Embroidery Modelling and Rendering

by

Xinling Chen

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2011

Abstract

Embroidery is a traditional non-photorealistic art form in which threads of different colours stitched into a base material are used to create an image. This thesis presents techniques for automatically producing embroidery layouts from line drawings and for rendering those layouts in real time on potentially deformable 3D objects with hardware acceleration. Layout of stitches is based on automatic extraction of contours from line drawings followed by a set of stitch-placement procedures based on traditional embroidery techniques. Rendering first captures the lighting environment on the surface of the target object and renders it as an image in texture space. Stitches are rendered in this space using a lighting model suitable for threads at a resolution that avoids geometric and highlight aliasing. It is also possible to render stitches in layers to capture the 2.5D nature of embroidery. A filtered texture pyramid is constructed from the resulting texture and applied to the 3D object. Aliasing of fine stitch structure and highlights is avoided by this process. The result is a realistic embroidered image that properly responds to lighting.

Acknowledgements

First I would like to thank my supervisor Professor Michael McCool for suggesting this topic to me and generously offering ideas, guidance, and support during the past two years. I would also like to thank my co-supervisor Professor Stephen Mann for providing ideas to improve this work and helping me with editing the thesis again and again with patience. To my readers, Professor Craig Kaplan and Professor George Labahn, thanks for your time and valuable opinions.

Thanks to all the CGL members, especially to Eugene for offering advice and discussing with me about the implementation of the surface space rendering with OpenGL API, when I was frustrated, getting nowhere...

Thanks to my boyfriend, Naoki who encouraged me to take this project and has been there for me when I lost confidence and direction.

In the end, to my parents, who bought a computer home when I was 11 and helped me develop a strong interest in video games and computer graphics ever since then, thanks for understanding and supporting me to pursue computer graphics for a Master's degree.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Many rendering algorithms have been developed to simulate different traditional artistic styles including painting, mosaics, and stippling. However, there is at least one traditional approach to rendering images non-photorealistically that has not been simulated: embroidery. Embroidery creates images by stitching threads of different colours into a base cloth. There are two kinds of embroidery. One is machine embroidery, which we can easily find in real life, Figure 1.1 shows a sample of machine embroidery on a cotton T-shirt. The other is freehand embroidery, which can use a wider variety of thread thicknesses, and can also use one kind of thread to modulate the effect of another. Traditional embroidery is an art form whose practitioners use a free combination of various styles of stitching techniques to achieve specific effects (Figure 1.2).

This research project attempts to emulate the ancient art form of free hand embroidery. The goal is to transform a traditional embroidery pattern (typically consisting mainly of a line drawing of region outlines) into a high-quality embroidery image generated in real time. Real time rendering enables the user to navigate around the image and manipulate it. We also want to be able to place the embroidery on curved 3D objects and use realistic and dynamic lighting models for the threads.

1.1 Challenges

The nature of embroidery and the difficulty of rendering thin objects poses challenges for this research in four ways.

First, we should respect the conventions of traditional embroidery. For example, unlike expressive painting, where boundaries are often not well-defined, due to the discrete nature of the rendering primitive (the stitch), embroidery normally clearly separates regions from

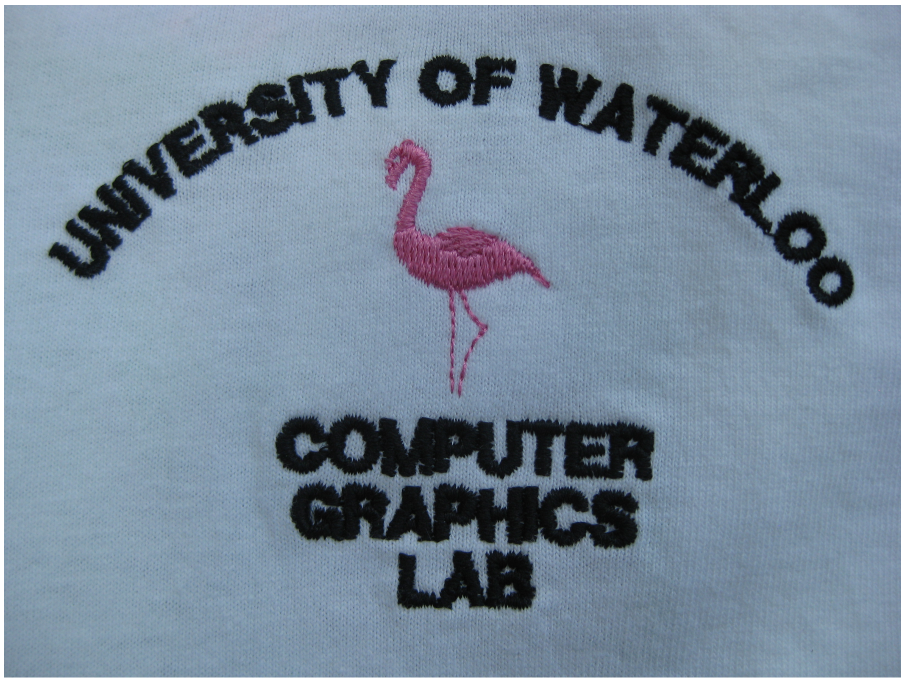

Figure 1.1: *A sample of machine embroidery on a cotton T-shirt.*

each other, and often uses explicitly rendered boundaries. Many hand embroiderers make each leaf and petal stand out distinctly by outlining edges so that they will not seem to run into each other, which raises the work and develops its highlights and shadows better than a flat treatment. We also want to drive the embroidery modelling process from computer vision analysis of traditional drawn patterns, with limited human input. In other words, we want to start from the same input that a human embroiderer would use. As part of this process we need to segment the input pattern into regions and boundaries and treat them separately. However, closed regions are not universally used in embroidery, and unclosed lines and "fuzzy" boundaries sometimes do appear. This makes region labelling more challenging. For example, in Figure 1.3, an embroidery pattern attributed to Kurenai, a prestigious embroidery organization in Japan, demonstrates this problem. In Figure 1.4, there are many small details, such as leaf veins or stamens, that are difficult to label.

Second, multiple stitching techniques need to be studied and simulated, since in practice many stitching styles are used to achieve different artistic effects, just as different types of brush strokes are used in painting or calligraphy. Each stitching style has various parameters that need to be estimated, and the stitching style used for a particular region needs to be selected.

Third, the orientation of stitches needs to be considered. Each stitch should align with each other in an orderly fashion, usually without overlaps but also without significant gaps.

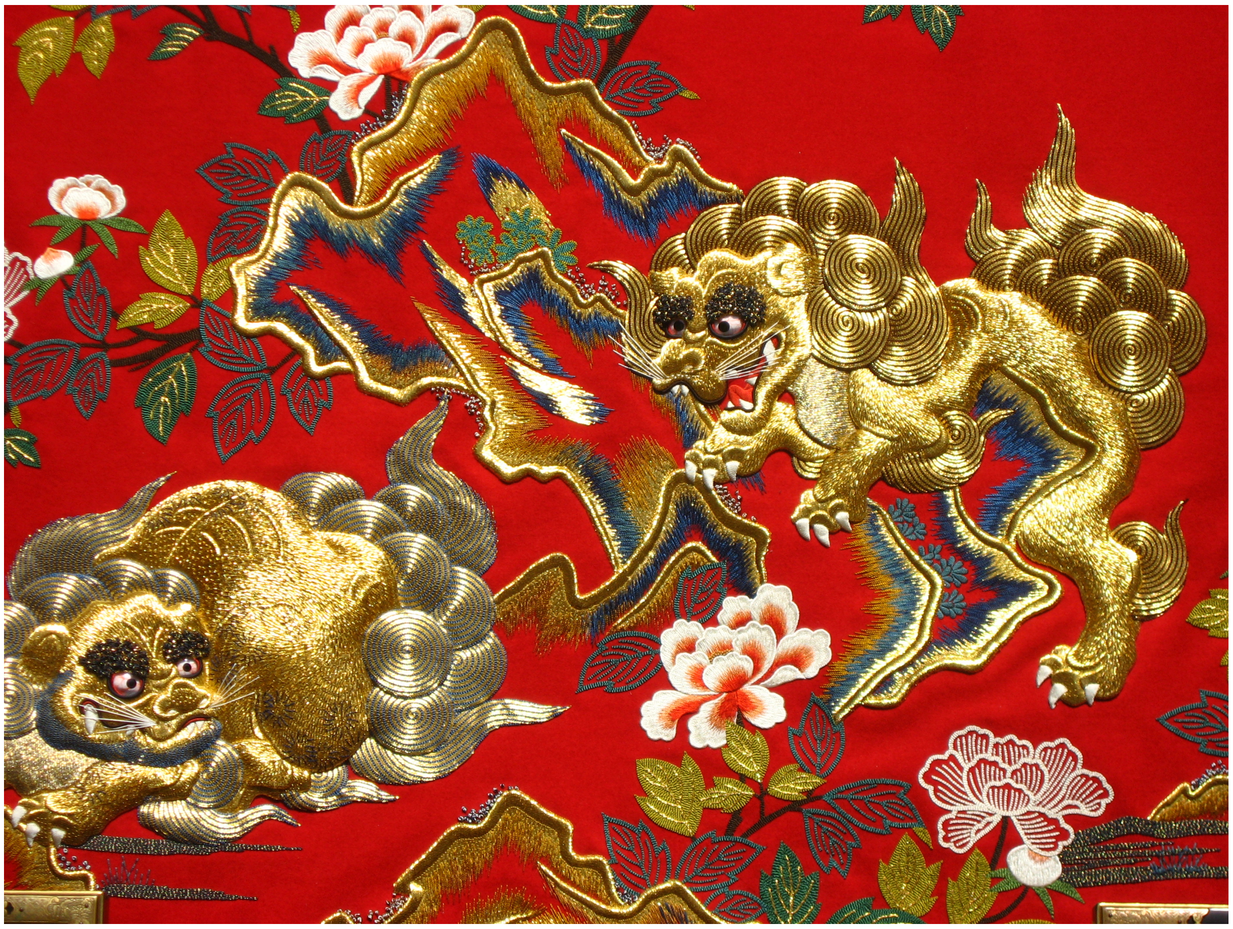

Figure 1.2: *A sample of traditional Japanese hand embroidery, employing a range of stitching styles. Image made available under Creative Commons license.*

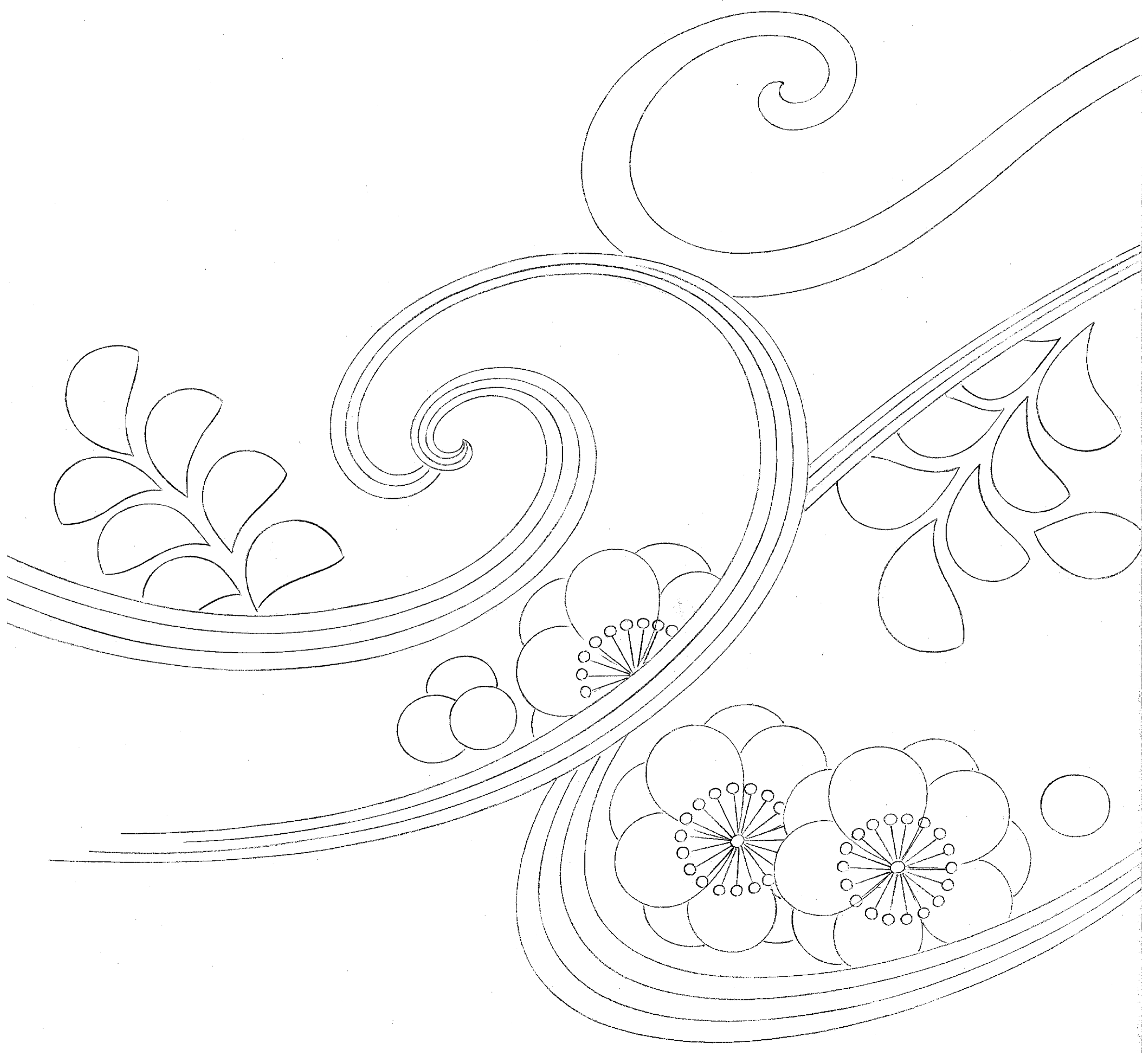

Figure 1.3: *A traditional embroidery pattern specified as a line drawing.*

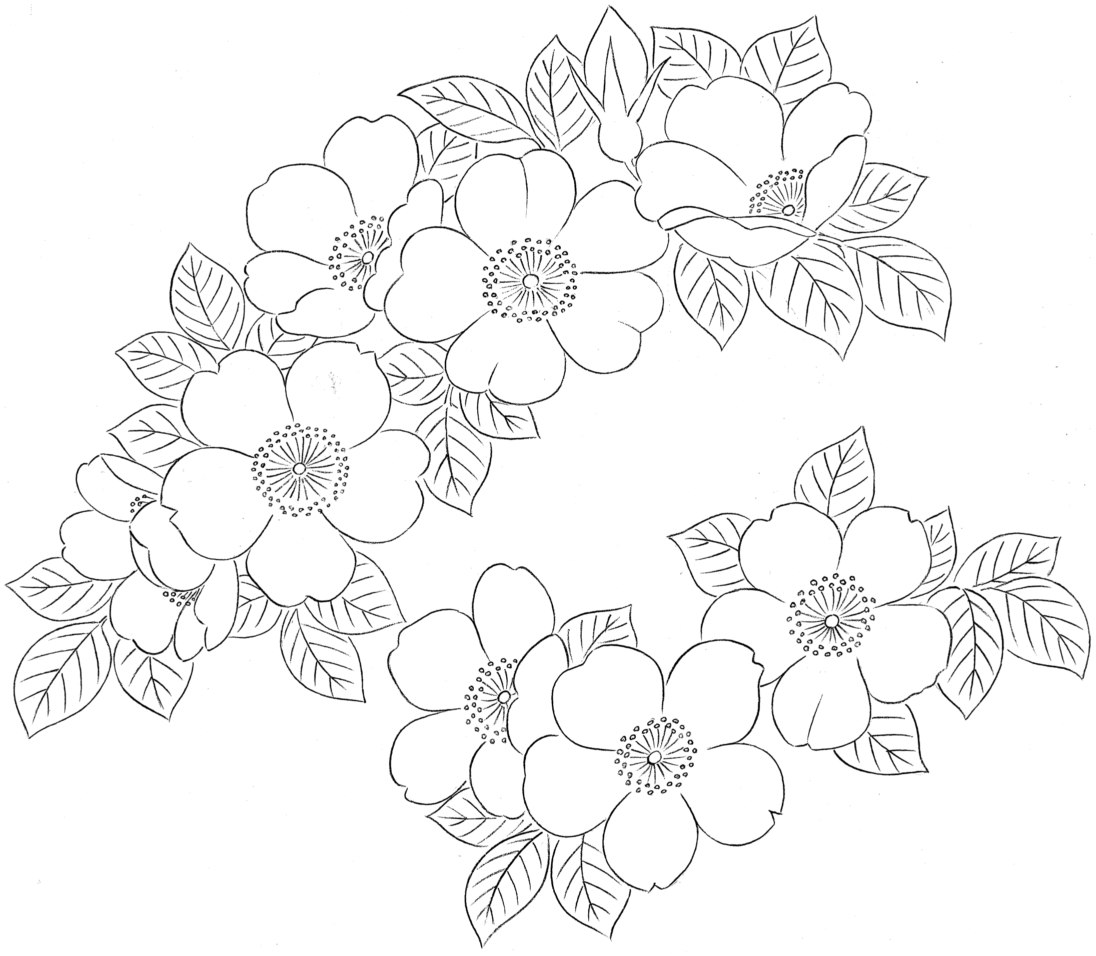

Figure 1.4: *A traditional embroidery pattern specified as a line drawing. How to properly treat the non-region-outline details in the pattern such as the stamens of flowers or veins of leaves is an interesting question.*

The way that light reflects off threads makes the orientation of the threads clearly visible, even from a distance, and the placement of the stitches lends a different texture and overall appearance to different stitching styles. Moreover, each style has its own rules regarding orientation. For example, the style called "separated layer", which is often used to decorate leaf-like shapes, requires stitches placed roughly 30 degrees away from central line of the region. This requires finding and defining the "central line" and dealing with often curved central lines.

Fourth, there are several rendering challenges. It is challenging to render embroidery well as the fine structure tends to alias, and there are potential performance problems due to the large number of primitives involved in embroidery. Also, there are many other minor details that affect the final look of the embroidery. For example, a stitch's end points exit and enter the basic fabric, which generates a curve and modifies the tangent of the stitch and in turn affects the lighting of the stitch. We do not handle all such details in this work—several details remain for later work, such as interstitch shadowing and other interstitch modelling dependencies—but we have attempted to handle enough for a convincing reproduction of the essential look of embroidery.

Some hobbyists are still doing craft projects for fun. However, such personal sewing usually lacks the variety of stitches that appeared in traditional embroidery. It would be meaningful if we could apply computer techniques to present this traditional art form and propagate it to the public. Also, it would be a useful system for previewing embroidery models during the design of embroidery.

This work has a practical application in the animation of characters in historical costume. In such an application, simulated embroidery could be used to add realism. One could attempt to scan an embroidery work and texture map it on a costume. However, using just a single image will not capture the true look of embroidery with its complex shifting highlights; the look of embroidery is a complex function of view direction and light direction. It is necessary to capture the light field which is four dimensional function [18]. Figure 1.5 shows that real embroidery looks different from different angles. Also, embroidery is high-frequency, so it would require a high sampling rate, and hence scanning it properly would be very expensive. Even if this worked, if we wanted to use a new embroidery pattern that has no physical embroidery work, we would need some embroiderers to create it manually.

1.2 Related Work

Distributing objects on the plane is not a new topic in the field of non-photorealistic rendering. Much work has been done to develop effective methods to imitate what the artist does to decorate a plane in various art styles using discrete rendering primitives.

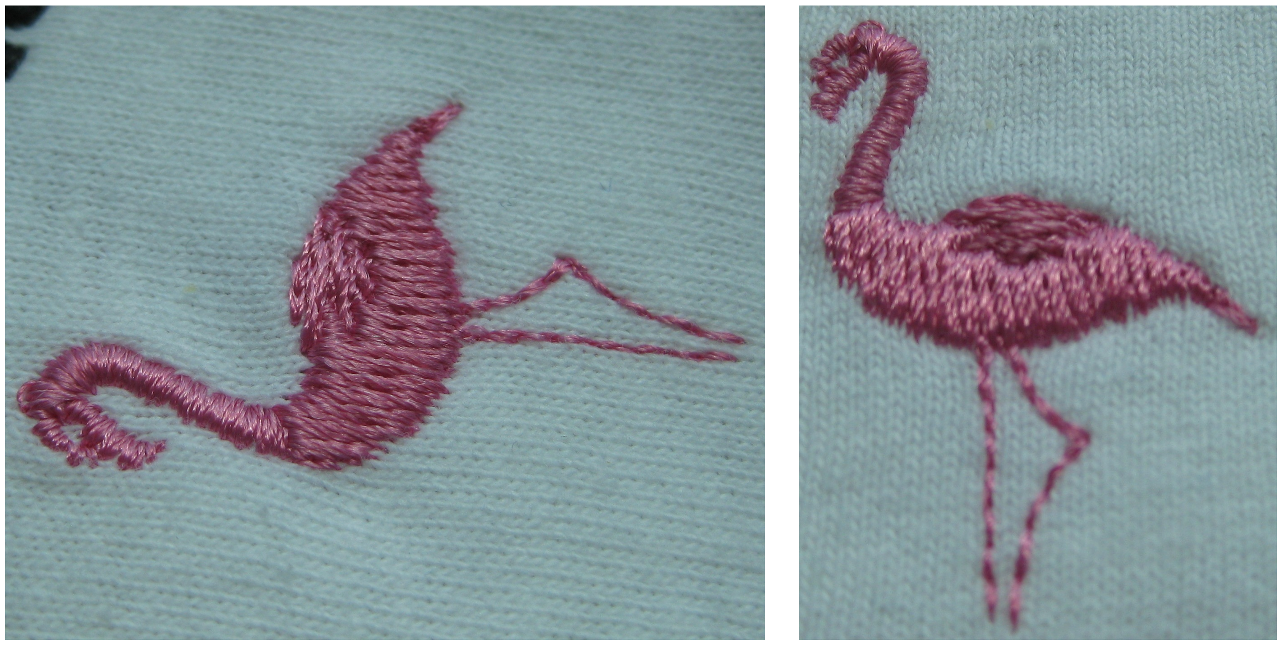

Figure 1.5: *Comparison of two images of the same embroidary from different viewpoints. For the image on the left, the wing is much brighter than the body of flamingo. For the image on the right, the light direction relative to the camera remained the same, but the sample was rotated 90 degree. The wing turned dark while the body turned bright.*

Deussen et al. [4] and Secord [25] generated attractive stippling drawings from images, while Winkenbach et al. [29, 30] and Salisbury et al. [23] generated pen-and-ink effects using stroke textures. Hausner [9], Elber et al. [5] and Kim et al. [14] explored methods to simulate mosaics.

There is no previous work on simulating traditional embroidery in 3D computer graphics, which involves both stitch placement and rendering. However, we can relate the problem of stitch placement to mosaic packing. Previous work dealing with mosaic simulation [9, 14, 5] assumed a color photograph as input and required the user to specify segmentations and feature curves, roughly guiding the orientation of tiles placed on the whole image. However, in the case of embroidery, the input image is usually a line drawing such as in Figure 1.3 or Figure 1.4. Given such an input, manual segmentation is a tedious, time-consuming task. Moreover, boundaries and regions should be treated separately, while normally the background in the pattern should be left untouched. It is also considered less important to cover the background completely in most styles of embroidery than in mosaics, and traditional stitch styles may add additional constraints. For example, the classic fill stitch, long-short stitch always starts with alternating half-long and full-long stitches in the first column, and then alterates on later columns (Figure 3.1). Also, primitives in embroidery are long and slender rather than compact as in mosaics, and can also vary relatively freely in length. These factors make mosaic placement algorithms not entirely suitable for this problem, since general packing may or may not generate an alternating structure with well-oriented primitives. *Traditional Japanese Patterns* [15] defines a set of sewing techniques used in free embroidery for Japanese embroidery. Other embroidery traditions define other embroidery styles.

A method of relaxation called Lloyd's method [21] is often used in mosaic and stippling methods. A variation of this approach places primitives using cellular automata based on partial differential equations [6]. The latter work also discussed the possibility of modeling a variety of surface details by designing a set of rules to control the growth of geometric elements. We experimented with this method but it failed to produce satisfactory stitch placements. In particular, it failed to preserve or generate the characteristic regularities that appear in traditional stitching styles.

Embroidery modelling also can be approached from another perspective: it is essentially a process of adding color and stitches to segmented regions of a monochrome image specified as a line drawing. In this sense, it is similar to colorization, which typically also involves segmenting images into regions. Existing colorization methods mainly rely on intensity continuity [17, 27] or more generally pattern-continuity [22] to segment the image into color regions. In recent work on Manga colorization [22], a texture-based level set method is applied to segmentation. However, the level set method performs well only over disjoint closed regions. In the case of embroidery patterns, overlapped regions and unclosed curves are quite common, which makes general region labelling methods fail. In this thesis, we propose a segmentation solution that can handle this type of input.

To produce realistic results, the embroidery threads also need to be lit correctly. Many lighting models for thin objects and fur have been developed [13, 12, 20, 1]. For simplicity and performance we use the lighting model presented by Mallo et al. [19], since it is efficient and generates reasonably realistic results, although a more sophisticated model may be needed if more realistic renderings are desired. In addition, to achieve high-quality results, it is essential that we avoid aliasing. It is useful but not sufficient to antialias each stitch separately since when viewed from a distance the stitches might suffer from both geometric aliasing (stitches falling between pixel centers and disappearing in a hit-and-miss fashion) and highlight aliasing (sharp highlights falling between pixel centers), so we also use hardware-accelerated texture antialiasing of the entire lit pattern of stitches. We render the lit embroidery in a flat texture space and at a resolution that avoids these problems, and handle further minification in image space.

1.3 Contribution

The main contribution of this thesis is a set of techniques for modeling and rendering traditional embroidery directly from standard drawn embroidery patterns. Our system takes a raster scan of an embroidery pattern in the form of a line drawing and automatically analyzes and extracts the boundaries and regions in this pattern. Further, an interactive interface is used to allow modification of the color and stitching style for each region, and generally to allow artistic control. Finally, the generated embroidery model can be rendered realistically in real time on a plane or on a 3D object. Particularly, our results capture the orientation-dependent interaction of the embroidery with the light (Figures 6.4 and 6.8). While we do not yet approach the sophistication of the best examples of traditional embroidery (as in Figure 1.2) our framework is extendable.

1.4 Framework

Given a pattern specified as a line drawing, human embroiderers usually choose a region, decide on a stitching style for this region, and select the thread color and thickness before starting to place stitches on the base cloth. Some regions might be left blank or half decorated, but more often than not, all the lines and contours on the pattern will be treated with stitches. Our approach to free embroidery simulation is based on this process.

As Figure 1.5 shows, free embroidery simulation based on traditional patterns can be broken into three phases. My simulation uses the same three phases.

First, we segment the input image of a pattern (such as in Figure 1.2) into a list of boundaries and a list of regions. Since in traditional embroidery different regions are often treated with different colors and stitching techniques, it is indispensible to divide the pattern into regions and label each region with a distinct identification number for further selection and treatment. Since embroiderers also usually make outlines distinct, boundaries in the pattern also need to be extracted.

Second, stitches are distributed in the identified regions. In this step, algorithms are developed to simulate different stitching styles to treat the boundaries and to fill the interiors of regions. An interactive interface is available for the user to decide each region's color and stitching style or leave it blank. These parameters are not specified in the input pattern. So we either have to infer them or seek user input.

Third, the generated model is rendered in real-time. A thread lighting model is selected to light the generated embroidery model in 3D. We use simple proxy geometry for each stitch, but techniques such as alpha mapping and bump mapping are used to add visual complexity at the mesoscale and in particular to generate the illusion that the stitch is curved in depth and that the two end points of each stitch enter into the base fabric.

These phases will be discussed sequentially in later chapters.

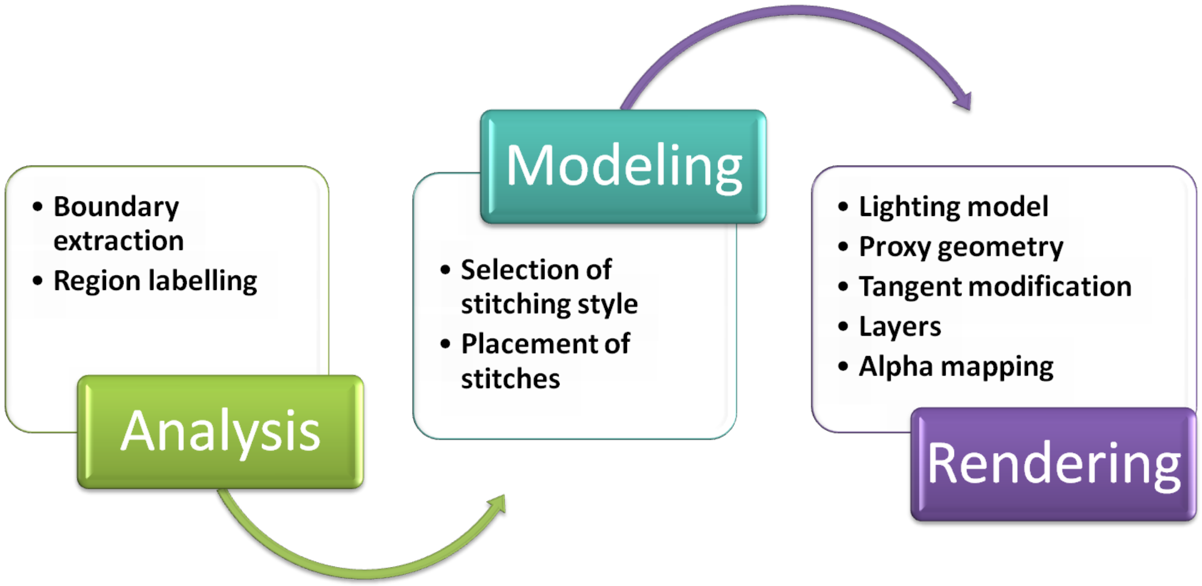

Figure 1.6: *Software architecture of our prototype embroidery system, which is broken into three phases.*

.

Chapter 2

Input Image Analysis

To analyze traditional embroidery patterns, we need to extract boundaries and distinctively labelled regions from scanned input images.

Essentially we applied a connected region labelling algorithm [26] to distinctly label each region. This algorithm has two passes. The first pass records equivalences and assigns temporary labels, while the second pass replaces each temporary label by the label of its equivalence class. However, this method fails to identify nested regions. Therefore we developed our own boundary tracking algorithm to label nested regions. In this chapter, I will first discuss region labelling algorithm and then the edge tracking method.

2.1 Region Labelling

The method we used is based on connected region labelling with 8-connectivity [24]. In a first pass, a temporary label is assigned to each pixel based on its neighbors above and the one neighbor to the left. In a second pass, adjacent temporary labels are merged into a final label for the region (Figure 2.2).

In the first pass, the image is scanned from top-left to bottom right row by row. For each white pixel, the algorithm looks at its North-East, North, North-West and West neighbouring pixel's id number, and assigns the smallest label among them to the current pixel (Figure 2.1, left). If these four pixels are black, then assign a new sequential ID number to this white pixel because it might belong to a new region. For example, if we had assigned label 1 to the previous connected white pixels and meet another pixel that has four neighbouring black pixels, then we increment the current label to 2 and assign it to this pixel.

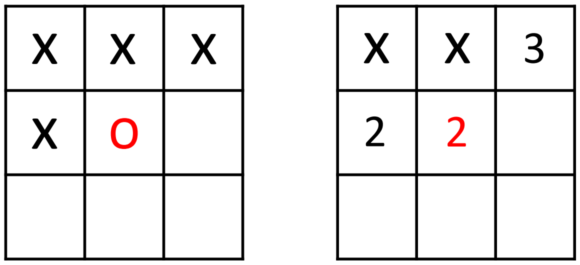

Figure 2.1: *Left: 8-connectivity: checking the labels of pixels that are North-East, North, North-West and West of the current pixel (red). Assign the minimum label among these four labelled pixels to the current pixel. Right: Store the connected area in a lookup table for further update. In this example, the current pixel (in the center) in fact serves as a bridge that connects Region 3 and Region 2. We store the equivalence information during labelling, then update all these connected labels as identifying the same region in a post-process.*

During the first pass, a lookup table is kept to store the equivalence between connected labels. For instance, in the diagram on the right of Figure 2.1, when the algorithm labels the pixel in the centre as 2 because the minimum of 2 and 3 is 2, the algorithm stores the information that Label 2 and Label 3 are connected in the lookup table.

During the second pass, the algorithm uses the lookup table to find all the connected areas and then updates each pixel's id number to the minimum label in each equivalence class. For example, if Label 2 and Label 3 are connected, and Label 3 and Label 4 are connected, then the pixels labelled as 3 and 4 are all connected to Region 2, and hence are updated as area 2. In this way, we can divide the whole image into a list of connected regions. Distinct id numbers for regions makes it easier to specify particular operations on a certain area.

The trouble with the above method alone is that when there are regions nested inside one another, this method fails to differentiate regions from background. Since regions are surrounded by boundaries, we extract the closed boundaries first to provide this information for region labelling as discussed in the following section.

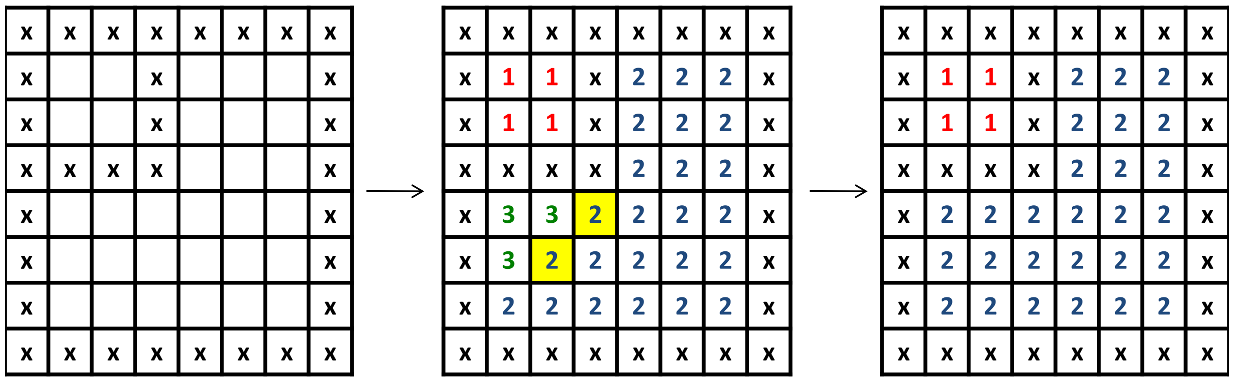

Figure 2.2: *This figure illustrates the process of connected region labelling. The figure in the left is an unlabelled image. The first pass assigns temporary labels to each pixel, and keeps a look-up table to store the equivalence between connected labels. The second pass updates all these connected labels.*

2.2 Boundary Extraction

To extract edges, the first algorithm that comes to mind is Canny edge detection [3]. However, Canny edge detection is intended to detect sharp changes in image brightness in a greyscale image. An embroidery pattern is not a general greyscale image and the edges in the line drawing are clearly defined. The core task here is to assist the region labelling algorithm detecting nested regions. While a scan of an embroidery pattern looks like a black-and-white image, there always exist some pixels of intermediate tones surrounding the boundaries. Therefore, we use simple thresholding [7] to create a true binary image before we track the edges. Our algorithm follows a clockwise order to label the boundary; when it meets a bifurcation, it always turns to the right. A closed region is identified when this tracking process returns to its start point. Otherwise, an open feature is identified. We can use the information of closed region boundaries to identify the nested regions.

The above tracking process returns several lists of boundary pixels. Each list contains points that describe a "curve" and are used as control points of a uniform cubic B-spline. When we are placing stitches, we will calcuate points on the spline at a fixed stepsize, which is the length of boundary stitch.

The disadvantage of this method is that it introduces double lines when a curve is shared by a inner region and an outer region such as in the flower pattern in Figure 3.8, left. However, the double lines along the boundaries do not prevent us from generating a single line of stem stitch as we can skeletonise the boundary using the hit-and-miss method [8]. The thinned boundary eliminates overlapping stitches.

Chapter 3

Modelling

In this chapter, we describe how to generate stitch layouts given a set of regions. This is a semi-interactive process since some artistic choice is involved. Artistic choices include but are not limited to stitching style, thread parameters, and colours. These are not specified in the input pattern, so need to either be inferred or provided as additional input by the user.

Traditional embroidery or embroidered clothing products exist in many cultures, such as China, Japan, India, and Europe. The material, fabrics and yarns used in traditional embroidery vary from place to place. For example, Asian embroidery tends to use silk that gives a shiny appearance. Also, different cultures have preferences in stitching styles in decorating a layout. However, some stitching styles are shared throughout the world, such as stem stitch, running stitch, long short stitch, satin stitch, etc. In this work, stitch placement and algorithms for running stitch, stem stitch, long short stitch, satin stitch, and separated layers are developed. In the following sections, I will give background on stitches and then describe the stitch placement algorithms in detail.

3.1 Background: Traditional Stitches

Traditional embroidery differentiates itself from machine embroidery by its varying stitching styles. For filling a region, styles include satin stitch, couching, separated layers, and long-short stitching. For embroidering contours, there are various techniques, including staggered diagonals and held thread. Among these stitches, long-short stitch is the most used technique for sewing regions.

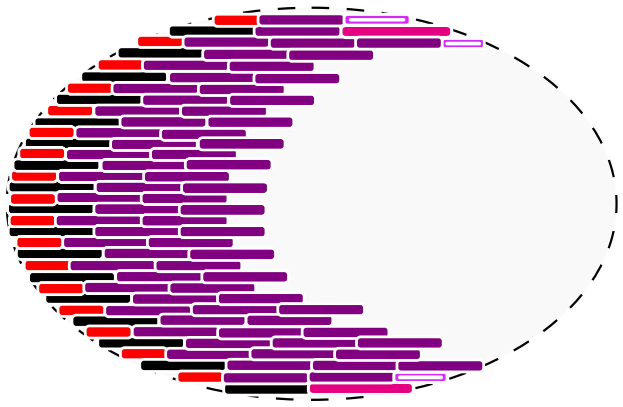

Figure 3.1: *Long-short stitches.*

3.1.1 Long-Short Stitches

The long-short stitch lies at the foundation of all solid embroidery. It is also one of the oldest stitches in embroidery. This technique is often used for fine shading. In the long-short stitch in the first column of stitches, every other stitch is half the length of its neighbours above and below. In the following columns of stitches, the stitches are roughly equal in size and worked to achieve a smooth appearance, using overlapping stitches without gaps, as in a brick wall. In general, an embroidery pattern has to fill space evenly and neither crowd stitches too close together nor leave gaps (Figure 3.1).

3.1.2 Other Stitches

There are several other styles possible; for example, in a "satin stitch", the stitches go entirely across the regions from edge to edge, and for certain shapes (such as leaves) we may break regions into subregions with different orientations. Figure 3.2 shows another four region-filling stitching styles, namely couching (tacking a stiff thread to the surface), separated layer, vertical layer, and diagonal layer. Vertical layer style and diagonal layer style are essentially satin stitch and can be achieved with machine embroidery, too.

There are three common boundary stitches. The first is a running stitch, used to embroider exactly along the contour using connected small stitches. It is used on clothes because machine embroidery can achieve this style. The other two methods to sew a curved line in embroidery are shown in Figure 3.3 which might be used either for details or for

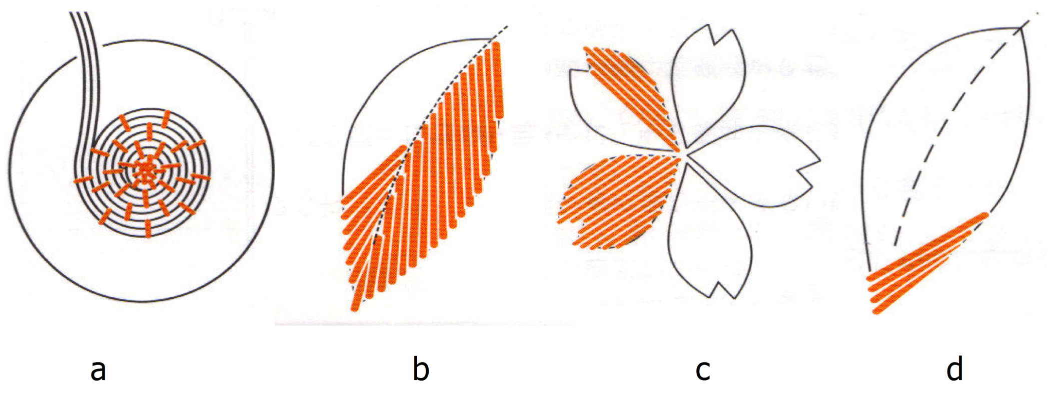

Figure 3.2: *Some other traditional stitching styles: (a) couching, (b) separated layer, (c) vertical layer, (d) diagonal layer.*

boundaries. Curved lines can be composed of many small stitches or a single thread tacked to the surface. The technique on the right in Figure 3.3 is called stem stitch, which uses overlapping short stitches placed at an angle to the curve.

3.2 Boundary Stitches Placement

We simulated two stitching styles for boundary stitches. One is the "running stitch". Given a spline describing the curve, we can sample the points on the spline in even steps. Between every pair of samples, we can place a stitch. Note that there are various ways to sew this in practice: hand embroidery doubles back, whereas machine stitch uses a backing thread. In general we do not worry about the mechanics of forming stitches from a continuous thread, as it does not significantly affect the appearance of visible stitches.

The other style we emulated is the stem stitch as shown in Figure 3.3 (right). This produces a thicker line than the running stitch. For this style the midpoint between every pair of sampled points is found. We place one stitch centered on every such point and then rotate and extend it slightly (Figure 3.4).

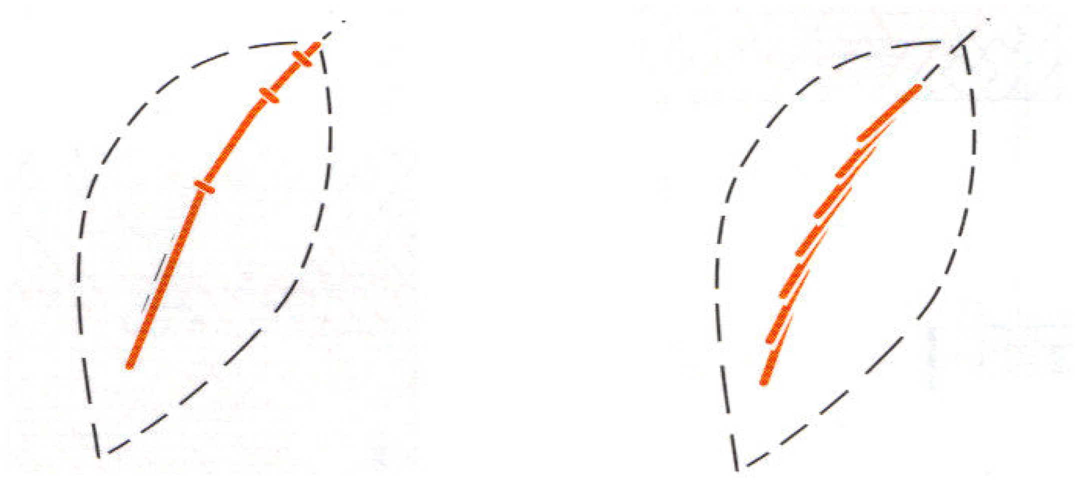

Figure 3.3: *Two classic methods for embroidering lines: Line of held thread and stem stitch.*

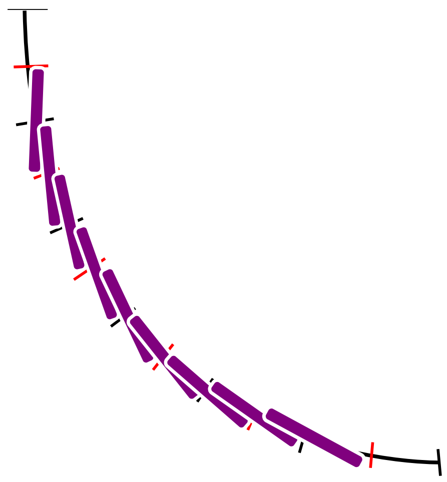

Figure 3.4: *Simulation of stem stitch. Each small red mark identifies the midpoint of each line segment.*

3.3 Fill Stitch Placement

3.3.1 Long-short stitches

To arrange stitches in the style of long-short stitches, we designed several rules to control the generation of stitches.

First, stitches in the first column are placed, alternating long and short. The starting positions of the first column are found by intersecting scan-lines with the region boundaries. Second, additional stitches are generated in each row until the surface is covered, clipping the stitches to the boundaries. Third, we reject stitches that are too short, merging these stitches with adjacent stitches. This rule applies to the last stitch in each row and will require its previous stitch to extend to the boundary.

The scanlines can be generated in various ways. In our implementation, we used a fixed orientation rather than a general vector field, and we did this for several reasons. Ijiri et al.'s work on 2D arrangements of elements [11] used flow fields to synthesize a larger pattern, and is to some extent similar to the problem we encounter here. However, their system requires the user to provide the flow field and still fails if the flow fields contain several edge lines. We experimented with this but in the case of embroidery, multiple vector field lines will lead to eye-catching gaps between elements. If we place a stitch to fill the gap, unpleasant overlapping will arise. Since artists often put the stitches along the longest axis of regions, we compute the main axis of inertia of each segmented region [10] and use this as a constant vector field.

To be specific, the object A is represented pixelwise by a characteristic function defined by

$$\psi_A(x,y) = \begin{cases} 1, & \text{if } (x,y) \in A \\ 0, & \text{if } (x,y) \notin A. \end{cases} \tag{3.1}$$

.

The following formulae determine the moments up to order one. The center of gravity of A is computed by $(c_x,\, c_y)$:

$$m_{0,0} = \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} \psi_A(x,y), \tag{3.2}$$

$$m_{1,0} = \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} x \cdot \psi_A(x,y), \tag{3.3}$$

$$m_{0,1} = \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} y \cdot \psi_A(x,y), \tag{3.4}$$

$$c_x = m_{1,0}/m_{0,0}, \tag{3.5}$$

$$c_y = m_{0,1}/m_{0,0}. \tag{3.6}$$

$$\tag{3.7}$$

A central coordinate is defined as $\tilde{p} = (\tilde{x}, \tilde{y}) = (x\text{-}c_x,\ y\text{-}c_y)$. Similarly, we compute the central moments of order two:

$$\mu_{1,1} = \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} \tilde{x} \cdot \tilde{y} \cdot \psi_A(x,y), \tag{3.8}$$

$$\mu_{2,0} = \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} \tilde{x}^2 \cdot \psi_A(x,y), \tag{3.9}$$

$$\mu_{0,2} = \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} \tilde{y}^2 \cdot \psi_A(x,y). \tag{3.10}$$

Then the angle of the main inertia axis is

$$\phi = \frac{1}{2} \arctan\left(\frac{2\mu_{1,1}}{\mu_{2,0} - \mu_{0,2}}\right). \tag{3.11}$$

For more details, please refer to Appendix B in the paper by Hiller et al. [10].

Finally, a small amount of randomness is used to modify each stitch's length and the exact position of its endpoints to make the embroidery more natural. Exact adherence to the brick pattern can result in disturbing patterns of diagonals, so we also randomize the starting offset. Furthermore, without randomization, in some styles, such as satin stitch, the stitches are hard to distinguish.

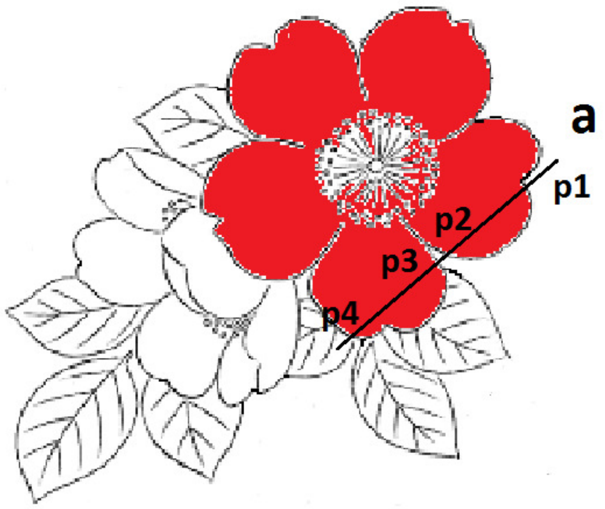

Figure 3.5: *Simulation of satin stitches (and related stitches): Line a intersects the boundary at points p_1, p_2, p_3, p_4. The line segment $\overline{p_2, p_3}$ should not be treated with stitches, and it can be eliminated by checking if it is in the interior of the region to be stitched using the even-odd rule. In practice, we know p_1 is at the start of an interior segment, and we then alternate interior/exterior segments.*

Since the boundaries of regions are represented with splines, we need to implement a line-and-spline intersection routine to clip the stitches (Figure 3.5). A simple and robust way to do this uses spline subdivision. In addition, we need to reject very short stitches near boundaries and lengthen neighbouring stitches as appropriate to fill the space.

3.3.2 Separated layer style

The separated layer style sometimes appears in patterns such as a leaf (Figure 3.2b). In this style, the stitches have two orientations, each at a certain angle away from the central line, usually about 30 degrees. To simulate this style, we first need to find the region's medial curve, and then determine which end of that curve is the "stem" in order to angle the stitches properly. We compute the axis of inertia in the same way as we do for long-short stitches. Then we parameterize the axis, $l(t)$, and shoot a set of lines that are perpendicular to $l(t)$, intersecting at $c_1(t)$ and $c_2(t)$ (Figure 3.6).

Then we compute the medial curve as

$$c_m(t) \quad = \quad \frac{c_1(t) + c_2(t)}{2}.$$

To sample the points on the medial curve in order, we find the "tip points" of the region (the extreme points in the direction of the medial axis) and then go from one to the other to collect medial axis points sequentially. The first and last elements in the resulting list are the two tip points (Figure 3.6, left).

As this pattern is imitating a natural leaf vein, it is essential also to determine which tip point attaches to the "stem". To decide between the two candidates, we search over a circle centered at the mass center of the region with a radius a bit longer than half the length of the longest main axis for outline pixels not on the boundary of the current region. We assume any pixels found not on the region outline belong to an associated "stem". By tracking them, we can detect the start point of the medial curve since the point will be closest to the pixels on the stem (Figure 3.6, right).

Finally, with the discrete points on the medial curve, we sample points in fixed step using uniform cubic B-splines, generating both positions and orientations. Each of these serves as a base orientation for a positive and negative 30 degree rotation. These are the final two orientations for the stitches placed in that section. Figures 6.6 and 6.7 show our simulation of this style.

3.3.3 Satin stitch

This style is the quickest way to solidly fill in an area, for it is simply a series of stitches laid side by side in a fixed direction. It is straightforward to simulate this style. Given a point in the labelled region, we can cast rays in the direction of the orientation and in its opposite direction. If the two rays intersect with the boundaries with two intersections, the connected line segment is where we place a stitch. By moving the point in the direction perpendicular to the orientation, we get a satin stitch pattern that covers the whole area. Note that it is essential to add some randomness to the orientation otherwise the stitches are hard to distinguish.

3.3.4 Couching

This couching style is similar to an Archimedean spiral (Figure 3.7). In embroidery, couching is created by a long thread and several equidistantly spaced small holding stitches. We need to chop the spiral into segments of constant arc length and then draw each segment

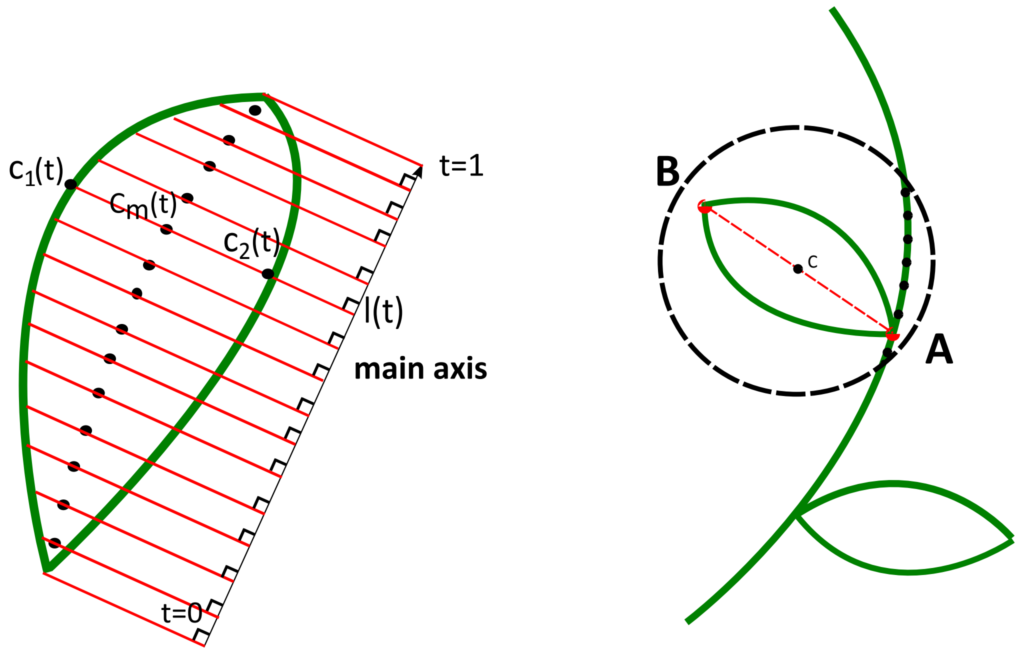

Figure 3.6: *Simulation of separated layer style. Left: Parameterize the main axis of inertia, $l(t)$, and shoot rays perpendicular to $l(t)$, intersecting at $c_1(t)$ and $c_2(t)$. Then we can compute midpoint $c_m(t)$. Right: Deciding the start point of the medial curve: search over a circle centered at the region's center, with a radius a bit longer than half of the length of the longest main axis for boundary pixels excluding those for the current region. We track these pixels and detect point A, which is the start point because it is closest to the pixels on the "stem".*

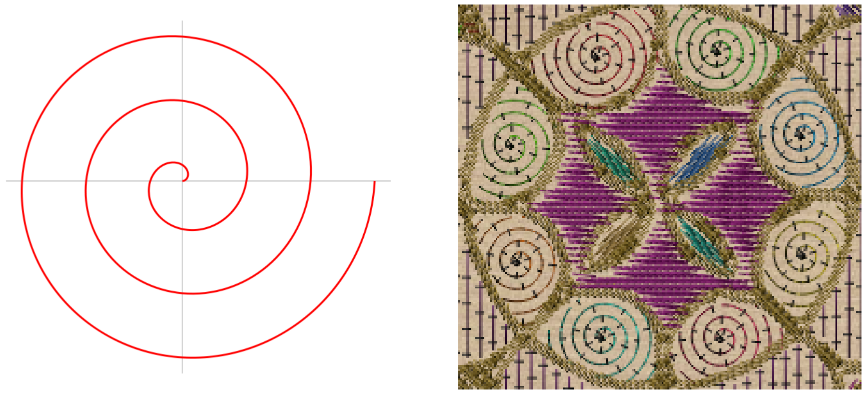

Figure 3.7: *This couching style (Figure 3.2a) is similar to an Archimedean spiral (left). We experimented with simulating couching style using polylines*

in a base layer, with holding stitches in an upper layer. Assuming the radius of a region is r and (c_x, c_y) is the centerpoint of the spiral, we can compute the positions of points along the spiral (x, y) by

$$(x, y) \quad = \quad r(\sin\alpha, \cos\alpha) + (c_x, c_y),$$

where $r = wt$ and $\alpha = 2\pi t$. Here w defines the width of the thread, and t is a parameter value (not, unfortunately, the arc length) that varies over $[0, 1]$ for each rotation, and r equals the radius of the region. Note that for longer segments, we may want to use multiple rendering primitives, since the thick thread in couching may have a significant curve between hold points.

3.4 Layers

The above methods fail to properly treat non-region-outline details in the embroidery pattern such as the stamens of flowers or veins of leaves. If these are also processed as boundaries they will break up the large areas. Forcing the system to fill in the broken up regions with small stitches would fail to give a neat result as shown in Figure 3.8 on the left. In fact, real embroiderers often do not treat all the lines in the pattern equally. They will stitch large regions first and then embroider details on top of such base layers. To simulate such techniques, we manually annotated the input to separate lines that represent region boundaries from those that represent details. When we render the embroidery model, a small z offset can be added to detail stitches making them appear above the fill layer. Alternatively, layers can be rendered back-to-front in texture space. Figure 3.8 compares the results generated by single layer and by using a second layer for details.

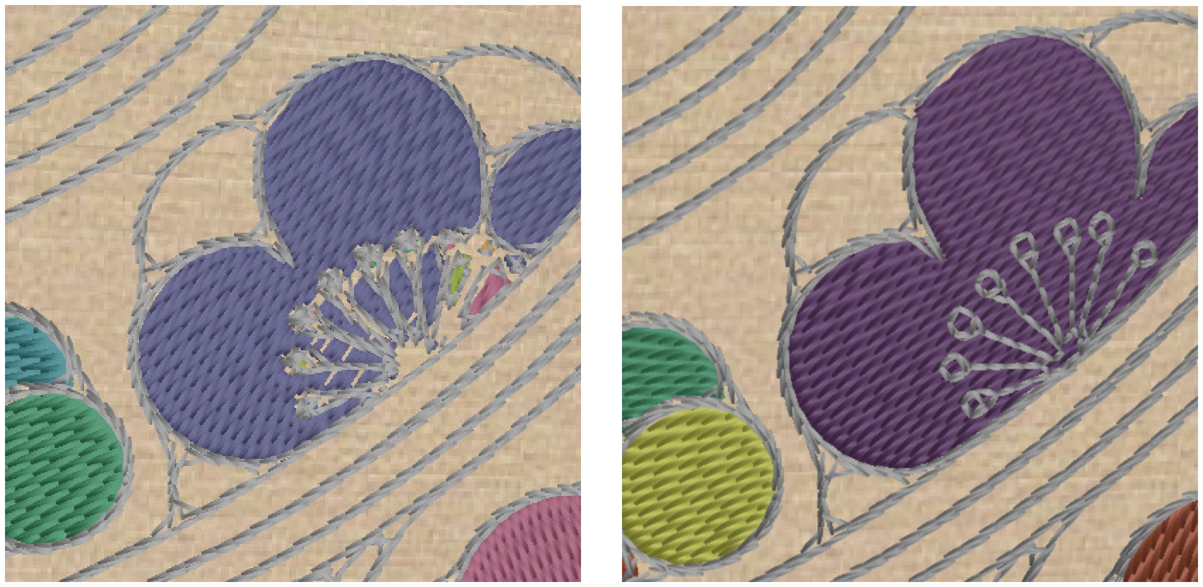

Figure 3.8: *Result generated by single layer is demonstrated on the left while the result generated by double layer is on the right.*

3.5 Coloring

Choosing the right color combination is vital for any artwork. In traditional embroidery, the colors are usually not specified in the input pattern but are often chosen from standard palettes. We decided simply to choose random colors from standard palettes. However, users can select their preferred color interactively for each region, or specify it in an annotated input.

3.6 Interactive Interface

A preliminary interactive interface was built to enable the user to decide the color and stitching style of a region. When the application starts, the system randomly chooses colors from standard palettes to fill in each region. It also initializes each region's stitching style when it starts. The user can freely change the sewing technique or color of a region by clicking on it, or choose to leave a region without any stitches (Figures 3.9 and 3.10).

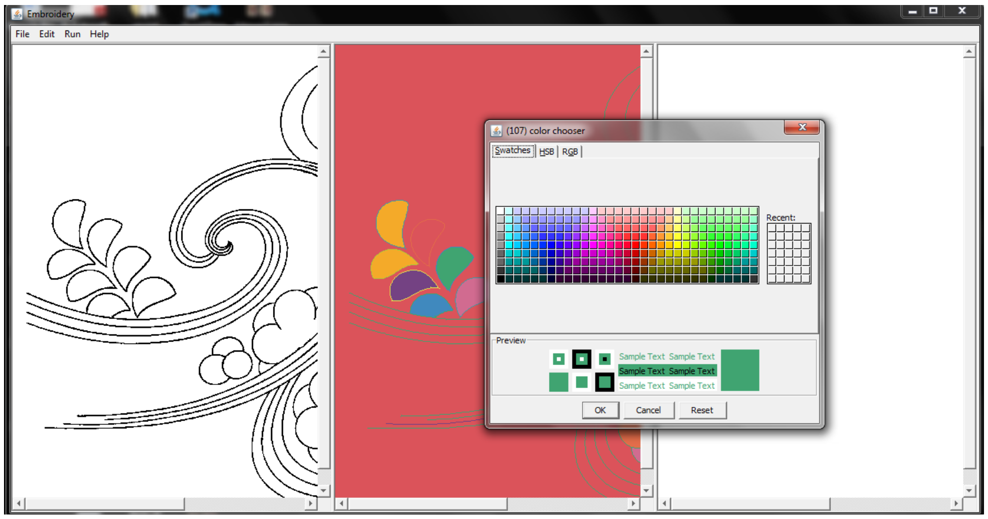

Figure 3.9: *Interactive interface: The image chosen by the user is displayed in the left column. In the middle column, the user may select a color by clicking the right mouse button on that region.*

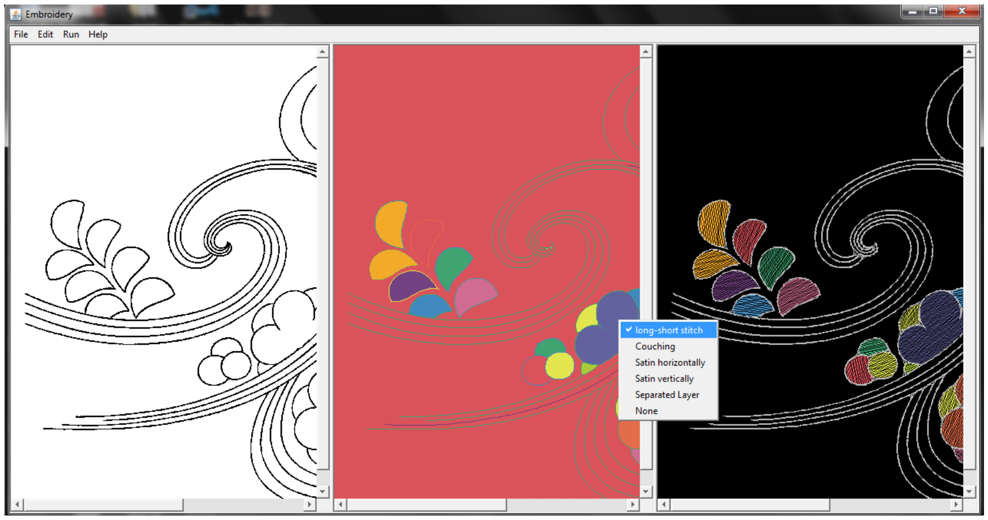

Figure 3.10: *The user may also select from a list of styles to sew a region. After clicking Run in the navigation bar, the result can be saved as an embroidery model as shown in the third column.*

Chapter 4

Stitch Rendering

Rendering the layout has to be efficient since it needs to be performed at a high frame rate and online, along with other processing.

There are three scales at which the stitch rendering process takes place.

The first one is microgeometry. This scale is accounted for by lighting. Since stitches are made of threads, we assume each thread is a thin cylinder. This microgemetry is reflected mathematically in the thread lighting model [19] we selected as discussed in Section 4.1.

The second scale is mesogeometry, the intermediate scale. This scale consists of discrete rendering primitives, but the details of these primitives are not clearly visible at normal viewing distances. Instead, it is the overall effect that is important. For embroidery, in real life one can barely perceive individual stitches or even stitching patterns. Instead what is important is the overall effect. Our rendering solution focuses on giving pleasing results as efficiently as possible. In particular, rather than rendering each stitch accurately (as, for example, a generalized cylinder, which would require a large number of polygons depending on the size of cylinder), we use some simple proxy geometry instead. Proxy geometry is a substitute rendering primitive; we use a simple quadrilateral in place of a tapered cylinder for stitches. However, we modify the tangent of each quadrilateral to simulate the curve of the stitch and also modulate its outline with alpha mapping to simulate the tapered shape of the cylinder. Both of these are inexpensive.

The third scale is macrogeometry. This is the object, curved or flat, on which we place the embroidery. Our system isolates the embroidery modelling from the macrogeometry. In this chapter, rendering embroidery layout on a flat plane for viewing will be discussed and rendering embroidery layout on a curved 3D object will be discussed in the next chapter.

4.1 Microgeometry: Lighting Model

The relatively simple lighting model we selected [19] is based on averaged Phong/Blinn lighting over the surface of an infinitesimally thin cylinder. This produced reasonable results and fast line rendering at a speed and quality comparable to the standard OpenGL rendering of polygons. Lighting is computed from three unit vectors: $\mathbf{V}$ (pointing from a point $\mathbf{P}$ on the curve towards the camera); $\mathbf{L}$ (pointing from a point $\mathbf{P}$ on the curve towards the light source); and the tangent $\mathbf{T}$ to the curve at the point $\mathbf{P}$ (Figure 4.1). Note that a curve does not have a uniquely defined normal, instead it has a plane of normals at every point. However, it does have a unique tangent at every point, so this is what is used in lighting. Diffuse and specular reflection are calculated by integrating over the visible part of the perimeter. The diffuse component is a view dependent version of the one introduced by Kajiya and Kay [13].

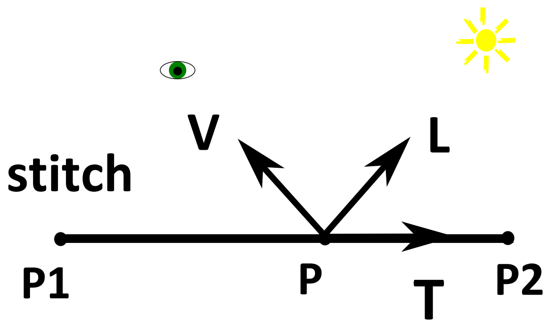

Figure 4.1: Side view of a stitch. Lighting is computed from three unit vectors: the view vector $\mathbf{V}$, the light vector $\mathbf{L}$, and the tangent $\mathbf{T}$.

For more detail and implementation of this lighting model, please refer to Appendix A, Lighting Model.

4.2 Proxy Mesogeometry: Quadrilaterals

Technically, each stitch is roughly a tapered cylinder. However, this would be expensive to render and since stitches are not normally clearly visible, we can use a simpler proxy instead. Two possible proxy geometries are possible: lines and quadrilaterals. We render each stitch as a quadrilateral proxy to have better control over size and endpoint placement than would be possible with line primitives. The other disadvantage of line primitives is that their widths are specified in screen space pixels, not physical width. We can also put an alpha texture on quadrilaterals to control shape and simulate tapering. With a smooth step at the edge of the matte, we can also antialias the shape.

Note that the lighting model captures the effect of the microgeometry, so these primitives are just acting as a substrate for that lighting model. It is not necessary to render stitches as small cylinders since the effect of a cylindrical thread is already captured in the lighting model: the lighting of the cylinders would be the same as the proxy geometry, the only approximation we are making is a reduction in parallax and a flattening of depth, both of which are small effects at normal viewing distances. The three-dimensional nature of threads is not visible at most normal viewing distances, even in real life.

4.3 Tangent Modification

In real embroidery work, a stitch's end points exit and enter the base fabric, which generates a curve. To simulate such an effect inexpensively (that is, without using extra geometric primitives), each quadrilateral's vertex tangents (not the geometric tangent, but the parameter used for the lighting model) is modified before being linearly interpolated. This produces an illusion that the stitch is not lying flat on the surface. Figure 4.2 illustrates this process and Figure 4.3 compares the image rendered with this technique and without.

Note that this modification is simular to bump mapping. But instead of perturbing the surface normals, we modify the tangent to affect the final illumination calculation because the lighting model is view-dependent and only uses the tangent to compute the final result. It would be possible to use a spline in a shader instead to get additional control over the apparent shape of the stitch.

4.4 Alpha Mapping

The Alpha channel in the RGBA color system represents the transparency of the color, with a value between 0 and 1. A value of 0 means that the pixel is completely transparent, or does not cover this pixel. A value of 1 means that the pixel is opaque because the geometry has completely overlapped the pixel. Intermediate values represent different percentages of coverage. Alpha mapping is a technique in 3D computer graphics where an image is mapped to a 3D object, and designates certain areas of the object to be transparent [28]. In particular, by using a "decal" the apparent silhouette of an object can be modified. By properly setting the decal texture's alpha, we can simulate a curved silhouette. In the case of a stitch, the two end points need to enter the fabric, which compresses the threads. Therefore, the apparent width of the stitch should get thinner when it gets closer to its

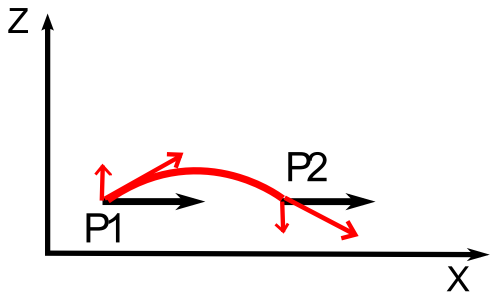

Figure 4.2: Tangent modification: p_1 and p_2 are two end points of a stitch. The original tangent is the black vector parallel to the x axis. After we add a small value pointing towards positive z to the tangent at p_1, and a small value pointing towards negative z to the tangent at p_2, we can get two new tangents that resemble the tangents of a curve's two end points.

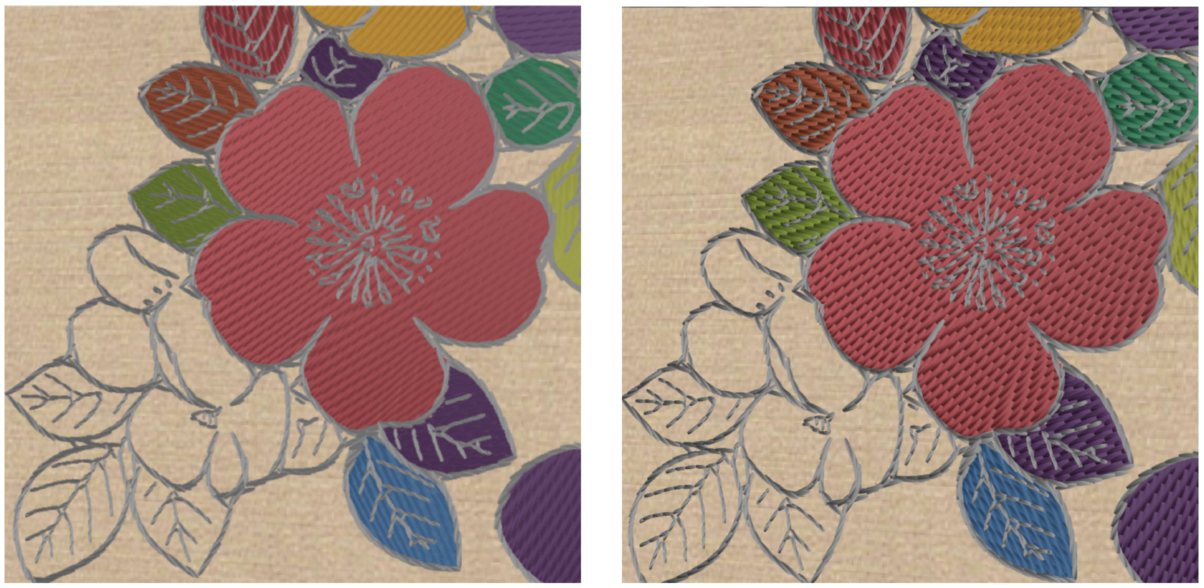

Figure 4.3: *Result generated without tangent modification (left) and result generated with tangent modification (right).*

end points. To generate an alpha texture to simulate this, the function $f(d) = (1 - d^a)^b$ was used. The value d can be interpreted as the distance from a point on the rectangle to the center, calculated by

$$d = w_x * \frac{|x - c_x|}{0.5 * l} + w_y * \frac{|y - c_y|}{0.5 * h}, \tag{4.1}$$

where (c_x, c_y) are the coordinates of the center of the rectangle, l is the length of the rectangle, h is the height of the rectangle, parameters w_x and w_y adjust the weight of x and y in computing this "distance". The alpha image used in the final results is generated in advance with the values $a = 11$, $b = 11$, $w_x = 0.3$, $w_y = 0.3$, values we determined experimentally. Then we mipmap alpha texture, as shown in Figure 4.4 and sample the value through the shader so as to mask it upon each stitch. Figure 4.5 shows stitches with and without this alpha blending.

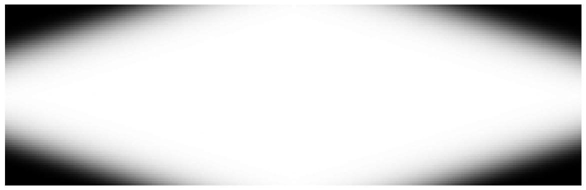

Figure 4.4: *An alpha texture is generated in advance and later applied to each stitch to taper its silhouette. In this figure, the alpha texture is placed on top of a black background. To make the alpha mask easily viewable, R, G, and B are set to the same value as the alpha.*

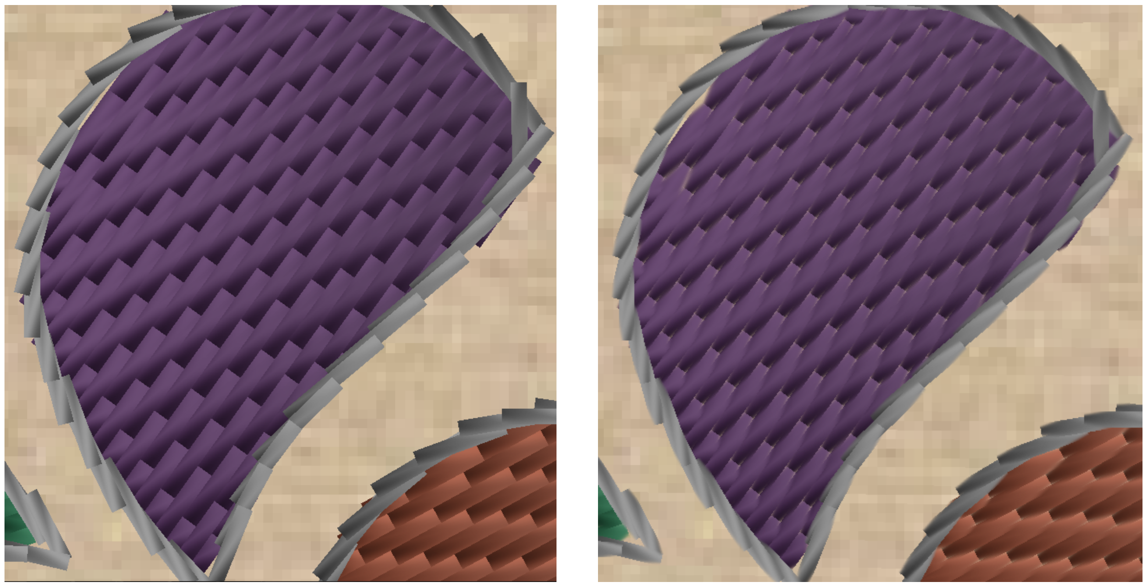

Figure 4.5: *By comparing The image on the left generated without alpha mapping while the image on the right generated with alpha mapping, we can see that alpha texture is essential for rendering the stitches.*

Chapter 5

Surface Space Embroidery Rendering

So far we have discussed how to render individual stitches. One approach to rendering embroidery would be to instantiate the stitches in object space on top of a 3D model. However, this would result in poor image quality and low performance.

First, placing stitches directly on a 3D model is difficult based on an embroidery line drawing. It is possible to imagine the input pattern wrapped around the model; however, how to place stitches neatly become a new problem. In reality the base cloth is often stretched out flat before the embroiderer starts placing stitches on it.

Second, the stitches would have to be placed so close to the model as to cause z-buffer precision issues. This could be dealt with by standard techniques, such as offsetting the stitches in the direction of the eye, but such an approach would cause various other problems. This is complicated and creates a dependency between the macrogeometry and the mesogeometry

Third, when placed on a tessellated model, the stitches would also have to be broken up to align with the tessellation. Otherwise they would tend to penetrate the surface even in the absence of z-buffer precision issues.

Fourth, if the embroidered surface were far from the eye, each stitch would be small, which would cause geometric aliasing. If instead we rendered the stitches at constant pixel width regardless of the distance from the eye (which is typical for the built-in line primitives), we would get an incorrect overall reflectance.

For these reasons we use the alternative approach of rendering the stitches in a separate "surface space". As is illustrated by Figure 5.1, essentially we render a texture containing the embroidery, and then apply this texture to the model. Before rendering, we compute the tangents from the texture parameterization. Then we render the 3D object and capture the light and view vectors as well as the local surface frame. Next, we project the light and view vectors onto the local surface frame and light the stitches using a thread lighting model. In the end, we map the texture back onto the object. This needs to be done dynamically since stitches in the embroidery respond to the lighting environment. As the lighting changes the texture needs to be updated. However, the resolution of the texture can be adjusted relative to the size of the stitches to avoid geometric aliasing in surface space. After the lit embroidery is rendered, the resulting image can be used as the basis of a MIP-map pyramid and standard texture filtering can be applied to back onto the object to get a high-quality rendering.

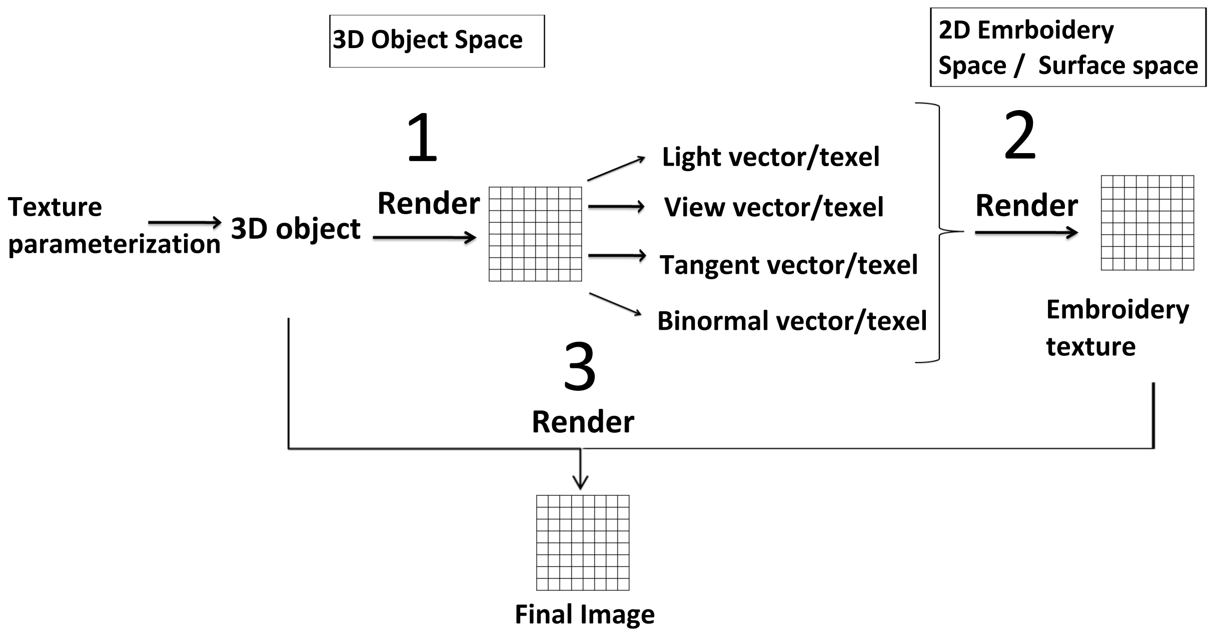

Figure 5.1: *This pipeline is used to render the embroidery layout on a 3D object. In step 1, the lighting environment on the 3D object is captured. In step 2, the lighting environment is used to render stitches in surface space, creating a lit embroidery texture. In step 3, the embroidery texture is mapped back onto the 3D object.*

5.1 Implementation

To implement the above approach, as with bump-mapping, we need to compute a surface frame per texel: a tangent $\mathbf{T} = [t_x, t_y, t_z]$, a binormal $\mathbf{B} = [b_x, b_y, b_z]$ and a normal $\mathbf{N} = [n_x, n_y, n_z]$. Each of these, expressed in object space coordinates, is placed as a column in a matrix S, which is called (in the context of bump mapping, for example) the TBN matrix:

$$S = \begin{bmatrix} t_x & b_x & n_x \\ t_y & b_y & n_y \\ t_z & b_z & n_z \end{bmatrix}.$$

Figure 5.2 shows the relationship between the different spaces and how to convert between them with different matrices. The TBN matrix S is used to transform lighting parameters from tangent space, also called surface space, to object space. Given a vector $\mathbf{V}_s$ in surface space (specified relative to the surface frame given by $(\mathbf{T}, \mathbf{B}, \mathbf{N})$), we can transform it into object space as $\mathbf{V}_o$ using

$$\mathbf{V}_o = S\mathbf{V}_s.$$

In bump mapping, the TBN matrix S is used to transform a normal on a bump-map texture from tangent space into object space, so that the light interacts correctly with the modified normal and so this normal can be used for lighting. In our case, the surface tangent specified for each stitch exists in surface space while the light vector and view vectors are captured in object space. We need these vectors to be in the same space. There are multiple options depending on where we choose to compute lighting. One option is to transform the light and view vectors into surface space; another is to transform surface tangents into surface space. The second option is more desirable because we only have to perform the tangent vector conversion as opposed to transforming all the light and view vectors to object space. This is the approach we use.

To calculate the tangent basis at each vertex of a mesh, we use the approach presented by Lengyel [16], which calculates the tangent orientations using the texture mapping coordinates. Note that the issue of having a smooth tangent field is related to the problem of having a smooth texture map parameterization, so we can use solutions for the latter problem to address the former. Of course, it is not possible in general to assign a smooth tangent field to arbitrary three-dimensional objects, so we will inherit any singularities or seams from the underlying texture parameterization.

We render the 3D object and use vertex and fragment shaders to compute the light, view, tangent, and bitangent value and render them per texel into buffers/textures (Figure 5.3). Next, we render the embroidered texture with information from the 3D object. In this stage, another pair of vertex and fragment shaders is used to compute the lighting. Each vertex of a stitch is assigned a texture coordinate, with which we do texture look-up for tangent, bitangent, light, and view vectors. We obtain the normal to form the TBN matrix by computing the cross product of the tangent and the bitangent vectors. With the surface frame, we transform the surface tangent to object space and further transform it to view space, where the final lighting is computed. In this way even high-frequency variation

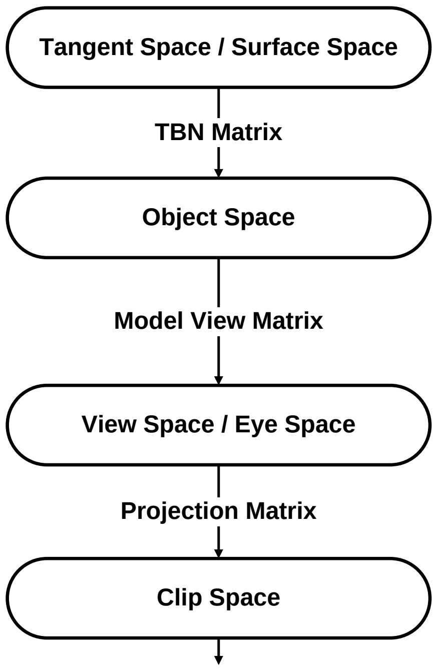

Figure 5.2: *Matrix space conversion. While we have Model View Matrix to transform vectors from Object Space into View Space and Projection Matrix to further transform vectors from View Space into Clip Space, TBN Matrix transforms vectors from tangent space into object space.*

in the lighting due to surface geometry can be accounted for. However, it would also be possible to make approximations to this, for example, by sampling the lighting from these buffers only at the endpoints of the stitches and interpolating (which would be sufficient for short stitches), or by using a lower resolution for these light buffers than for the output image.

We currently re-render all the stitches for every frame to construct the surface texture. This is sufficient for our prototype, but may not scale well for more complex embroidery, or for systems where the GPU is needed for other tasks. A more scalable approach would be to use deferred shading, where the tangent and other parameters of the stitches lighting model are first rendered into buffers, then lighting is done in a separate pass as a single global per-texel operation. This approach would be more efficient as the stitches would only have to be rendered once, and the evaluation of the lighting model would be regular (and therefore efficient), since it would only involve streaming a computation over a set of buffers. However, depending on the number of lighting parameters needed and how many bits are needed to store them, it might be relatively expensive in terms of storage and memory bandwidth.

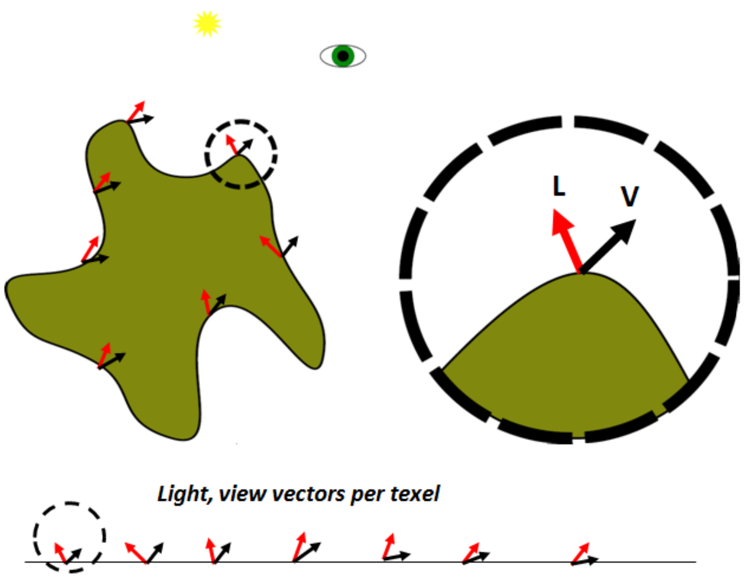

Figure 5.3: *The surface is unfolded and lighting is computed in texture space. Each vertex's light and view vectors are computed in 3D space and written at the vertex's texture coordinate.*

If the model is rigid, then we also do not need to rerender the surface frame buffers on every frame. Instead we could apply suitable transformation vectors to buffers containing a sampling of the surface tangents and model-space positions in texture space and then compute the view and light vectors dynamically, again using more regular "array" computations. It is not clear that this would always be faster: it will depend on the number of texels per polygon. In our implementation, we also re-render the light and view buffers for every frame.

Chapter 6

Discussion

6.1 Results and Discussion

To test our system, two embroidery line drawings (Figures 1.3 and 1.4) attributed to Kurenai Kai were used. Each image is about 600 by 600 pixels. Figures 6.1 shows the result generated from Figures 1.3. In this embroidery layout, long short stitches are applied to all the identified regions; running stitches are used to decorate the details inside the regions; and stem stitches are placed on the boundaries and outlines. These rules are also applied to the result in Figure 6.3 generated from Figure 1.4. Note that the user can use other styles such as a satin stitch to decorate the regions. Performance (in terms of stitches per second) should be comparable for these styles, although the number of stitches would be different. Colors in the two results were randomly chosen by the system from standard palettes. We can see that our rendering approach produces the look of embroidery at the scale of not more than a few pixels per stitch (see Figures 6.2 and 6.5).

Figure 6.6 shows that various styles can be used to decorate a leaf pattern. In this generated embroidery layout, the separated layer style, satin stitch style, and long-short stitch style were used to fill regions. The stem stitch style is applied on the boundaries in all cases. The green color was intentionally chosen for the leaves.

Figure 6.8 is a screenshot showing surface space embroidery on a curved object. The generated embroidery layout from the input pattern shown in Figure 1.4 is lit in surface space with the lighting information of the target 3D model, a teddy bear. Then, as described in previous chapters, the rendered and lit embroidery texture is dynamically mapped onto the 3D model with appropriate texture antialiasing.

	image analysis time in seconds	stitch placement time in seconds
Input Figure 1.3	5.51	24.0
Input Figure 1.4	5.44	251

Table 6.1: Performance of the embroidery modelling system.

6.2 Performance discussion

The system was tested on a laptop with an ATI Mobility Radeon HD 3400. The processor was an Intel(R) Core(TM)2 Duo CPU P8800 @ 2.66GHz. The installed memory (RAM) was 4GB. When we discuss measured time in the following, we mean wall-clock time.

The emphasis of this system was not performance, but image quality. Nevertheless, we need the modelling system to be fast enough to be usable and for the rendering to be fast enough to be used in real-time applications. We also need to demonstrate that our system is scalable to larger embroidery patterns with more stitches, so we would like to see at most a linear increase in modelling and rendering time for more complex embroidery. While there are many ways in which our prototype could be improved, these performance measurements provide a baseline.

6.2.1 Embroidery layout generation speed

There are two stages to modelling, and we have measured the performance of each stage. The first stage is image analysis. The implementation of this stage was performed using Sun Microsystem's implementation of Java. We measured the time from when an image is selected by the user to the time the system finishes labelling each region and extracting the boundaries. The second stage is stitch placement time, from the time the user asks the system to start placing stitches until all stitches have been placed. This does not include the time it takes to write the results to disk. Results are shown in Table 6.1.

For the input image in Figure 1.4, with an input resolution of 600×600, the image analysis time is 5.44 seconds. For this image and the output in Figure 6.1, with 12,645 stitches, the stitch placement time was 251 seconds, which results in a stitch placement rate of 50.3 stitches/sec. For the input image in Figure 1.3, with an input resolution of 600×600, the image analysis time was 5.51 seconds. For this image and the output in Figure 6.3, with 11,142 stitches, the stitch placement time was 24.0 seconds. The second example executed in a significantly shorter time than the first example because this second imput image has fewer regions and details for the system to process.

	Stitching rendering (fps)	Texture Space stitch rendering (fps)
Input Figure 1.3	36.4	6.1
Input Figure 1.4	36.1	5.0

Table 6.2: Performance of the embroidery rendering system.

6.2.2 Embroidery rendering performance discussion

Table 6.2 shows the performance of the rendering system. The rendering system was implemented in C++, using the Microsoft Visual Studio Express 2010 compiler. To test performance, we rendered a simple scene with the embroidery texture mapped to a screen-spanning flat polygon. When rendering stitches on a plane, the averaged frame rate is 36.1 fps for input image 1.3 and 36.4 fps for input image 1.4 for a 1280×800 pixel image. When rendering embroidery layout on a 3D object, the averaged frame rate dropped severely to about 5.0 fps and 6.1 fps respectively, for the embroidery is generated in texture space with necessary lighting parameter textures generated from the 3D object in the previous pass. Also, our current implementation updates the lighting texture every frame. Note that 12,645 primitives are rendered for the embroidery layout generated from Figure 1.3 and 11,142 primitives are rendered for the layout from Figure 1.4.

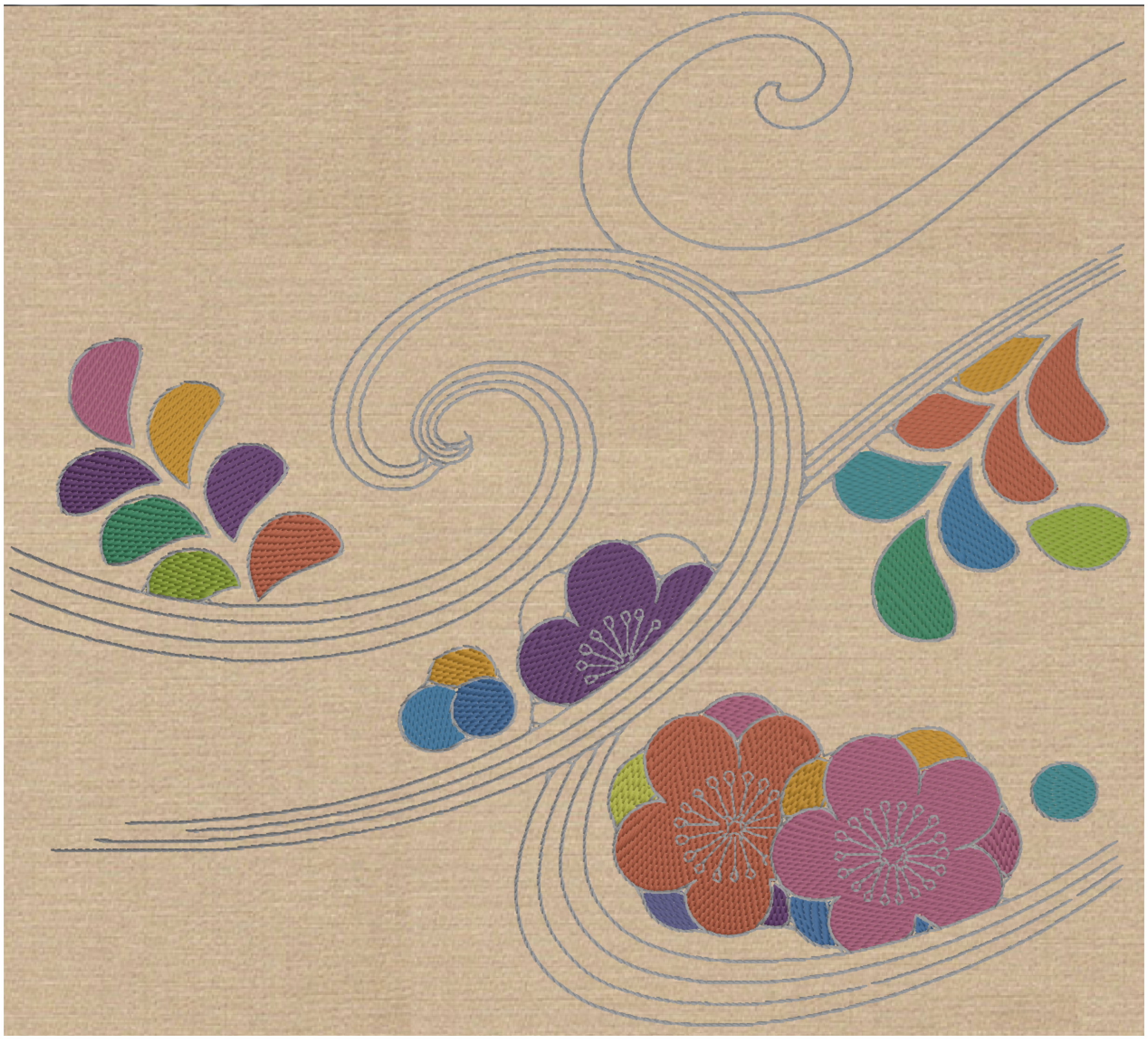

Figure 6.1: *Generated result from the embroidery pattern in Figure 1.3.*

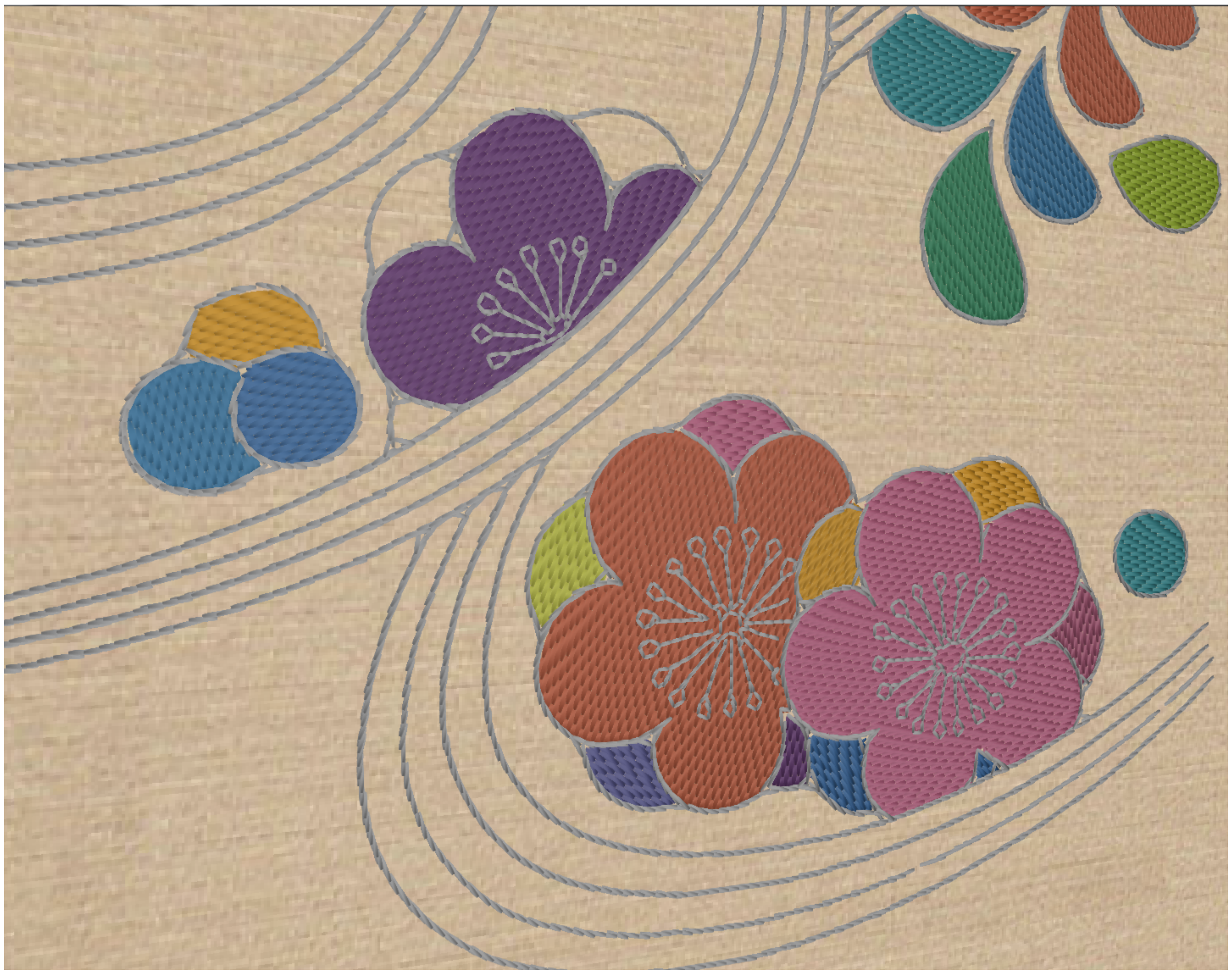

Figure 6.2: *A close up look at Figure 6.1.*

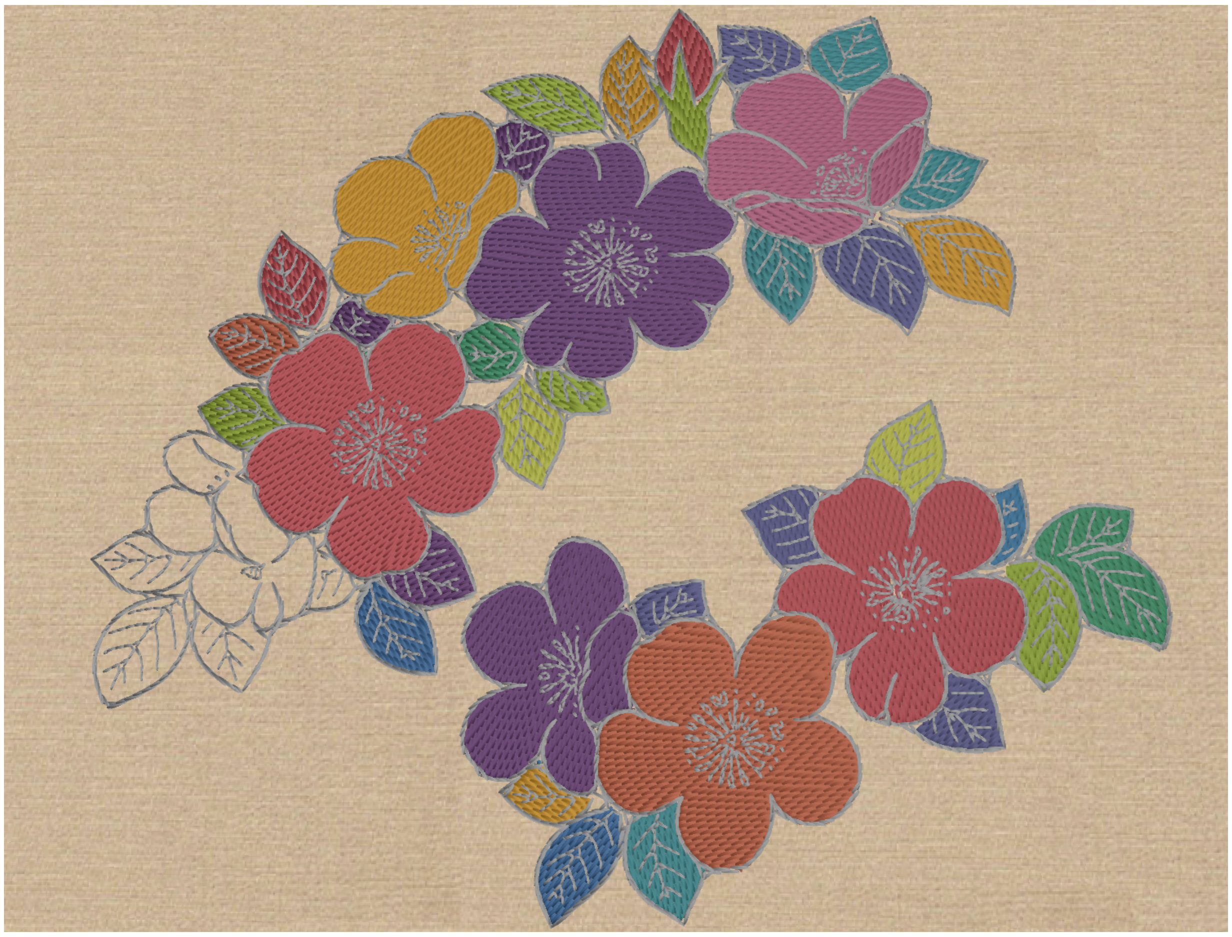

Figure 6.3: *Generated result from the embroidery pattern in Figure 1.4.*

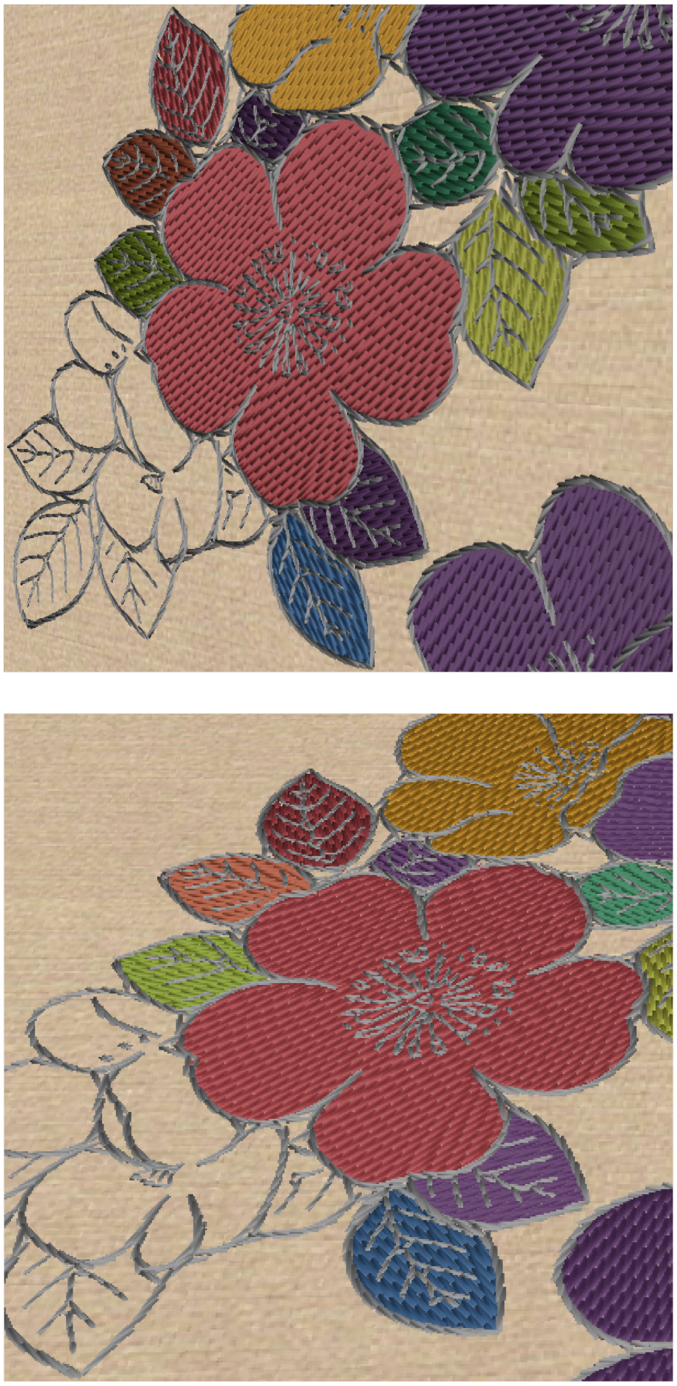

Figure 6.4: The simulated embroidery properly responds to the lighting. For example, the green leaf gets brighter in the image on the bottom while the yellow flower gets darker.

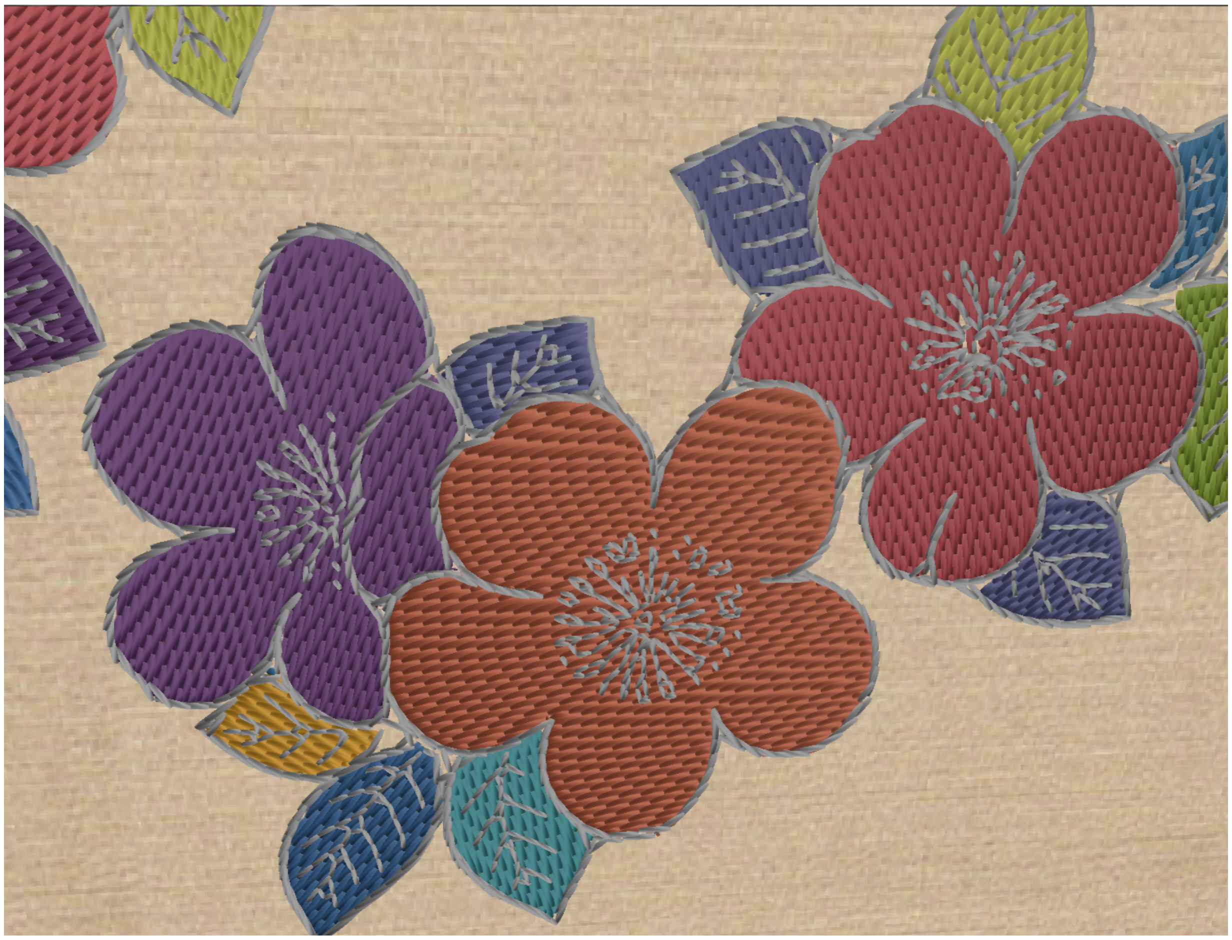

Figure 6.5: *A close up look at Figure 6.3.*

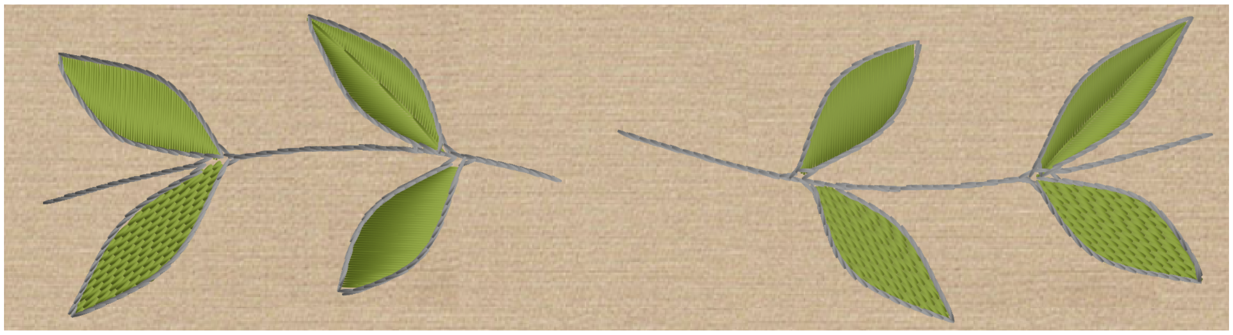

Figure 6.6: *Pattern generated for testing various styles: separated layers, satin stitch and long-short stitches.*

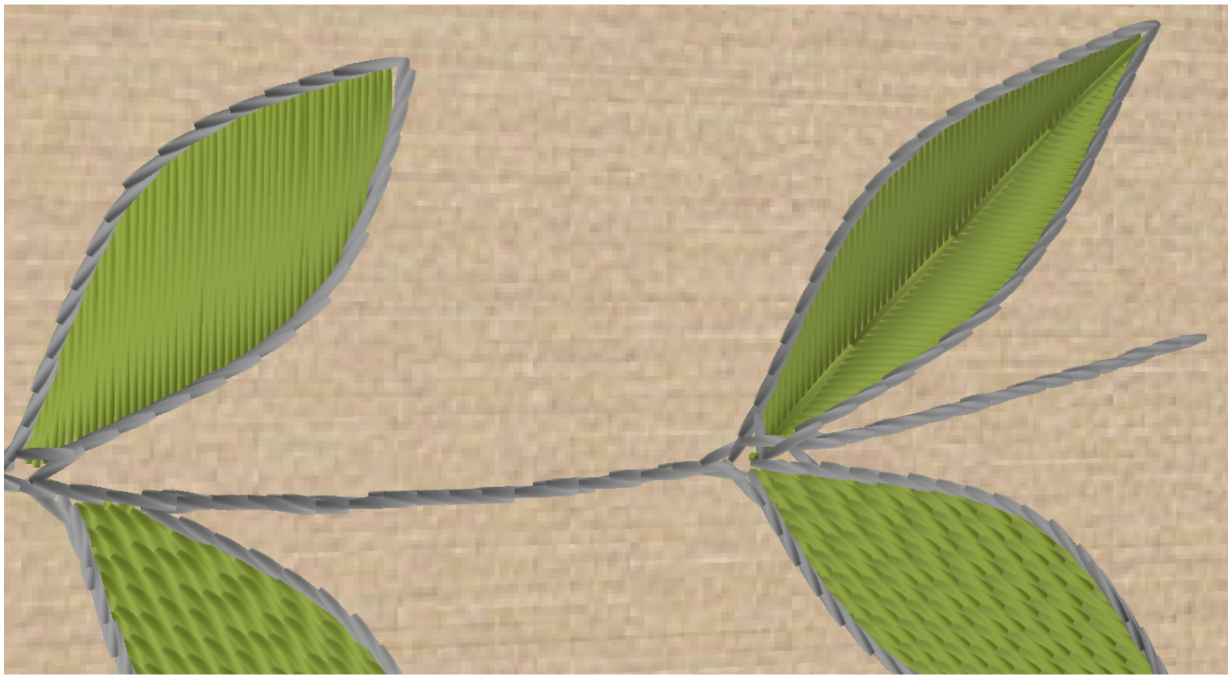

Figure 6.7: *A zoomed in image of Figure 6.6. The leaf on top-left shows the simulation of the satin stitch. The leaf on top-right shows the simulation of separated layers style.*

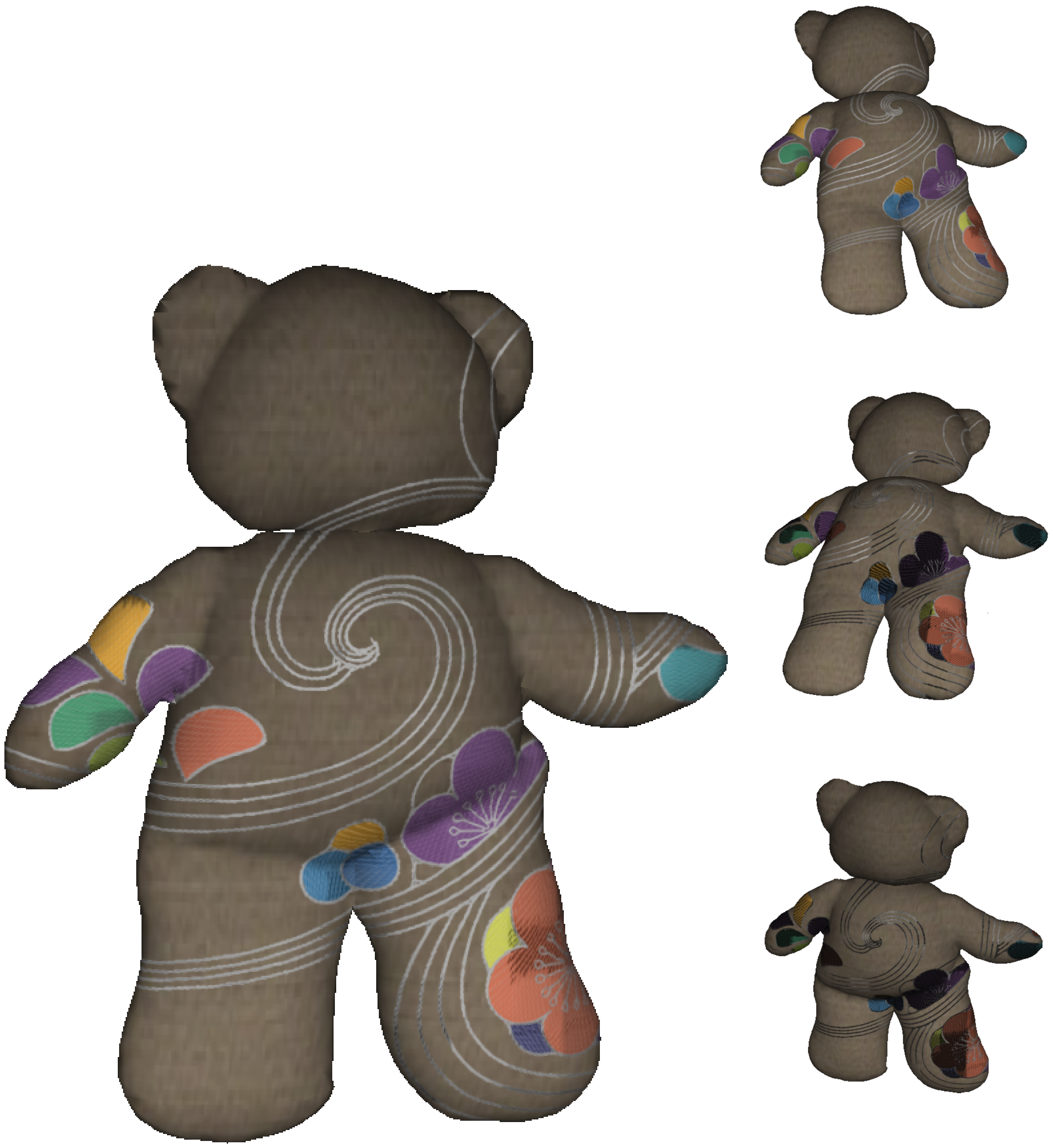

Figure 6.8: *Embroidered teddy bear: We render stitches in a separate "surface space" and dynamically map the embroidered texture to the 3D model. We can see that the embroidery on the teddy bear properly responds to the lighting. In particular, note that the embroidery is sometimes darker, sometimes lighter than the base cloth.*

Chapter 7

Conclusion

Embroidery is a common decoration on textiles. Since such textiles may appear in many scenes of interest in computer graphics applications, such as animation of historical costumes, it is useful to generate it automatically and render it well. This thesis has presented techniques for mostly automatic generation of decorative embroidery patterns, following the style of traditional embroidery, given an input line drawing. We have also presented techniques for rendering embroidery in real time at high quality, including anisotropic lighting and antialiasing. Further, an interactive interface was built to allow the user to exert artistic control over the result.

7.1 Contributions

This thesis has presented a set of techniques to generate an embroidery decorated model, following the styles of traditional Japanese embroidery, although our results should work for most embroidery styles. Vision techniques were used to extract regions from line drawings of embroidery patterns. We also presented techniques to render the embroidery in real time, using appropriate shading for the microgeometry and texture-space rendering for the mesogeometry. Appropriate strategies for antialiasing and lighting were used at each scale.

First, we developed techniques for analysing line drawings of embroidery patterns. Algorithms were developed for labelling regions and extracting boundaries from the input line drawing. We started with a connected region labelling algorithm based on flood fill. However, to distinctly label each region, we developed an edge tracking algorithm to solve a problem with the connected region labelling algorithm that failed to detect regions nested inside another. The combined algorithm can effectively identify each region in the input pattern. To follow the tradition of using single lines to decorate outlines, we also applied a thinning algorithm to the output of the edge tracking algorithm. This technique allows us to decorate boundaries with either a single-line stem stitch or a running stitch.

Second, a variety of traditional embroidery techniques were studied and simulated for automatic generation of decorative embroidery. For boundary stitches, running stitches and stem stitches, placement methods were developed. For filling stitches, we explored stitch placement for long-short stitches, satin stitches, separated layers, and couching. A user interface tool was also developed to allow the user to decide color and stitching style for each labelled region. These parameters are not included in the input line drawing, but are artistic choices.

Third, we developed techniques to render individual stitches using appropriate lighting, fine-scale geometry, and antialiasing. To model each stitch, a simple geometric proxy was chosen since the details of the shape are not visible at normal viewing distances. We chose a quadrilateral for the proxy geometry to get fine control over width and to allow texture mapping. We selected an anisotropic lighting model based on averaged Phong/Blinn lighting over the surface of an infinitesimally thin cylinder. An alpha texture was also generated and applied to the stitch's proxy geometry to mimic the tightening of the thread when it goes into the fabric. This alpha texture is also bounded by a smooth step to provide antialiasing around the boundaries of the stitch. Finally, the tangent of the stitch is modified to mimic the curve of a thread coming out of, lying on, and then going back into the cloth base.

Fourth, we developed techniques to render a set of stitches laid out in a pattern and apply them to a potentially curved and deforming 3D object in real time. We first captured the lighting environment (incoming light directions and view directions) on the surface of the target object and stored this in texture space. Stitches are then rendered in this "flat" space at a resolution that avoids geometric and highlight aliasing. Rendering stitches in a geometrically flat space also prevents surface and stitch interpenetration and allows us to use simple proxy geometry (quadrilaterals) for stitches. However, the lighting and view directions vary on a per-texel basis, so when the resulting texture is applied to the object, which is also the point where we multiply the reflectance by the light intensity, the embroidery is correctly lit for the object.

7.2 Future Work

Our current system could be extended in various ways.

First, multi-layered embroidery layouts cannot be created automatically yet, although we can specify them manually. Multiple layers are important for properly rendering details on top of background areas. It would be interesting to explore computer vision techniques to identify layers. Computer vision techniques could also be used to choose stitching styles based on the shapes of regions.

Second, rather than processing a raster scan of a line drawing, we could also accept vector graphics input (such as SVG) directly, and this would have many advantages. However, we explored support for raster input since traditional embroidery designs are usually only available as printed images. We could, however, experiment with existing software for converting raster images to SVG, although the output would still require interpretation.

Third, the rendering could be improved in various ways. Pre-rendering the stitches into regularly sampled parameter buffers (e.g., using deferred shading) would improve performance, since the per-frame rendering would only have to process images, not individual stitches. However, this approach might lead to an increase in aliasing. Also, recently several improved lighting models for hair and thread have been developed, and these could be applied, but the more expensive models may impact performance. For our prototype we intentionally selected a simple model.

Fourth, the current system can't use the separated layer style to decorate "occluded" parts of leaves. A method to infer the missing boundary would be needed to estimate central line and orientation. Another extension would be using image recognition algorithms to identify leaves and other special regions instead of leaving the task to the user to select an appropriate stitching style.

We have not taken into account several factors that may affect appearance, such as fiber twisting, fuzziness, and translucency of different stitch materials. We also did not consider inter-stitch shadowing or local light transport. However, since we generate a 2.5D image at an intermediate step and also capture local view and light directions, a technique such as horizon maps could be used to compute local shadowing, and would also be applicable to real time rendering. Likewise, processing of the 2.5D image could be used, as with skin rendering, to simulate local interreflections. It might also be useful to blur the 2.5D depth map before shadowing to more accurately represent the mesogeometry, and an alpha test should also be used to capture the modification of the outline of the proxy geometry.

7.3 Conclusion

The techniques we have developed for embroidery modelling and rendering are still relatively simple. There is much more that could be done with this topic, particularly in the area of automatic stitch placement and interpretation of traditional patterns given as line drawings. However, we feel our system does generate reasonable images and its performance is high enough to be usable in many interesting applications.

Appendix

Appendix A

Lighting Model

The lighting model proposed by Mallo et al. [19] is based on curve/view aligned coordinate frame (**T**,**N**,**B**), with the binormal **B**=**T** $\times$ **V**/|**T** $\times$ **V**| and the normal **N**= **B** $\times$ **T**. For a point **P** on a curve in space, the situation differs in that a curve does not have a uniquely defined normal. Lighting must be instead be computed from the three unit vector **V**, **L** and the tangent **T** to the curve at point **P**. Note that k_a is ambient reflection coefficient, k_d is diffuse reflection coefficient, and k_s is specular reflection coefficient.

The lighting model is given by

$$C_{out} = C_{in}(k_a + k_d F_d(\cos\alpha, L_T)) + k_s\Big(\sqrt{1 - H_T^2}\Big)^n F_s(\cos\alpha, \cos\beta), \tag{A.1}$$

where C_{in} is the incoming fragment color and the C_{out} is the final color of the illuminated fragment. For F_d, we have

$$F_d(\cos\alpha, L_T) = \sqrt{1 - L_T^2}\frac{\sin\alpha - (\pi - \alpha)\cos\alpha}{4}. \tag{A.2}$$

$$\tag{A.3}$$

For F_s, we have

$$F_s(\cos\alpha, \cos\beta) = \int_{\alpha-\pi/2}^{\pi/2} \cos^n(\theta - \beta)\frac{\cos\theta}{2}d\theta, \tag{A.4}$$

where

$$L_T = \vec{L} \cdot \vec{T}, \tag{A.5}$$

$$\cos\alpha = (\vec{L} \cdot \vec{N})/\sqrt{1 - (\vec{L} \cdot \vec{T})^2}, \tag{A.6}$$

$$\cos\beta = (\vec{H}\cdot\vec{N})/\sqrt{1-(\vec{H}\cdot\vec{T})^2}, \tag{A.7}$$

where $\vec{H}$ is the halfway vector, can be computed by

$$\vec{H} = \frac{\vec{V}+\vec{L}}{|\vec{V}+\vec{L}|}. \tag{A.8}$$

Bibliography

[1] David C. Banks. Illumination in diverse codimensions. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, pages 327–334, New York, NY, USA, 1994. ACM. 9

[2] Pascal Barla, Simon Breslav, J Thollot, F Sillion, and Lee Markosian. Stroke pattern analysis and synthesis. *Computer Graphics Forum*, 25(3):663–671, 2006.

[3] J Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8:679–698, November 1986. 15

[4] Oliver Deussen, Stefan Hiller, Cornelius W. A. M. van Overveld, and Thomas Strothotte. Floating points: A method for computing stipple drawings. *Comput. Graph. Forum*, 2000. 8

[5] Gershon Elber and George Wolberg. Rendering traditional mosaics. *The Visual Computer*, 19:67–78, 2003. 10.1007/s00371-002-0175-x. 8

[6] Kurt W. Fleischer, David H. Laidlaw, Bena L. Currin, and Alan H. Barr. Cellular texture generation. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '95, pages 239–248, New York, NY, USA, 1995. ACM. 8

[7] Rafael C. Gonzalez and Richard E. Woods. *Thresholding In Digital Image Processing*, pages 595–611. Pearson Education, 2002. 15

[8] R. Haralick and L. Shapiro. *Computer and Robot Vision*, pages Vol. 1, Chap 5, 168 – 173. Addison-Wesley Publishing Company, 1992. 15

[9] Alejo Hausner. Simulating decorative mosaics. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 573–580, New York, NY, USA, 2001. ACM. 8

[10] Stefan Hiller, Heino Hellwig, and Oliver Deussen. Beyond stippling-methods for distributing objects on the plane. *Computer Graphics Forum*, 22(3):515–522, 2003. 20, 21

[11] Takashi Ijiri, Mech Radomir, Takeo Igarashi, and Gavin Miller. An example-based procedural system for element arrangement. *Computer Graphics Forum*, 27(2):429–436, 2008. 20

[12] Wenzel Jakob, Adam Arbree, Jonathan T. Moon, Kavita Bala, and Steve Marschner. A radiative transfer framework for rendering materials with anisotropic structure. In *ACM SIGGRAPH 2010 papers*, SIGGRAPH '10, pages 53:1–53:13, New York, NY, USA, 2010. ACM. 9

[13] J. T. Kajiya and T. L. Kay. Rendering fur with three dimensional textures. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '89, pages 271–280, New York, NY, USA, 1989. ACM. 9, 29

[14] Junhwan Kim and Fabio Pellacini. Jigsaw image mosaics. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02, pages 657–664, New York, NY, USA, 2002. ACM. 8

[15] Kurenai-Kai. *Traditional Japanese Patterns 1.* Seigensha, 2005. 8

[16] Eric Lengyel. *Computing Tangent Space Basis Vectors for an Arbitrary Mesh.* Terathon Software 3D Graphics Library, http://www.terathon.com/code/tangent.html, 2001. 36

[17] Anat Levin, Dani Lischinski, and Yair Weiss. Colorization using optimization. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 689–694, New York, NY, USA, 2004. ACM. 8

[18] Marc Levoy and Pat Hanrahan. Light field rendering. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 31–42, New York, NY, USA, 1996. ACM. 6

[19] O. Mallo, R. Peikert, C. Sigg, and F. Sadlo. Illuminated lines revisited. In *Visualization, 2005. VIS 05. IEEE*, pages 19–26, October 2005. 9, 28, 29, 53

[20] Stephen R. Marschner, Henrik Wann Jensen, Mike Cammarano, Steve Worley, and Pat Hanrahan. Light scattering from human hair fibers. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, pages 780–791, New York, NY, USA, 2003. ACM. 9

[21] Michael McCool and Fiu Eugene. Hierarchical Poisson disk sampling distributions. In *Proceedings of the conference on Graphics interface '92*, pages 94–105, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc. 8

[22] Yingge Qu, Tien-Tsin Wong, and Pheng-Ann Heng. Manga colorization. In *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, pages 1214–1220, New York, NY, USA, 2006. ACM. 8

[23] Michael P. Salisbury, Michael T. Wong, John F. Hughes, and David H. Salesin. Orientable textures for image-based pen-and-ink illustration. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pages 401–406, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co. 8

[24] H. Samet and M. Tamminen. Efficient component labeling of images of arbitrary dimension represented by linear bintrees. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 10(4):579 –586, July 1988. 12

[25] Adrian Secord. Weighted Voronoi stippling. In *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, NPAR '02, pages 37–43, New York, NY, USA, 2002. ACM. 8

[26] L. Shapiro and G. Stockman. *Computer Vision*, pages 69–73. Prentice Hall, 2002. 12

[27] Daniel Sýkora, Jan Buriánek, and Jiří Žára. Unsupervised colorization of black-and-white cartoons. In *Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, NPAR '04, pages 121–127, New York, NY, USA, 2004. ACM. 8

[28] Naty Hoffman Tomas Akenine-Moller, Eric Haines. *Real-Time Rendering.* A K Peters/CRC Press; 3 edition (July 25, 2008), 2008. 30

[29] Georges Winkenbach and David H. Salesin. Computer-generated pen-and-ink illustration. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, pages 91–100, New York, NY, USA, 1994. ACM. 8

[30] Georges Winkenbach and David H. Salesin. Rendering parametric surfaces in pen and ink. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 469–476, New York, NY, USA, 1996. ACM. 8