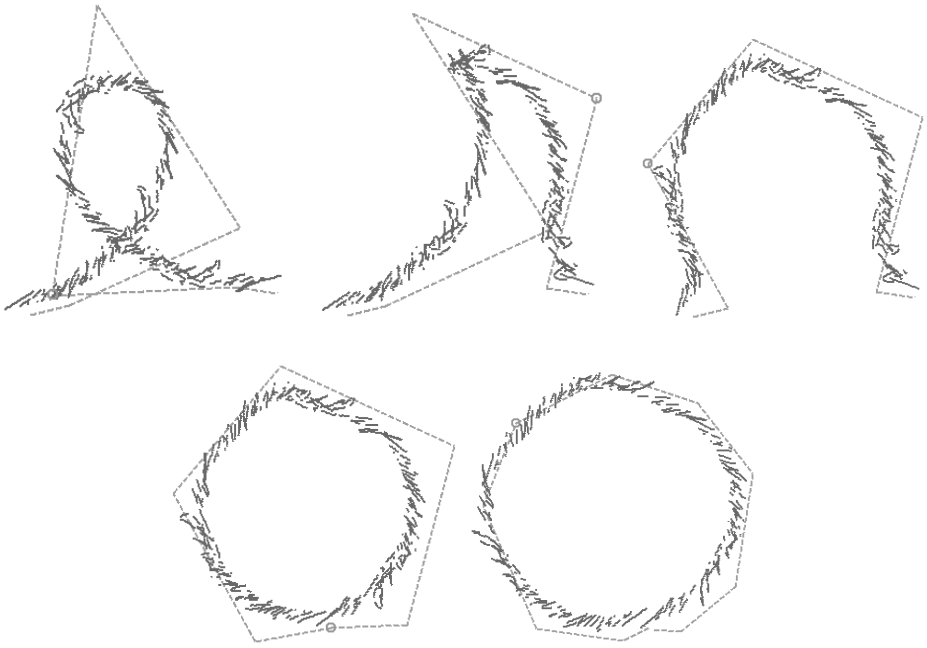


Figure 6.11: Interactively editing the sweep of a base curve at multiple levels of detail within our style applying system. From left to right, top to bottom: We unwind the curve's loop, styled with a discontinuous hatching pattern, by dragging vertices from the base path's lowest detail level. After roughly fashioning a circle, we increase the resolution of the control curve to adjust its final shape.

display the vertices for interactive manipulation. As the user manipulates these multiresolution "control points," we construct another filter bank from them and replace the detail levels of the original filter bank with those from the edited point filter bank. This merges in the changed points at the level they were modified and at all the levels below. When we then reconstruct the modified filter bank to its full resolution, it contains the changes made at the edit level. The level replacement process is illustrated in Figure 6.10. Finally, we rebuild the current style and its attributes upon the modified curve, showing how the changes would affect the overall

image. Figure 6.11 shows several steps in interactive base path editing within our system. The styled curve is shown updated in real-time as the user manipulates multiresolution control points of the curve's base path.

Chapter 7

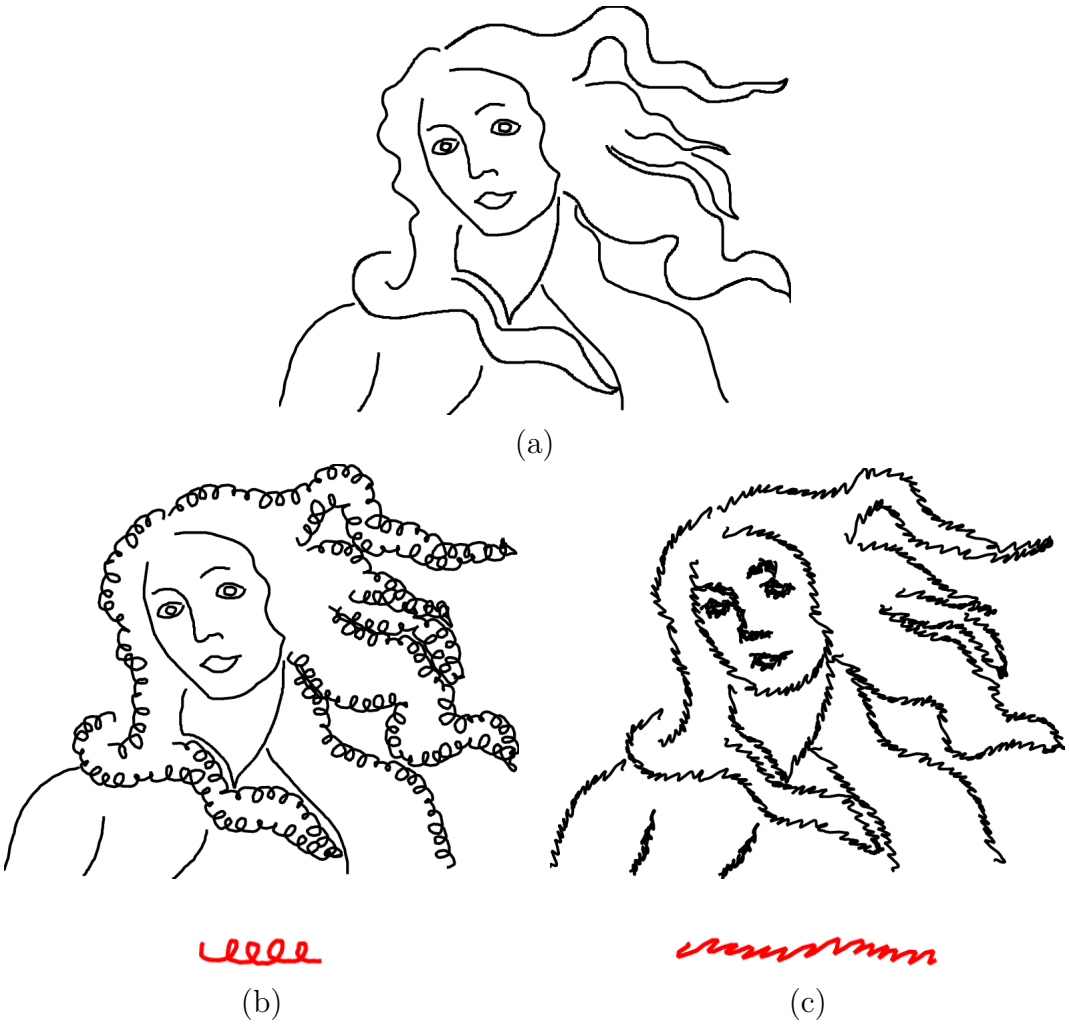
Results and Discussion

All our results have been generated with the style applier on a 2.65 GHz Pentium IV using the Qt software 2D drawing API. The applier and extractor are real-time, running at interactive rates for all our examples. We recommend printing the results at least at 200dpi on a 600dpi laser printer.

Figure 7.1 shows a complete illustration using sketched styles extracted with the global Chaikin filters. We have used style repetition to cover the longer strokes of the illustration with the short captured style segments.

Figure 7.2 shows a leaf silhouette style captured from a single stroke using the local Chaikin scheme. Note that the venation is not re-styled—our goal is to reproduce the more complex serrated pattern of the leaf boundaries and silhouettes. We first reproduce the original and then apply the style to completely new bases. Comparing the new base path and the final result of the two leaf pairs shows the effect of the artistic style on the final composition. Our system can also be used to extract individual style segments from a single larger illustration.

Figures 7.3 and 7.4 show the same process with several more examples. Figure 7.3 uses the global Chaikin and cubic B-spline schemes for the left and right silhouette pieces and Figure 7.4 uses the local Chaikin scheme. For each figure, the top row shows the scanned input image and the corresponding extracted style. For simplicity, we have separated the scans into two pieces to extract as a left and right half. On the very complex styles for the full tree silhouettes, this give us more leeway when



(a)

(b)

(c)

Figure 7.1: Sketched styles applied to alter the look of an illustration. (a) The base curves of the drawing—traced detail from Botticelli—as created in our interactive drawing system. (b) and (c): stylized versions of (a) created by applying curly and sketchy styles to selected strokes of the image. For both results a shorter style segment (red, below) has been shown repeated over the paths to cover the longer base curves.

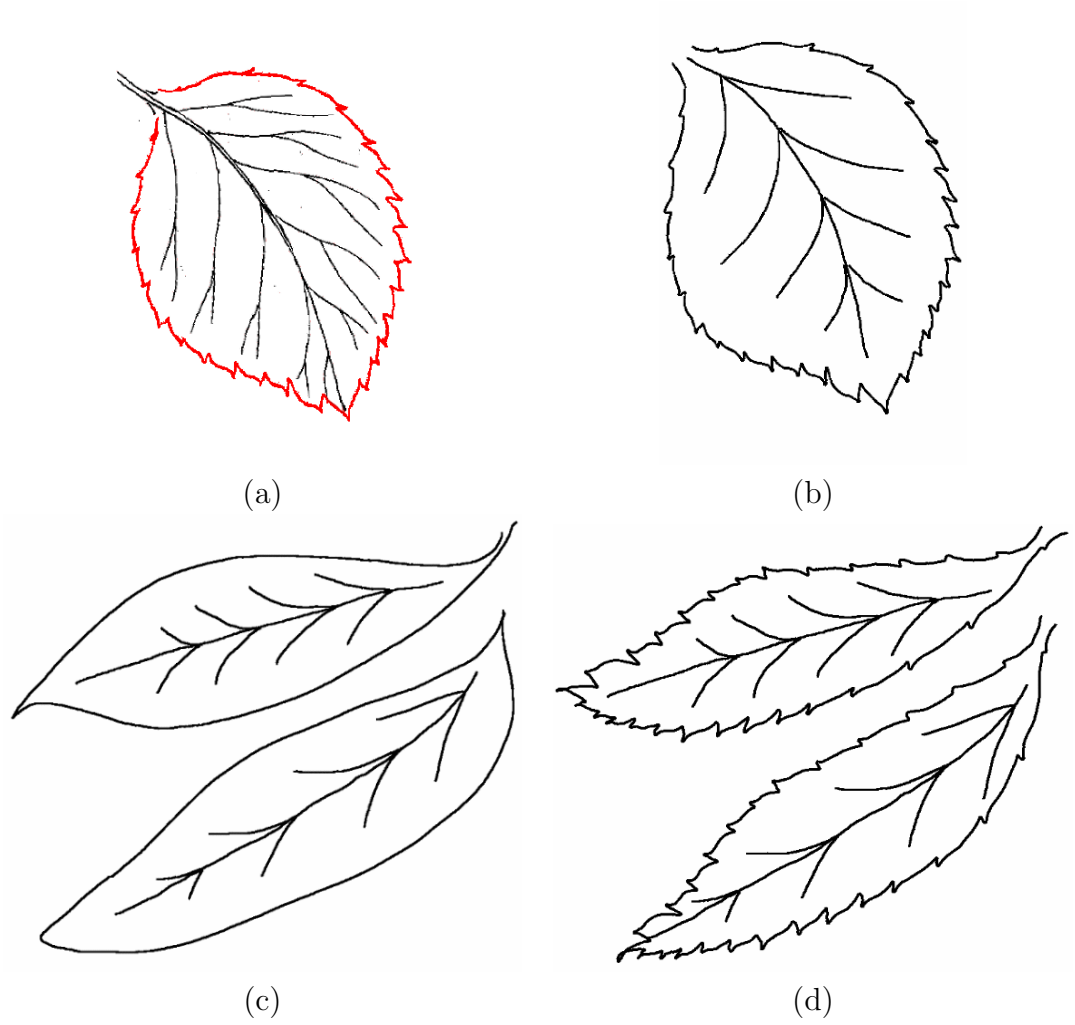


Figure 7.2: Capturing and re-using a leaf silhouette style. (a) Extracted silhouette (in red) of the leaf illustration with a single continuous stroke, drawn over the scanned input image (758x942 pixels). (b) Application to a new base to reproduce the original. The venation has been added by hand to complete the illustration. (c) Two new leaf base paths for the style. (d) Captured style from (a) applied to create new leaf illustrations with the look of the original.

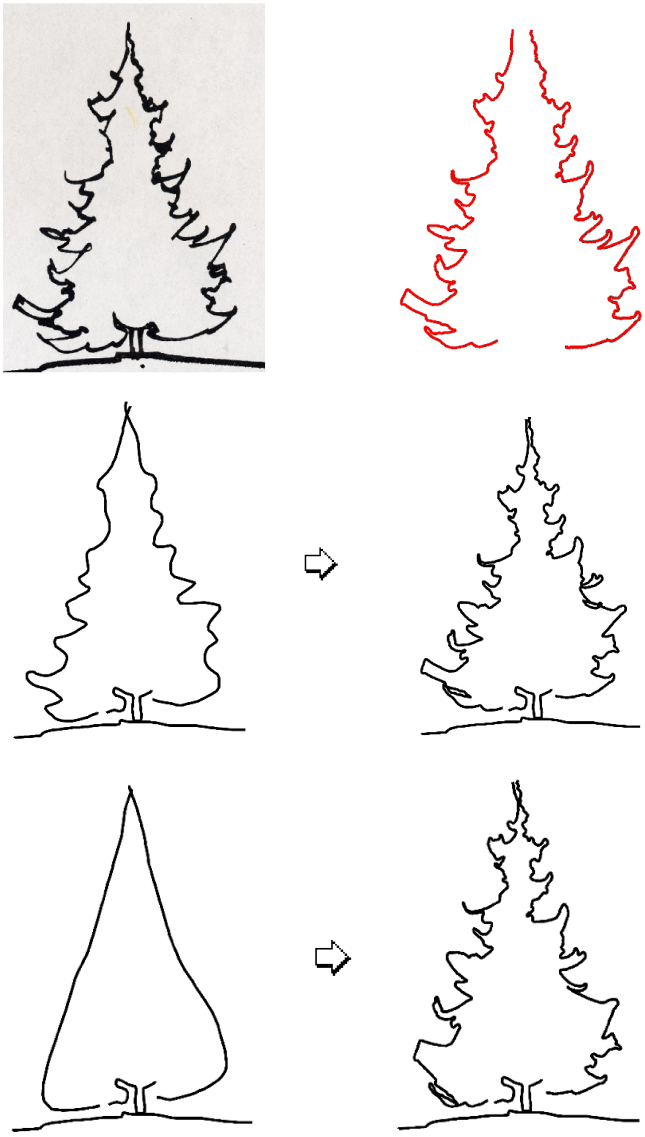


Figure 7.3: Extraction and re-application of a coniferous tree silhouette style. Top row, left: original scanned illustration (320x884 pixel JPEG image), and right: extracted silhouette style in two parts. Middle row: detailed base path following the original and result of style application. Bottom row: simple, rough, base path in the nature of a design sketch, showing a credible reproduction of the overall silhouette look.

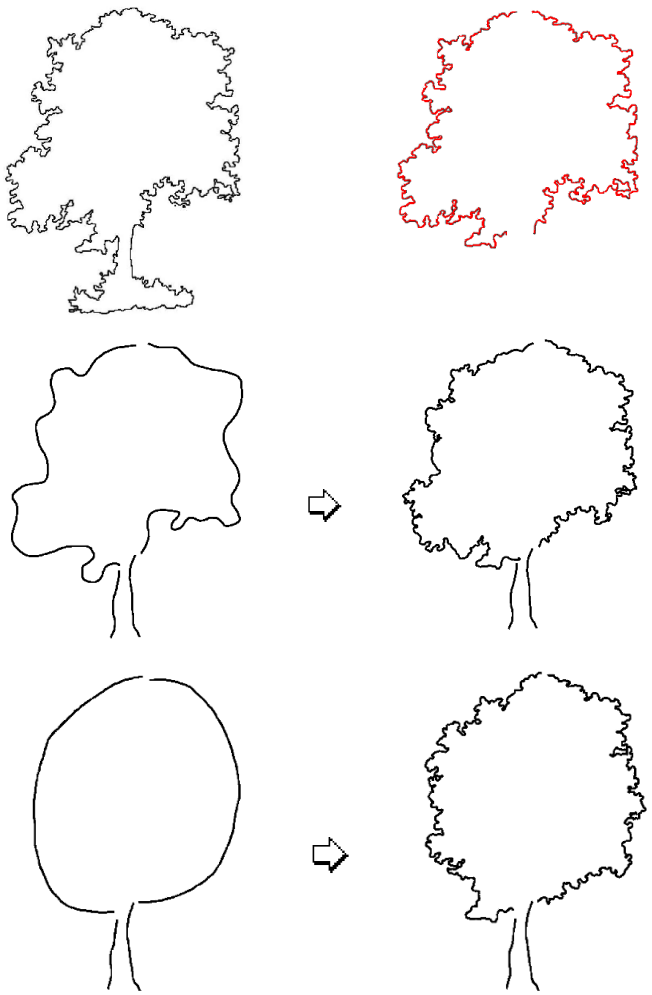


Figure 7.4: Extraction and re-application of a deciduous tree silhouette style. Top row, left: original scanned illustration (645x853 pixel JPEG), and right: extracted silhouette style in two parts. Middle row: detailed base path and result of style application. Bottom row: simple, sketched, path and a good reproduction of the style's detail.

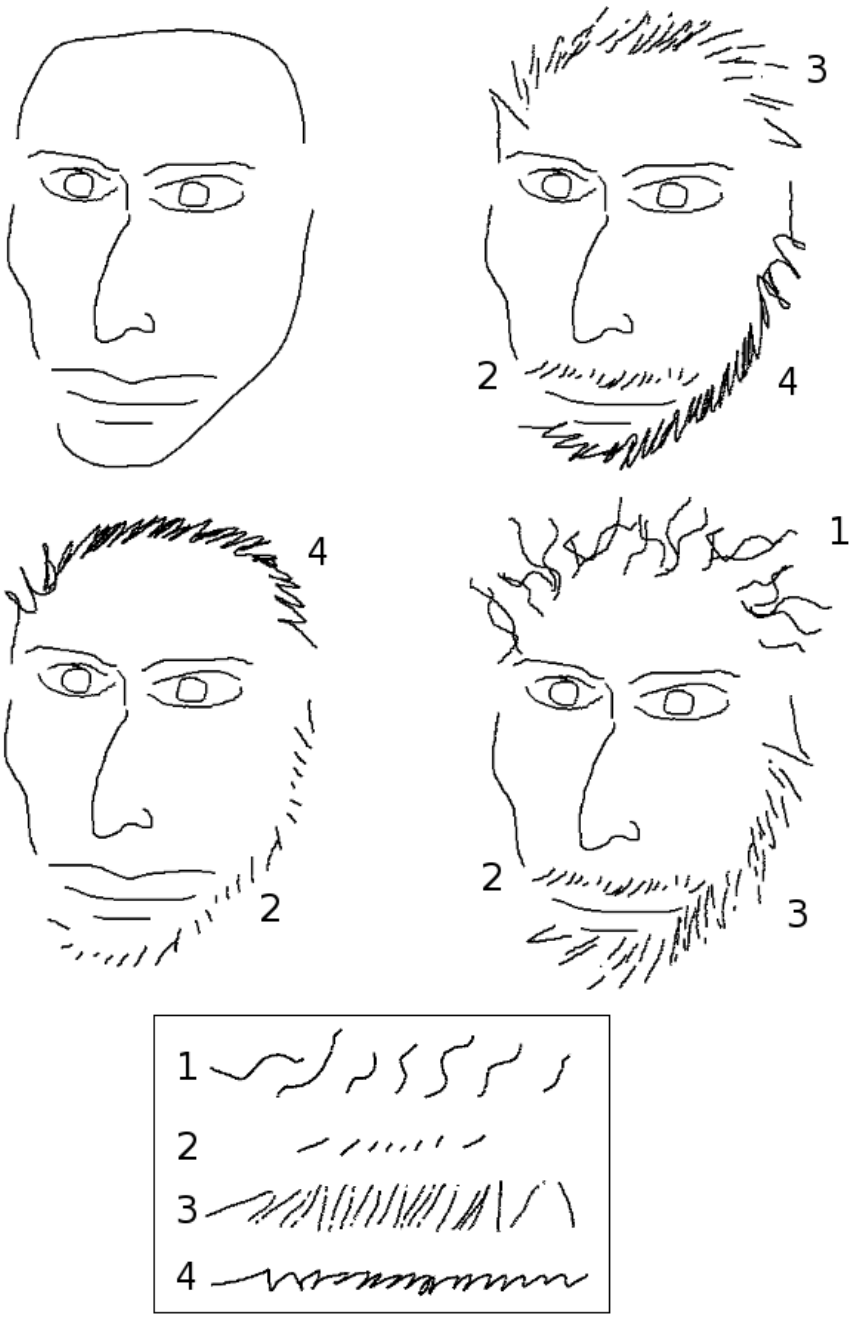


Figure 7.5: Sketched discontinuous hatching styles captured and re-used in an illustration.

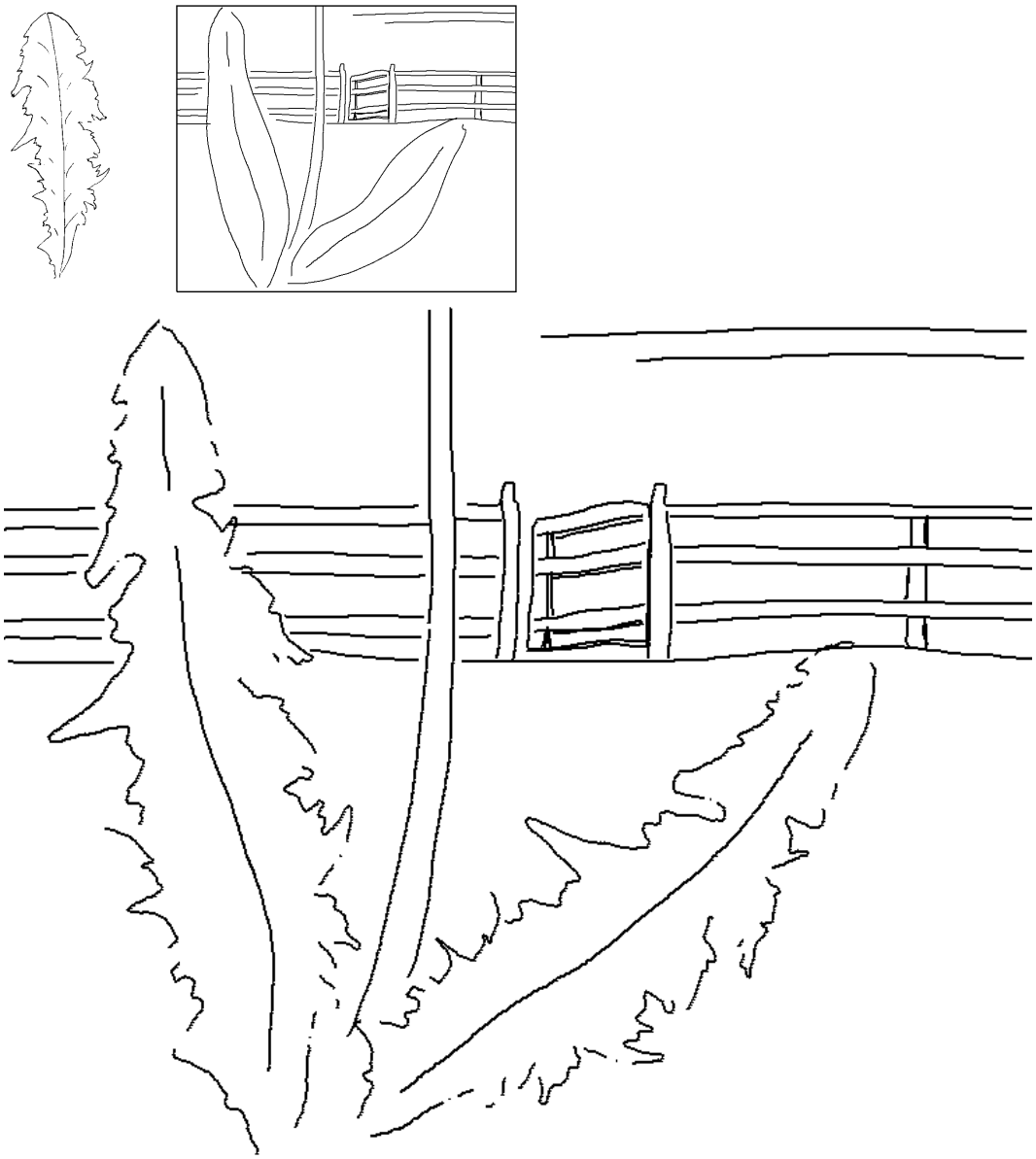


Figure 7.6: Capturing and re-using a scanned discontinuous silhouette style. The dandelion sketch was automatically vectorized from a 282x675 pixel scan. We then used the global cubic B-spline scheme to generated the style's filter bank.

creating the more intricate base paths to match the originals. A full silhouette can also be used, as demonstrated by Figure 7.2. Note that for the extracted style segments on the right side of the top row, the style is reconstructed directly on the extracted base path, leading to perfect reconstruction. In the second and third rows of each figure, the reconstruction on a new base path is shown. We have attempted to reproduce the original by using a rough sketch (left, second row), capturing only the large-scale variations of the original in our new base path. The result of applying the styles to this sketch is shown on the right. In each case a close approximation to the original is achieved. On the left side of the third row of each figure, we use only the broadest outline of the original scan as a new base. The result, at right, shows how the style is flexibly preserved even with such a rough approximation to the original. While a detailed base path that follows the tree's outline yields a nearly exact reproduction, a generic sketch still preserves most recognizable details from the original. In the context of our illustration system, this property of our multiresolution technique allows even non-experts to create drawings with the look of a captured style, while not giving the result the appearance of an exact copy.

In Figures 7.5 and 7.6 we show some additional results from scenes sketched in our interactive system. The former uses sketched and latter uses scanned discontinuous styles. Note that the sketched styles are complex hatching strokes that would be difficult to automatically extract from scanned images.

7.1 Comparison to Wavelet-based Multiresolutions

The advantages of constructing multiresolutions based on reverse subdivision have been discussed in Bartels and Samavati [BS00, SB99]. Here we compare the method of Finkelstein and Salesin [FS94] to our method. While both approaches have linear-time algorithms for decomposition and reconstruction, in practice the reverse subdivision approach is much faster. Table 7.1 shows the average result of several timing runs for a full decomposition and reconstruction of curves with 35, 8195, and 32771 points. We find the global method to be almost six times faster and the local method almost nine times faster. This is primarily due to the shorter bands of the filter matrices we use, as discussed in Section 3.3.

Another advantage is that the simplicity and generality of the reverse subdivision method allows us to use different filters, including both local and global Chaikin and cubic B-splines. Other schemes, such as Dyn-Levin, are also available [BS00]. Finkelstein and Salesin [FS94] only consider cubic B-splines and do not offer a way to employ other curve schemes.

Figure 7.7 graphically shows that our new approach suffers no loss in quality while achieving its speed advantage. We extract a tree silhouette style using wavelet-based, global, and local cubic B-spline schemes and then re-apply the style to a single new base path, representing a perturbed, or "wind-swept," version of the original. Our global technique provides identical visual results to the wavelet-based approach, while the local method still provides a close approximation with some distortion due to a coarser base path.

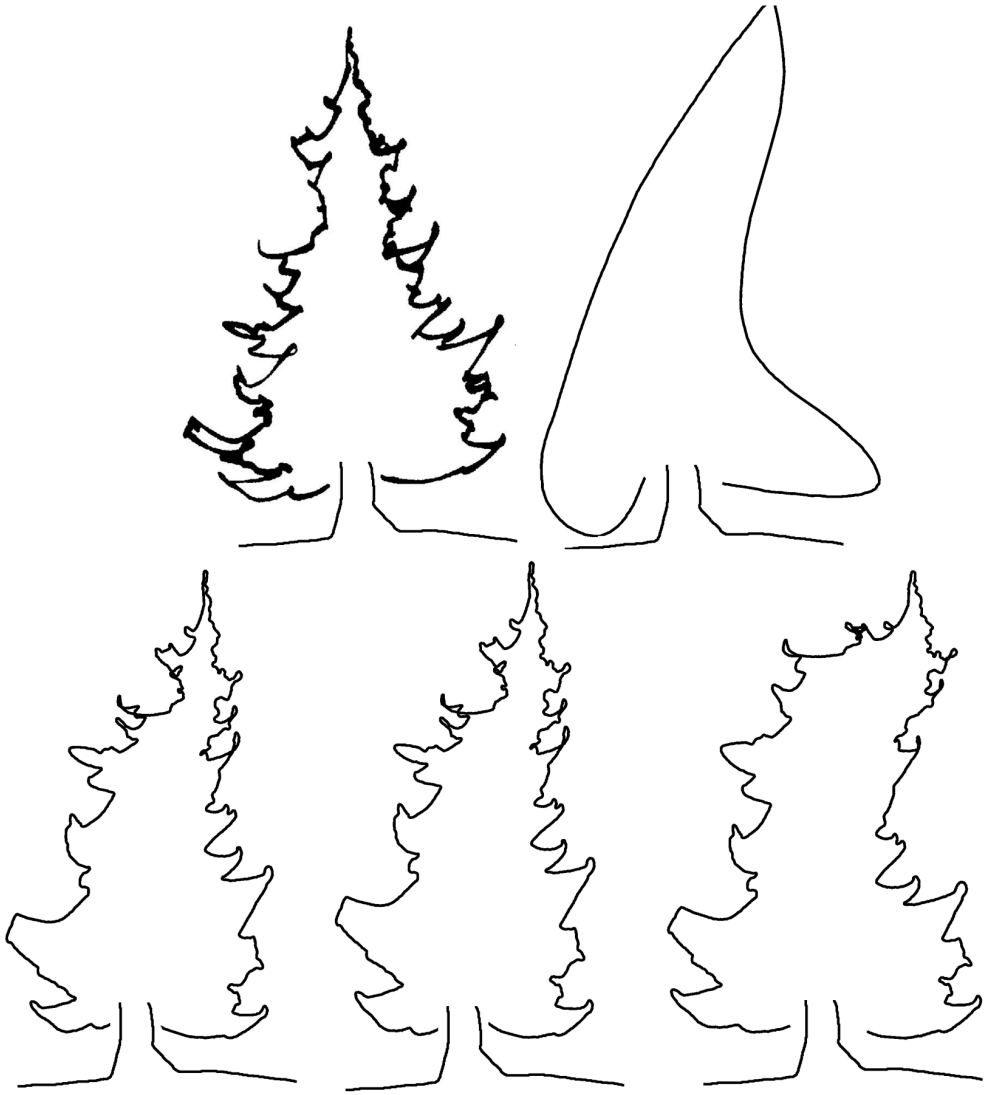


Figure 7.7: Graphical comparison of wavelet and reverse-subdivision multiresolution methods for style capture and re-use. From left to right: the original sketch that the style was extracted from; a new hand-sketched base path for the style; the new base path with style applied using wavelet-based, global, and local cubic B-spline filters.

#points	Wavelets	Global RS	Local RS
35	$311\mu s$	$55\mu s$	$37\mu s$
8195	$73ms$	$12ms$	$8ms$
32771	$309ms$	$51ms$	$34ms$

Table 7.1: Timing comparison of wavelet and reverse-subdivision multiresolution methods for curves of different sizes using cubic B-spline schemes. Both the global and local implementations of Samavati and Bartels [SB99, BS00] achieve significantly faster times than our implementation of the filters of Finkelstein and Salesin [FS94]. All timings are the mean of five runs performed on an unloaded 2.65GHz Pentium IV with 1GB of RAM.

Chapter 8

Conclusions

We have presented a technique for NPR-by-example that allows exact reproduction of artistic stroke styles. Our method does not require training sets and does not rely on statistical similarity to synthesize new strokes with the extracted style. The multiresolution technique was have adopted gives us both a simple and automatic way to separate a style from a curve and yet still preserves artistic stylization by giving multiple rendering options to the user. For example: using longer or shorter style segments, using a wide range of existing styles that may be easily imported into the system, and allowing arbitrary base paths. Combined, these naturally encapsulate variations intrinsic to hand-drawn styles. We have demonstrated our method with a variety of silhouette styles showing the flexibility of the capturing and applying techniques.

The principle advantages of our method are its conceptual simplicity and the ability to achieve efficient implementation of extraction and application. Using the filters of Bartels and Samavati [BS00, SB99] allows us guaranteed linear time multiresolution decomposition and reconstruction. Additionally, the filtering and banded linear system solving operations are simple and easily-optimized, and we show that they are significantly faster than previous multiresolution techniques using wavelets. Our system can perform real-time extraction and application of the styles, providing a useful basis for interactive stroke-style render-by-example. We have found that our new techniques provide results comparable to previous work for style extraction and

re-application. In particular, the global filters and local cubic B-spline filter generate the smoothest base paths, allowing good reconstructions.

Currently, one limitation of the system lies in the extraction process: we have found that one of the biggest problems in getting a good re-application of a style is the smoothness of the base path extracted from the filters. As shown in Figure 5.2, while the best filters result in only the broadest features remaining, there may be significant base path variations left after a filter has reached its minimum width. If the base path is not smooth then too much detail is lost when the re-application replaces the base points, leading to a distorted style with a barely recognizable look. This problem also tends the system toward capturing the local details of curve variations better than larger features, as these macro-level variations are usually extracted as part of the curve's base path. These then end up being replaced and lost when the style is reconstructed on the new base. The availability of multiple filters and the interactive preview within the style extractor mitigates this by allowing the user to quickly determine if there will be an acceptable extracted style or not. We have often found that such problems can be rectified by using a global filter and a style curve with at least several hundred points, such as from a longer sketch or a high resolution scan. This generally results in the base path being smooth enough for successful style reconstruction. In the absence of filters with shorter minimum widths, one possible automatic solution to this problem is *resolution enhancement* of the final base path. Repeatedly resampling the points of the final path "up" a level, and then decomposing again would allow extraction of more details. Curvature, or another measure of smoothness, such as the magnitude of the extracted details, would determine when this process was no longer progressing and could be stopped.

Another area where we would like to extend the system involves automating the finding of seamless repetitions for captured styles, currently an interactive, user-driven task. A heuristic for determining an overlap distance for style blending that results in a seamless repetition would allow this. And, while we have not experimented with it, an obvious future direction for our blending solution for style repetition is to allow the joining of different styles. This would necessitate the a more sophisticated method of matching detail vectors. One possible approach is to re-parameterize those within the overlap region, guaranteeing a match at each reconstruction level.

The use of flags for discontinuous styles presents several other avenues as well. Our current rendering technique uses a fixed threshold of 0.5, an experimentally derived value. There is further opportunity to refine this value, or determine a dynamic threshold from the input curve that would maximize the number of levels that allow good reconstruction.

Another area to pursue lies in generalizing the flag technique for allowing discontinuous styles into other display-related attributes. Capturing line qualities such as weight or thickness, colour, and stroke rendering brush styles would enhance our interactive illustration system by allowing more expressive final renderings. For example, by using brush-stroke styles to render the final curves, creating painterly styled outputs.

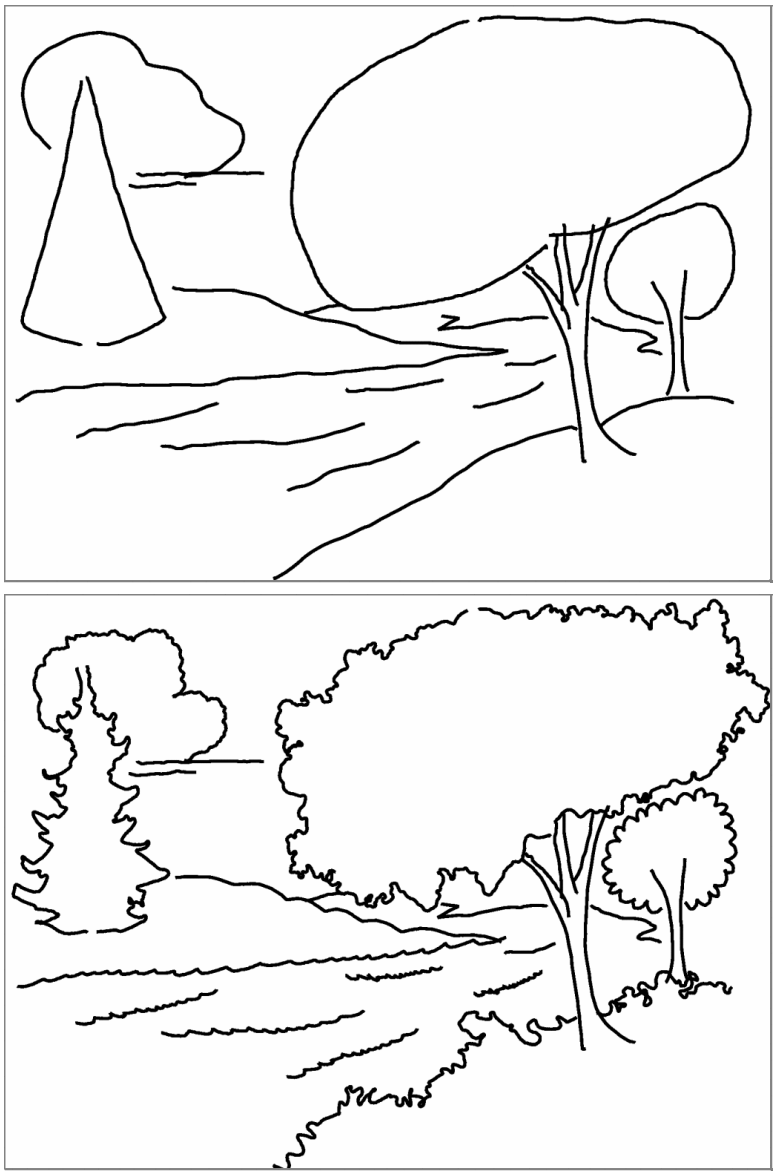


Figure 8.1: Composition produced with our system. Top: roughly-sketched base curves. Bottom: result of applying the captured styles from the cloud and bush of Figures 1.2 and 7.3; the foliage of Figure 7.4, and using sketched styles for the water, far shoreline, and small tree.

Bibliography

[Act97] Acton M.: *Learning To Look At Paintings*. Routledge, New York, 1997.

[BGS06] Bartels R. H., Golub G. H., Samavati F. F.: Some observations on local least squares. *BIT Numerical Mathematics Journal* (2006).

[BLCD02] Bregler C., Loeb L., Chuang E., Deshpande H.: Turning to the Masters: Motion Capturing Cartoons. In *Proceedings of ACM SIGGRAPH 2002* (July 2002), pp. 399–407.

[BS00] Bartels R. H., Samavati F. F.: Reversing subdivision rules: local linear conditions and observations on inner products. *Journal of Computational and Applied Mathematics 119*, 1–2 (2000), 29–67.

[CAS*97] Curtis C. J., Anderson S. E., Seims J. E., Fleischer K. W., Salesin D. H.: Computer-generated watercolor. *Computer Graphics 31*, Annual Conference Series (1997), 421–430.

[Coc91] Cockshott T.: *Wet and Sticky: A Novel Model for Computer-Based Painting*. PhD Thesis, University of Glasgow, 1991.

[Cra00] Crane W.: *Line & Form*. George Bell & Sons, London, 1900.

[DHvOS00] Deussen O., Hiller S., van Overveld C., Strothotte T.: Floating points: A method for computing stipple drawings. *Computer Graphics Forum 19*, 3 (2000), 41–50.

[Ebe05] Eberly D.: *Magic Software*. http://www.magic-software.com, (Web site), 2005.

[FS94] Finkelstein A., Salesin D. H.: Multiresolution Curves. In *Proceedings of ACM SIGGRAPH 94* (July 1994), pp. 261–268.

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

[GG01] Gooch B., Gooch A.: *Non-Photorealistic Rendering*. AK Peters, Natick, MA, USA, 2001.

[GW87] Gonzalez R. C., Wintz P.: *Digital Image Processing*, 2nd ed. Addison-Wesley, Don Mills, Ontario, 1987.

[Hae90] Haeberli P.: Paint by numbers: abstract image representations. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques* (1990), ACM Press, pp. 207–214.

[HB97] Hearn D., Baker M. P.: *Computer Graphics*. Prentice Hall, New Jersey, USA, 1997.

[HOCS02] Hertzmann A., Oliver N., Curless B., Seitz S. M.: Curve Analogies. In *Proceedings of the Thirteenth Eurographics Workshop on Rendering* (2002), pp. 233–246.

[JEGPO02] Jodoin P.-M., Epstein E., Granger-Piché M., Ostromoukhov V.: Hatching by Example: a Statistical Approach. In *Proceedings of*

NPAR 2002 (June 2002), pp. 29–36.

[KGC00] Kaplan M., Gooch B., Cohen E.: Interactive artistic rendering. In *Proceedings of NPAR 2000* (2000), pp. 67–74.

[KMM*02] Kalnins R. D., Markosian L., Meier B. J., Kowalski M. A., Lee J. C., Davidson P. L., Webb M., Hughes J. F., Finkelstein A.: WYSIWYG NPR: Drawing Strokes Directly on 3D models. In *Proceedings of ACM SIGGRAPH 2002* (2002), pp. 755–762.

[Lit97] Litwinowicz P.: Processing images and video for an impressionist effect. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (1997), ACM Press/Addison-Wesley Publishing Co., pp. 407–414.

[Nic41] Nicolaides K.: *The Natural Way to Draw*. Houghton Mifflin, Boston, 1941.

[Pav82] Pavlidis T.: *Algorithms for Graphics and Image Processing*. Computer Science Press, Rockville, MD, 1982.

[PTVF92] Press W. H., Teukolsky S. A., Vetterling W. T., Flannery B. P.: *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1992.

[Rei87] Reid G. W.: *Landscape Graphics*. Watson-Guptill Publications, New York, 1987.

[SB99] Samavati F. F., Bartels R. H.: Multiresolution Curve and Surface Representation: Reversing Subdivision Rules by Least-Squares Data Fitting. *Computer Graphics Forum 18*, 2 (June 1999), 97–119.

[SB04] Samavati F. F., Bartels R. H.: Local Filters of B-spline Wavelets. In *Biometric 2004* (2004).

[SD04] Simhon S., Dudek G.: Sketch interpretation and refinement using statistical models. In *Proceedings of the Eurographics Symposium on Rendering '04* (2004).

[SDS96] Stollnitz E. J., DeRose T. D., Salesin D. H.: *Wavelets for Computer Graphics: Theory and Applications*. Morgan Kaufmann, San Francisco, Calif., 1996.

[Sec02] Secord A.: Weighted voronoi stippling. In *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering* (2002), ACM Press, pp. 37–43.

[SMGG01] Sloan P.-P., Martin W., Gooch A., Gooch B.: The Lit Sphere: A Model for Capturing NPR Shading from Art. In *Proceedings of Graphics Interface 2001* (June 2001), pp. 143–150.

[Spe72] Speed H.: *The Practise and Science of Drawing*. Dover Publications, New York, 1972.

[SS02] Strothotte T., Schlechtweg S.: *Non-Photorealistic Computer Graphics*. Morgan Kaufmann, Boston, MA, USA, 2002.

[Sul97] SULLIVAN C.: *Drawing the Landscape*, 2nd ed. Van Nostrand Reinhold, Toronto, 1997.

[Whi94] WHITAKER S.: *The Encyclopedia of Cartooning Techniques*. Running Press, Philadelphia, 1994.

[WS94] WINKENBACH G., SALESIN D. H.: Computer-generated pen-and-ink illustration. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques* (1994), ACM Press, pp. 91–100.

Appendix A

System Description

This appendix illustrates the use of the programs for capturing and re-using a scanned style. The system consists of two main programs: the style extractor, `newstyles`, and the style applier, `applymrd`. Both are interactive, graphical, user-oriented applications which use a common library of multiresolution classes for creating and manipulating filter bank libraries. Additionally, a non-interactive command-line program, `wmskel`, is used to automatically vectorize raster images for use within the style extractor.

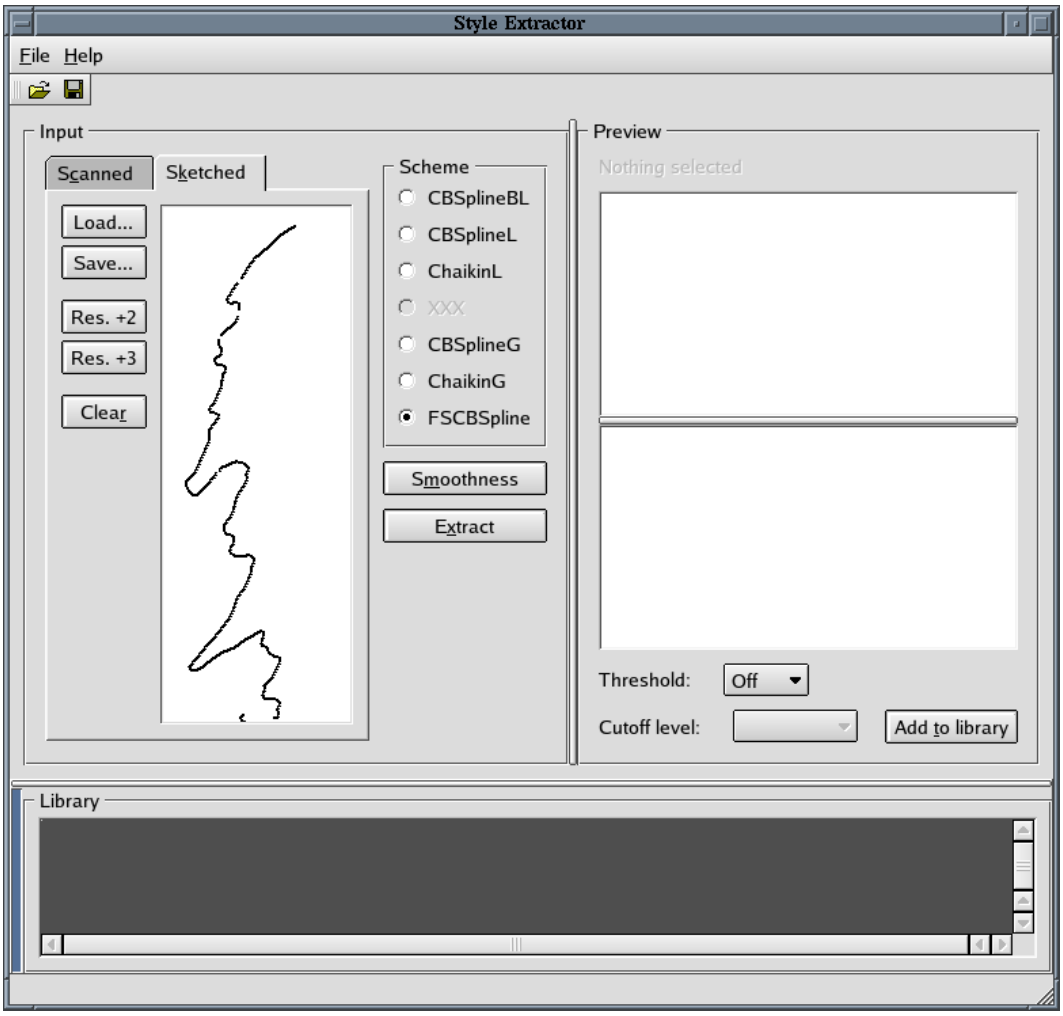


Figure A.1: In this view of the style extractor, a style curve has been loaded into the "Sketched" input tab. The available multiresolution schemes are shown beside it and at far right the extracted levels and a preview of the style application will be displayed. The filter bank library area is below.

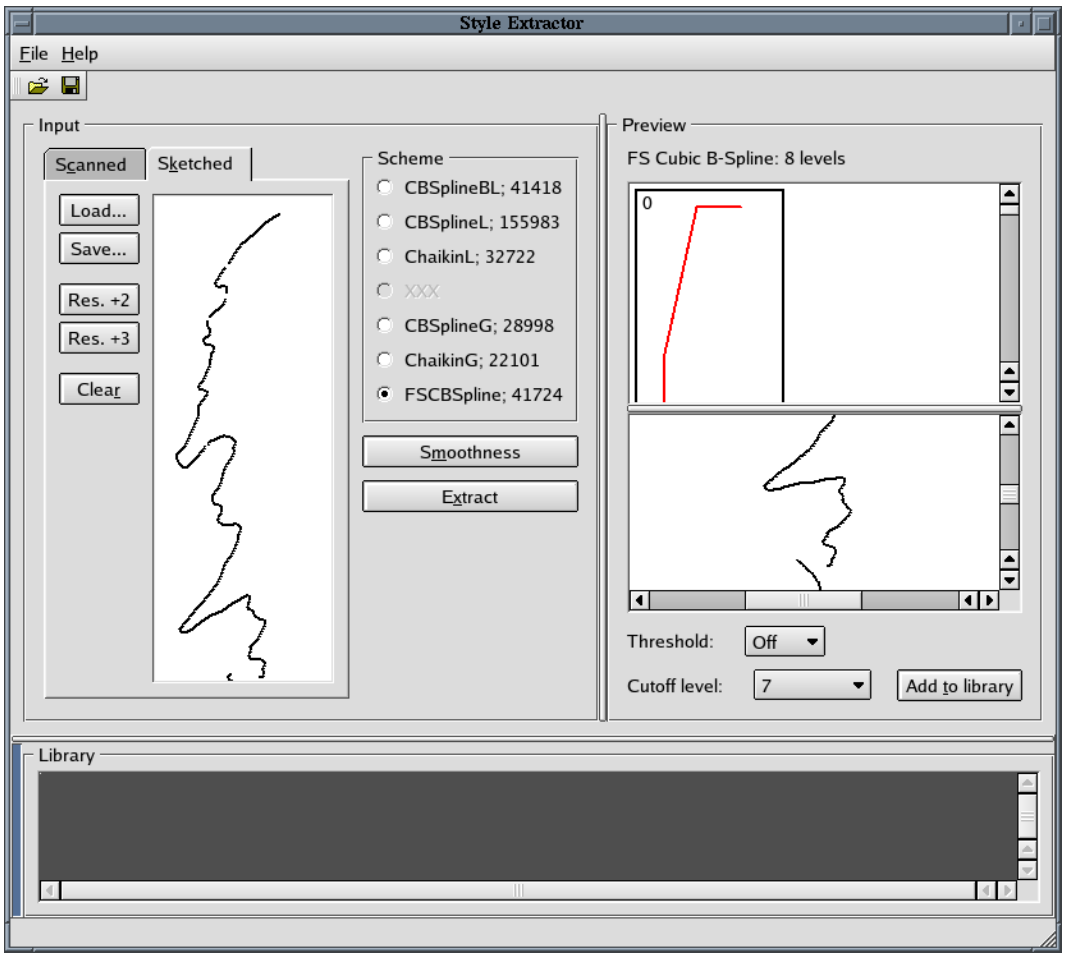


Figure A.2: After extraction the levels of decomposition are displayed, with the base path shown first, at top right. A sample application of the style is shown by automatically rebuilding it upon a simple vertical or horizontal path below. This provides a quick preview as to the quality of the extraction.

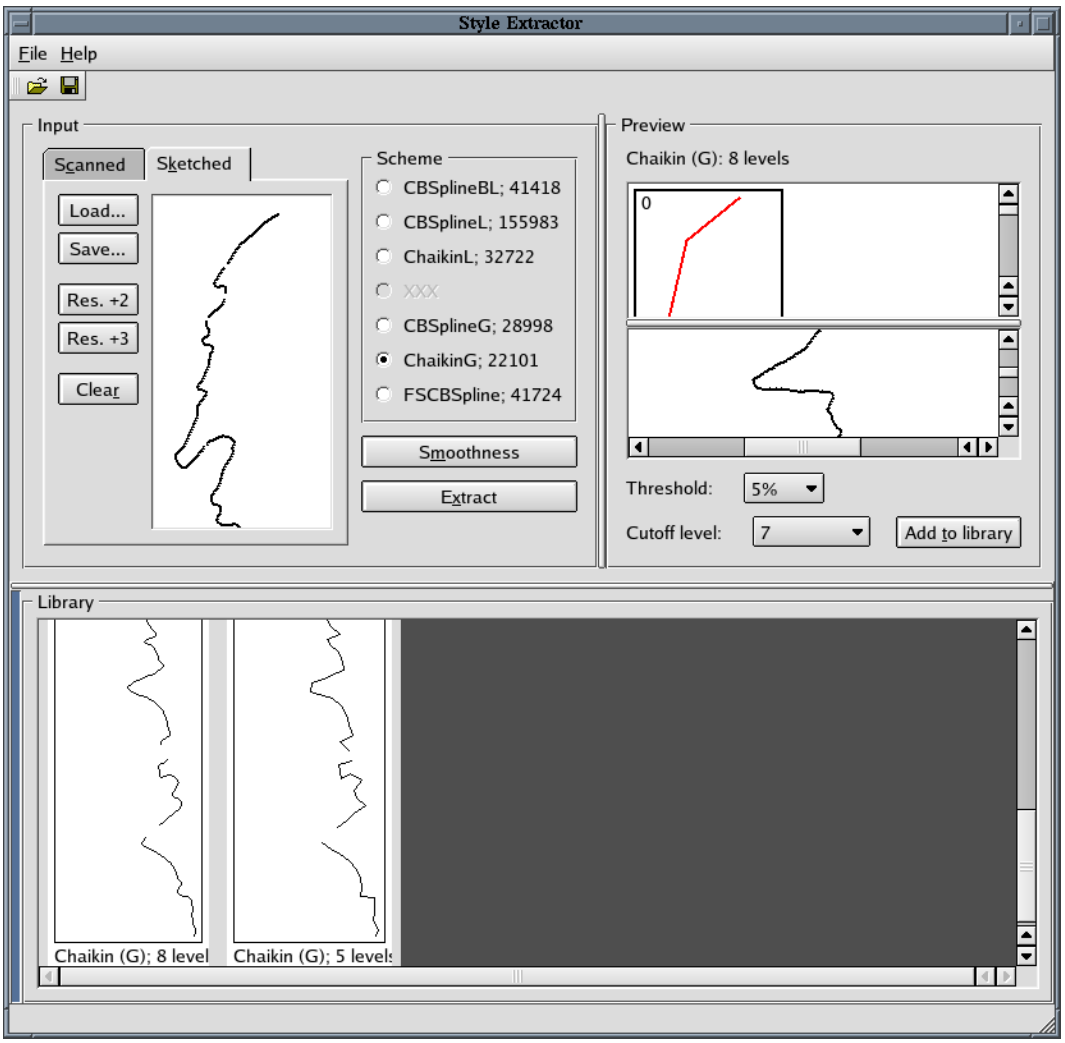


Figure A.3: Once an acceptable filter scheme has been determined, the highest detail levels of the style can be dropped and the result added to the current library of filter banks. The "Threshold" and "Cutoff level" options specify how many of the extracted levels to discard before adding.

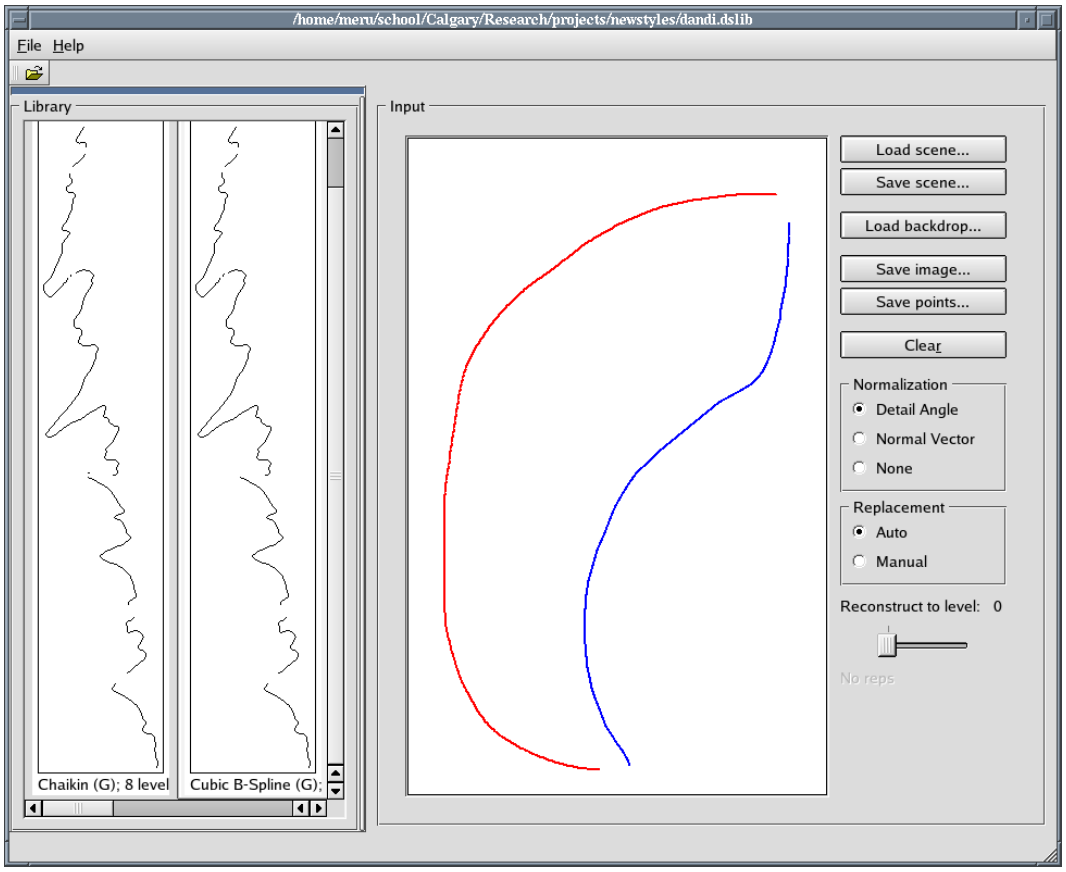


Figure A.4: In the style applier, we load the saved filter bank library (left) and sketch new base paths for the captured styles (right). At far right, options for saving the drawn scene and manipulating the applied style are available.

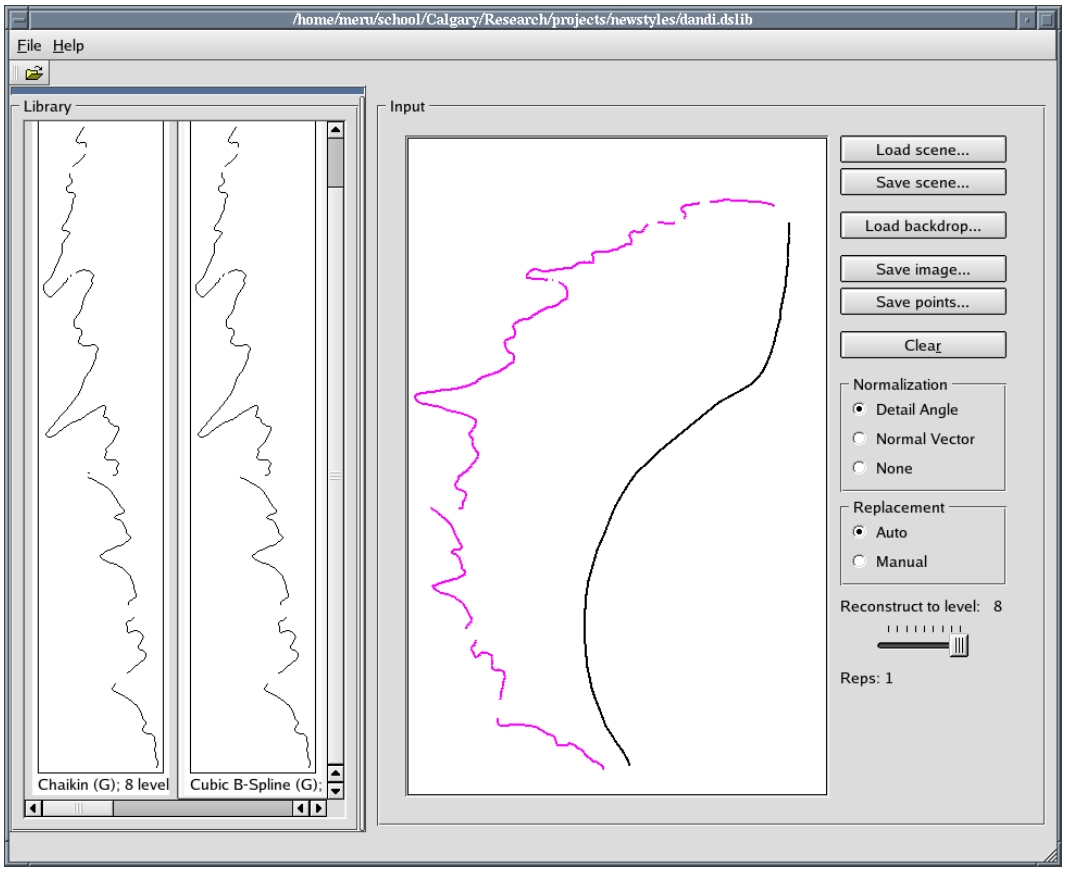


Figure A.5: We select a base curve using the right mouse button, and then click on an item in the filter bank library pane to attach that style to the curve. Once applied, the options for changing the number of style repetitions and number of reconstruction levels are enabled. The style can be changed by clicking any of the available extracted styles in the currently-loaded filter bank library. The multiresolution editor for the current base curve is invoked by pressing "e."

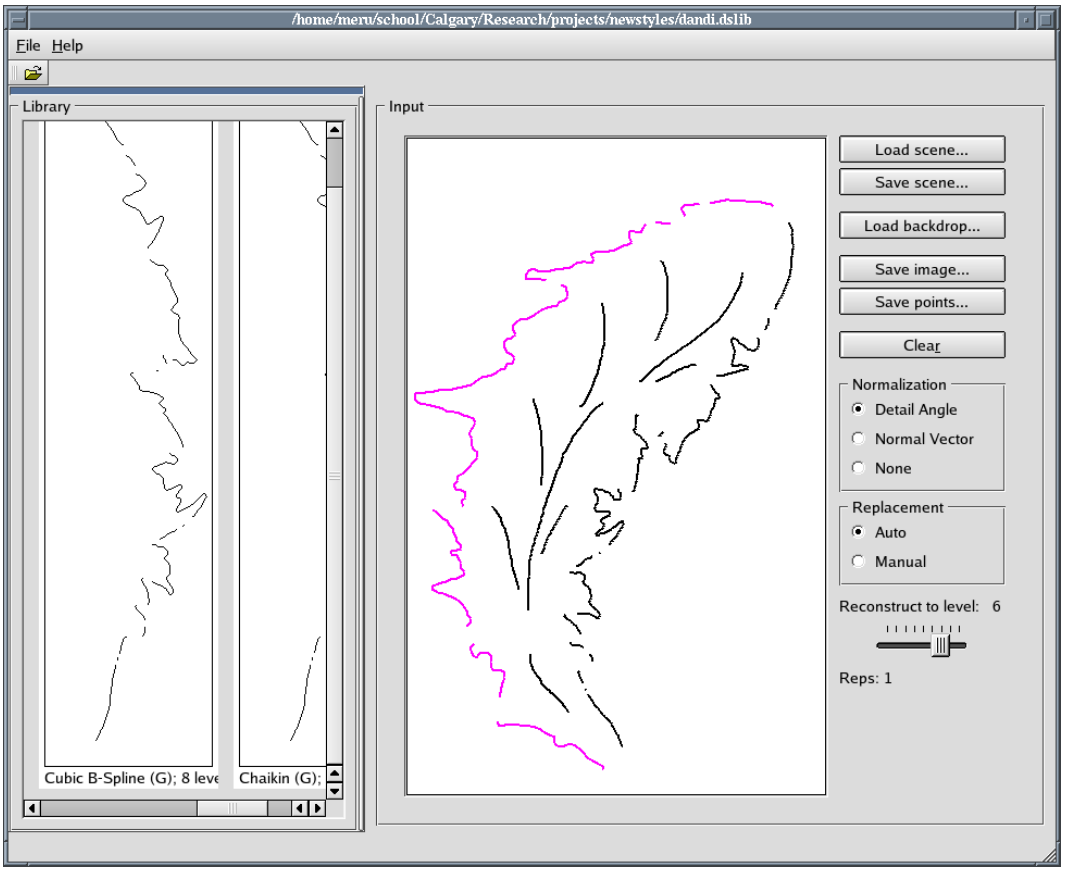


Figure A.6: After the base curves and their styles are acceptable, additional non-styled strokes can be added to the illustration for extra detail. The result can be saved in an output-oriented vector or raster format, or as a "scene" file that can be re-loaded and edited in a subsequent session.