Computer-Assisted Bargello Quilt Design

by

Marge Coahran

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

Computer-Assisted Bargello Quilt Design

Marge Coahran

Master of Science

Graduate Department of Computer Science

University of Toronto

2005

Quilting is an art form in which amateur craftspeople explore geometry and color to create stunning designs. Bargello is a specific quilt style, defined by a clever construction method that imposes constraints on the designs and gives them a characteristic appearance. The current process for designing Bargello patterns is laborious and time-consuming. We have developed a prototype system in which users can design Bargello quilts quickly and easily by sketching curves with a mouse. The system has three constituent parts: an algorithm that produces visually smooth curves composed of corner-connected axis-aligned rectangles, a module that supports the creation of Bargello quilt designs, and a module that supports the exploration of color combinations using real fabric textures. We conducted a set of informal focus group sessions with quilters from the community to evaluate the system. The participants were enthusiastic about the system and also provided suggestions for its improvement.

Acknowledgements

Many people contributed to the success of this project and to the fullness of my experience here. My heartfelt thanks to you all:

To the many quilters who participated in focus groups and especially to Bette of the Simcoe County Quilt Shoppe, whose enthusiasm for this project was rivaled by none.

To the folk from Vermont who encouraged me to take this step, supported me in it, and gave me respite: Myra Cohen, Charlie Colbourn, Craig Damon, Alison Pechenick, Robert Snapp, Kathy Desjardins, and Cait Lloyd.

To the many dgp students and staff who taught me much and occasionally let me teach them as well: Wael Aboelsaadat, Gerard Baron, Hanieh Bastani, Anastasia Bezerianos, Patrick Coleman, Kevin Forbes, Irene Fung, John Hancock, Maciej Kalisiak, Alex Kolliopoulos, Azeem Lakdawalla, Joe Laszlo, Xia Liu, Noah Lockwood, Shahzad Malik, Mike McGuffin, Nigel Morris, Michael Neff, Mike Pratscher, Faisal Qureshi, Gonzalo Ramos, Abhishek Ranjan, Alan Rosenthal, Russell Schick, Patricio Simari, Maryam Tohidi, Chris Trendall, Fanis Tsandilas, Mike Tsang, Winnie Tsang, Dan Vogel, Jack Wang, Naiqi Weng, Daniel Wigdor, Mike Wu, Paul Yang, and Quinxin Yu. Especial thanks go to the "eccentric old codgers" among you, my fellow elfgroup students.

To the coffee klatch and lunch bunch gals who shared their stories and their desserts: Afra Alishahi, Faye Baron, Tobi Kral, Mahsa Moallem, Nina Theissen, Amber Wilcox-O'Hearn, and Jane You.

To the faculty who gave me special challenges and helped me meet them: Derek Corneil, Suzanne Stevenson, Diane Horton, and Tom Fairgrieve. Also to Ron Baecker who, as second reader, pressed me to claim my contributions and mean it.

Most especially to Eugene Fiume who was just the supervisor I needed – for enthusiastically supporting this unorthodox project, for valuing authentic applicability in research, and for caring more about students than computer graphics. Your consistent kindness, inexhaustible cheerfulness, and apparent imperturbability challenge us all.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Quilting is an art form in which amateur craftspeople explore geometry and color to create stunning designs in fabric. Bargello is a specific style within the world of art quilts defined by a unique construction process that imposes geometric constraints on the designs and gives them a characteristic appearance: row upon row of gracefully flowing curves composed solely of rectangular pieces. Currently, designing Bargello quilts is a time-consuming and labor-intensive process. Our goal is to provide a system that allows users to create Bargello designs quickly and easily.

We have developed a prototype computer-assisted design system for Bargello quilts. The system is composed of three constituent parts: an algorithm that produces graceful Bargello curves from sketched input data, a design module that allows users to create both traditional and contemporary Bargello quilt designs by sketching curves with a mouse or stylus, and a fabric selection module that allows users to explore fabric and color combinations using a database of real fabric textures.

In this chapter, we introduce quilting as an art form and describe the Bargello style in detail. We then discuss related work in the computer graphics literature and present the contributions made by this thesis.

1.1 The art of quilting

There is a common misperception, fueled perhaps by nostalgia for “simpler days,” that quilts are primarily utilitarian objects pieced together by elderly women out of leftover fabric scraps – but the truth is much more exciting. Quilting today is an art form in which amateur craftspeople of all ages work with beautiful high-quality fabrics to explore compositions in geometry and color, challenge and develop their technical abilities to fabricate the designs they envision, and gather socially to teach and learn, compete, and share their achievements. The work they create is often stunning.

Moreover, this has always been the case. The fact that quilts were traditionally used as bed coverings does not diminish the artistry and technical achievement involved in creating them. Over the past two centuries quilters entered their prize works in competitions at local agricultural fairs, and some surviving quilts from this period are signed in embroidery or ink, indicating that the creators viewed them as works of art. Advances in transportation have made possible the international juried competitions of today, and industrialization has allowed hundreds of thousands of modern quilters to create works that are purely decorative wall hangings, but quilting has always been a creative outlet for amateur artisans.

A few illustrative examples follow. The honeycomb quilt in Figure 1.1 is composed of 15,876 regular hexagons, each exactly 1” from point to point [1]. It was created circa 1815, using the *English template* method in which each fabric piece is wrapped around a paper template and whip-stitched to its neighbors. The (unknown) quilter painstakingly centered the printed design on each hexagon such that each piece cut from the same fabric looks identical. Figure 1.2 shows a quilt made circa 1840 using a technique known as *applique* [1]. Each flower spray in the central region of the design was carefully cut from an expensive chintz fabric and sewn onto a white background fabric with tiny embroidery stitches. The quilt in Figure 1.3 was created with the *block piecing* method between 1880 and 1900 [60]. Many small triangles, squares, and diamonds were cut from

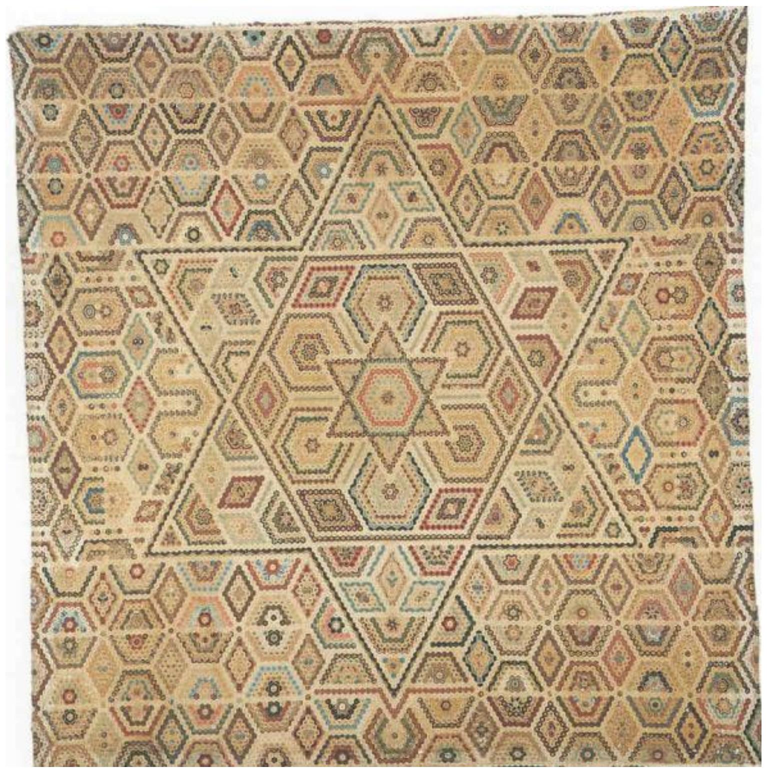


Figure 1.1: Honeycomb quilt, c. 1815, 108" x 108", from [1] page 54

various fabrics and stitched together into stars. Each star in the two concentric borders occupies a larger square, called a block. Each block was constructed separately, and the blocks were then stitched together to complete the quilt top. Figure 1.4 shows a free form quilt made in 1995 [62]. In this quilt, the fabrics representing scenery were first pieced together, and then the figures of children were appliqued on top. Although all but the first quilt example include *quilting stitches*, the stitching is most noticeable on the beach and around the border of Figure 1.4. These stitches are sewn through all layers of the quilt to secure the batting in place and to superimpose another subtle design element on the quilt top.

Notice that the colors in the fabrics of each successive example get brighter and more varied. In 1815, the few fabric dyes available were all subtle earth tones created from organic materials [1]. Beginning in the 1820's, a new class of mineral colorants were developed, including a colorfast green that appears in Figure 1.2. The first aniline

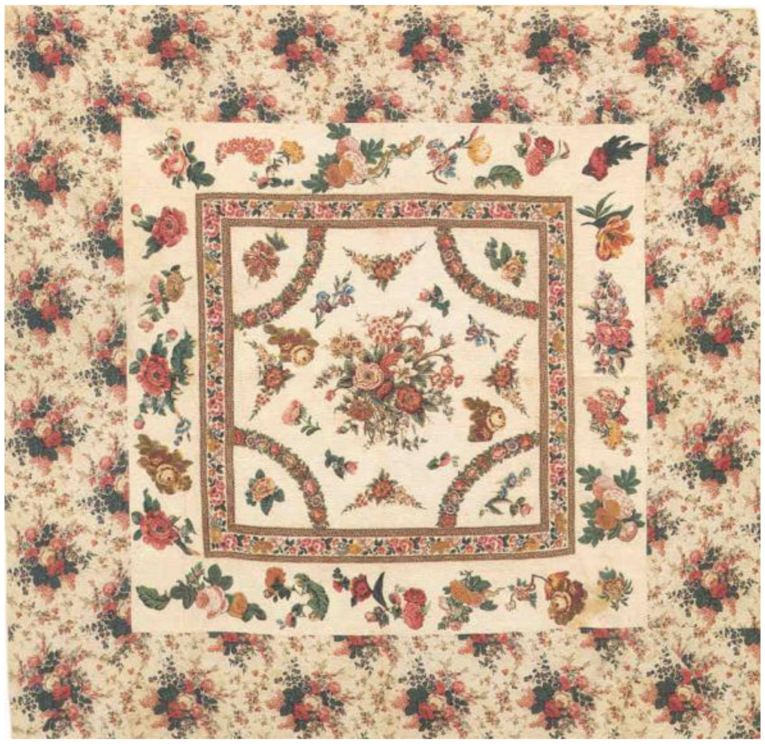


Figure 1.2: Chintz applique quilt, c. 1840, $89\frac{3}{4}$"x 86", Annie R. Smith, from [1] page 96

dyes were invented in 1865, and these can be seen in the brighter colors of Figure 1.3. Today fabrics are manufactured in a vast array of colors, from subtle shades to vibrant bursts in every possible hue. This adds a new dimension to the artistry and challenge of quiltmaking that did not exist in previous centuries.

Although individual colors can be defined mathematically in absolute terms, the appearance of a given color depends on the context in which it is viewed. Many factors affect the way that colors are perceived, including the ambient illumination level, the size of the color patch, and the colors surrounding it [46]. A given color will appear lighter when surrounded by dark colors, and darker when surrounded by light colors. Similarly, it will appear more red or yellow when surrounded by greens or blues, respectively, and vice-versa in both cases. Further, small color patches appear less saturated than larger ones. Figure 1.5 gives a striking example of the complex shifts in color perception that occur when colors are juxtaposed with one another in varying proportions: amazingly,

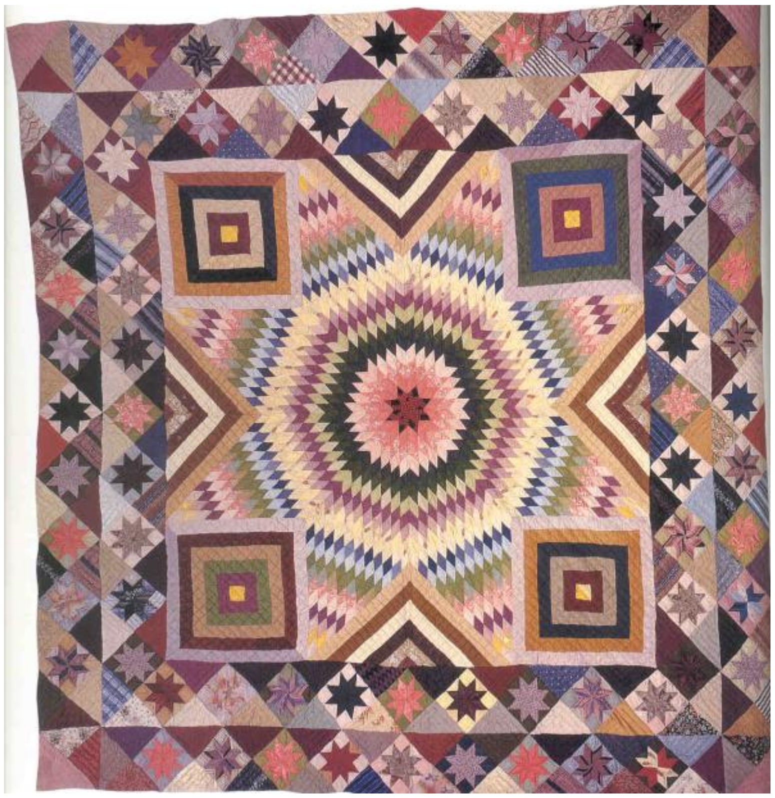


Figure 1.3: Pieced quilt, c. 1880-1900, 99” x 94 $\frac{1}{4}$”, from [60] page 74

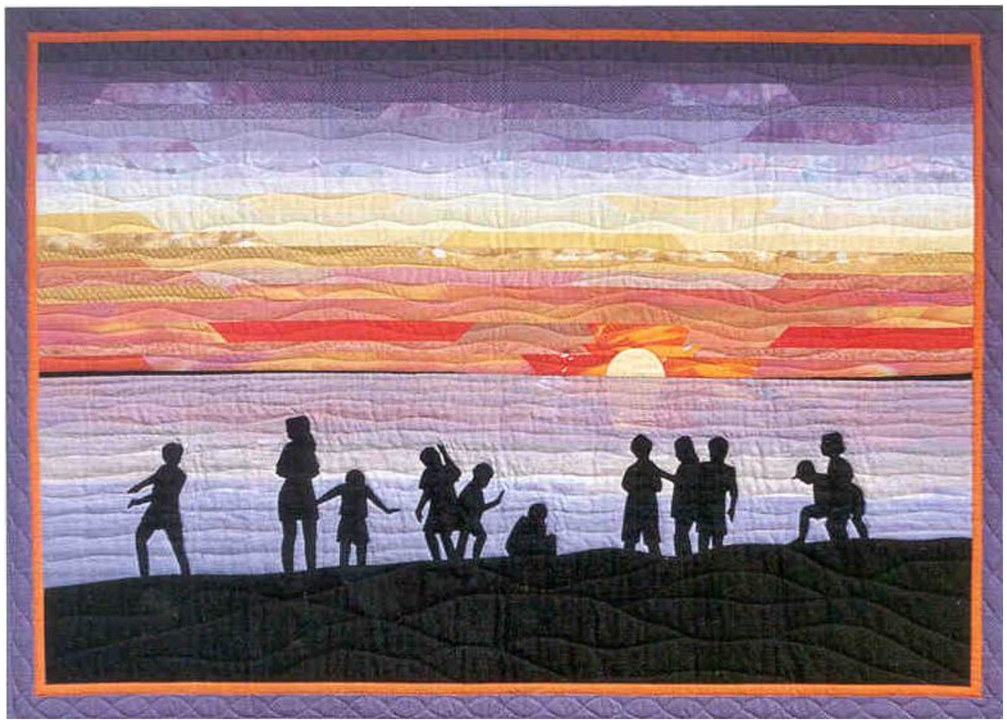


Figure 1.4: Landscape quilt, 1995, 50” x 36, by Helen Courtice, from [62] page 7

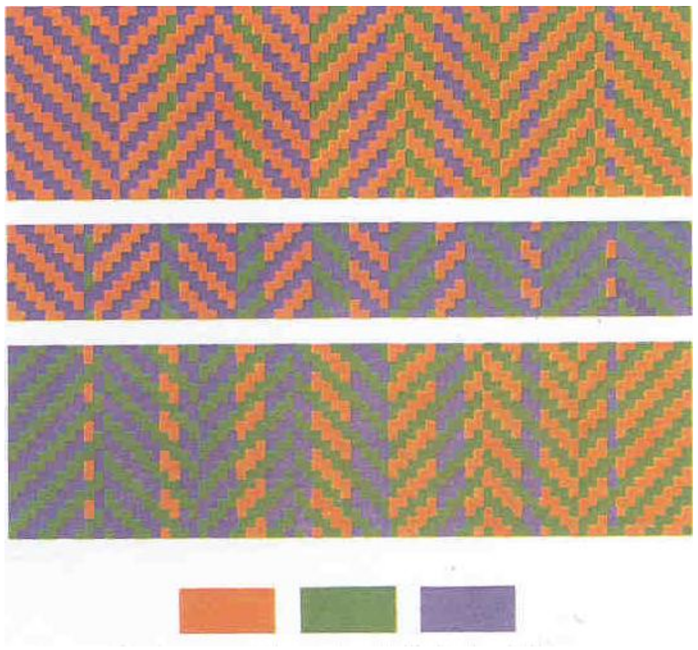


Figure 1.5: Simultaneous color contrast. Design by S. Harry. From [46] page 9

this figure is composed of only three colors. Designers who work with color must pay careful attention to these color shifting effects.

A quilter's task is to combine small pieces of several multi-colored fabrics to create a coherent whole. Clearly, the complexities of color perception complicate this task, as evidenced by the fact that quilters frequently speak of color in anthropomorphic terms, saying that a color "died" when placed next to another one, or that yellows and oranges "take over" in a quilt, or that solid colors should be avoided because they "jump out." Many books and workshops have been devoted to teaching quilters the art of color usage (e.g., [48, 62]), yet color remains one of the foremost challenges of quilting (and computer graphics), and even expert quilters claim that they "do not have good color sense." In part, this arises because comparing whole fabric bolts with one another in the fabric shop is insufficient to get an accurate impression of the final appearance of a completed quilt. As a result, quilters are frequently disappointed by finished quilts that look different than they envisioned. Software that allows quilters to view their quilt designs in the intended fabric textures before commencing new projects can help to alleviate this problem.

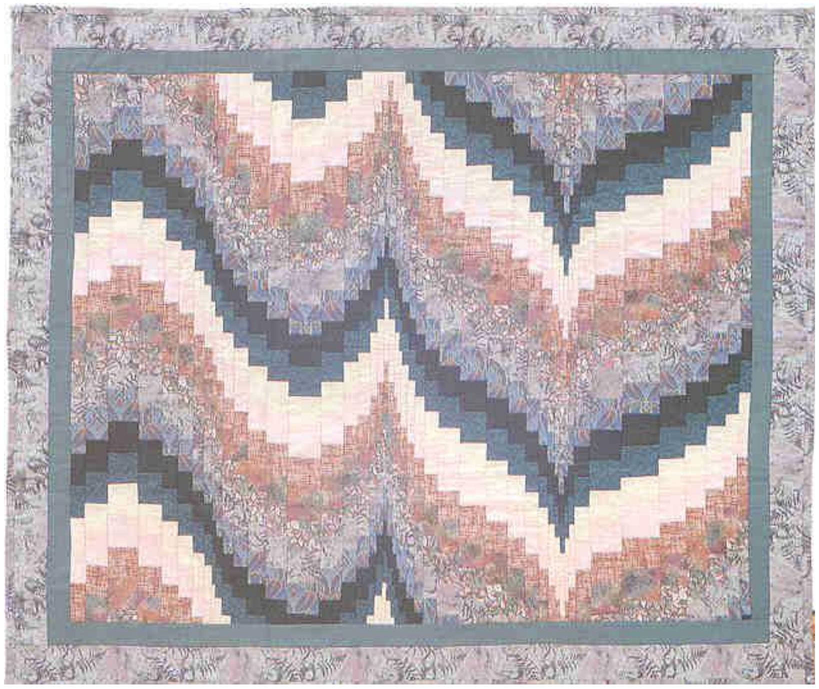


Figure 1.6: Traditional Bargello design, from [19] page 40

1.2 Bargello quilts

Definition of a style

Bargello is a specific quilt style defined by two intertwined characteristics. First, Bargello quilt designs frequently exhibit a characteristic appearance that is loosely evocative of the Italian Renaissance tapestry designs from which they descend: row upon row of gracefully flowing curves composed solely of rectangular pieces (Figure 1.6). Second, Bargello quilts are crafted via a clever construction method, highly tuned for both efficiency and precision, which results in designs that exhibit the characteristic Bargello appearance.

Unfortunately, there is no standardized vocabulary for the concepts intrinsic to Bargello quilt designs. Some terms are used consistently, but others are not. Therefore, we have adopted various terms for use in this text: some borrowed from existing Bargello texts, and others invented in collaboration with local quilters. They will be defined as they are needed. To begin, quilters describe the narrow extrema in Bargello curves as *peaks* and

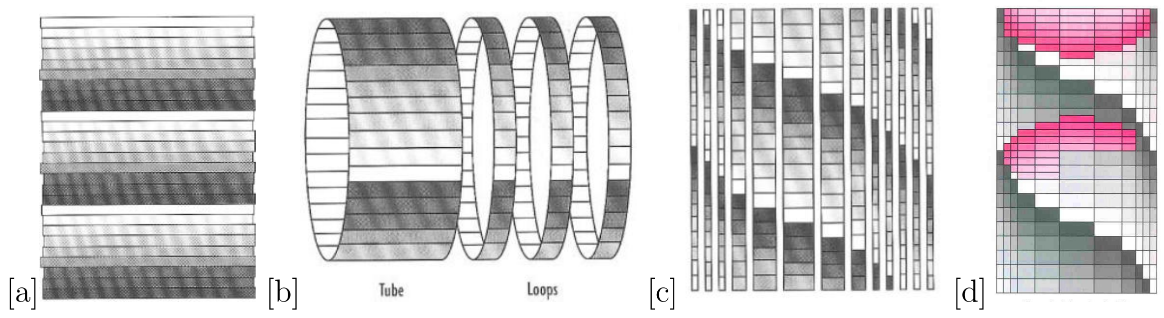


Figure 1.7: Bargello construction process, from [19] pages 5,7,31.

the wide extrema as *waves*.

Bargello construction process

To construct a Bargello quilt, one starts by cutting long narrow strips from each of several fabrics [19, 61]. These are called *color strips.* The color strips are then stitched together lengthwise, into *color runs* or *color strata* (Figure 1.7[a]). Frequently, three or four color runs are stitched together, creating a repeating pattern, and then the bottom color strip is wrapped around and stitched to the top color strip, forming a tube (Figure 1.7[b]). Next, patchwork loops of various widths are created by cutting perpendicular slices off the tube. As a result, the individual fabric pieces that compose each loop are rectangular. Finally, each loop is opened out into a linear strip, called a *vertical strip*, by removing the stitching in one seam or by cutting one fabric piece in two (Figure 1.7[c]). Note that if each loop were opened at the same point in the repeating pattern, the resulting vertical strips would be identical except for their widths. Instead, consecutive loops are opened at consecutive fabric positions, creating vertical strips that exhibit different phases of the same repeating pattern. At this point, the vertical strips may be sewn together lengthwise to complete a quilt top consisting of a single set of uninterrupted parallel curves, as in Figure 1.6. Alternatively, one can *embellish* the design by modifying the vertical strips before stitching them together (Figure 1.7[d]). This process will be described in more detail below.

The Bargello construction process is extremely efficient in that a finished quilt top may consist of hundreds of individual pieces, yet be produced with relatively few seams. Let n be the number of colors in a stratum, m be the number of times the stratum is repeated, and s be the number of vertical strips in the design. The quilt top will have nms pieces, but can be constructed with $(nm + s)$ seams. In contrast, a naïve construction method in which each piece is cut and stitched individually requires $(nms + s)$ seams. Moreover, the Bargello construction method results in seam lines that are much straighter, and a finished appearance that is much crisper, than would be possible if each fabric piece were stitched into the design individually.

Design constraints

The Bargello construction process imposes several constraints on Bargello designs: all fabric pieces are rectangular, all fabric pieces are aligned in vertical columns, and the fabric pieces that comprise a given color strip all have the same height. In addition, although it is not required for construction, it is common in Bargello designs that all color strips have the same height and that the fabric pieces in adjacent vertical strips are either aligned horizontally to form rows (*matched seams*) or offset by one half the height of a color strip (*staggered seams*). As a result, there is frequently only one geometric degree of freedom in a Bargello design: the width of the vertical strips.

At this point, the reader may wonder whether Bargello designs with any degree of complexity or visual interest can be created at all. The answer is an emphatic "Yes." Many artistic styles are defined by design constraints, and the artistic challenge is to explore the space and generate designs that fascinate even while adhering to the constraints. By way of analogy, consider the remarkable number and variety of poems that have been generated within the constraints of Haiku.

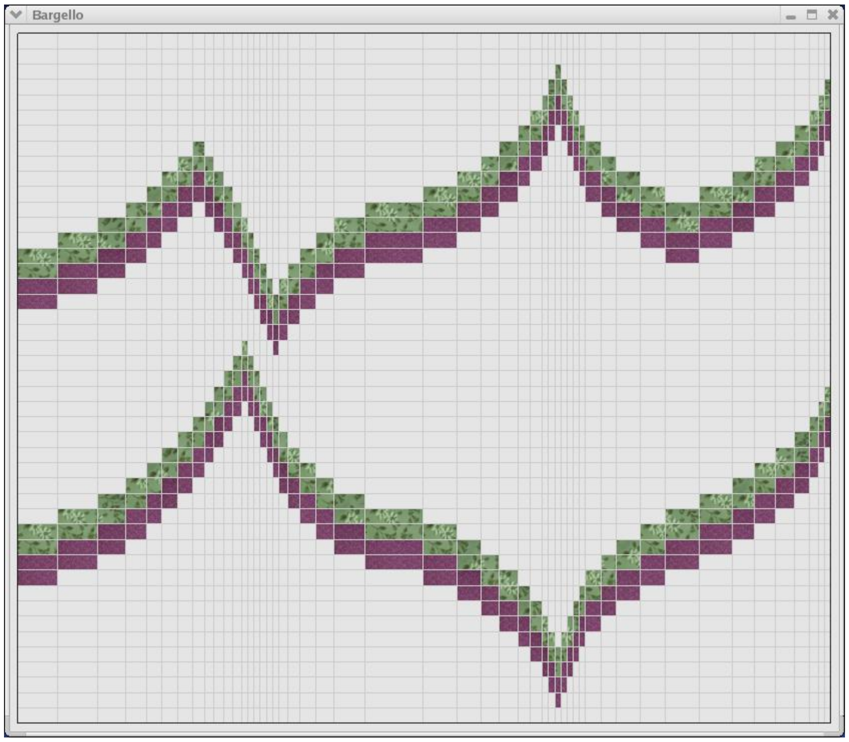


Figure 1.8: Local parallelity and symmetry in Bargello curves.

Design possibilities

For a moment, let us suspend our knowledge of the Bargello construction process beyond the fact that it imposes the constraints just described, and consider the following. Each color strip in a Bargello design forms a curve that runs through the design. Because the pieces in the color strip are corner-connected, the tangent of each curve as it passes through a given fabric piece is defined by the height and width of the piece. In a design where all color strips (and therefore all fabric pieces) have the same height, since the fabric pieces in a given vertical strip all have the same width, the tangent of each curve as it passes through a vertical strip is either parallel or symmetric to every other curve that passes through the same strip.

This can result in designs consisting of a single set of parallel curves or in vertically symmetric designs. However, these are not the only possibilities. The design constraints only require that the curves be locally parallel or locally symmetric. As shown in Figures

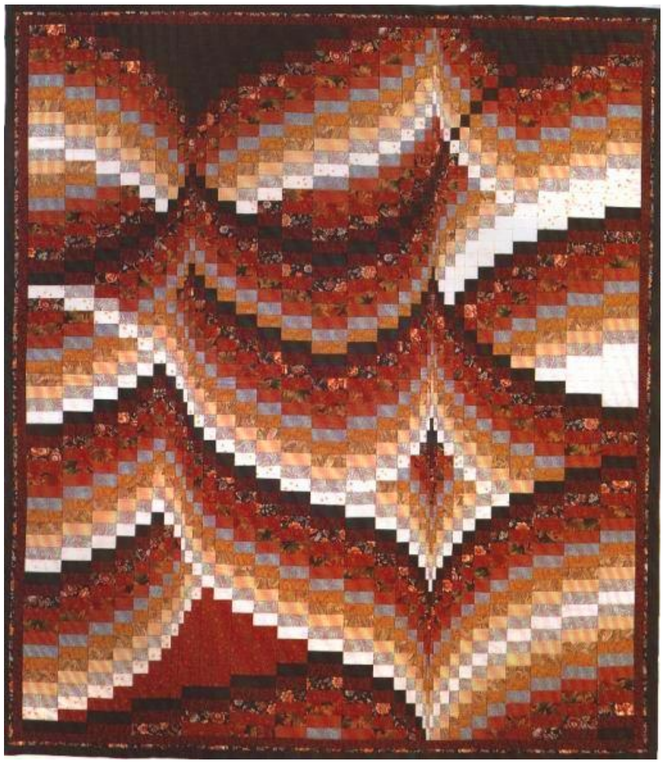


Figure 1.9: Contemporary Bargello design, from [19] page 72

1.8 and 1.9, if two curves contain extrema that are not aligned horizontally (i.e., their apices do not fall within the same vertical strips), then the curves will be neither globally parallel nor globally symmetric. This allows Bargello designs to become arbitrarily complex, including individual color strips that run parallel to one another in some regions and symmetric to one another in others. Similarly, while Bargello designs frequently consist of color strips that are connected over their whole length, this is not required.

But how are quilts with complex designs such as these to be constructed? As mentioned previously, embellishments can be added to Bargello quilts by modifying the vertical strips before they are sewn together (Figure 1.7[d]). For example, one can

- create vertically symmetric designs by dividing each vertical strip in half, reversing the order of one segment, and stitching the two segments back together,
- create designs with disconnected curves by opening adjacent vertical strips at non-

consecutive positions in the repeating pattern such that there is a discontinuity in the phases of adjacent vertical strips,

- create designs where the color strata do not repeat periodically throughout a vertical strip by removing a sequence of fabrics and replacing it with some other sequence. This type of embellishment can be used to create designs in which color strips switchback over themselves or designs in which multiple color strips run in counterpoint to one another.

This list of embellishments only scratches the surface of what is possible. However, it is clear that the more embellishments a given design calls for, the more effort is required to physically construct the quilt. It is sometimes said that Bargello quilters do more "unsewing" than sewing. As a result, quilters often have strong feelings about this design style – one way or the other.

Bargello design process

Edie [19] describes the process of designing a Bargello curve as follows. First, the desired curve is sketched on graph paper, and then in another location on the paper, an appropriate set of grid squares is penciled in to represent the curve. This process is more difficult than it sounds; Edie warns novices that it takes practice, suggests they have an eraser handy, and adds that curves will look more graceful if they do not include "jumps" in tile width.

In the language of computer graphics, *scan conversion* is the process of selecting a set of discrete pixels from a two-dimensional pixel "raster" with which to depict a line, curve, or other geometric object, and the resulting pixel set is called a *rasterization* of the object. Thus, the task at hand for a Bargello designer is to create an attractive rasterization of the sketched curve.

One difficulty arises from the fact that graph paper is of much lower resolution than the sketched curve, so a naïve scan-conversion algorithm (i.e., penciling in every grid

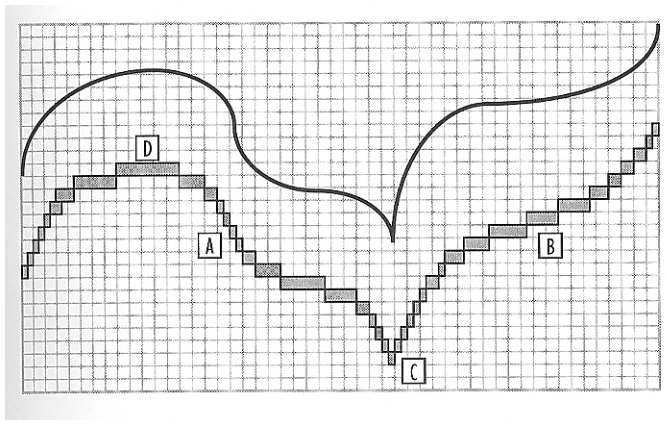


Figure 1.10: Bargello design process, from [19] page 13

square the curve passes through) can result in an unattractive rasterization. If the apex of a curve extends nearly but not quite to the top of a grid row, a naïve scan-conversion algorithm will result in an overly wide tile at the apex. Similarly, if the curve grazes the highest row, it will result in an overly narrow tile. Neither of these rasterizations represents the shape of the sketched curve well. In addition, the sketched curve may include a segment that is too steep to be represented by a set of corner-connected tiles, yet this connectivity is necessary if the quilt is to be constructed without embellishments. Finally, large width differences between adjacent tiles can give a curve a blocky, rather than a graceful, appearance.

Therefore, an accomplished Bargello designer will create a stylistic abstraction of the sketch, rather than a mechanical scan conversion. Edie [19] illustrates this with a figure (reproduced in Figure 1.10) that gives an example sketch and a corresponding suggested tile path. In several places, the suggested tile path deviates from the path followed by the sketch. For example, near labels "A" and "C" the suggested tile path is less steep than the sketch, as is necessary if the tile path is to be corner-connected. Near "B" the tile widths in the suggested tile path vary gradually, whereas a scan conversion would have produced large width differences between adjacent tiles. Finally, near "D" the uppermost tile of the arc has been narrowed to give the curve a more gently rounded appearance.

We are aware of two commercial software packages that currently allow quilters to create Bargello designs [4, 20], but neither of them allow users to sketch Bargello curves with a mouse or stylus. Rather, in both cases, the geometry of the desired curve must be specified by manually adjusting the width of each vertical strip, either by dragging a slider or typing data into a dialog box. In our experience, it is difficult to create the desired curve in this manner. Essentially, these systems require users to perform a manual scan conversion of an imagined curve into an invisible grid. Again, in our experience, it is simpler to design the curve on paper and then enter the completed geometry into the program. Even then, the data entry process is time-consuming. Thus, we find that these systems provide little support for the artistic design process, and they certainly do not lend themselves to design exploration.

A software solution that automatically generated graceful Bargello curves from sketched input, while respecting the physical and design constraints of the Bargello style, would constitute a major improvement over the current state-of-the-art Bargello design process.

1.3 Computer graphics: related work

We now turn to the computer graphics literature, seeking previous work related to the problems we have posed. Given that quilting is a textile craft and Bargello designs tend to have a characteristic appearance, at first glance our problem seems connected to the body of work that simulates the appearance of specific artistic styles and craft media. Such endeavors are currently quite popular in the computer graphics community, and several colleagues have suggested that we add features leading to the "Bargello-ization" of existing images. This trend was started in 1990 when Haeberli introduced a "paint-by-numbers" system [26] that reproduced source images in a painterly style as the user swept a mouse over the image. In his system, since the size of the simulated brush strokes varied in proportion to the mouse speed, the user could exert a small amount of control over the appearance of the final image. Subsequently, the trend has leaned toward reproducing images completely automatically in an ever expanding array of styles and media. By now, a given source image can be reproduced in various painterly styles [29] or used as the basis for a plane tiling reminiscent of the work of M.C. Escher [35]. Alternatively, it can be reproduced with the look of a plethora of media, including watercolor, pen and ink, copperplate engraving, mosaic tilings, stained glass, and Batik [16, 53, 45, 27, 43, 64].

The primary purpose of this body of work seems to be self-entertainment on the part of the researchers. However, these capabilities could also prove useful in set production for animated films or computer games, or even for creating personalized prints for interior decoration. These are reasonable goals, yet none of them speak to the problems we have posed. Certainly, one can imagine that a quilter might want to make a quilt reproduction of an existing image; indeed, Figure 1.4 may well have been based on a photograph. However, Bargello is probably not a good choice of quilt style in which to do so. Because Bargello designs are always aligned in columns, their peaks and valleys express a preferred orientation that is not typical of most natural images. Further, while Bargello could theoretically be used to recreate any image in pixelized form (assuming an unlimited number of construction embellishments), mosaic tilings or embroidery are both more appropriate media for producing pixelized images. Neither of them depend on any degree of connectedness between cells of similar colors, while Bargello does. Without the connectivity inherent in Bargello designs, the efficiency of the Bargello construction process is lost. Therefore, we do not seek to reproduce arbitrary images as Bargello quilt designs.

Next, we turn our attention to interactive software systems for drawing, painting, and sculpting. These systems provide a setting in which users generate virtual artworks, and as such, they seem connected to our system for creating Bargello designs. The earliest paint programs were developed over the course of the 1970's [55]. These allowed users to draw digital pictures with a mouse or stylus, by compositing a small rasterized brush “footprint” with image pixels at the current input location. Subsequent work has focused on realism, simulating the behavior of the paint, the brushes, and their interaction with virtual media. Brushes were given bristles that received and deposited ink individually and deformed under pressure [58, 41]. Paint models simulated the flow of wet paint from pixel to pixel, including both surface flow and capillary action along paper fibers [13, 16]. Eventually, a complex physically-based simulation modeled multiple brush shapes, multiple paint layers, and paint transfer from brush to surface and back again [6].

Interactive sculpting systems have followed a similar trajectory. In 1990, Naylor produced the first interactive 3D sculpting system, using a solid modeling approach in which analytically defined shapes were combined via boolean set operations to create arbitrary polyhedra [44]. Voxel-based sculpting systems soon prevailed, however, since they allowed the sculpted object to undergo arbitrary topology changes [24]. Over the years, more tools have been added to the virtual sculpting tool set [59, 21], and the clay has taken on more realistic behavior [18]. Further, haptic force-feedback input devices have been introduced in both painting and sculpting systems. Thus, by now, these systems not only allow users to create virtual artworks, they also simulate the experience of working with the tools and media in each of these crafts.

There are many reasons why amateur artists might prefer to paint or sculpt within the computer than outside it. First, virtual paint and clay are less messy than their physical counterparts. Second, interactive software systems are free to ignore undesirable behaviors of physical media. For example, virtual paint and clay do not dry before the work has been completed, and virtual clay never needs to be kiln fired. Interestingly, the lack of physical realism is a significant part of what makes these virtual crafts attractive. Of course, a disadvantage of virtual art is that the tactile pleasure associated with working with real materials and fabricating real objects is lost. One cannot enjoy a morning cup of coffee in a virtual mug. Increasing the realism of the simulation does nothing to alleviate this, and one wonders how far to push the realistic simulation of activities that are neither unsafe nor difficult to experience in the physical world. In such cases, users who seek a “realistic” experience may be better served by a real one.

Our interest is in real fabric, real quilts, and real quilters. Rather than simulating the experience of quiltmaking, we seek to assist quilters in the process of creating quilt designs which may subsequently be used to produce physical quilts. To this end, we must provide a visualization that is sufficiently accurate to allow users to make informed design decisions, but beyond that point, additional physical realism does not further our goal. As such, the system presents real fabric textures, but does not simulate the physical behavior of cloth.

Currently, designing Bargello quilts is a time-consuming and labor-intensive process; our goal is to provide a system that allows users to create Bargello designs quickly and easily. With a shortened design cycle, users will be encouraged to explore new ideas, keeping some and discarding others, and the resulting evolution of design ideas will likely culminate in better final outcomes. As such, the true lineage of this work is in the literature of sketch-based design.

The guiding principle in the field of sketch-based design is to allow users to create designs via free form sketches, bypassing the laborious process of individual control point manipulation that is required by most computer-aided design (CAD) systems. The resulting designs necessarily lose the high precision offered by CAD; however, in many settings high precision is not required, and the artistic benefits can be substantial. Designs can be created more quickly, and designers can focus more attention on the creative aspects of the design process, when they are freed from mandatory micro-manipulation of their created objects. In addition, sketch-based design systems tend to be easier to learn, and therefore more accessible to beginning designers, than CAD systems. However, the less information the designer articulates, the more the system needs to infer since most interesting designs contain more internal richness than a simple sketch can convey on its own. Therefore, research in this field has explored various kinds of inferences that can be drawn in various contexts.

Some sketch-based systems focus on recognizing low-level primitives and the relationships between them. These systems do not presume particular application domains, but they do make assumptions about attributes that are desirable in drawings. An early example was given by Pavlidis and Van Wyk [47]. In their system, sketched strokes are resolved into lines, and the lines are examined for each of a set of possible relationships between them. When approximate relationships are recognized, the system "beautifies" the drawing, adjusting the lines to make the relationships precise. For example, lines with similar orientations are brought into exact alignment, lines that almost meet are brought into contact with one another, and a set of lines may be repositioned such that their endpoints are aligned horizontally or vertically. *Pegasus* [33] expands on this work in two ways. First, it recognizes an extended set of relationships among lines that includes symmetries, congruences, and perpendicularities. Second, when multiple conflicting relationships are discovered, it handles the ambiguity by suggesting multiple beautified line placements and letting the user specify the line that was intended.

Fluid Sketches [3] focuses on recognizing individual strokes, rather than the relationships among them, and classifies each stroke as a line, circle, or box. However, the real contribution of this system is that it continuously re-examines strokes as they are being drawn, rather than waiting until the pen is lifted to attempt recognition. When each input point is received, the system reconsiders the stroke classification, finds the best-fit instance of the class, and transforms the sketch a small distance toward the current ideal shape. As a result, a roughly drawn circle will gradually be transformed into a tidy circle in the wake of the pen. Moreover, an ambiguous sketch may begin transforming toward a circle but later converge on a box as more information is received.

Another direction that has been explored in sketch-based design is the creation of 3D objects based on 2D input strokes. To manage the inherent ambiguities in this problem, each system must make limiting assumptions about the domain of shapes it can create. *SKETCH* [65] allows users to create and manipulate three-dimensional geometric solids, based on analytic primitives, with gestural commands. *Teddy* [34] makes an astonishing leap, allowing users to create 3D objects based solely on free form sketches. In this system, free form strokes are interpreted as object silhouettes, and 3D geometry is inferred from the silhouettes. To make this possible, Teddy makes strong assumptions about the objects in its domain, and under those assumptions all objects acquire a similar symmetric "rotund" morphology. Nearly circular silhouettes beget nearly spherical objects, while oblong silhouettes produce more cylindrical results. The authors point out that many natural objects, such as heads and limbs, fit this description reasonably well, and in practice the system is well suited to drawing 3D cartoon-like characters.

The last category of systems we consider are those intended for domain specific, early-stage design. These systems intentionally retain the "sketchy" appearance of the original input strokes, rather than beautifying them, in order to capture and reflect the amorphous nature of early stage designs. It is hoped that retaining ambiguities will allow designers to continue developing their ideas abstractly, rather than leading them to consider design specifics prematurely. *SILK* [39] is intended for early-stage user interface design, and this domain specificity allows it to recognize high-level objects such as scroll bars and buttons in the sketch. Once these objects are recognized, they are imbued with object-specific behaviors. For example, scroll bars can be dragged, and buttons become highlighted when the mouse hovers over them. This is an especially fun result given that SILK renders these objects in their original "sketchy" form. The *Electronic Cocktail Napkin* [25] extends the ideas in SILK by allowing users to specify new application domains and teach the system stroke configurations relevant to each domain. In addition, the system intentionally retains ambiguity in drawings by simply not categorizing sketched objects that do not match known configurations well enough. This allows users to draw items they have not yet taught the system, and it also allows for in-between stages in which some, but not all, of the strokes needed to specify a configuration have been drawn.

Our Bargello design system applies a sketch-based design approach to a new application domain [12]. Similar to [34], in this domain all input curves are known *a priori* to be Bargello curves, so our task is not object classification; rather we seek the best-fit instance of each input curve and create a 2D Bargello virtual fabric tiling from it. Like [3], we continuously re-examine the input curve and update the resulting tiling as the sketch is being drawn. To ensure interactive response rates, a lightweight beautification algorithm is used as the sketch progresses, and a more rigorous algorithm is applied after the pen is lifted. Like [39] our focus is on real users creating real designs, and as such, we encounter and respect the constraints within the application domain.

Finally, we also draw on previous work in curve fitting and rasterization, which will be discussed more fully in Chapter 2. These fields have been well established for quite some time, so we do not seek to improve on existing methods that solve previous problems. Rather, we draw on lessons learned and apply them to a new domain: creating graceful curves composed of rectangular tiles from sketched input.

1.4 Contributions

This thesis makes specific technical and broad systems contributions.

Our broad contribution is a prototype system in which users can design Bargello quilt patterns quickly and easily by sketching curves with a mouse or stylus. The system addresses an authentic need, given that the current design process for Bargello quilts is laborious and time-consuming. The system contains three constituent parts: an algorithm that creates visually smooth curves composed solely of rectangular primitives, a design module in which users can create both traditional and contemporary Bargello quilt designs, and a fabric selection module in which users can explore fabric and color combinations using a database of real fabric textures. However, the problem of fabric and color selection in the domain of quilt design is an open-ended one which is not fully solved in the thesis. Finally, we conducted an informal evaluation of the system via a set of focus group sessions with quilters from the community. We found that participants were quite enthusiastic about the system, and they also provided suggestions for its improvement.

The technical contribution of the thesis is an algorithm that generates graceful curves composed of corner-connected, axis-aligned rectangles. Note that all tiles in a Bargello design must be aligned in vertical columns, so the constituent rectangles must be axis-aligned; they cannot be rotated to align with the curve tangents they approximate. Neither can anti-aliasing techniques be used, since the Bargello style intentionally retains the (very large) "jaggies" created by large corner-connected tiles. Therefore, what we seek is not a curve that is literally smooth, but rather an optical illusion of sorts. The eye will merge corner-connected tiles into a large-scale curve, and we ensure that the result has a graceful appearance. The solution to this problem required several steps: the Bargello design constraints were analyzed and codified, a precise definition of what constitutes a "graceful" Bargello curve was generated, and an algorithm was developed that produces graceful Bargello curves from sketched data points, while maintaining both physical and design constraints from the Bargello quilt domain. The algorithm incorporates novel curve fitting, rasterization, and beautification schemes specific to Bargello curves. Finally, the algorithm was extended to cover the case of multiple independent curves (or curve segments) that share the same column space within a Bargello design. This problem is more difficult since the independent curves must be coordinated to create a single set of vertical Bargello design columns.

The remainder of the thesis proceeds as follows. Chapter 2 discusses technical background on curve-fitting and rasterization as well as our approach to each of these tasks in the context of Bargello curves. We introduce our software implementation in Chapter 3, followed in Chapter 4 by a description of the focus group sessions that were conducted and the lessons that were learned. Chapter 5 concludes with a discussion of our results and possible directions for future work.

Chapter 2

Bargello Curve Generation

The core technical challenge addressed by this thesis is to generate graceful Bargello tilings based on the data points received as users draw curves with a mouse or stylus in our system. We solve this problem in two phases. First we generate a concise mathematical primitive, a piecewise parametric curve, that approximates the discrete input data. We then rasterize the curve into a low-resolution Bargello grid to produce a stylistic rendering consisting of corner-connected rectangular tiles. This tile path may then be propogated in vertically-aligned columns to form a complete Bargello tiling.

In this chapter we discuss the two constituent problems, fitting curves to data points and rasterizing mathematical primitives, in some detail. In each case we provide technical background, discussing previous work on problem instances similar to our own, and then we present our solution, which is tailored to the specific task of generating Bargello designs from interactive input.

2.1 Fitting curves to data points

Given a sequence of discrete data points, we would like to find a compact mathematical representation, a curve, that reflects the continuous process underlying the discrete data. This problem is intriguing because it does not have a unique solution. As a result, many curve formulations and many algorithms have been developed that provide a variety of solutions, each tailored to the needs of specific application domains. For example, interpolating curves intersect every data point, and they are useful when the data are known to be exact, but when the data are imprecise or noisy, curves that only approximate the data provide more accurate representations of the underlying generative processes [38]. Similarly, explicit curve formulations (i.e., those of the form $y = f(x)$) represent single-valued functions, and they are useful for modeling many kinds of experimental data [38]. However, in computer graphics parametric curve representations (i.e., those of the form $F = (x(t), y(t))$) are used almost exclusively since they can model curves that are not single-valued in any coordinate system; such curves are common in this field [23].

Piecewise polynomials are commonly used for modeling data since they provide a compact representation, and they are easy to evaluate and differentiate. A piecewise polynomial consists of a sequence of polynomial segments joined together at *knots*, and it is said to be C^n*-continuous* if it is constructed such that its n^{th} derivative is continuous at the knots. Piecewise cubics are popular since they are the lowest degree polynomial that allow variation in curvature. Higher-degree polynomials are prone to excessive oscillation.

Within the family of piecewise cubic polynomials, there are still many formulations to choose from. Cubic splines, including cubic B-splines, typically have C^2 continuity, which makes them ideal for applications in computer-aided geometric design where curves with a high degree of smoothness are needed for manufacturing purposes [17, 5]; however, C^1 continuity is generally sufficient for visual smoothness. A formulation that allows tangent discontinuities is needed for applications, such as Bargello, in which curves contain corners or cusps. While B-splines can be made to exhibit tangent discontinuities by superposing multiple knots in the same location, the Hermite curve formulation is more convenient for this purpose. Since each Hermite curve segment is explicitly defined by two endpoints and the tangents at those points, C^1 continuity can be ensured by equating the tangents on adjacent segments, and tangent discontinuities can be allowed by refraining from doing so.

In addition, in many computer graphics applications users are given the ability to manipulate existing curves via a set of *control points*, and each curve formulation defines its control points differently. Manipulating B-splines can be awkward for novice users since their control points are not interpolated. In contrast, Catmull-Rom splines do interpolate their control points, so the curve can be manipulated by dragging “handles” on the curve itself. However, Catmull-Rom splines are always C^1-continuous, and while it is easy to generate a Catmull-Rom spline that interpolates every data point, because this formulation consists of overlapping curve segments, it is cumbersome to generate one that approximates the input data. All in all, the Hermite formulation is a good choice for our purpose: it can exhibit C^1 continuity or tangent discontinuities, it provides an intuitive control mechanism since both the position and the tangent at the endpoints of each curve segment are interpolated, and it is affords a tractable least-squares error function with which to approximate input data points.

Even within the relatively narrow realm of piecewise Hermites, a variety of algorithms have been developed to generate curves from data points. Each of these algorithms must solve two constituent problems: it must choose the number and placement of the knots, and it must generate curve segments between the knots. Various algorithms accomplish these tasks in various ways, reflecting the different assumptions and requirements of different application domains.

We next examine a few specific Hermite curve-generation algorithms in more detail to get a taste for the problem and the variety of solutions possible. Then we present our own solution to this problem in the context of Bargello curves.

2.1.1 Text character contours

Plass and Stone developed a curve-fitting algorithm for a situaton in which the data points were known with high precision [49]. Their aim was to generate an improved representation for text characters that had previously been stored as bitmaps. A bitmap character representation requires a large amount of storage because a value must be stored for each pixel in the rectangular area covered by the character, and separate bitmaps must be stored for each size instance of the character. In contrast, a piecewise curve that models the contour of a character requires much less storage and provides more flexibility since it requires relatively few stored data points (knots), and it can be transformed mathematically to produce characters in various sizes and orientations. Thus, the authors proposed an algorithm to fit piecewise curves around the contours of characters stored in bitmaps.

In this application, the input data was known to contain a small amount of noise due to rasterization effects, yet it was nevertheless considered to be highly accurate given that each character had been carefully designed at the individual pixel level to give it a particular appearance; thus, a high degree of visual fidelity to the data was required. At the same time, each text character only needed to be processed once, so computational expense was not a concern. Therefore, the authors sought a globally optimal solution, intentionally sacrificing computational speed in favor of precision.

The proposed algorithm generates a piecewise parametric Hermite curve which may contain corners but is G^1-continuous between corners. *Geometric continuity* is a concept that arises only in parametric curve representations. The tangent of a parametric space curve is vector-valued: $F'(t) = (x'(t), y'(t))$. However, only the projection of the curve is visible in the xy plane, so the tangent of the visible curve, $y'(x) = \frac{y'(t)}{x'(t)}$, depends only on the relative sizes of the two vector components. Therefore, a piecewise parametric curve will appear to have a continuous tangent if the tangent vectors of adjacent segments have the same direction: they need not have the same magnitude. In practice, this means that adjacent tangent vectors can differ by a positive scale factor, and the curve will remain visually smooth. A piecewise curve with this property is said to be G^1-continuous.

To generate individual curve segments, the authors proposed an iterative two-step algorithm that simultaneously solves for the correct parameterization of the data and for the curve segment given that parameterization. In this approach, each data point is initially assigned a parameter value based on chord-length: the first and last data points are assigned $t = 0$ and $t = 1$, respectively, and each intermediate point is assigned a value that reflects its relative distance from the first point. Then the two-step iteration begins. In the first step, given the current parameterization, a curve segment is generated via least-squares approximation. In the second step, the parameter value associated with each data point is updated by finding the closest point on the newly generated curve and assigning the parameter value at that location to the data point. The iteration converges since each step will either reduce the total distance between the curve and the data points in parametric space, or leave that distance unchanged.

Each step in the algorithm warrants further discussion. First, least-squares approximation can be used for generating constrained or unconstrained curve segments, and in the course of their overall algorithm the authors do both. In the constrained case, they solve for curve segments between known knots with known tangent directions at those knots. If an explicit curve representation were being used, there would be no free coefficients left! However, in a parametric representation, it is possible to re-state the Hermite basis functions such that for each endpoint, one coefficient governs the direction of the tangent vector and another governs its magnitude. In this way, the authors retain two free coefficients (the magnitudes of the two tangent vectors) to be determined via least-squares approximation.

Second, to update the parameter value associated with each data point, the closest point on the newly generated curve must be found, which is itself a non-trivial problem. The squared distance between a given point (x_i, y_i) and an arbitrary point on the curve $F = (x(t), y(t))$ is given by

$$D^2 = (x(t) - x_i)^2 + (y(t) - y_i)^2 \ . \tag{2.1}$$

To find the point on F closest to (x_i, y_i), Eq. 2.1 is differentiated and set to zero, yielding a fifth-degree polynomial in t:

$$(x(t) - x_i)x'(t) + (y(t) - y_i)y'(t) = 0. \tag{2.2}$$

To find the root of Eq. 2.2 nearest (x_i, y_i), the authors initialize t to the value currently associated with (x_i, y_i) and then apply Newton's method [14]:

$$t^{(j+1)} = t^{(j)} - \frac{f(t)}{f'(t)} = t^{(j)} - \frac{(x(t) - x_i)x'(t) + (y(t) - y_i)y'(t)}{x'(t)^2 + y'(t)^2 + (x(t) - x_i)x''(t) + (y(t) - y_i)y''(t)} \; . \tag{2.3}$$

Recall that the authors' objective is to fit curves to character fonts stored as bitmaps. The high-level algorithm proceeds as follows. First, since many characters include sharp corners, the data points are scanned to locate corners, and knots are placed at corner points. Corners are recognized as those data points where the internal angle between the point and its neighbors is less than 135^{O}. Next, to reduce the amount of computation necessary, some of the remaining data points are selected as potential knots. To determine tangents for each of the non-corner potential knots, an unconstrained curve is generated from a set of neighboring data points centered at the knot, and the tangent at the closest point on the resulting curve is taken to be the tangent at the knot.

Finally, a dynamic programming algorithm [15] is used to select the optimal set of knots from the larger set of potential knots. The cost e_{jk} of placing a curve segment between knots j and k is determined by generating a constrained curve segment between the two knots and adding a constant τ to the least-squares error over the curve segment. The effect of τ is to impose a cost for the number of segments used. The minimum cost E_{ik} associated with any of the possible piecewise curves that span knots i through k is given by the following recursive cost definition:

$$E_{ik} = min(E_{ij} + e_{jk}) \text{ for } i < j < k \; . \tag{2.4}$$

Ultimately, the piecewise curve used to represent the input character is the union of the minimum cost piecewise curves between adjacent corners.

2.1.2 Experimental data streams

Ichida et al. addressed a different problem [32]. Their aim was to develop an algorithm that could be used in a variety of experimental situations to model the underlying process responsible for generating a given sequence of data points. They anticipated that such data would contain random noise and postulated that the ideal curve would be one that used as few segments as possible (in accordance with the principle of Occam's Razor) while retaining the property that the measured data points are randomly distributed about the curve.

In the language of statisticians, the *residual* associated with a given data point is the (signed) vertical distance between that point and the approximating curve. Note that a curve with enough segments can be made to interpolate every data point such that every residual is zero; however, such a curve is not desirable in this setting since it models the noise in the data. On the other hand, a curve with too few segments will not have the flexibility to follow even the underlying trend of the data, and it will take "shortcuts" such that several consecutive data points fall on the same side of the curve in one or more regions. In this case, it is said that there is a "trend in the residuals," a situation which can be detected by examining the sum of the residuals in local neighborhoods along the curve. When the data points are randomly distributed about the curve, the sum of the residuals in a given neighborhood remains small since the residuals cancel one another. In contrast, when the sum of the residuals is large, it indicates that there is a trend in the residuals, and the curve does not fit the data well. The authors presented an algorithm that fits a sequence of curve segments to the data, extending each segment as far as possible while ensuring that the sum of the residuals associated with each segment remains small with respect to the variance in the same data.

Another interesting aspect of this algorithm is that it is intended for long streams of data, for which it would be infeasible to wait until all the data had been collected before processing it. Therefore, the data are processed within a moving window into the data stream, and each newly generated curve segment is appended to a growing piecewise curve. This complicates the problem, given that the authors also sought a curve with C^1 continuity. If each approximating curve segment is generated independently, the tangent at the endpoint of one segment may be inappropriate as a starting tangent for the next segment. In that case, the subsequent segment may swing wildly to meet the pre-determined tangent and also fit the available data points. Then, even though the residuals may be small, the curve still will not fit the underlying trend of the data well. To avoid this repercussion, the presented algorithm considers data from adjacent curve segments simultaneously to ensure that the resulting tangent at the shared knot is appropriate for both segments.

Although this work uses an explicit curve representation, it is an interesting solution to the problem, and it is extensible to the parametric case. In this algorithm, each polynomial segment is computed sequentially. "Past" and "future" data are incorporated into the solution for each segment to contribute to the smoothness of the resulting curve and to ensure stability, respectively. Consider segment $S(x)$ on interval $I = [s,t]$ with associated data points (x_k, F_k) where $x_k \,\epsilon\, [s,t]$. Future data is included in the solution for $S(x)$ by solving simultaneously for the subsequent segment $S_1(x)$ on interval $I_1 = [t,u]$; however, the solution for $S_1(x)$ is tentative. $S(s)$ and $S'(s)$ are fixed by the previous curve segment. C^1 continuity is ensured at t by setting $S_1(t) = S(t)$ and $S_1'(t) = S'(t)$. The four remaining parameters (i.e., $S(t), S'(t), S_1(u)$, and $S_1'(u)$) are computed via least-squares approximation, minimizing the error function

$$E = \sum_{\delta I + I} [S(x_k) - F_k]^2 + \sum_{\delta I_1 + I_1} [S_1(x_k) - F_k]^2 \ . \tag{2.5}$$

Here, past data are incorporated into the solution for each segment by extending the domain of each sum backwards over an additional subinterval, δI or δI_1, whose length is half the length of the previous curve segment.

To find knots that allow each segment to be as long as possible without incurring a statistical trend in the residuals, Eq. 2.5 is solved multiple times with potential knot placements. Each solution is tested against the criterion

$$\sum_{k=p+1}^{q} r_{k-1}r_k < \sum_{k=p}^{q} \frac{r_k^2}{\sqrt{q-p}} \ , \tag{2.6}$$

where p and q are the indices of the first and last data points included in interval $(\delta I + I)$, and r_k is the residual associated with data point k. The left-hand side of the inequality gives the trend in the residuals over interval $(\delta I + I)$, and the right-hand side is closely related to the variance over the interval. If the inequality holds for both segments, $S(x)$ and $S_1(x)$, the data included in interval I is said to "belong" to segment $S(x)$, and interval I is extended; otherwise, interval I is decreased. Interval I_1 is taken to include the same number of data points as interval I.

A bisection approach is used to determine the knot placements and corresponding intervals that will be tested. Let n be the number of data points in interval I. As an initial estimate, n_0 is set to the value of n accepted for the previous curve segment. $S(x)$ is generated and tested against the criterion in Eq. 2.6, and K is set to $+1$ or -1 if the criterion is passed or failed, respectively. Subsequent values of n are determined via

$$n^{(j)} = n^{(j-1)} + K\ 2^{j-1} \text{ for } (j = 1, 2, ..., \theta_1), \tag{2.7}$$

which doubles the step size taken in n on each iteration. Here, θ_1 indicates that this iterative process is finite. Eventually, the correct value of n will be passed, which is indicated when the criterion outcome for one iteration differs from that of the previous iterations. At that point, a new iterative process begins with the final value of n produced by Eq. 2.7. The direction is reversed, and subsequent iterations halve the step size to hone in on the correct value of n:

$$\begin{aligned} n^{(0)} &= n^{(\theta_1)}, \\ n^{(j)} &= n^{(j-1)} - K\ \frac{2^{\theta_1 - 1}}{2^j} \text{ for } (j = 1, 2, ..., \theta_2). \end{aligned} \tag{2.8}$$

If the direction must be reversed again, the step sizes continue to be halved:

$$n^{(0)} = n^{(\theta_2)},$$

$$n^{(j)} \;=\; n^{(j-1)} + K\,\frac{2^{\theta_1 - 1}/2^{\theta_2}}{2^j} \text{ for } (j = 1, 2, ..., \theta_3), \tag{2.9}$$

and so forth. The process stops when the final step size is 1, at which point the knot t is placed at the midpoint between the last data point included in the current interval I and last data point included in the previously tested interval; one of these intervals incurred a statistical trend in the residuals, and the other did not. Minor adaptations to this scheme are needed to generate the first and last curve segments.

2.1.3 Interactive drawings

Reeves addressed the problem of capturing interactive drawings sketched with a stylus on a digitizing tablet [51]. In this application, the data are received as a stream of slightly noisy input points that track the stylus position over time. As such, this problem bears some resemblance to that addressed by Ichida et al., and the algorithm they presented would presumably have produced nice results here. Nevertheless, it would have been inappropriate for this application given its computational intensity. Delays between user input and system response are highly undesirable in interactive systems, so computational speed is a primary concern. As a result, the algorithm Reeves proposed employs a less rigorous heuristic approach.

It is worth noting that this work was done in 1980 when computers were not as fast as they currently are, and some of the heuristic shortcuts taken would not have been necessary today. Even so, in the examples shown, the resulting curves appear to represent the input sketches accurately at the minor expense of using more curve segments than would be needed by a more exacting approach.

Reeves presented and compared several curve representations and knot selection algorithms, then concluded that piecewise cubic Hermite curves, along with the generation algorithm described below, were the most appropriate choice for his purpose. One important reason was that hand-drawn curves frequently contain cusps, and Hermite curves are well suited to representing curves with tangent discontinuities.

Reeves generates individual curve segments between knots i and j as follows. Approximate one-sided tangents are determined for each knot by computing the weighted average of a set of data points in the one-sided neighborhood of the knot and finding the slope of the line segment between the resulting pseudo-point and the knot itself. The curve segment is then uniquely specified by the two endpoints and their associated tangents. Clearly, a curve segment defined in this way may deviate significantly from the data internal to the segment unless the knots are placed relatively close to one another.

The deviation between the resulting curve and the input data is approximated by

$$D^2 = \left(S_x(\frac{1}{2}) - x_{\frac{i+j}{2}}\right)^2 + \left(S_y(\frac{1}{2}) - y_{\frac{i+j}{2}}\right)^2, \tag{2.10}$$

where $S_x(t)$ and $S_y(t)$ are the x and y coordinates of the curve segment for $t \epsilon [0,1]$, and data point i is given by (x_i, y_i). Specifically, this heuristic measures the distance between the midpoint of the curve segment and the median of the data points. It approximates the maximum deviation between any data point and the curve since the maximum deviation will be located at the midpoint of the segment if the segment has low curvature.

The high-level algorithm begins by scanning the data points to detect cusps and placing knots at each cusp. At each data point i the *incremental curvature*, which measures the change in direction between the line segment from point $i-1$ to point i and the line segment from point i to point $i+1$, is computed:

$$\delta_i = \arctan\left(\frac{y_{i+1} - y_i}{x_{i+1} - x_i}\right) - \arctan\left(\frac{y_i - y_{i-1}}{x_i - x_{i-1}}\right). \tag{2.11}$$

A cusp is detected at point i if $|\delta_i|$ exceeds some threshold, such as $\frac{\pi}{2}$.

Next, piecewise Hermite curves are generated between cusps by adding segments sequentially, extending each one as far as possible without exceeding a pre-defined error tolerance. When placing each segment, the initial knot is known and the final knot is found as follows. Let $Error(i,j)$ be the maximum distance between any data point and the associated curve segment spanning knots i and j. Then the optimal location for the knot following i is given by z in

$$G(z) = Error(i, z) - Tol = 0. \tag{2.12}$$

G can be evaluated (heuristically) at any point z by generating a Hermite segment between knots i and z, computing the error at the segment midpoint, and subtracting Tol. The task is then to find the root of G nearest i. To begin, G is evaluated at a sequence of data points that follow i, spaced approximately 30-40 points apart. The first point σ, for which $G(\sigma) > 0$, is known to be beyond the desired root; thus, the root is bracketed between i and σ. The modified regula-falsi method [14] is then used to find the bracketed root. This method is similar to, but converges more quickly than, the standard bisection method, and as such it is appropriate for functions that are continuous but may not be everywhere differentiable. Ultimately, the piecewise curve representing the entire sketch is the union of the curves between each pair of adjacent cusps.

The algorithm does not enforce C^1 continuity, even at non-cusp knots. Rather it depends on the fact that if the curve near a knot was drawn smoothly (i.e., without a cusp), then the heuristically determined tangents on each side of the knot will be visually similar. Allowing small differences in the two tangents incident on a knot frees the algorithm from the stability problems that typically plague schemes which "grow" piecewise curves sequentially.

2.1.4 Bargello curves

Our objective is to generate graceful Bargello curves that stylistically approximate the data points produced when users draw with a mouse or stylus in our system. As previously mentioned, we attack this problem in two phases: first we fit a piecewise cubic Hermite curve to the data received from the input device, and then we render that curve in a grid of virtual fabric tiles.

All of the curve-fitting algorithms described above model the input data according to their own intrinsic properties, without regard for the subsequent rendering environment. In each case, it is assumed that the curve will be rendered in a high-resolution raster (i.e., a standard computer screen) that is capable of representing the generated curve. The distinguishing feature of our curve generation problem is that we anticipate rendering our curves with virtual fabric tiles that are very large in comparison with standard pixels. The resulting Bargello grid is of very low resolution, so it is incapable of accurately representing curves that oscillate rapidly. It turns out that curves can be represented accurately in a raster display only when they oscillate slowly with respect to the size of the individual elements in the raster. This concept was codified mathematically by Shannon [54], but for our purposes, what is important is that Hermite curves that oscillate too quickly will result in Bargello curves with a ragged appearance. Since our goal is to produce graceful Bargello curves, we limit the *frequency* (i.e., the rate of oscillation) of the Hermite curves generated in this phase of our algorithm to ensure that they can be represented accurately and gracefully in a Bargello grid.

Another way to view this issue is as follows. The resolution of the Bargello grid is bounded by external design decisions, namely the height of the fabric tiles and the total height of the finished design. Further, Bargello designs are typically composed of only twenty to sixty rows of fabric tiles, and as a result, they contain large sweeping curves (with respect to the height of the design space). Although Bargello quilters may not conceptualize these issues in terms of curve frequency and grid resolution, they nevertheless know or quickly learn that graceful Bargello curves must be large. Therefore, we anticipate that users will draw curves with large sweeping arcs in our system, yet they may have difficulty drawing the curves smoothly given that a mouse is sensitive to small deviations in hand motion. Thus, we also anticipate that the input data may contain high frequency oscillations that cannot be represented in a graceful Bargello design, and we assume that such oscillations result from unintended "jitter" in the mouse motion.

Finally then, another way to state the distinguishing feature of our problem is that we wish to generate piecewise cubic Hermite curves that match the frequency of the *intended* Bargello curve.

Of course, it is impossible to determine algorithmically what the user intended, but in practice, for a Bargello curve to appear graceful the segments between each pair of adjacent extrema and any intervening inflection points should be smooth. We have found that if an arc between characteristic points in the Bargello curve is spanned by multiple Hermite segments, the resulting rasterization tends to have an ragged appearance. Therefore, we propose a solution in which the Hermite curve has knots located (only) at each of the local extrema and inflection points of the Bargello curve. Thus, each Hermite segment is monotonic in both x and y. Such a curve produces a relatively smooth initial rasterization and provides the additional benefit of recording the location of the various characteristic points: these points will be used for beautification purposes in the subsequent rasterization phase of our scheme. We have developed the following algorithm to generate piecewise cubic Hermite curves with the aforementioned properties.

The algorithm begins by thinning the input data, which are originally sampled at approximately 20-40 Hz, such that each remaining point is at least $\frac{3h}{4}$ from its neighbors, where h is the row height in the current Bargello grid (Figure 2.1). This reduces the amount of subsequent computation, yet retains sufficiently many points to specify the curve at the desired resolution. Since the location of apices will be perturbed during rasterization, retaining their precise locations now is not necessary, so we allow apex points to be culled.

Next, the data are scanned to find points that represent local extrema and inflection points of the curve (Figure 2.2). The peaks and valleys characteristic of Bargello curves are of necessity aligned with the vertical columns of the Bargello design. In addition, Bargello curves may contain switchbacks that create horizontal extrema, but they cannot contain cusps at other angles without rotating the entire design. Therefore, we seek

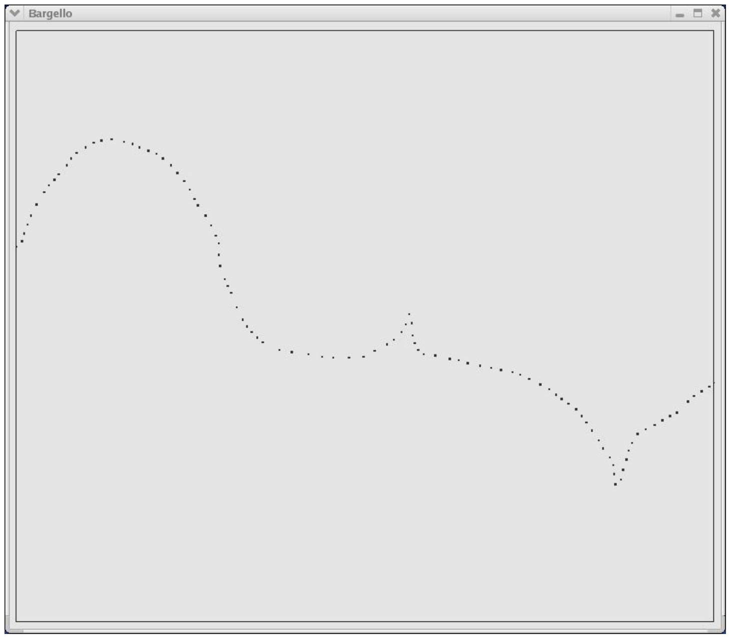


Figure 2.1: Curve generation: culled data points.

extrema in x and y, but we do not concern ourselves with other directions. The data are scanned in three passes, one each to detect x-extrema, y-extrema, and inflection points between them. The separate passes allow "false extrema" to be detected and discarded immediately.

The first pass records the first and last data points as endpoints of the curve and seeks x-extrema between them. Let P_{x_i} be the x coordinate of data point P_i, and let σ_i be defined as

$$\sigma_i = (P_{x_{i+1}} - P_{x_i})(P_{x_i} - P_{x_{i-1}}). \tag{2.13}$$

For each P_i, if σ_i is negative, P_i is recorded as an x extremum. However, if the distance in x between P_i and the previously recorded x extremum is below a threshold (e.g., $\frac{3h}{4}$), P_i is not accepted as an extremum, and the previously recorded extremum is discarded. This ensures that the accepted extrema represent sustained direction changes, rather than noise with respect to the low resolution Bargello design. The second pass scans the data between adjacent x-extrema, seeking local extrema in y. It is analogous to the first pass, with x's replaced by y's in Eq. 2.13.

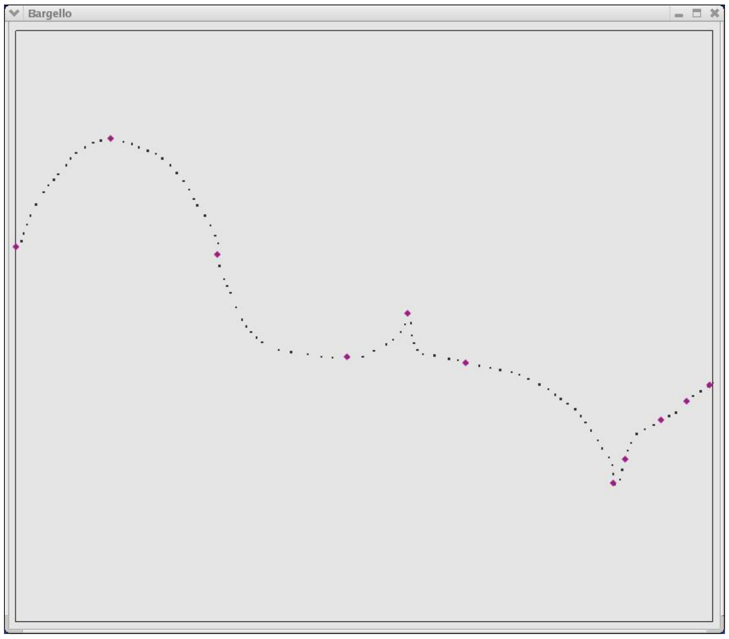


Figure 2.2: Curve generation: characteristic points.

The third pass seeks inflection points between any of the discovered extrema. For each P_i between adjacent extrema, a bounded proxy b_i for the slope of line segment P_iP_{i+1} is computed according to Eq. 2.14, where K is either $+1$ or -1. A proxy b_{i-1} for line segment $P_{i-1}P_i$ is computed analogously:

$$b_i = K \, \frac{(P_{y_{i+1}} - P_{y_i})^2}{(P_{x_{i+1}} - P_{x_i})^2 + (P_{y_{i+1}} - P_{y_i})^2} \; . \tag{2.14}$$

Then a measure of the discrete curvature c_i is given by $c_i = b_i - b_{i-1}$. The resulting c_i are passed through a three-point moving average filter since noise in the input data is exaggerated by discrete curvatures. Finally, the filtered c_i are scanned for sign changes. When a sign change is detected, the associated data point is recorded as an inflection point of the curve, with a similar caveat to that used for extrema. If the new inflection point is too close to the previous one, both are discarded. For this test, we have found that a distance threshold of approximately h works well.

Finally, cubic Hermite curve segments are generated between adjacent knots (Figure 2.3). C^1 continuity is not enforced at the knots for two reasons. First, Bargello curves

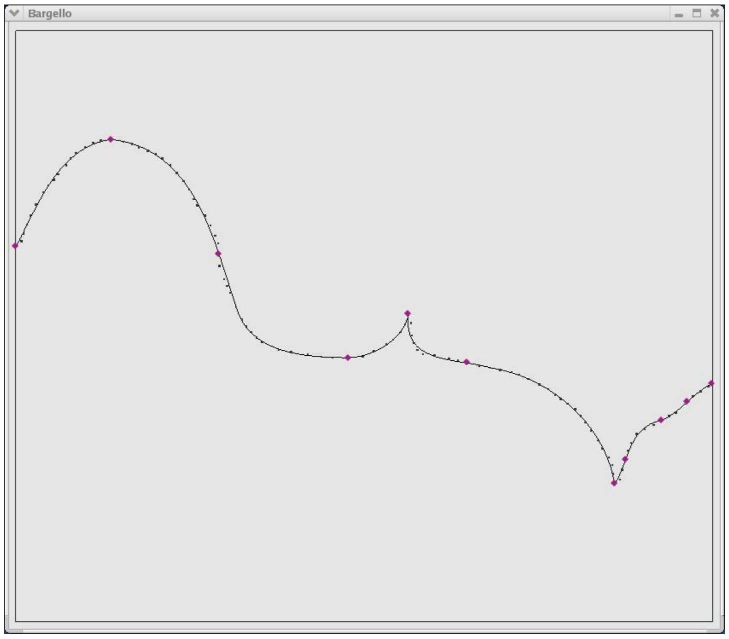


Figure 2.3: Curve generation: piecewise cubic Hermite curve.

frequently contain cusps, and tangent discontinuities should be retained at these points. Second, the Hermite curve is never displayed, and the relatively large fabric tiles of the Bargello curve obscure minor tangent discontinuities at non-cusp knots. In practice, Bargello curves appear smooth at non-cusp knots, even though the underlying Hermite would not if it were visible.

Two methods are used to generate individual Hermite curve segments, depending on the number of data points associated with each segment. If the segment spans at least four data points, a parameter value t is assigned to each point based on chord length, and a curve segment is generated via least-squares approximation. In this calculation, the position of each knot is constrained, and we solve for the associated tangents. Alternatively, if a curve segment spans at most three data points, a Hermite segment is defined directly from the positions and tangents of the two outer data points, where the tangents are defined as vectors pointing from each endpoint toward the adjacent data point. Segments with fewer than two data points are disallowed.

Thus, we have generated a piecewise cubic Hermite curve in which every segment spans an adjacent pair of characteristic points of the curve, and the characteristic points have a minimum distance between them. Although these properties are not common requirements for piecewise curves, in our application they ensure that each arc of the resulting Bargello tile path is founded on a single (smooth) arc in the Hermite curve, and further, that each arc is long enough to allow for a graceful Bargello tiling. This completes the first phase of our Bargello curve generation algorithm.

2.2 Rasterizing geometry

Rasterization is the process of converting the mathematical description of a scene into a set of pixels in a two-dimensional raster display that depicts the scene [23]. At the level of individual lines, curves, and other geometric primitives this process is also known as "scan conversion." Various *geometric primitives* (i.e., atomic units of geometry that are combined to represent complex scenes) present various rasterization challenges.

We begin this section by summarizing the issues with and algorithms for the scan conversion of lines, curves, and text characters since previous work in these areas informed our approach to the rasterization of Bargello curves. We then present the second phase of our Bargello curve generation scheme: an algorithm for rasterizing a piecewise Hermite curve into a low-resolution grid to produce a graceful Bargello curve composed of corner-connected rectangular tiles.

Anti-aliasing is a process closely related to rasterization in which pixels near the ideal geometry are drawn with reduced intensity to blur the jagged appearance that otherwise results when only pixels exactly on the ideal geometry are illuminated. Many rasterization algorithms have been extended to incorporate anti-aliasing during the scan-conversion process; however, since the Bargello style intentionally retains the jagged edges typical of aliased curves, we do not discuss work in the field of anti-aliasing.

2.2.1 Lines

Lines are fundamental to and ubiquitous in computer-generated images; thus, seminal rasterization algorithms addressed the problem of scan-converting lines. Since a variety of rasterization schemes are possible [22], aesthetic decisions were needed and made regarding the nature of an "optimal" line rasterization. Over time, these decisions have gained a sense of seeming inevitability, and it is unanimously agreed that an optimal line rasterization includes the one pixel in each raster column (or row) whose center lies closest to the true line. Such a rasterization results in the thinnest and most accurate line image possible. Subsequent work in this area, which continues to this day, has focused on ever-increasing computational speed.

In 1965, Bresenham [8] presented a classic scan-conversion algorithm to compute the optimal rasterization of a line expressed in its implicit form, $f(x, y) = 0$. The algorithm is defined on the first octant of the Cartesian plane, where $x \geq y \geq 0$, and it selects one of two possible candidate pixels on each iteration. If the most recently selected pixel is $p_i = (x_i, y_i)$, then the two candidates are $p_a = (x_i + 1, y_i + 1)$, the pixel diagonally right and above p_i, and $p_b = (x_i + 1, y_i)$, the pixel directly right of p_i. The algorithm computes $d = |f(x, y)|$ at the center of each candidate pixel and selects the one that lies closest to the line (i.e., the one for which d is smallest). Scan conversion of lines in other octants proceeds similarly, based on symmetry.

Van Aken and Novak [2] presented another classic algorithm, the "midpoint" algorithm, in 1985. It also selects between p_a and p_b on each iteration to produce optimal lines in the first octant of the Cartesian plane, but it computes only one value per iteration: $d_m = f(x_m, y_m)$ at the midpoint between p_a and p_b. If $d_m > 0$, the line passes above the midpoint, and p_a is chosen; otherwise, p_b is chosen. Further, an incremental approach is used such that each subsequent d_m is realized with a single addition. Thus, the algorithm produces lines identical to Bresenham's, but does so more quickly.

Since that time, work in the field of line rasterization has focused on increasing the speed of line drawing with algorithms that draw multiple pixels per iteration. One approach uses Euclid's (greatest common divisor) algorithm to find repeating segments within a line. For example, Castle and Pitteway [10] showed that if a line is drawn from one pixel center p_s to another pixel center p_d and the line passes through a pixel center p_m along the way, then the rasterized line segment from p_s to p_m will be repeated, perhaps multiply, between p_m and p_d. Euclid's algorithm can be used to find p_m efficiently, if it exists, and the remainder of the line can then be drawn immediately.

However, the largest gains have been made by algorithms based on the concept of pixel runs. A *pixel run* is the set of selected pixels in a given raster row. An optimal line rasterization can contain at most two pixel run lengths, and the longer run always includes one more pixel than does the shorter run. For example, a line with slope $\frac{2}{3}$ is represented by an alternating sequence of one-pixel and two-pixel runs, which can be seen by the fact that a consecutive pair of such runs cover two rows and three columns in the raster. Reggiori [52] proposed an algorithm that computes the length of the next pixel run and draws the full run on each iteration. This reduces the time complexity significantly when pixel runs are lengthy, but for lines with slopes near one, it produces little gain. To counter this problem, Bresenham [9] proposed an algorithm that uses both horizontal and diagonal runs, or *run-length slices*. Stephenson and Litow [57] extended the idea in another direction with an algorithm that draws a *run of runs* on each iteration. Again, an optimal line can contain at most two lengths of runs of runs, the longer of which includes one more run than does the shorter. Given that the minimum number of runs in a run of runs is two, this algorithm theoretically cuts the computational time by at least another half. Further, the hierarchy can be scaled indefinitely. However, the authors note that initialization costs increase with each abstraction level, so the gains in doing so are, at present, limited.

2.2.2 Cubic curves

Cubic curves are also common in computer-generated scenes, thus warranting the development of a class of rasterization algorithms specific to these curves. Since the geometry of cubic curves is less predictable than that of lines, they present a different rasterization challenge: naïve scan-conversion algorithms can miss portions of the curve or leave gaps in the resulting rasterization. Thus, while speed and aesthetics remain important, cubic curve rasterization algorithms have focused primarily on accuracy.

Hobby [30] proposed an algorithm to scan-convert implicit cubic curves, (i.e., those of the form $f(x, y) = 0$), based on the midpoint algorithm for line scan conversion. To begin, a curve is partitioned into segments that are monotonic in both x and y, such that two adjacent candidate pixels are known *a priori* at each iteration. (However, details are not provided regarding the method used to achieve this segmentation.) The midpoint between the candidate pixels is then tested to determine whether it lies above or below the curve, and a pixel is selected accordingly. Thus, the algorithm produces an optimal rasterization for nonparametric cubic curves. However, the algorithm is not suitable for use with parametric curves because it is difficult to determine whether a given point lies above or below a parametric curve. Further, *implicitization*, the process of converting a parametric curve to its implicit form, is computationally expensive and prone to numerical difficulties. Yet, in computer graphics, parametric curves are used almost exclusively, so most of the work in the field of cubic curve rasterization has taken a different approach.

In 1974, Chaikin [11] proposed an algorithm for scan-converting parametric curves that approximates each curve with a series of short line segments. To accomplish this, the curve is recursively subdivided until the two endpoints of a given segment are within two pixel units of one another, and then a line segment is drawn between the endpoints. Unfortunately, given that a parametric cubic curve can self-intersect, the algorithm is not guaranteed to draw the entire curve. Rather, subdivision may proceed until a segment is found with both endpoints close to an intersection point but outside the resulting loop, at which point a line will be drawn between the endpoints, and the loop will be omitted.

Lane and Riesenfeld [40] developed a similar algorithm from a more theoretical perspective. Their version uses the Bezier curve representation and subdivision continues until the distance between each control point and the approximating line segment is within a given tolerance. Since a Bezier curve is completely contained within the convex hull of its control points, the deviation between the polyline approximation and the true curve can be bounded to ensure that non-trivial loops will not be missed. However, another difficulty remains: the curve subdivision process is computationally expensive and not amenable to hardware implementation because it requires a memory stack.

In response, another family of algorithms was developed that traverse the curve and paint the pixels responsible for each evaluated point. *Forward differencing* is an incremental method for evaluating a sequence of points along a curve that takes advantage of knowledge about the curve derivatives at each step. As a result, each subsequent point can be found with only three additions, and the approach is amenable to hardware implementation. For this reason, all of the algorithms in this family use forward differencing to evaluate the curve. The differences between them lie primarily in the step size taken.

Kaufman [36] proposed an algorithm that uses a uniform step size that ensures no pixels are skipped and, as a result, also ensures no portion of the curve is missed. If a step δt is taken in parameter space, then $\delta x = \delta t\ (\frac{dx}{dt})$ is the corresponding horizontal step distance in screen space. Therefore, an upper bound on the step size that can be taken while ensuring that no pixel is overstepped, either horizontally or vertically, is given by

$$\delta t = \frac{1}{max(max_{0\leq t\leq 1}|\frac{dx}{dt}|,\ max_{0\leq t\leq 1}|\frac{dy}{dt}|)}\ . \tag{2.15}$$

Here, the numerator represents one pixel unit.

Lien et al.[42] proposed an algorithm that uses an adaptive step size. An initial step size is determined by curve subdivision, then at each subsequent step if the step distance based on the current δt would be more than 1 pixel unit or less than $\frac{1}{2}$ pixel unit, δt is halved or doubled, respectively. The authors found that when drawing a wireframe mesh of the Utah teapot the number of steps taken was approximately half the number that would have been required with a uniform step size. However, this approach re-opens the possibility that portions of the curve may be missed.

Klassen [37] modified the approach of [42] to make it more robust. First, before scan conversion, the curve is split into segments that are monotonic in both x and y to ensure that no segment contains a loop. Second, the step size is adjusted iteratively until the step distance is within the allowed window. In rare instances when the current step distance is less than $\frac{1}{2}$ pixel unit but doubling the step size would cause it to be greater than 1 pixel unit, or vice-versa, a binary search is used to find an appropriate step size. In this way, the algorithm retains the efficiency of an adaptive approach, while ensuring that no portion of the curve is missed.

In summary, few scan-conversion algorithms for cubic curves strive to produce curves that are as thin as possible; rather they strive to ensure that no portion of the curve is missed and that no gaps appear in the rasterization. As a result, a curve rasterization may include "elbows" where, for example, the pixels (x_i, y_i), (x_i+1, y_i), and (x_i+1, y_i+1) are all illuminated. However, this result is less unsightly than a rasterization with gaps, and less inaccurate than one that misses a portion of the curve.

2.2.3 Text characters

Text characters are also extremely common on computer displays, and some of the specific issues that arise in text character rasterization are similar to those that arise in the rasterization of Bargello curves. In the early days of computer graphics, text characters were stored as rasterization patterns (i.e., bitmaps), but since the 1980's, they have been represented internally by piecewise cubic curves that delineate their contours. This approach allows a single character representation to be scaled to draw character instances in different sizes. Character rasterization is then a matter of scan-converting and filling the spline contours.

However, a unique set of problems arises when scan-converting text characters. Each character in a given font has been especially designed to have a specific appearance, and it should maintain that appearance regardless of the font size or the raster resolution at which it is drawn. Further, each instance of a character should look identical to other instances of the same character when drawn at the same size and resolution. Similarly, some characters contain repeated parts, such as serifs, that should look identical to one another within a given character instance.

A naïve scan-conversion algorithm would not maintain these properties, particularly when characters are rendered with a very small footprint on a raster screen such that individual strokes are only a few pixels wide or less. In this case, each illuminated pixel contributes significantly to the final appearance of the character. One consequence of this is that small deviations in the placement of a character with respect to the raster grid can make a significant difference in the character's appearance. Consider a stroke that is two pixel units wide. If it is placed such that one edge falls on the division between two pixels, it will cover two pixel centers, but if it is translated by $\frac{1}{2}$ pixel, it will cover three pixel centers. If drawn with a naïve scan-conversion algorithm, the stroke would appear half again as wide after the translation as it did before, and if a full paragraph were drawn this way, the resulting text would look as if random strokes had been emboldened.

Thus, text rasterization algorithms make slight modifications to both the placement and the underlying geometry of each text character to achieve the desired appearance. In essence, characters are often scaled nonuniformly to meet typographic and perceptual needs. Note that this approach is fundamentally different from that used in the scan-conversion of cubic curves, where the correct appearance is assumed to be that which most accurately reflects the underlying geometry.

Hersch [28] proposed a solution in which constraint specifications are included in the internal description of each character, and subpixel displacements and subpixel stretches are applied to parts of the character during scan conversion to enforce the constraints. Two types of constraints were proposed. First, horizontal and vertical strokes are centered with respect to the underlying pixel grid. This ensures that the same number of pixels are selected each time the stroke is drawn at a given size and resolution, and it also preserves size relations between various strokes in a character. However, to make the constraint satisfiable, certain segments of some characters must be deemed “stretchable.” For example, in the character “E”, all three horizontal strokes should be centered, and to allow this, one or both halves of the vertical stroke may need to stretch a subpixel amount during scan conversion.

The second type of constraint Hersch proposed relates to characters with curved contours, such as “C”. If horizontal or vertical apices of such curves lie close to the division between two pixels, the resulting rasterization can include long pixel runs or isolated pixels, which manifest as unsightly flat spots or blips, respectively. In response, Hersch constrains curve apices that are convex upward to lie between $\frac{1}{8}h$ and $\frac{5}{8}h$ from the bottom of a given pixel, where h is the pixel height. Apices with different orientations (both horizontal and vertical) are constrained symmetrically.

Finally, Hersch advocated that CAD systems for font design should allow designers to add such constraint specifications to characters.

In contrast, Hobby [31] proposed an algorithm that automates the process. He also added several constraints to the cadre. For example, many characters have internal symmetries that should be preserved, others have multiple parts that should be aligned horizontally, and the bottoms and tops of characters across the entire font should be aligned vertically. For each of the defined constraints, a feature detection pass finds applicable parts of each character, error measures are associated with vertex positions, and penalties are added to character-specific objective functions. The solution to the resulting optimization problem specifies an intermediate encoding of each character. Once the scale factor is known for a given scan-conversion task, final adjustments to each character are made. Thus, while the encoding algorithm is computationally expensive, much of it is performed only once. Further, Hobby argues that the constraints are sufficiently abstract that the algorithm can be applied to any character set, and he shows that it works for Japanese Kanji as well as English characters.

It is worth noting, however, that the intra- and inter-character relationships codified in this approach were included based on their common usage in Western typography. Other typographical customs may not be well served by an algorithm that forces particular character alignments, for example.

2.2.4 Tiled Bargello curves

In the second phase of our Bargello curve generation scheme, we rasterize a piecewise cubic Hermite curve into a low-resolution Bargello grid to produce a stylistic rendering of the curve consisting of a corner-connected path of rectangular tiles. This tile path can then be propagated vertically to create a complete Bargello tiling.

There are two ways to envision this problem. First, given that all tiles in a Bargello design must be aligned in vertical columns due to the constraints of the Bargello style, we can think of each tile as an atomic element in a Bargello grid. Then the rasterization problem takes place in a grid of "pixels" that are neither uniform, nor fixed size. In our first attempt, we tried approaching the problem this way, but we soon came to think better of it! Thereafter, we have envisioned an underlying grid of uniform, fixed-size (but non-square) "pixels" such that each virtual fabric tile consists of one or more consecutive pixels in a given pixel row. Approached from this angle, the problem more closely resembles a traditional rasterization problem, so we can draw on lessons from the rasterization schemes for lines, curves, and text characters to accomplish the task.

Nevertheless, there are several Bargello-specific goals to fulfill. In order to produce a rasterization that fits our notion of "graceful" and that conforms to the physical and design constraints of the Bargello style, we place a unique set of constraints on the pixel configurations that can be selected to represent a Bargello curve. We take small liberties with the placement and geometry of the underlying Hermite curve, and we may also select a set of pixels that do not completely cover the curve. Ample precedent has been set for such stylistic renderings in the rasterization algorithms used for text characters. Indeed, even the seminal line rasterization algorithms are founded on aesthetically motivated considerations.

We now turn to the domain constraints placed on the rasterization problem. To begin, a minimum tile width should be imposed, and only a discrete set of tile widths should be allowed. These requirements arise because physical fabric pieces are cut with discretely marked rulers, and they must be wide enough to be manipulated by hand. To conform to the Bargello construction process, all tiles in the design should be aligned in columns, and the tiles used to represent a given Bargello curve should have the same height. Further, consecutive tiles should occupy different design columns (i.e., there should be no vertically adjacent tiles in a Bargello tile path). The latter constraint implies that the rasterization should not contain any “elbows.”

Finally, the Bargello curve should be “graceful.” The requirements for a graceful curve are less clear, but we have defined the following soft constraints in pursuit of this goal. First, on each arc between inflection points of the Bargello curve, the constituent tiles should exhibit either a monotonically increasing or a monotonically decreasing sequence of tile widths, allowing repeated widths. Second, no tile should be more than double the width of its adjacent neighbors, except when an adjacent tile is the minimum allowed width. The neighbors of a minimum width tile may be as much as three times the minimum tile width. Third, in a design that includes multiple independent Bargello curves, tiles in one curve may need to be subdivided into multiple design columns to accommodate the tangent requirements of another curve. In this case, subdivisions that result in overly narrow tiles should be avoided.

In summary, the task we face is this. Given a piecewise cubic Hermite curve, create a corner-connected path of rectangular tiles that constitutes a stylized rendering of the curve. The tile path must maintain certain hard constraints, such as a minimum tile width, and it should also have a graceful appearance. Subsequently, this tile path can be propagated vertically to create a complete 2D Bargello tiling. If multiple independent (i.e, non-parallel) curves co-exist in the design, the representative tile paths should be coordinated to form a mutually compatible Bargello grid. In the remainder of the section, we describe the data structures and algorithms used to solve this problem.

Data structures

In our system, the current Bargello design is stored in a “bitmap” of rectangular, but rarely square, cells. We refer to the geometry underlying the bitmap as the *Bargello grid* and reserve the term *bitmap* to refer to the data structure itself. The rows and columns in the grid have somewhat different semantics. Each grid row represents either the full height or half the height of fabric pieces in the physical quilt, and as such, grid row height is a design-dependent variable. In contrast, grid columns represent the finest allowable width increment of fabric pieces. Since this is a physical constraint of the problem, the width of the grid columns is constant across all designs. Thus, a fabric tile will occupy either one or two rows in the bitmap, for “matched seams” or “staggered seams” designs, respectively, but it may occupy several bitmap columns. This mechanism is used to enforce the two physical constraints: that there should exist a minimum tile width, and that there should exist a minimum width increment. Further, the minimum tile width can be set to any positive integer multiple of the minimum width increment.

Unlike standard scan-conversion algorithms that write their output directly to the frame buffer, we first store the output of the scan-conversion process into a data structure called a *tile path*. This initial tile path is used to initialize a beautification process which generates a second tile path, and the second tile path is painted into the bitmap with a z-buffer algorithm for subsequent display. Both tile paths are stored in a data structure called a *BargelloCurve*. Storing the beautified tile path allows it to be written into the bitmap repeatedly, without requiring repeated beautification. However, sometimes re-beautification is necessary, so the initial tile path is retained for subsequent initializations.

Algorithms

Bargello curves suffer some of the same scan-conversion hazards text characters do. If a local extremum of the piecewise Hermite curve lies close to a division between rows in the Bargello grid, the resulting rasterization is likely to contain either a long flat tile or an isolated thin tile at that point. (For example, see Figure 2.4 near "A".) Neither outcome reflects the shape of the underlying curve well. Further, because the scale of the Bargello grid is large, the visual defect is more pronounced in Bargello curves than it is in text characters. Therefore, it is not possible to retain simultaneously the visual shape of the curve and the exact locations of the extrema. We conclude that retaining the shape of the curve is the more important goal.

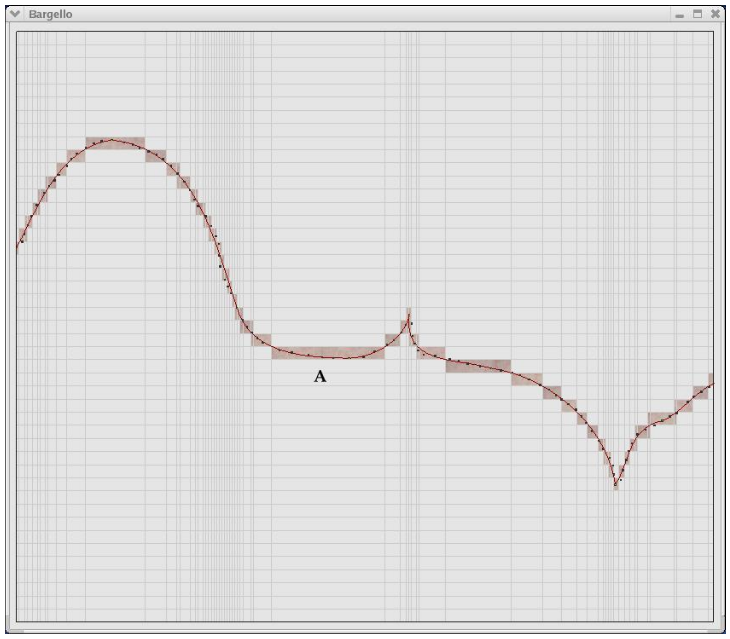


Figure 2.4: Scan-conversion hazards.

Note that a uniform, global translation of the whole curve does not repair the problem

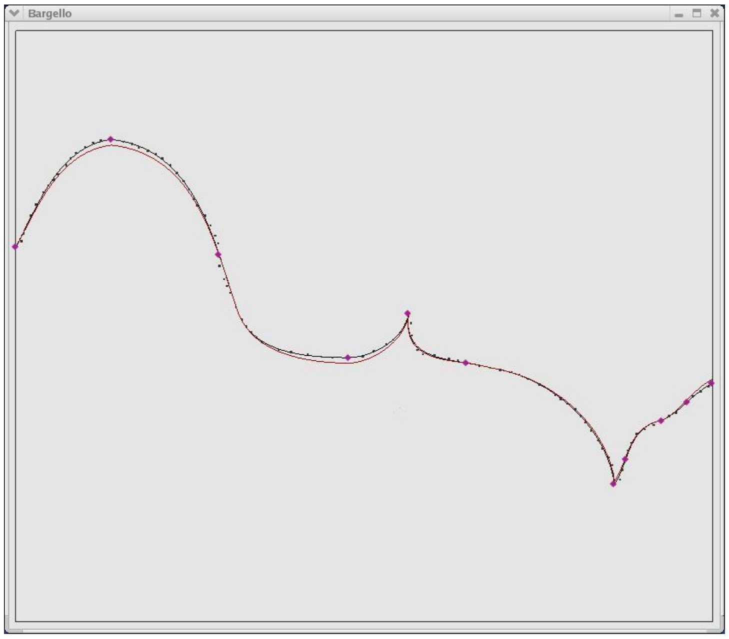


Figure 2.5: Bargello rasterization: stretched Hermite curve.

since various extrema within the curve fall at various phases of the grid. Therefore, following an approach used in the rasterization of text characters [28], prior to scan conversion, the Hermite curve is stretched to place each y-maximum and y-minimum at the nearest point that lies $\frac{3}{10}h$ and $\frac{7}{10}h$, respectively, from the bottom of a grid row, where h is the grid row height (Figure 2.5). These values were found empirically to produce pleasing results. Naturally, other values are possible. Since the piecewise Hermite curve has a knot at each extremum, to stretch the curve these knots are simply translated to the desired positions. Although extrema may be moved to adjacent rows, all moves are "subpixel" with respect to the Bargello grid since all moves are at most $\frac{1}{2}h$. Further, there is no danger of moving a local maximum to a position below an adjacent local minimum because such a pair would have been recognized as "false extrema" and discarded during the Hermite curve generation process.

Next, a rasterization of the stretched curve is computed and stored in a tile path (Figure 2.6). To ensure that the tile path is connected, each Hermite segment is traversed with a uniform step size δt that ensures no grid cell the curve passes through is missed.

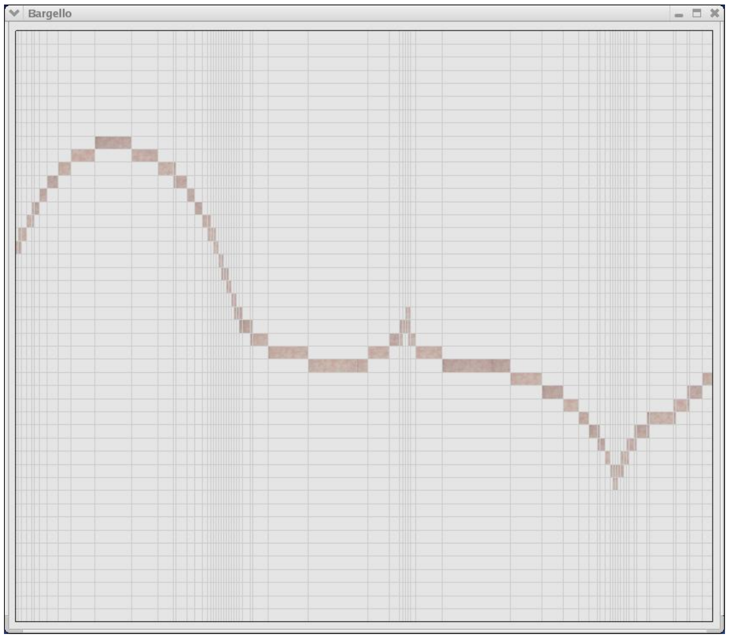


Figure 2.6: Bargello rasterization: initial tile path.

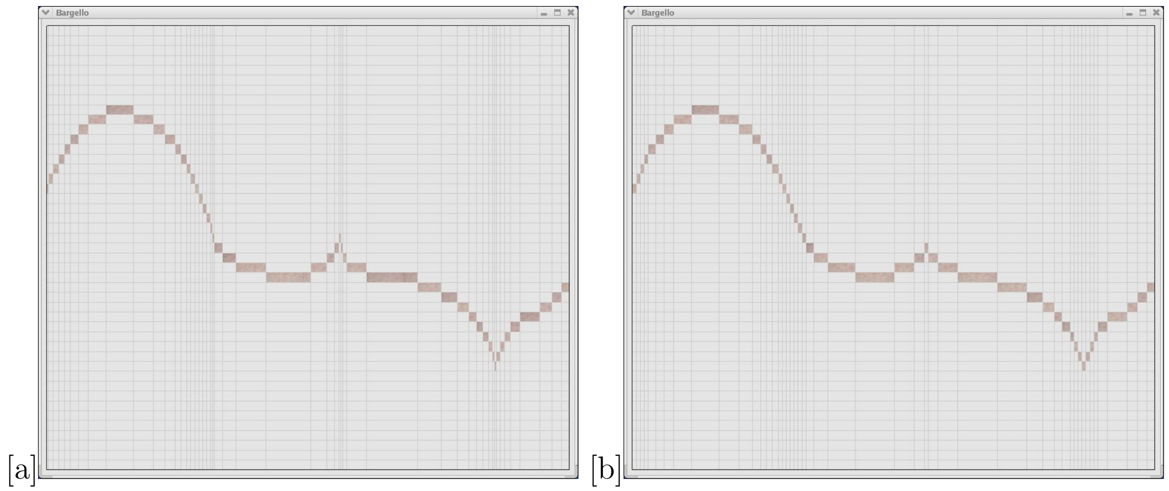
The step size is defined as

$$\delta t = min(\frac{w}{max_{0\leq t\leq 1}|\frac{dx}{dt}|}, \frac{h}{max_{0\leq t\leq 1}|\frac{dy}{dt}|}), \tag{2.16}$$

where each of the maxima are taken over the curve segment, and w and h are the width and height of each grid cell, respectively. This choice of step size is based on Eq. 2.15 given by Kaufman [36] but has been adapted to accommodate rectangular grid cells. Essentially, we take the smaller of the two step sizes required to ensure no pixel will be overstepped horizontally or vertically.

Once the initial tile path has been computed, a copy of it is made, and the copy undergoes a beautification process. Two beautification algorithms are used: a greedy algorithm that runs while the curve is being drawn, and a dynamic programming algorithm that runs when the pen is lifted.

Greedy beautification is performed as the curve is drawn because each new data point can influence the entire previous curve segment (Figure 2.7 [a]). Therefore, the Bargello curve is continuously refreshed as it emerges behind the mouse or stylus. However, only

[a]


[b]

Figure 2.7: Bargello rasterization: [a] greedy beautification, [b] final beautification.

a subset of the design constraints are enforced at this time. First "elbows" are removed, and then the tile path is massaged until it meets the monotonicity constraint. Recall that on each arc between inflection points, the tiles should exhibit either a monotonically increasing or a monotonically decreasing sequence of widths. Because each arc is monotonic in both x and y, the appropriate sequence direction can be determined by the tangents at the endpoints of the underlying Hermite segment. The tile path is traversed and grid cells are "pushed" from one tile to the next if the width differential between them does not meet the constraint.

The second beautification algorithm is more rigorous (Figure 2.7 [b]). It seeks a globally optimal solution via dynamic programming, and it considers all the Bargello design constraints. The recursive cost definition is given by

$$E_{i,j} = min[E_{i-1,k} \ + \ e_{k,j}], \text{ over all feasible } k, \tag{2.17}$$

where $E_{i,j}$ is the minimum cost for any tile path up to and including tile i such that tile i has width j, and $e_{k,j}$ is the cost of a tile having width j given that the previous tile had width k. Details of the cost function $e_{k,j}$ will be given presently. Thus, the algorithm seeks an optimal sequence of tile widths, while the number of tiles and their vertical positions are taken directly from the reference tile path.

A two-dimensional table is used to store the minimum cost solutions to each constituent subproblem. In this table, each row represents a tile in the beautified tile path, and each column represents a potential tile width, where widths refer to the number of grid cells occupied. However, row zero is an exception. It represents an invisible virtual tile that extends from the left edge of the design space to the leading edge of the first tile, and this virtual tile is allowed to have zero, positive, or negative width. Thus, a tile path is allowed to start at the edge of the design space, interior to the design space, or outside the design space. (The latter case can not occur at the time the curve is drawn, but the tile path may be dragged partially off the design space later.)

Two constraints are enforced implicitly in the problem formulation. First, tile widths are stated in terms of the number of grid cells occupied. Thus, the discrete problem formulation ensures that tile widths come in discrete increments. Second, the first grid cell occupied by a given tile is assumed to be horizontally adjacent to the last cell occupied by the previous tile. Thus, by definition, the tile path does not contain any rasterization "elbows."

In addition, several constraints are enforced by assigning infinite cost to infeasible options. First, for each design, some minimum tile width is specified. Tiles that fall short of the minimum width are disallowed, and tiles that end less than the minimum width away from the edge of the design space are also disallowed since they render the subsequent tile infeasible. Second, while a user may drag a tile path partially off the design space, no tile is allowed to straddle the edge. This ensures that every tile is either completely visible or completely invisible, and therefore also ensures that the computed cost $e_{k,j}$ is correct for the visible portion of the tile. Finally, if a tile in the reference tile path abuts the edge of the design space, some tile in the beautified tile path must as well. This ensures that a tile path intended to cover the design space does so and does not get "sucked away" from the edge by a complicated combination of cost factors.

The four remaining soft constraints are intended to give the tile path a graceful appearance while retaining the shape of the underlying Hermite curve. These objectives are blended together in cost function $e_{k,j}$ as follows:

$$e_{k,j} = w_d c_d + w_m c_m + w_r c_r + w_s c_s. \tag{2.18}$$

As tile i is placed in the tile path, the position of its leading edge has already been determined, and we seek the best position for its trailing edge. The algorithm works for curves drawn right to left or left to right, so either edge of a tile can be the "leading" or "trailing" edge, depending on which direction the curve was drawn.

We now motivate and define c_d, c_m, c_r and c_s. The *distance cost*, c_d, keeps the tile path close to the underlying Hermite curve:

$$c_d = |x_{r_i} - x_{b_i}|. \tag{2.19}$$

Here, x_{r_i} and x_{b_i} indicate the horizontal position of the trailing edge of tile i in the reference and beautified tile paths, respectively, where the units are the number of grid cells from the left edge of the design space.

The *monotonicity cost*, c_m, encourages the tile path to exhibit monotonic tile width sequences between inflection points:

$$c_m = \left| \frac{j-k}{j+k} \right|. \tag{2.20}$$

The cost is imposed if tile i is wider than tile $i-1$ when it should be narrower, or vice-versa; otherwise, $c_m = 0$. In this equation, j and k are the widths of tiles i and $i-1$, respectively, stated in terms of the number of grid cells occupied. The cost c_m is greater in steep regions of the tile path than in shallow regions because a lack of monotonicity is more apparent between narrow tiles.

The *width ratio cost*, c_r, encourages the tile path to exhibit smooth width transitions between adjacent tiles:

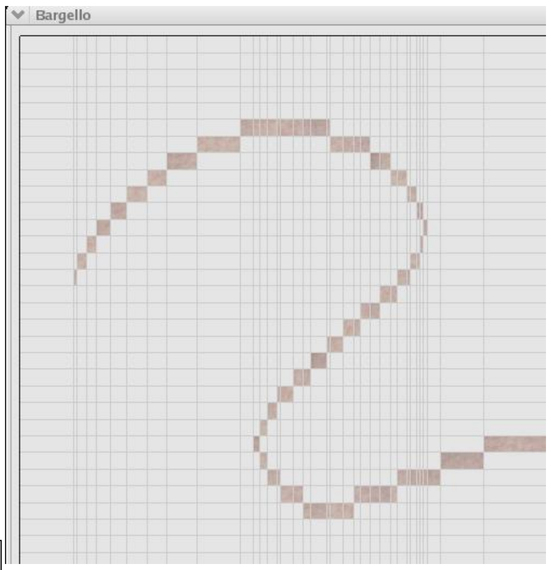
$$c_r = max \left\{ \frac{j}{k}, \frac{k}{j} \right\}. \tag{2.21}$$

Here, j and k retain their previous definitions, and c_r increases as the difference between them increases. The cost is imposed if $max[\frac{j}{k}, \frac{k}{j}] > 2$; otherwise, $c_r = 0$. However, an exception is made when either j or k is the minimum width allowed. To keep the curve from losing flexibility in steep regions, tiles adjacent to minimum width tiles are allowed three times the minimum width without penalty.

These three cost terms relate to the current Bargello curve alone. If this curve is the only one in the design space, they are sufficient. However, when multiple Bargello curves share the design space, a *subdivision cost*, c_s, is introduced, which we now discuss.

After beautification, a Bargello curve is "painted" into the bitmap for subsequent display. Since all tiles in a Bargello design must be aligned in columns, the vertical edges between adjacent tiles in the tile path imply boundaries between vertical columns in the design that run the full height of the design space. If a Bargello curve is drawn into a design space that already contains another curve, then the space has already been partitioned into columns. In all likelihood, the tangents of the new curve will differ from those of the pre-existing one(s) in many places, such that many of the edges between adjacent tiles in the new tile path will not be aligned with pre-existing column boundaries. Then new column boundaries will be inserted into the design space, subdividing the pre-existing columns. When this happens, the data structure of the pre-existing curve is not modified, but the corresponding tiles are subdivided *de facto* in the design, and they would be constructed from separate fabric pieces in the physical quilt.

This mechanism adds richness to a design by allowing multiple independent curves to co-exist. However, arbitrary subdivisions are unattractive and unnecessarily complicate quilt construction (Figure 2.8 [a]). Optimally, we would like narrow tiles from one tile path to fit snugly within the boundaries of a wider tile from the other tile path (Figure 2.8 [b]). In contrast, when tiles from two tile paths interlock like brickwork the subdivisions are unnecessary, especially if the resulting design columns are narrow. In that case, it would be preferable to collapse the narrow columns and bring the tiles of the two curves

[a] 

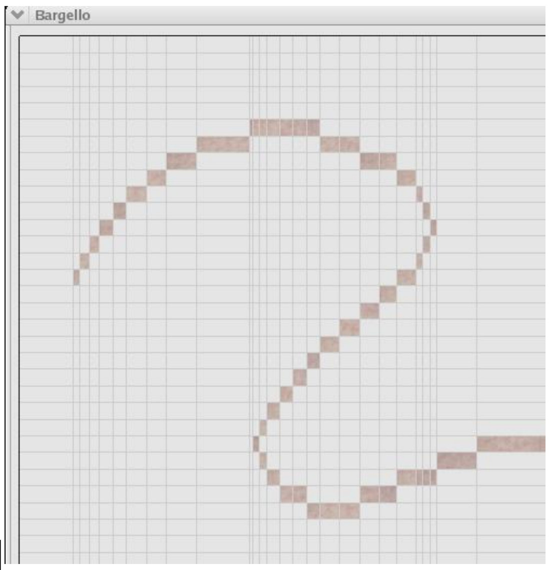
[b] 

Figure 2.8: Bargello rasterization: [a] arbitrary subdivisions, [b] snug subdivisions.

into alignment.

Therefore, a *subdivision cost*, c_s, is imposed on tile paths that subdivide existing columns in the design space. This cost discourages brickwork-style subdivisions, especially when the resulting columns are narrow, but it does not discourage subdivisions that are completely internal to an existing column since these may contribute to a snug-fit set of subdivisions.

Let x_0 and x_1 be the horizontal positions of the left and right edges of tile i, respectively. Let x_a through x_d be the horizontal positions of column boundaries that already exist in the design space, where x_a indicates the rightmost boundary such that $x_a \leq x_0$, x_b indicates the leftmost boundary such that $x_0 < x_b$, x_c indicates the rightmost boundary such that $x_c \leq x_1$, and x_d indicates the leftmost boundary such that $x_1 < x_d$. Then, by definition, $x_a \leq x_0 < x_1 < x_d$. However, there are several scenarios for the placement of x_b and x_c, four of which are shown in Figure 2.9.

If x_0 lands inside the design column delimited by x_a and x_b, and x_1 lands outside it (i.e., $x_a < x_0 < x_b < x_1$), then tile i straddles the right edge of the column. Cost c_{s_0}, the

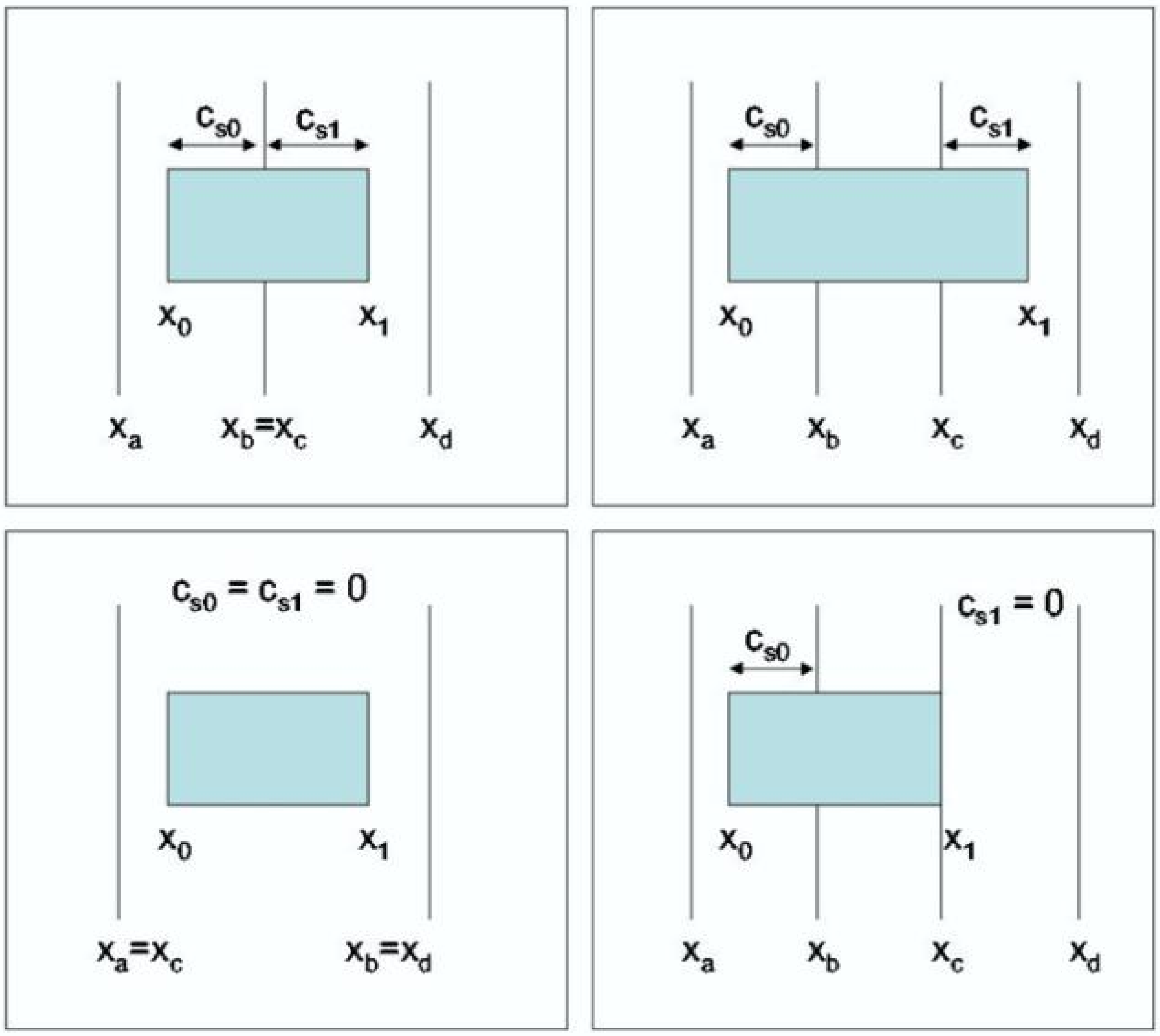


Figure 2.9: Design column subdivisions

"left-hand" cost of subdivision, which we define as

$$c_{s_0} = \begin{cases} \frac{1}{x_b - x_0} & \text{for } x_a \neq x_0 \text{ and } x_b \leq x_1 \\ 0 & \text{otherwise,} \end{cases} \tag{2.22}$$

imposes a penalty inversely proportional to the width of the column partition between x_0 and x_b. In contrast, if x_1 lands within the same design column (i.e., $x_1 < x_b$), then the column is partitioned in at least three pieces, and no cost is imposed on tile i. If a subsequent tile straddles the boundary, cost c_{s_0} will be imposed on the subsequent tile.

Similarly, if x_1 lands within an existing design column and x_0 lands outside it (i.e., $x_0 < x_c < x_1 < x_d$), then tile i straddles the left edge of the design column. Cost c_{s_1}, the "right-hand" cost of subdivision, imposes a penalty inversely proportional to the width of the column partition between x_c and x_1:

$$c_{s_1} = \begin{cases} \frac{1}{x_1 - x_c} & \text{for } x_0 < x_c < x_1 \\ 0 & \text{otherwise.} \end{cases} \tag{2.23}$$

Again, if x_0 lands within the same design column (i.e., $x_c < x_0$), no cost is imposed on tile i.

Finally, the subdivision cost c_s is given by $c_s = c_{s_0} + c_{s_1}$. Note that the two terms need not be zero or non-zero simultaneously.

The weights in Eq. 2.18 can be varied to modify the balance between the beautification objectives. We set the values $w_d = 0.5$ (distance cost), $w_m = 30$ (monotonicity cost), and $w_r = 5$ (width ratio cost), while the value of w_s (subdivision cost) is varied from 0 to 4, as will be explained next. These settings result in monotonicity and width ratio costs that are typically approximately equal, while the subdivision cost is somewhat smaller, and the distance cost is smaller still. This reflects our preference for Bargello curves that are graceful over ones that follow the sketch exactly, given that sketches are typically imprecise. Similarly, while we strive to reduce the number of tile subdivisions, we do not do so at the expense of gracefulness.

In this algorithm, the current tile path adjusts itself to align with pre-existing columns in the design space, but pre-existing tile paths do not reciprocate. It would be preferable if all the tile paths adjusted themselves to one another simultaneously. However, a dynamic programming algorithm to find a globally optimal fit among an arbitrary number of tile paths would require a matrix of arbitrary dimensions! Thus, we take a different approach. A high-level multi-pass algorithm cycles through the tile paths and invokes the dynamic programming algorithm to fit each tile path in turn to the design columns defined by the others. On the first cycle, the weight w_s associated with the subdivision cost is set to zero, so each tile path finds its own ideal shape. Thereafter, on each cycle w_s is increased, and the tile paths move into alignment with one another to the extent possible while balancing this objective with the others.

There is one more case to consider. A Bargello curve that is not single-valued in x presents a challenge similar to that presented by multiple independent curves. Various segments of the curve share columnar regions within the design space, and assuming they have tangents that differ, they each insert column boundaries that subdivide one another's tiles. However, the algorithm as described so far has no knowledge of design columns defined by other segments of the same curve. To rectify this, curves that are not single-valued in x are parsed into segments between x-extrema. Then, on each pass of the high-level algorithm, each segment is fit to the design columns defined by the others. As a result, tiles in each segment are aligned with tiles in the others, but the segment endpoints may move with respect to one another, causing the tile path to become disconnected at the x-extrema. Therefore, after each segment has been fit individually, the full curve is fit to the design columns established by the individual segments, so the final beautified tile path follows the beautified segment templates to the extent possible while maintaining connectivity.

Note that the same effect could not be ensured by simply boosting w_d, the weight associated with the distance cost, to encourage the tile path to follow the underlying

Hermite curve closely. Although doing so would keep the segment endpoints in close proximity, unless w_d was boosted to infinity it would not ensure connectivity, and even then it would not ensure corner-connectivity. Moreover, to encourage gracefulness in the resulting curves, the “beautification” weights, w_m and w_r, are ordinarily maintained at higher levels than w_d. Thus, boosting w_d to encourage connectivity at x-extrema may result in a lack of grace throughout the remainder of the curve. Therefore, although a multi-pass algorithm can be expensive, it seems necessary to allow multiple independent curves, or curves that are not single-valued in x, to co-exist in a design while retaining the desired properties in each Bargello curve.

The beautification algorithm produces graceful Bargello curves that approximate the input sketches well in many cases. However, it is possible to draw a curve that is too steep to follow with a corner-connected tile path and a minimum width constraint. In this case, the algorithm responds by producing a tile path that meets the domain constraints but does not follow the input sketch: it produces a linear segment of minimum width tiles. This is an appropriate algorithmic response, and if the linear segment is not long, its deviation from the (invisible) underlying curve may go unnoticed. However, arbitrarily long linear segments can deviate arbitrarily far from the input sketch, and this result can be confusing to users who are not accustomed to thinking in terms of “Bargello design constraints” and may not realize they have drawn an unattainable curve. Potential improvements for this situation include providing guidelines while users draw that indicate whether a given curve can be followed, and allowing tile path connectivity to be broken in extreme cases. This may also surprise users, but it could offer a visual explanation of the underlying difficulty.

Figure 2.10 shows a Bargello tile path created with our rasterization algorithm. The input data points, the piecewise Hermite curve, and the implicit “Bargello grid” formed by the tiles in the path are also shown. Careful examination of the figure reveals features of the algorithm. For example, near the label “A” the piecewise Hermite curve has been pulled away from the input data to ensure that its local extremum falls at an appropriate phase of the Bargello grid. Near “B” the tile path does not cover the Hermite curve because the curve is too steep to follow with discrete corner-connected tiles, so the algorithm has produced a short linear segment of minimum width tiles instead. Near “C” the uppermost tile in the arc is not quite wide enough to completely cover the Hermite curve; rather, the tile has been made narrower to ensure that it is not more than twice the width of the subsequent tile in the path. Thus, the algorithm has produced a tile path that deviates slightly from the input sketch but is “graceful” and meets the constraints of the Bargello style.

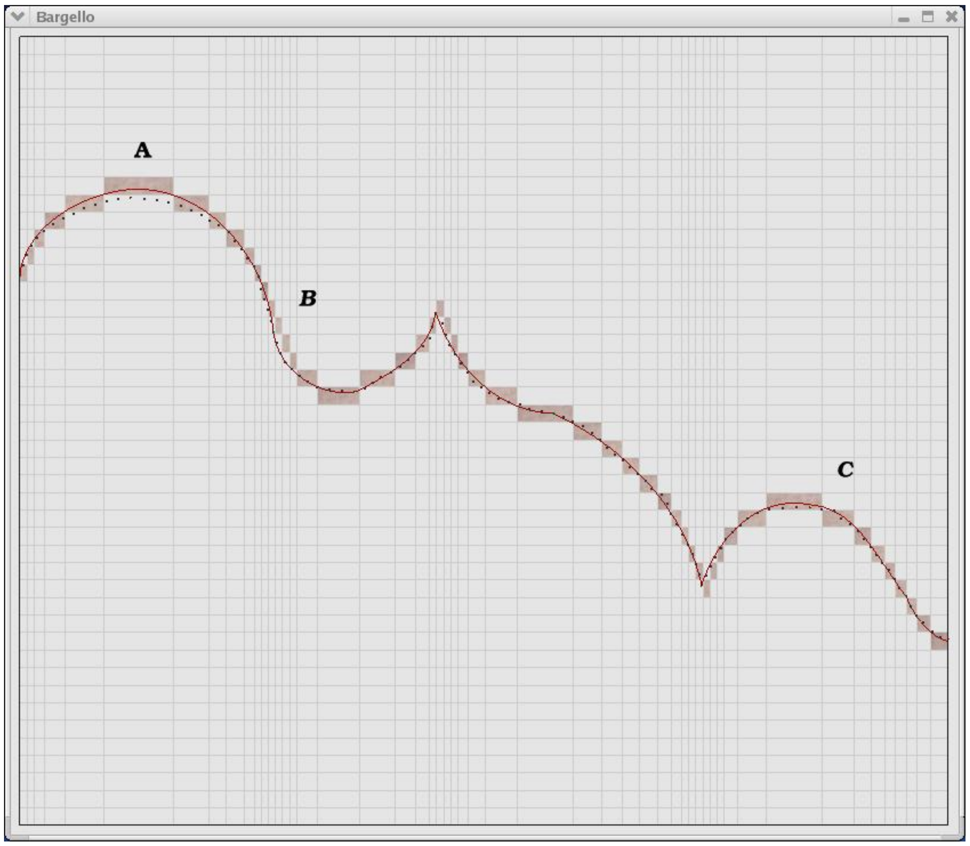


Figure 2.10: Bargello tile path.

Figure 2.11 shows two Bargello tile paths that co-exist within the same Bargello design. In this case, the two curves were drawn independently, and the algorithm fit them together to form a mutually amenable Bargello grid. In many places, for example near “A,” multiple tiles within the yellow tile path have been fit snugly within the bounds of a single tile in the other tile path. Near “B” tiles in both paths have been given the same width, so they both fit snugly within the same vertical column. Thus, two graceful tile paths were produced which cooperatively define a single Bargello grid. In this way, we avoid unnecessary work during the construction of the physical quilt.

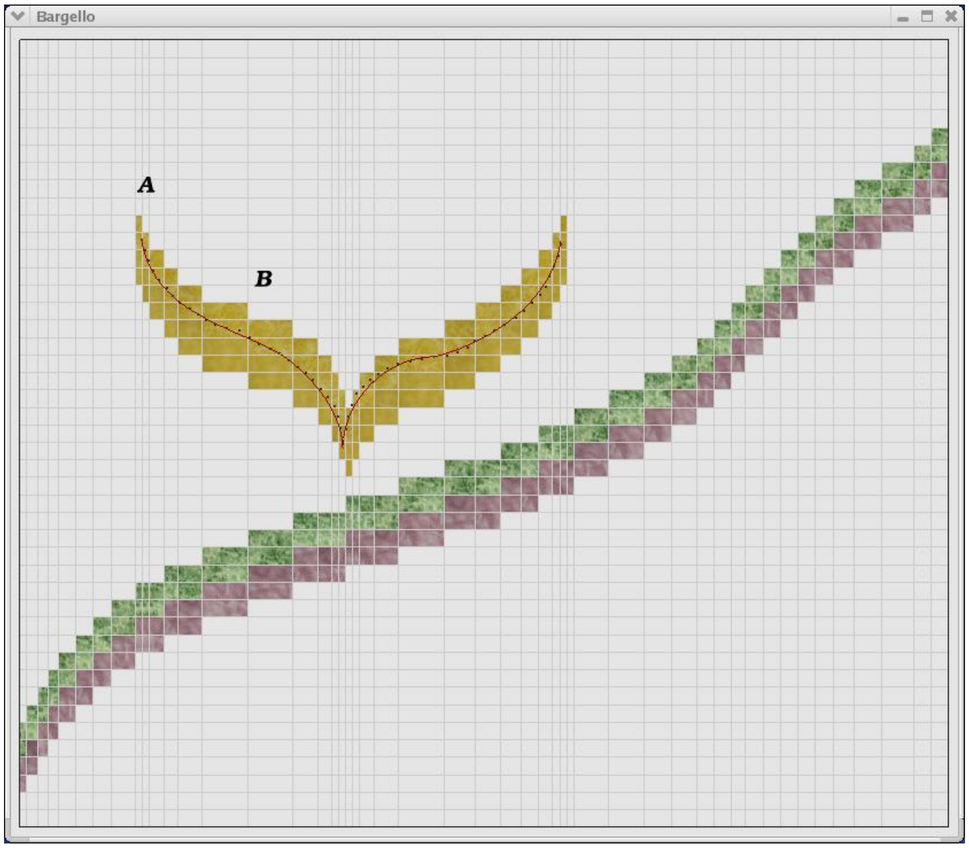


Figure 2.11: Independent Bargello tile paths.

Chapter 3

Computer-Assisted Bargello Quilt Design

We have implemented a computer-assisted design system for Bargello quilts. The system is intended for use by quilters with a broad range of quilting experience, from novice to expert, both with the Bargello style and with quilting in general. In addition, the system is intended to be accessible to quilters without presupposing extensive computer experience: many quilters are expert computer users, but others are not.

The problems addressed by the system include some that are specific to the Bargello style and others that are prevalent in any quilting style. For example, currently quilters typically use a difficult manual process, described in Chapter 1, to design Bargello quilt patterns [19]. One quilter we spoke with described this process as requiring "pencil, graph paper, and a *lot* of eraser." Further, we are aware of two commercial software packages for creating Bargello designs [4, 20], but neither of them allow users to sketch Bargello curves with a mouse or stylus. Rather, they require users to specify their curves by manually adjusting the width of each vertical strip. In our experience, designing curves in this manner is so difficult that it is easier to design the curve on paper and then enter the completed geometry into the program.

Another problem in quilt design is that finished quilts frequently look differently than the designer had imagined them, which can be quite disappointing given that quilters frequently spend a significant amount of time and money to fabricate each quilt. This outcome typically results from using a set of fabrics that appear differently in combination than they do individually, which in turn results from the fact that color appearance depends on context. During our focus group sessions, one quilter remarked that it is common to have made a quilt that you "put on the guest room bed and close the door," and another said, "So many times we start a quilt and then just want to get it over with because it didn't come out the way we wanted it to."

In order to address these and other problems, we have established the following goals for the system.

- The system should be easy to learn and easy to use, especially for novice level designs. On the other hand, it should also support advanced users who are expected to invest more effort to create complex designs.

- The system should accelerate the design process to the point that it invites experimentation.

- The system should enable users to visualize their designs with the real fabric textures they intend to use in the physical quilt.

- Designs produced in the system should adhere to the physical constraints imposed by the artistic medium (fabric) and the design constraints of the Bargello style.

- The system should provide a rich design space, allowing users to create designs that range from simply elegant to intricately complex. In addition, typical Bargello design features such as symmetry and "matched-" or "staggered-seams" patterns should be supported.

Our computer-assisted design system meets these goals. It addresses the problems associated with Bargello quilt design by allowing users to create and experiment with designs quickly and easily, such that disappointing designs can be discarded early and painlessly, and only the most promising designs will be brought to fruition in fabric.

The system contains three constituent parts: an algorithm that produces graceful Bargello curves from sketched input data, a design module that allows users to create both traditional and contemporary Bargello quilt designs, and a fabric selection module that allows users to explore fabric and color combinations using a database of real fabric textures. While the design module is tailored specifically to Bargello designs, the fabric selection module would work equally well for designing quilts in any style. Our curve generation algorithm was introduced in Chapter 2. In the remainder of this chapter, we discuss the two system modules in more detail. We describe what each of them does from a user perspective and how it is done from a programmer perspective.

3.1 Fabric selection

The fabric selection module provides users with access to a database of approximately 450 real fabric texture images that were collected via digital photography from a local quilt shop. Various widgets provide information on the colors contained in each fabric texture and allow users to select fabrics, explore fabric and color combinations, and experiment with fabric orderings for use in Bargello designs.

To help readers understand the purpose of the widgets, we next briefly describe color usage in quilt design. Color usage is at the heart of the artistry of quilt design, and an expert quilter will develop an individual color sense and practice. Nevertheless, because color mastery is difficult, many quilters follow a common set of guidelines that aid in the fabric selection process [48, 62]. Using these guidelines, quilters frequently begin a new project by selecting a *focus fabric* that defines the color scheme of the quilt.

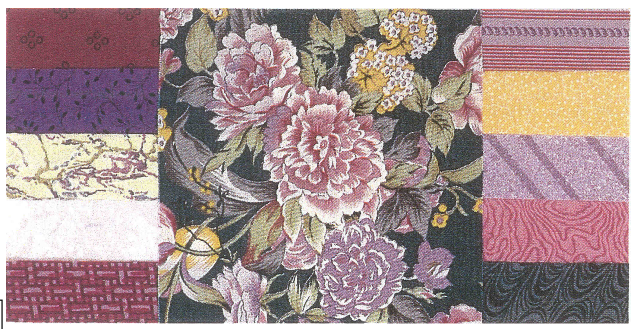
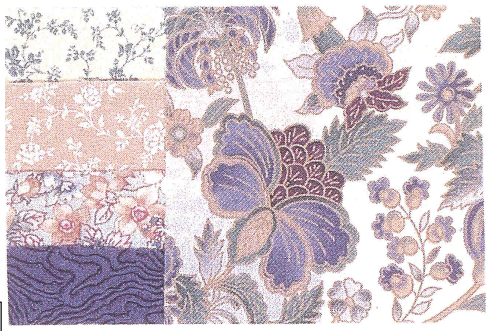
[a]  [b] 

Figure 3.1: Sample fabric combinations, from [48] page 84.

This multi-color fabric is chosen because the quilter finds its colors vibrantly engaging: quilters often say this fabric "speaks to them" or it is a fabric they "cannot live without." Next, *supporting fabrics* are selected that contain colors present in the focus fabric or variants of those colors, e.g., different values of the same hue. Finally, a quilter may select *background fabrics* for use in background regions of the design. These tend to be subtle tone-on-tone fabrics in muted colors such as creams, grays, or black. Figure 3.1 illustrates these concepts with sample fabric combinations.

The colors in each fabric combine to create an overall color scheme for the quilt. As described, many fabric combinations revolve around a central fabric that contains most of the colors present in the quilt. This fabric is sometimes said to "bridge" between various colors in the scheme. Sophisticated fabric combinations may include multiple bridge fabrics, creating a complex web of color throughout the design.

We now return to the fabric selection module and give a brief overview of the various widgets used in the fabric selection process (Figure 3.2); each of them will be described in more detail below. The *main color wheel*, in the upper left corner of the screen, is used to drive color selection. A color may be added to the current "color scheme" by clicking on the corresponding color tile in the main color wheel. In response, the selected color tile is outlined in white, a larger tile in the same color is added to the *color scheme palette* just right of the main color wheel, and fabrics containing that color are added to the *fabric browser*. The fabric browser consists of two large windows at the bottom of

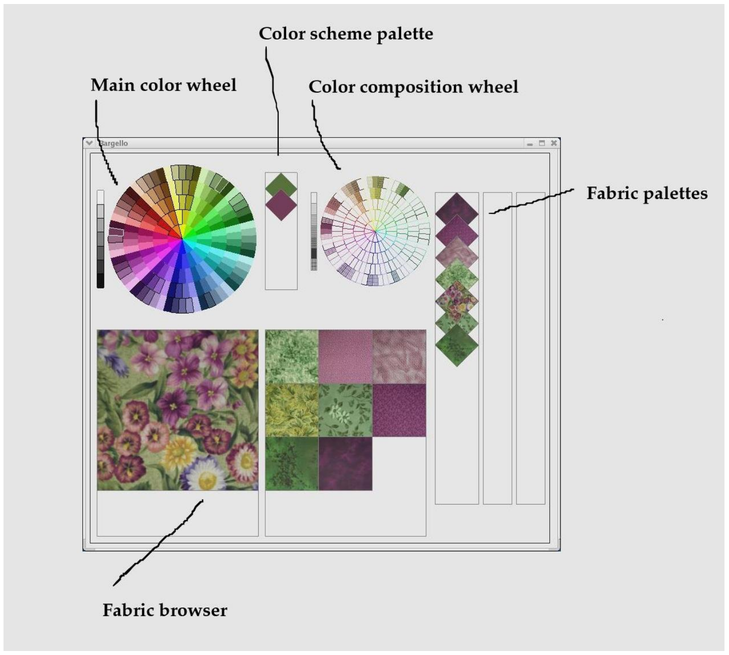


Figure 3.2: Fabric selection module.

the screen: fabrics that contain one or more than one of the colors in the current color scheme appear in the right or the left window, respectively. A fabric texture may be selected for inclusion in a Bargello design by clicking on the corresponding image in the fabric browser. In response, the fabric texture will be added to one of the three *fabric palettes* along the right edge of the screen. Three palettes are provided to accommodate quilt designs that contain up to three separate color strata. At any point in time, one of the palettes is deemed "current," and one of the fabric tiles within that palette is also current. Finally, the current fabric tile is featured in the *color composition wheel*, left of the fabric palettes at the top of the screen, which provides information regarding the colors in that tile and provides access to pop-up windows from which other fabrics containing those colors can be selected.

In the remainder of this section, we first discuss the data structures and processes that allow the fabric selection module to sort fabrics according to color, and then we return to the fabric selection widgets and describe them in more detail.

Categorizing fabrics by color

We acquired fabric textures for use in our system by photographing a selection of fabrics from the inventory of a local quilt shop. Because of the complexities of lighting and the vagaries of digital photography, many of the resulting jpeg images contain tens of thousands of colors, even though each fabric is printed with relatively few colors: generally, between two and two dozen. In contrast, quilters frequently describe fabrics according to a single dominant color category. For example, a floral fabric that includes several pinks and reds set on a cream background with green, yellow, and blue highlights may be categorized as a “red” fabric by quilters. Therefore, the main challenge behind the fabric selection module is to define an approach by which fabrics that contain tens of thousands of colors can be presented to users according to an “intuitive” categorization based on very few colors. This challenging problem has not been completely solved, but the approach we currently take is as follows.

To reduce the computational and memory requirements placed on the system at run-time, some pre-processing is performed once for each fabric image. The images are color-quantized to 256 colors, cropped to the largest possible square, and scaled to 256 x 256 pixels. In addition, color histogram information is computed and recorded in an external file for subsequent access. We have found that the resulting images have high visual fidelity when compared to the originals, so these images are displayed as fabric textures in the system.

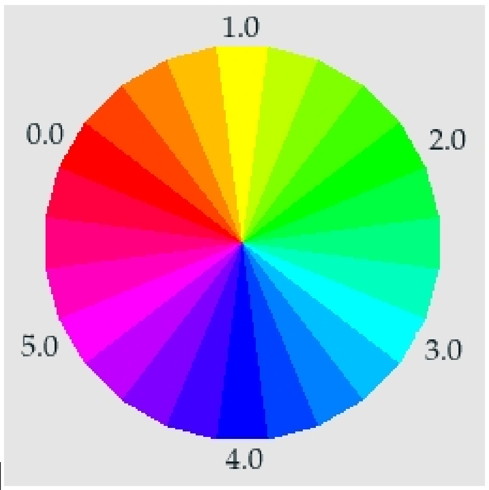
Ultimately, we want to sort images based on the colors represented in the module's main color wheel. Therefore, when the application is launched, the previously recorded histogram data undergoes a second quantization process, in which the constituent colors are accumulated into the color categories defined by the main color wheel. The results are stored in a *fabric color table* in which rows represent fabrics, columns represent colors from the main color wheel, and table entries record the percentage of a given fabric image covered by pixels that fall within a given color category. The table is then used by the widgets in the fabric selection module to locate fabrics that contain specific colors.

To allow the system to categorize fabrics according to their dominant colors and to locate fabrics that contain noticeable amounts of secondary colors, perhaps in highlight regions, two thresholds have been set based on the percentage of a given fabric covered by a given color. We say that a color is *dominant* in a fabric if it covers at least 6% of the image and that it is *significant* if it covers at least 1%. While these thresholds seem surprisingly low, we have found them to work reasonably well in practice. Indeed, the fact that a "dominant" color may cover as little as 6% of a fabric is testimony to the difficulty of the problem: fabric images may contain so many colors that 6% coverage represents a peak in the histogram, and it is reasonable for a fabric to appear in several similar color categories.

Main color wheel

The main color wheel, which was invented for use in this system, allows users to view the full three-dimensional color space in a single compact representation. Its arrangement is based on the HWB (hue-whiteness-blackness) color space [56], which in turn is based on color concepts used by artists in order to provide an intuitive approach to color selection. Artists commonly describe color with language based in the craft of mixing pigments [63]. In this context, "pure" (i.e., saturated) colors can be darkened by adding black to produce "shades," or lightened by adding white to produce "tints," of the original hue. Alternatively, pure colors can be muted ("grayed") by adding various combinations of black and white to produce various "tones" of the hue. All of these procedures produce colors that are desaturated compared to the original.

The HWB color space is based on these concepts (Figure 3.3). In this representation, *hue* is defined cyclically on the real-valued interval $[0, 6]$, such that both 0 and 6 indicate a full red, 2 indicates a full green, and 4 indicates a full blue in RGB space. *Blackness* and *whiteness* are both defined on the real-valued interval $[0, 1]$, and they are intended to

[a] 

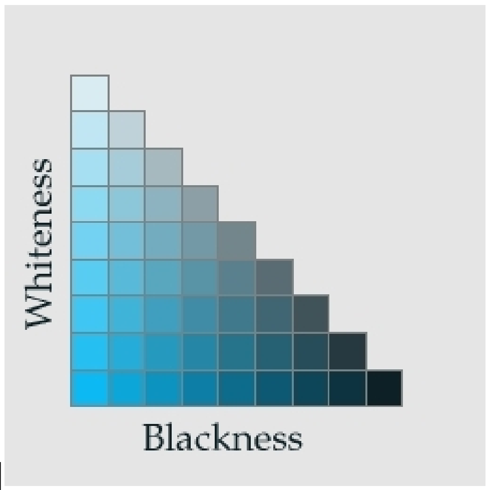
[b] 

Figure 3.3: HWB color space.

convey the amount of black or white that has been added to a given pure color. *Grayness* is defined as the sum of whiteness and blackness, $g = w + b$, and fully desaturated colors (i.e., gray scale) have grayness equal to 1. Since there is no meaning associated with a grayness value greater than 1, a single-hue slice in this space is triangular: if whiteness is placed on the vertical axis and blackness is placed on the horizontal, the fully saturated color is located at the origin, and the diagonal line from $(0, 1)$ to $(1, 0)$ defines the gray scale from white to black. Therefore, in the parlance of artists, tints are located along the vertical axis, shades are located along the horizontal axis, and tones occupy the remainder of the triangular space.

The HWB color space is closely related to the more common HSV (hue-saturation-value) color space [23] as follows: $h = h$, $b = 1 - v$, and $w = (1 - s)v$. As a result, grayness is given by $g = 1 - sv$. Therefore, blackness is simply a reverse indexing of the HSV value scale, but whiteness and grayness are linearly related to both value and saturation. Code for translating between the HWB, HSV, and RGB color spaces is given in the appendix of [56].

The main color wheel used in our system is a discretization of the HWB color space (Figure 3.4). Because the human eye has difficulty discerning colors in very small areas

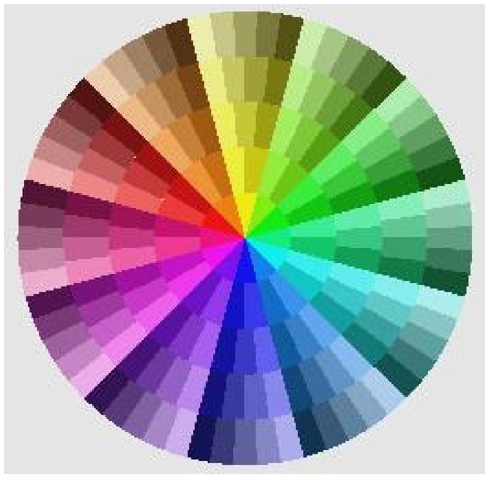


Figure 3.4: Main color wheel.

[23], we depict the color space with discrete color tiles rather than continuous color gradients. The triangular nature of the HWB space suits our purpose well, given that the color wheel is divided into discrete hue wedges, and each wedge depicts a compressed version of a triangular slice through HWB space.

The wheel is divided into twelve hue wedges, each wedge is divided into five concentric rings, and each ring is divided into individual color tiles. To index tiles within the wheel, we define i as the hue number, starting with $i = 0$ for red and proceeding clockwise; j as the ring number, starting with $j = 0$ at the tip of each wedge (i.e., at the center of the circle) and proceeding outwards; and k as the tile number within a given ring, starting with $k = 0$ for tiles along the counter-clockwise edge and proceeding clockwise. The HWB color associated with tile (i, j, k) is given by:

$$h = i\left(\frac{6.0}{n_h}\right) \quad (3.1)$$

$$b = \left(k + \frac{1}{2}\right) d \quad (3.2)$$

$$w = \left(j - k + \frac{1}{2}\right) d \quad (3.3)$$

where n_h is the number of hue wedges, and d is the distance in both the blackness and whiteness dimensions of the HWB space covered by a single color tile. As a result, the grayness associated with tile (i, j, k) is given by $g = (j + 1)d$. We use $d = 0.1585$ for aesthetic reasons. With this value, the color tiles display grayness values in the range $[0.1585, 0.9510]$, and we have found that colors outside this range appear either exceedingly bright (i.e., fabrics are not printed that brightly) or exceedingly gray (such that they appear misplaced) when included in the color wheel.

The triangular tile at the tip of each hue wedge represents the most saturated color, with both blackness and whiteness at minimum values. Tiles in a given ring have the same grayness, and outer rings represent more muted colors than inner rings do. Tiles along the counter-clockwise edge of a given wedge all have high value since blackness is at a minimum, and each tile moving clockwise within a given ring becomes increasingly darkened as blackness increases. Thus, in the parlance of artists, the tile in the tip of the wedge is the pure color, tiles along the counter-clockwise edge are the tints, tiles along the clockwise edge are the shades, and tiles in between are the tones associated with a given hue. The color wheel is oriented with yellow ($h = 1$) at the top to match a color wheel popular with quilters, given in [62].

At any given time, some color tiles in the main color wheel are outlined in black, and others are not. These outlines indicate the presence of at least one fabric in the database of fabric images, that is not currently displayed in the fabric browser, for which the outlined color is a "dominant" color. (Recall the definition of dominant colors given in the discussion regarding "Categorizing fabrics by color.") Users can select a color to add to the current color scheme by clicking on the associated color tile. In response, the selected color tile is outlined in white, a larger tile of the selected color is displayed in the color scheme palette, and the associated fabrics appear in the fabric browser. If a user selects a color tile that is not outlined in black, the color will be added to the color scheme palette, but no fabrics will be added to the fabric browser since there are no undisplayed fabrics associated with such a color tile.

An unlimited number of colors can be selected for inclusion in the color scheme.

Further, the white outline indicating that a color has been selected can be dragged from one color tile to another in order to replace one color with another in the color scheme. This operation can be used to browse the fabrics in the database one color at a time. If the white outline is dragged off the color wheel or onto a tile that is already selected, the outline disappears and the number of colors in the color scheme is reduced. For convenience, the color scheme can also be cleared with a single keystroke.

Just left of the main color wheel is an associated gray scale widget that represents colors too muted to belong to the chromatic portion of the color wheel. This gray scale widget works in tandem with the main color wheel and offers the same functionality.

Fabric browser

Fabrics associated with colors that have been selected for the current color scheme are displayed in the two windows of the fabric browser (Figure 3.5). Fabrics displayed in the left window contain significant amounts of more than one of the color scheme colors, so these fabrics are said to "bridge" colors in the scheme. Fabrics displayed in the right window contain only one of the color scheme colors, but for inclusion this color must be a dominant color of the fabric. Whenever the color scheme is modified, the fabrics in the fabric browser are automatically updated.

Within each window, fabric images are arranged in a rectilinear grid with high-value fabrics on the left, low-value fabrics on the right, muted fabrics on top, and brilliant (saturated) fabrics on the bottom. To accomplish this, a weighted average of the pixels in each fabric texture is computed and used for comparison. Sorting fabrics in this way provides information about the color relationships between various fabrics and also reduces the color distraction between neighboring fabric tiles.

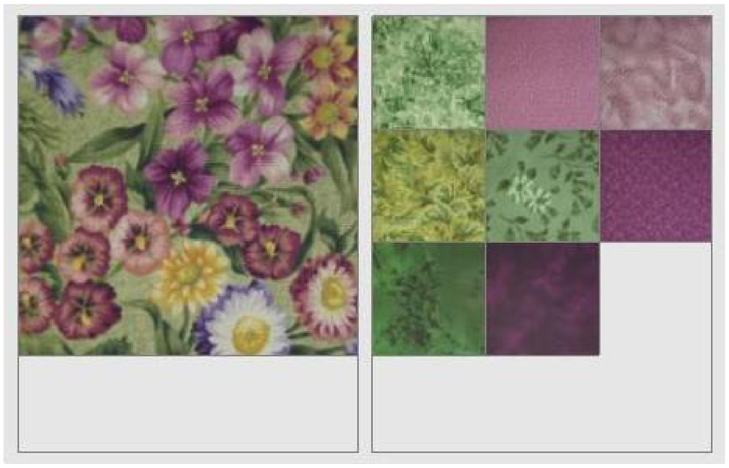


Figure 3.5: Fabric browser.

Fabric palettes

Fabrics can be selected for a given Bargello design by picking them from the fabric browser or a pop-up window associated with the color composition wheel. Selected fabrics are displayed in one of the three fabric palettes along the right edge of the fabric selection screen (Figure 3.6). At any given time, one of the palettes is wider than the other two, indicating that it is currently active, and fabrics are inserted into the active palette when they are selected. Fabric images are displayed as overlaid diamonds to allow an unlimited number of fabrics to be added to each palette: as needed, the images are simply overlaid more compactly. A user can change which palette is currently active by clicking on the desired palette. The active palette can be cleared with a single keystroke, or all fabric palettes can be cleared simultaneously with two keystrokes.

As long as the currently active palette contains at least one fabric, then some fabric in the palette is also deemed currently active. Clicking on a fabric makes it current and brings it to the front such that it partially occludes its neighbors. The fabric can then be removed from the palette or dragged to a new position within the palette.

When a Bargello curve is drawn in the design space, it is associated with the fabric palette that is active at the time of its instantiation. Thereafter, modifying the fabrics

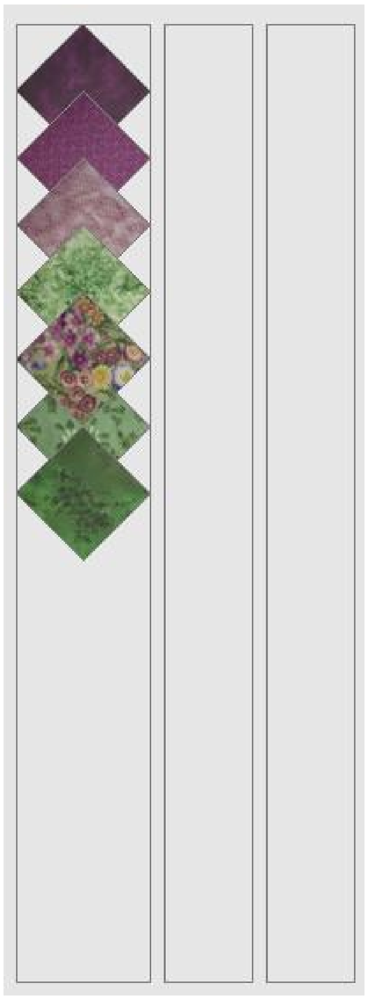


Figure 3.6: Fabric palettes.

or the order of fabrics within the palette causes a corresponding modification in the color strata associated with the Bargello curve. The existence of multiple fabric palettes allows designs to include multiple color strata. It would be straightforward to provide an unlimited number of fabric palettes, and therefore an unlimited number of color strata in each design, but in practice we have found three palettes to be sufficient. In designs where all three are used, one holds the dominant color strata, another holds a secondary strata, and the third is used for background areas.

Color composition wheel

The color composition wheel, shown in Figure 3.7, presents a decomposition of the colors in the active fabric (i.e, the fabric displayed "on top" in the active fabric palette). This color wheel has the same arrangement as that of the main color wheel, but at any given time various color tiles may be filled with solid colors, hatched with one of two hatching patterns, or represented with a wireframe display. Solid color tiles indicate colors that are dominant in the active fabric, a dense hatching pattern indicates colors that are significant (but not dominant), a sparse hatching pattern indicates colors that are present (but not significant), and wireframe tiles indicate colors not found in the active fabric. Presenting the color decomposition this way provides information about the relationships between the colors in the active fabric as well as the prevalence of each color.

In addition, the color composition wheel can be used to browse and select fabrics that include color matches with the currently active fabric. Clicking on a color tile invokes a pop-up window that displays fabrics containing the indicated color, and clicking on a fabric in the pop-up window adds the fabric to the currently active fabric palette. This mechanism allows users to select fabrics for the design without first adding them to the fabric browser.

Thus, the various widgets and windows in the fabric selection module work together to allow users to explore color combinations and make fabric selections in multiple ways.

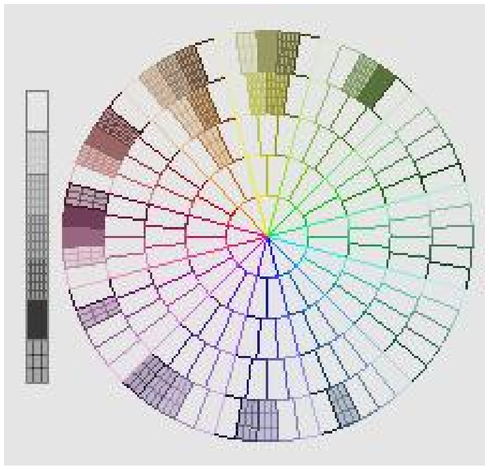


Figure 3.7: Color composition wheel.

One user may come to the design process with a specific color scheme in mind, in which case she can select the desired colors from the main color wheel and then select fabrics from the fabric browser that collectively represent that color scheme. Another user may prefer to begin by browsing the stock of available fabrics for an attractive focus fabric and then consult the color composition wheel to locate fabrics that provide color support for the selected focus fabric. The system supports various approaches to the fabric selection task with the expectation that doing so will lead to the most rewarding user experience during this challenging phase of quilt design.

3.2 Bargello design

The Bargello design module allows users to create and experiment with Bargello quilt designs. In this section we first describe the module's functionality from a user perspective and then its implementation from a programmer perspective.

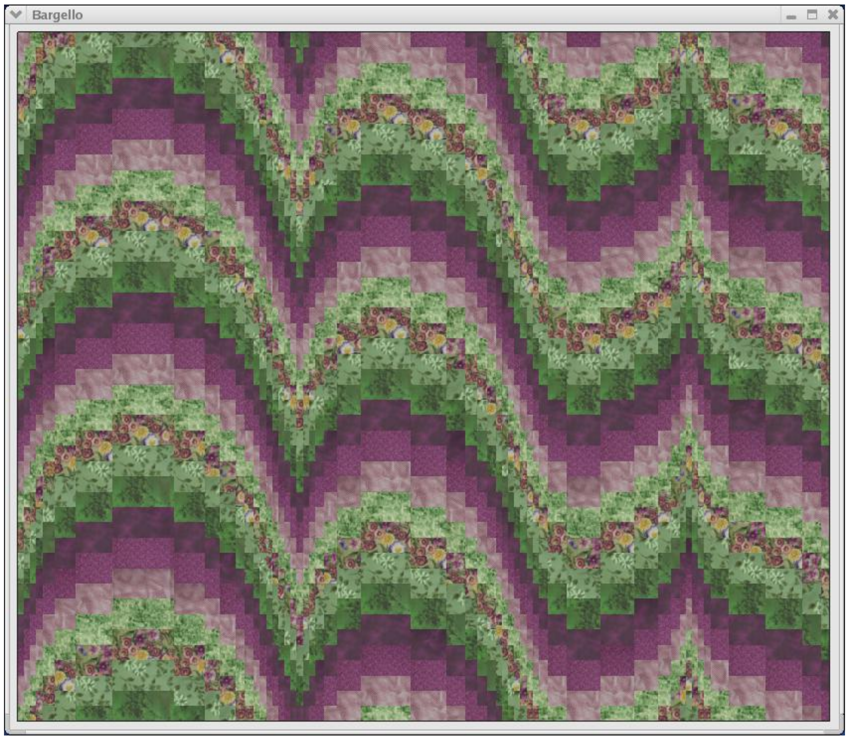


Figure 3.8: Bargello design: Traditional, with staggered seams.

3.2.1 Functionality

Traditional designs

All that is required to create a design with the simple elegance of Figure 3.8 is to draw the curve with a mouse or stylus, starting near either the left or right margin of the design space and proceeding to the opposite margin, tracing out the vertical inflections desired. As the user draws, the system transforms the input stroke into a graceful Bargello curve and spawns parallel curves to fill the vertical extent of the design space. The resulting curves conform to the Bargello design constraints described in Section 1.2. In addition, the system enforces a lower bound on the width of each vertical strip and ensures that each width is drawn from a discrete set of possible widths. These physical constraints ensure that the final design can be constructed in fabric: the pieces can be cut using a discretely marked ruler, and each vertical strip is wide enough to be manipulated by hand.

A user can then experiment with the design in several ways. Curves can be dragged vertically to place a specific color strip at a particular position in the design space. They can also be “flipped” (i.e., reflected through the horizontal midline). The fabric pieces can be arranged with matched seams or staggered seams (see Section 1.2). The choice of fabrics or the fabric order in the color strata can be changed by making the corresponding modifications to the active fabric palette in the fabric selection module, and finally, the design can be cleared to start again.

Some fine points related to the curve drawing process follow:

- When a curve is drawn, it is associated with the currently active fabric palette. If that fabric palette does not happen to contain any fabrics, the curve is represented with a default set of gray scale colors until such time as fabrics are added to the palette.

- If the user begins or ends the input stroke sufficiently near to the corresponding edge of the design space, the resulting Bargello curve is snapped to the edge. However, if the endpoints of the input stroke are well inside the design space, the system places the endpoints of the Bargello curve as the user indicated.

- While drawing a curve, if a user reverses the horizontal direction of the input stroke, by default the system responds by erasing the latter portion of the Bargello curve as the stroke backtracks over it. This allows users to correct a portion of the curve without lifting the pen.

Symmetrical designs

To support the creation of symmetrical designs, we have defined a set of *reflection templates.* Figure 3.9 shows an example of an empty reflection template. The design space is partitioned into regions, and each region contains a watermark indicating the type of symmetry that will be imposed on designs drawn in the template. In this case, the

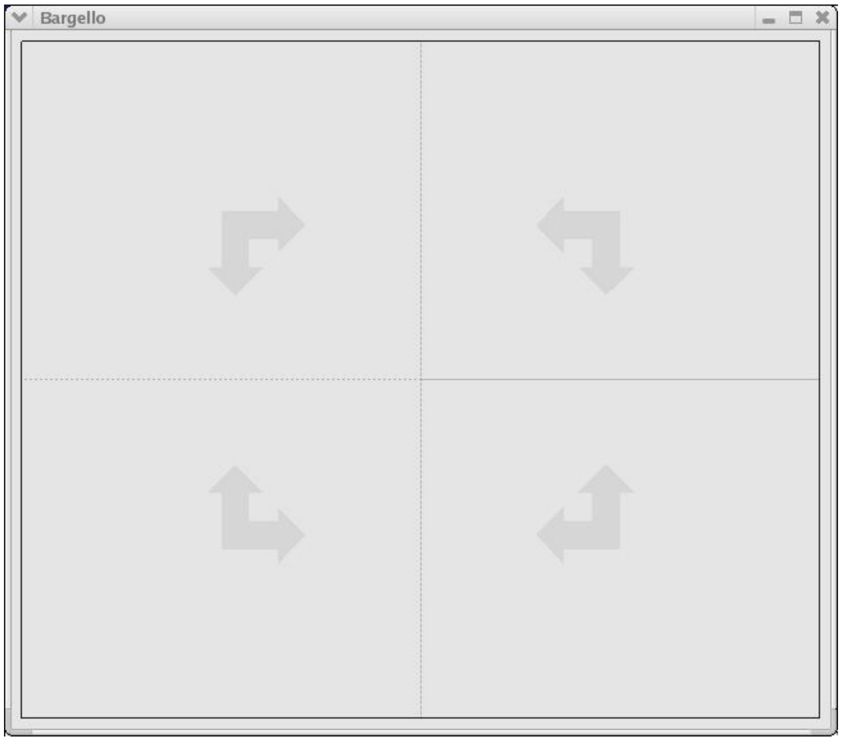


Figure 3.9: Reflection template.

resulting designs will have both vertical and horizontal symmetry, as does the design in Figure 3.10. Other reflection templates include: a template with two horizontally symmetric regions that extend the full height of the design space; a similar template with four full-height regions and horizontal symmetry between adjacent regions; and a template with eight regions, four above and four below the vertical midline of the design space, that impose both horizontal and vertical symmetries on the design.

When drawing in a reflection template, the user may begin drawing in any region. However, since an input stroke drawn in one region defines the entire design, the system expects the stroke to remain within the region in which it began. If it strays into another region, the system waits for it to return to the specified drawing region and then continues the curve from the point of re-entry. All the operations previously described for modifying Bargello designs are also available for designs created in the reflection templates.

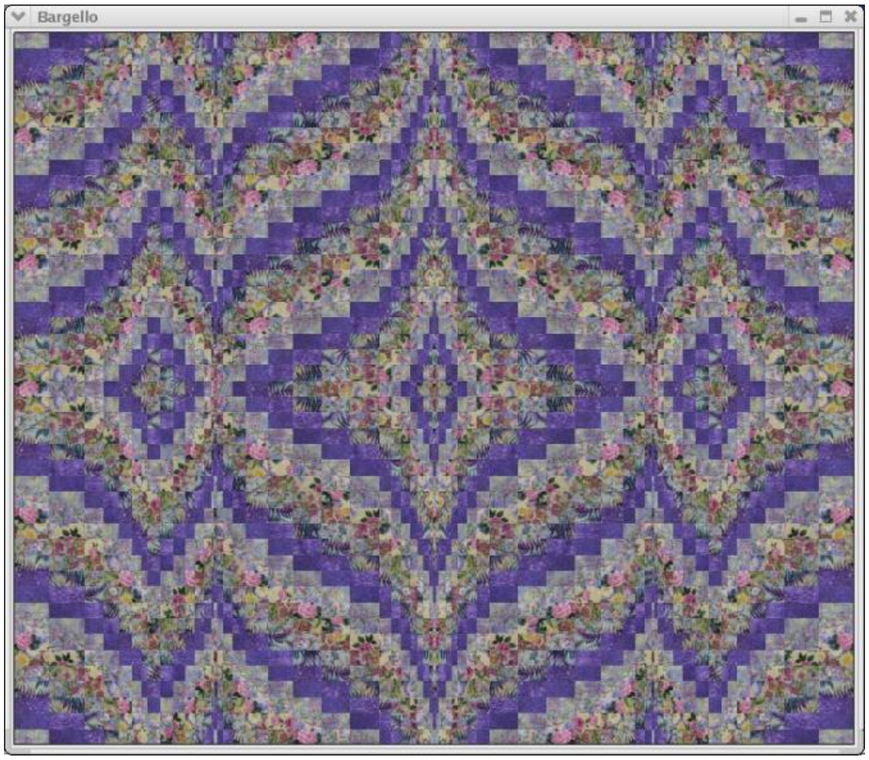


Figure 3.10: Bargello design: Symmetrical, with matched seams.

Contemporary designs

Edie [19] provides many examples of complex Bargello designs that fascinate the eye. In some, the fabrics used in a given color strata change over the length of a Bargello curve, in others, contrasting curves are overlaid on top of a motif curve, and in a striking few, the motif curve is repeated in variations throughout the quilt (e.g., Figure 1.9). Some designs give the illusion of depth, as secondary curves appear to emerge from behind the motif curve or a single curve switches back to create a 3D figure. Other quilts include horizontal stripes and solid regions as a background for a central design.

To create designs such as these, users must be able to draw discrete curves (i.e., curves that consist of a finite number of color strips that do not fill the vertical design space), they must be able to add multiple curves to a design, and they must be able to modify existing curves in a variety of ways. Further, they must be able to select an individual curve, or a set of tiles within a curve, to work with. Our system provides this functionality.

When a user draws a curve, the default system behavior is to produce an “infinite” set of parallel color strips that fill the vertical design space, but a preference can be set such that curves drawn thereafter are rendered with a single color strata. Further, a finite set of color strips within an “infinite” set can be selected to keep, and the remainder can be discarded. Thus, designs can include discrete curves, and these curves can include an arbitrary number of color strips.

Similarly, the default system behavior when the mouse or stylus reverses direction horizontally is to erase the latter portion of a curve, but this behavior is governed by a state variable that can be altered by the user. Thus, users can create curves with switchbacks (i.e., curves that are not single-valued horizontally), and this technique can be used to create curves with a 3D appearance (e.g., Figure 3.18).

Once an initial curve has been drawn, additional curves can be added to a design in three ways. First, users can draw additional curves from scratch. Second, users can select an existing curve, or a set of tiles within an existing curve, and create either an exact copy or a reflected copy of the selected tiles. Third, users can create horizontal stripes to fill space in background regions of the design.

In addition, the following operations are useful for modifying curves in complex designs. Users can

- reverse the fabric sequence within a color strata;
- reflect a curve, or a portion of a curve, vertically;
- alter the apparent depth of a curve, or a portion of a curve, either bringing it to the front or sending it to the back, such that it will occlude or be occluded by other curves in the design;
- substitute different fabrics into a set of tiles without modifying the basic color strata or the fabrics in neighboring tiles; and

- delete a set of tiles from a curve.

These operations require that users be able to select curves or sets of tiles within curves. To select an entire curve, the user simply double-clicks on it, and there are two ways to select a set of tiles within a curve. To quickly select all tiles in a color strip (row) or all tiles in a vertical strip (column), the user drags the mouse or stylus with the “control” modifier key pressed. An arbitrary “rectangular” set of tiles, including a set containing a single tile, can be selected by dragging the input device with the “shift” modifier key.

Curves can also be dragged horizontally; however, to ensure that this does not happen accidentally, the system provides two separate curve dragging modes. If a curve is dragged with the left mouse button depressed, it can only move vertically, but if it is dragged with the right mouse button (or a physical button on the stylus) depressed, it can move in any direction. Although the existence of two modes for curve dragging adds to the set of commands an advanced system user needs to master, this nuisance is offset by the convenience of vertical drags with the horizontal position locked in place. We have found this to be helpful in designs that include multiple curves since it allows one to be dragged vertically with respect to the others while ensuring that they retain their horizontal alignment.

Finally, we point out that all of the operations described in this section are available at all times in the Bargello design module. The design process has been described separately for different design styles as a matter of convenience.

3.2.2 Implementation

In this section, we describe the core data structures used by the Bargello design module and the related processes that enable users to create and modify Bargello designs.

Bargello curves

Prior to this point we have defined a Bargello curve as a connected path of rectangular virtual fabric tiles within a Bargello design. We now expand the definition to include an associated parallel set of such paths. The set can be finite or infinite, indicating Bargello curves that include a specific number of parallel paths, or curves that fill the vertical extent of the design space, respectively. Recall from Section 1.2 that a single path within such a parallel set of paths is called a *color strip*.

To represent such a data structure internally, we store a single tile path that defines the geometry of the curve and a pair of integer bounds that specify the number of associated color strips and their location with respect to the stored tile path. Each bound is given as a number of rows, counting upward from the stored path. So for example, a Bargello curve that includes four color strips, two above the stored path, one coincident with the stored path, and one below the stored path is bounded by the relative row numbers $[-1, 2]$. Bargello curves with an "infinite" set of associated color strips are bounded by $[-int\_max, +int\_max]$, and for convenience we also store variables to flag this situation.

In addition, each Bargello curve is associated with one of the fabric palettes in the fabric selection module. Fabric textures from that palette are associated with color strips in the curve in a repeating sequence, where the first (uppermost) texture in the palette is associated with the color strip coincident with the stored tile path, and subsequent textures in the palette are associated with subsequent color strips below the stored tile path. As a result, the color strip directly under the cursor as the user draws, which is coincident with the stored tilepath, is depicted with the first fabric texture in the palette, and the fabric ordering in the design matches that of the palette.

Table 3.1 summarizes the variables stored in a *BargelloCurve* object in our implementation. A *Tilepath* is an object that contains a vector of tiles, each of which occupies a specific location in the design space. A BargelloCurve contains two Tilepaths, *coverTP* and *displayTP*. As described in Section 2.2.4, the former is an initial rasterization of an input sketch, and the latter is a "beautified" version that defines the geometry of the Bargello curve. *loBound* and *hiBound* delineate the vertical positions of associated color strips with respect to the beautified tile path, while *fillUp* and *fillDown* indicate whether the Bargello curve fills the vertical extent of the design space above and below the beautified tile path, respectively. *palette* records the index of the associated fabric palette in the fabric selection module, and *reverseFKeys* indicates whether the color strips are displayed in the same order as, or are reversed from, the fabric sequence given in the associated fabric palette. *depth* is used in a z-buffer algorithm to determine the relative order of various Bargello curves in a design. Finally, *deletedTiles*, *fabricExceptions*, and *depthExceptions* each list individual tiles that are depicted non-standardly: tiles may be deleted (i.e., made invisible), associated with fabric textures that do not correspond with the associated fabric palette, or displayed at a depth other than the depth of the curve as a whole.

A new Bargello curve is generated each time an input curve is drawn in the design module. The BargelloCurve data object is created when the first data point is received. It is initialized with a link to the "current" fabric palette in the fabric selection module and a depth that places it "in front" of any pre-existing Bargello curves in the design. The vertical extent of the new BargelloCurve object depends on the value of a state variable: by default new curves fill the vertical design space, but they can be created as discrete curves in which the uppermost row is directly under the cursor and the curve extends downward to form a single color strata (i.e., a single period of the associated fabric sequence). As additional data points are received they are thinned, and the surviving points are stored temporarily. The two Bargello tile paths are re-generated from this set of input points each time a new point is accepted. Thus, as the user draws, the Bargello curve "grows" to follow the cursor. If the horizontal direction of the input stroke is reversed, a state variable governs the result: by default data points are removed from the input list such that subsequent iterations of the curve generation process produce

Variable name	*Data type*	*Purpose*
coverTP	Tilepath	initial rasterization of sketched input
displayTP	Tilepath	beautified tile path: stores geometry of Bargello design
hiBound	integer	highest row occupied, relative to displayTP
loBound	integer	lowest row occupied, relative to displayTP
fillUp	enumeration	curve extends upward to fill design space
fillDown	enumeration	curve extends downward to fill design space
palette	integer	index of associated fabric palette
reverseFkeys	boolean	fabric order is reversed from that in palette
depth	integer	used in z-buffer algorithm
deletedTiles	set$\langle int \rangle$	tiles that are not displayed: $\langle tileNum \rangle$
fabricExceptions	map$\langle int, int \rangle$	tiles with non-standard fabrics $\langle tileNum, palette \rangle$
depthExceptions	map$\langle int, int \rangle$	tiles with non-standard depths $\langle tileNum, depth \rangle$

Table 3.1: Data dictionary: BargelloCurve.

shortened tile paths (i.e., the latter portion of the Bargello curve is erased), but Bargello curves can also be created with switchbacks, in which case the new data points are simply added to the list and the curve generation process continues as normal. Details of the curve generation, rasterization, and beautification algorithms that transform the input data into Bargello tile paths are given in Sections 2.1.4 and 2.2.4.

Modifications can be made to various attributes of a BargelloCurve object via a set of design operations. However, we defer discussion of the implementation of these operations until the details of selecting tiles within a Bargello design have been presented.

An array of BargelloCurve objects is maintained in the Bargello design module to support designs that contain multiple Bargello curves.

Bitmap

The Bargello design module also maintains a "bitmap" of data regarding the current design. The bitmap partitions the design space into a two-dimensional array of rectangular cells arranged in a uniform rectilinear grid, and data is stored in each array element regarding the composition of the design in the corresponding location. The width of each cell corresponds to the minimum-width increment that fabric tiles may take on, and the height of each cell corresponds to either the full height, or half the height, of fabric tiles in designs with "matched-seams" or "staggered-seams" patterns, respectively. (See Section 1.2). Thus, a virtual fabric tile can occupy multiple cells in the bitmap, including either one or two grid rows, and perhaps several grid columns.

Table 3.2 provides a summary of the information stored in each bitmap element. The tile that currently occupies the corresponding cell is identified by *curveNum* and *tileNum*, where *curveNum* refers to the index of a Bargello curve within the array of curves stored in the Bargello design module, and *tileNum* identifies a specific tile within the Bargello curve. The depth of the tile stored in the cell is given by *depth*, and *fkey* indicates the fabric texture that will be used for subsequent display.

Variable name	*Data type*	*Purpose*
curveNum	integer	identifies Bargello curve
tileNum	integer	identifies a tile within a Bargello curve
depth	integer	used in z-buffer algorithm
fkey	integer	identifies a fabric texture

Table 3.2: Data definition: bitmap element.

Tiles within each Bargello curve are numbered as follows. First, the color strips that comprise the Bargello curve are given relative row numbers (which may be negative), counting upwards from the tile path stored in the BargelloCurve object. Then, tiles within the Bargello curve are numbered as

$$tileNum = n_t i + j, \tag{3.4}$$

where n_t is the number of tiles in the stored tile path, i is the row number of the associated color strip, and j is the index of the tile within the color strip (i.e., the index of the corresponding tile in the stored tile path).

When modifications are made to any Bargello curve or an associated fabric palette, the bitmap is cleared and all Bargello curves in the design are "painted" into the bitmap with a z-buffer algorithm. For each Bargello curve, the beautified tile path is traversed, and the tiles in the curve are painted column-wise, working from the color strip coincident with the tile path outwards in each direction to the vertical boundaries of the Bargello curve or the edges of the design space, whichever comes first. For each tile, the corresponding tile number is computed according to Eq. 3.4, the depth is taken from the Bargello curve, and the corresponding fabric key is determined by

$$f = \begin{cases} (n_f - (i \;\%\; n_f)) \;\%\; n_f & \text{for } i > 0 \\ -i \;\%\; n_f, & \text{for } i \leq 0. \end{cases} \tag{3.5}$$

In this equation, % is the modulus operator, f indexes the array of fabric keys in the associated fabric palette, n_f is the number of fabrics in the palette, and i is the row number of the associated color strip. Next, the three exception lists in the Bargello curve are checked to determine whether the tile's depth or fabric key have been overridden and whether the tile has been "deleted."

Finally, assuming the tile has not been deleted, the accumulated data is written into the bitmap cells covered by the tile. The number of grid columns a tile occupies is determined by its width, and the number of grid rows is determined by a state variable in the design module that indicates whether the design has "matched-" or "staggered-seams." The data must be written in each cell, rather than once per tile, because tiles in one Bargello curve may be subdivided into multiple horizontally-adjacent tiles as other Bargello curves are painted into the bitmap. In that case, some cell in each tile must contain the necessary data, yet at the time a given curve is painted we do not know which cells may be affected. Thus, the data is written into all occupied cells.

In addition, a vector *vLines* of boolean values is maintained to indicate which grid columns bound design columns (i.e., which grid columns coincide with vertical edges of tiles in the design). This information is used during the beautification of Bargello curves and also as the design is displayed.

To display the design, each virtual fabric tile is sent down the graphics pipeline individually. The tiles are displayed column-wise, using the *vLines* array to determine the boundaries of each design column. The height of each tile is determined by the number of contiguous grid rows occupied by the tile, as indicated by the stored values of $curveNum$ and $tileNum$. This approach is general enough to serve in the future when various color strips in the design may have differing heights. Tiles are ordinarily depicted with the associated fabric texture, and to avoid unrepresentative repetitiveness in the displayed design, various portions of a given fabric texture are used to display the different tiles depicted with that texture. However, if a tile has been selected for further processing at the time the design is displayed, it is drawn as a solid color tile in the average color of the associated fabric. This average color is computed once and stored in

Variable name	*Data type*	*Purpose*
lowLeft	Point	position of lower-left region corner (world coordinates)
width	double	region width (world coordinates)
height	double	region height (world coordinates)
scaleX	integer	indicates horizontal reflection status: values in $\{-1, +1\}$
scaleY	integer	indicates vertical reflection status: values in $\{-1, +1\}$

Table 3.3: Data definition: design region.

the associated fabric object.

Reflection templates

When users invoke one of the pre-defined reflection templates, the design space is partitioned into a set of equal-size regions, and a corresponding array of *region* objects is stored in the Bargello design module. Table 3.3 summarizes the data maintained for each region object. The location and size of a region is given by *lowLeft*, *width*, and *height*, and the symmetry properties of the region are given by *scaleX* and *scaleY*. The latter two variables can take on the values -1 and +1, indicating that the region is, or is not, reflected with respect to the region in the lower-left corner of the design, which we call region 0.

In addition, the bitmap is re-sized to store data regarding region 0 only, yet users may interact with the design in any of the regions. To support this, each input point received from the mouse is transposed to the corresponding point in region 0, regardless of the type of interaction. Thus, internally, all interactions with the design occur within region 0, and all modifications to the design are recorded in region 0.

To display designs created in reflection templates, the bitmap data recorded for region 0 is displayed repeatedly, and the result is translated and reflected according to the attributes of each region to cover the design space and produce the requisite symmetries in the final design.

Select range

The Bargello design module supports the creation of complex contemporary designs with a set of design operations that modify various attributes of existing Bargello curves. To invoke these operations on particular tiles within the design, users first select individual Bargello curves or tile sets within a given curve. To support this, a *SelectRange* object is maintained in the Bargello design module to delineate the current set of selected tiles.

Table 3.4 summarizes the data stored in the SelectRange object. A select range must consist of a "rectangular" set of tiles within a single Bargello curve; in other words, the set must consist of a contiguous set of color strips in a Bargello curve and a contiguous set of tiles in each color strip. Thus, *curveNum* specifies the Bargello curve from which tiles are selected, and four variables (*loTile, hiTile, loRow, hiRow*) delineate the selected tile set. Two boolean variables, *fullHgt* and *fullWid*, are used for convenience to record whether the select range includes an entire column or row of tiles, respectively. Finally, *tpTiles* gives the number of tiles in the beautified tile path of the Bargello curve: it is used to convert between tile numbers, as stored in the bitmap and defined by Eq. 3.4, and the opposite-corner approach to tile numbering used here.

We use the opposite-corner approach, rather than delineating select ranges with tile numbers, because select ranges that do not include full color strips contain tiles that are not contiguously numbered. In future, we may allow arbitrary tile sets to be selected, but to do so, an exhaustive list of the selected tiles will need to be maintained.

Users can select Bargello curves by double-clicking on them, or they can select tiles within a curve by clicking on one tile and dragging the mouse to another tile within the same curve while depressing a modifier key. In the latter case, as each data point is received from the mouse the bitmap is queried to determine which tile the cursor occupies. If the result is different from the previous query but remains within the selected Bargello

Variable name	*Data type*	*Purpose*
curveNum	integer	identifies Bargello curve from which tiles are selected
fullHgt	boolean	is full height of Bargello curve selected?
fullWid	boolean	is full length of Bargello curve selected?
tpTiles	integer	number of tiles in tile path of Bargello curve
loTile	integer	lowest tile number included in select range
hiTile	integer	highest tile number included in select range
loRow	integer	lowest row number included in select range
hiRow	integer	highest row number included in select range

Table 3.4: Data definition: SelectRange.

curve, the select range is updated accordingly.

Each time a Bargello design is displayed, the SelectRange object is queried to determine the selection status of each tile in the design. Tiles that are not selected are drawn with the associated fabric texture, and selected tiles are depicted with the average color in the fabric texture. This approach results in noticeably different appearances between selected and unselected tiles, yet maintains some continuity of appearance between tiles within a color strip. Without this continuity, selected tiles may no longer appear to belong to their respective curves given that texture is the single identifying attribute of tiles within a Bargello design.

Design operations

Each of the design operations supported in the Bargello design module has a straightforward implementation. The challenge was to define a set of operations that would allow quilters to create the designs they envision and to define the behavior of those operations such that they would seem "intuitive." This is an open-ended problem we have not solved completely. However, our approach has been to define operations that could be used to create virtuoso designs such as those presented by Edie in [19]. We also insist that all operations preserve the "integrity" of Bargello curves in the sense that they do not disconnect tile paths or break the parallelity between color strips. This ensures that sets of tiles that are internally represented as a single Bargello curve are also visually coherent. Otherwise, subsequent operations on the curve can have surprising results.

The first set of operations we present modify attributes of a Bargello curve. One operation toggles the boolean *reverseFKeys*. Another links the curve, or tiles within the curve, to a new fabric palette. If the full curve is selected, the variable *palette* is updated with the index of the "current" palette in the fabric selection module; otherwise, an element is added to the list of *fabricExceptions* for each tile in the select range. Modifications to the apparent depth of a curve, or tiles within the curve, are handled analogously. Curves can be moved only "to the front" or "to the back" of a design, so every Bargello curve is queried for the highest or lowest depth of any member tile, and the next larger or smaller integral depth is assigned. Deleting curves, or tiles within curves, works similarly. If the full curve is selected, the curve is removed from the array of Bargello curves; otherwise, the selected tiles are added to the list of *deletedTiles*. Finally, to extend discrete curves to fill the vertical design space, *hiBound* and *loBound* are set to int_max and $-int\_max$, respectively. Conversely, to discard color strips outside a selected range of an "infinite" curve, *hiBound* and *loBound* are set to the relative row numbers of the remaining color strips (i.e., to *hiRow* and *loRow* of the select range). In both cases, *fillUp* and *fillDown* are also modified accordingly.

To reflect a Bargello curve vertically, both tile paths within the curve are traversed and their tiles are displaced vertically. However, a complication arises in determining what horizontal line the curve should be reflected through. There are two cases. First, if a full Bargello curve is selected, the beautified tile path is queried for the locations of the highest and lowest tiles with respect to the design space, and the two tile paths are reflected around the horizontal midpoint between these tiles. As a result, the reflected curve occupies the same vertical region of the design space that it did previously, which ensures that the curve will remain within the design space. Alternative approaches such as reflecting the curve around the first tile in the path do not ensure this. Second, if a subset of tiles is selected that includes either the first or last column of tiles in the Bargello curve, then the selected portion of the tile path is reflected around the horizontal midpoint of the innermost tile in the select range, which ensures that connectivity is preserved. Finally, to ensure the color strips remain parallel, a reflect operation modifies all color strips within a Bargello curve even if they are not all included in the select range.

When a new Bargello curve is created as a copy of all or part of an existing Bargello curve, the new curve is linked to the parent's associated fabric palette, and it is given a depth that places it "in front" of other curves in the design. The variables governing vertical extent are set to reflect the selected portion of the parent curve, and both tile paths within the new curve are created as copies of the selected portion of the corresponding tile paths within the parent curve. Finally, the new curve is translated vertically such that its lowest color strip is adjacent to the parent's highest color strip: without this, the new curve would not be visible. The operation which creates new Bargello curves as reflections of selected portions of existing curves is essentially a composition of the copy operation and a subsequent reflection. Finally, to add a "horizontal" Bargello curve (i.e., a set of horizontal stripes) to a design, a curve is created with a tile path that consists of a single tile which covers the horizontal extent of the design space. In this case, the curve is given a depth that places it "behind" existing curves since this operation is used to fill background regions.

When a Bargello curve is dragged within the design space, the bitmap is queried for each data point received to determine whether the mouse has entered a new grid cell. If so, both tile paths of the Bargello curve are translated by an integral number of grid rows or grid columns, as needed. The tile paths must move in discrete jumps to satisfy the constraint that allows only a discrete set of possible tile widths. To maintain connectivity, the entire Bargello curve is translated even if only a subset of tiles in the curve is selected. Two drag operations are provided: one allows curves to be dragged in any direction, and another allows curves to be dragged vertically with the horizontal position locked in place. When a curve is dragged vertically only, the operation has no affect on the widths of the Bargello design columns, so no special processing is needed. However, when a curve is dragged horizontally, its tangents at each horizontal position in the design space are modified, which in turn modifies the Bargello columns. If multiple curves are present in the design, they all undergo "re-beautification" to re-establish a Bargello grid that is a mutual compromise between the various curves. This process is explained in detail in Section 2.2.4.

3.3 Image gallery

The Bargello designs shown in Figures 3.8, and 3.10 through 3.19 were created in our computer-assisted Bargello design system.

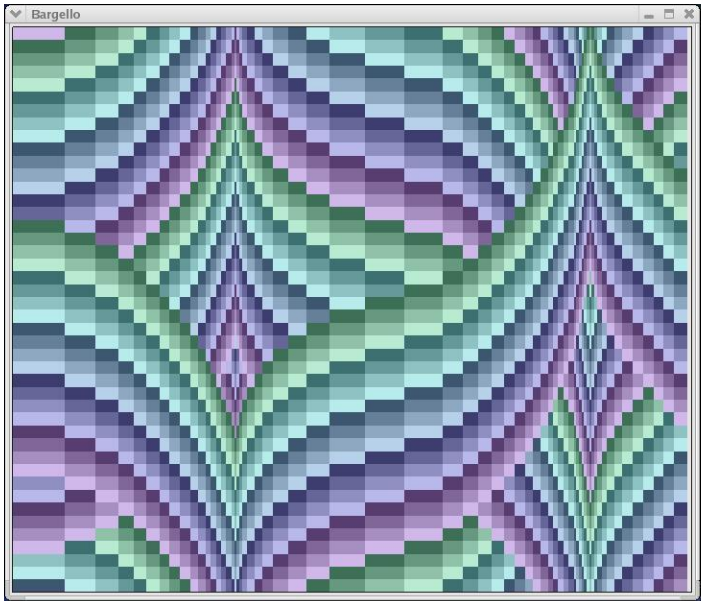


Figure 3.11: This design contains two discrete Bargello curves in solid color tiles. At the bottom of the design, tile depths have been modified to make the curves intermingle.

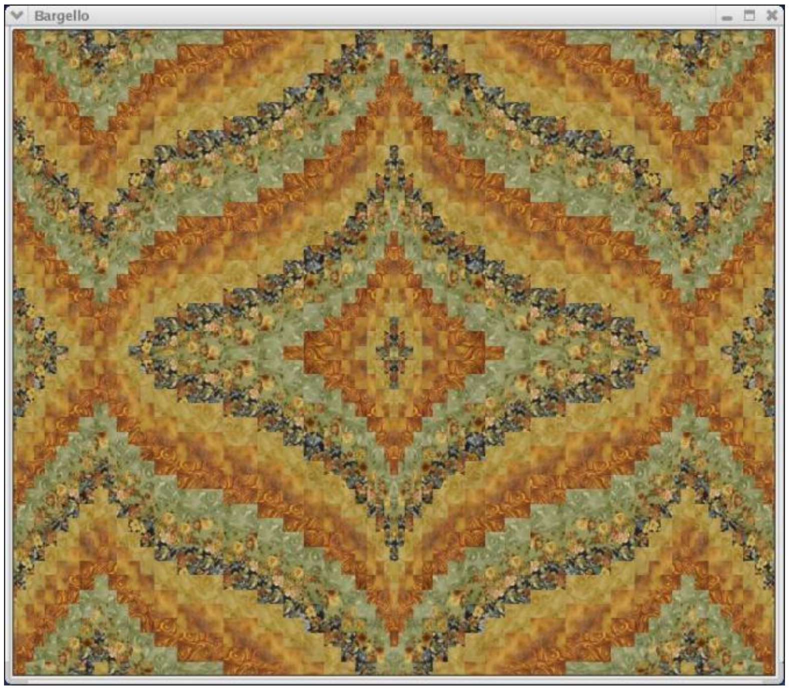


Figure 3.12: This design was drawn in a four-way symmetric reflection template with closely color-matched fabrics that blend together to produce a “watercolor” effect.

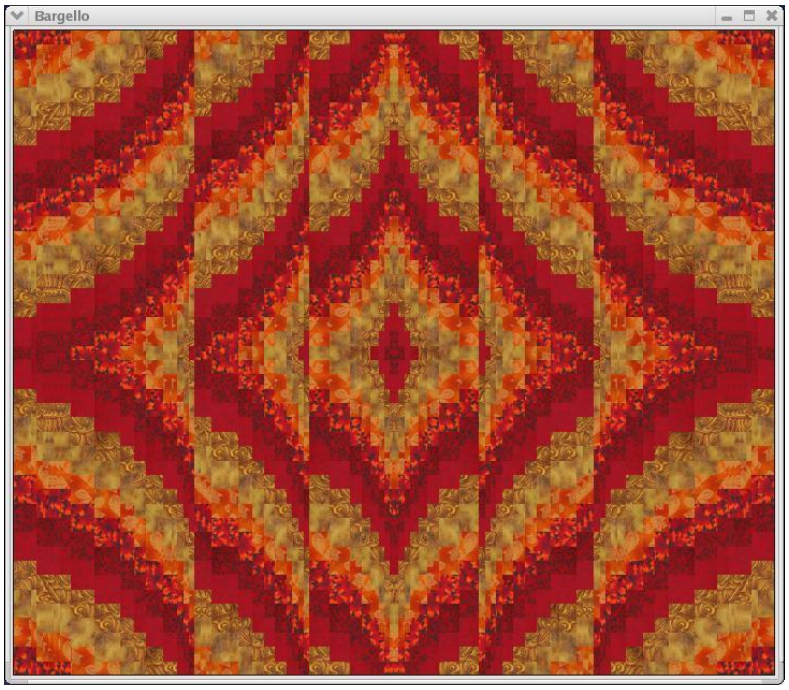


Figure 3.13: The design technique used here was discovered accidentally: the pen wove in and out of the “drawing region” of a reflection template in a elliptical motion.

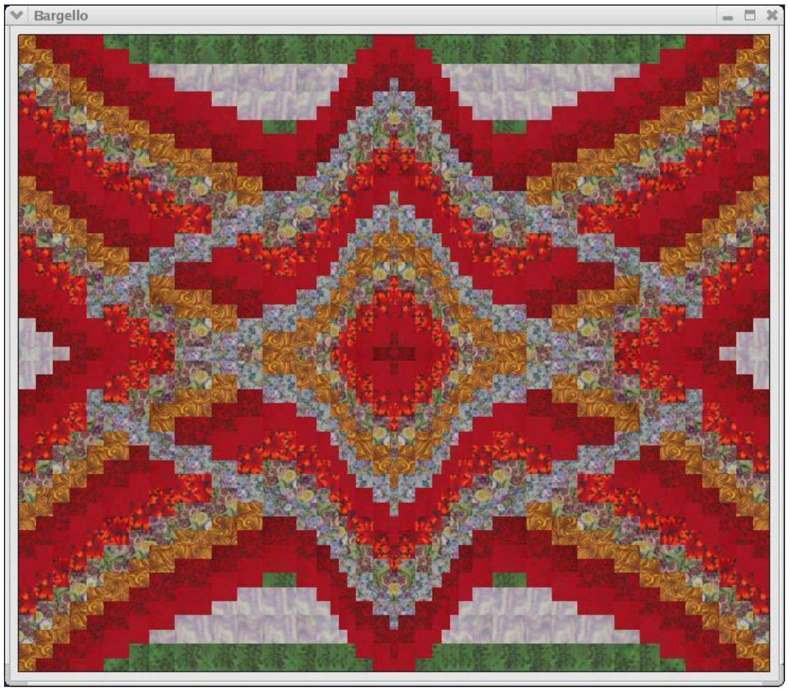


Figure 3.14: This design includes multiple layers of color, including horizontal stripes that fill background regions.

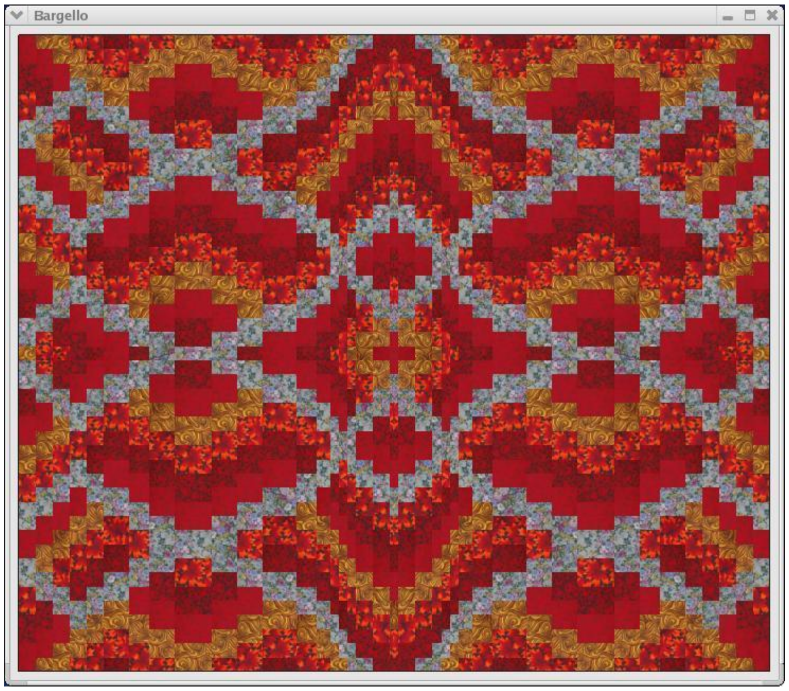


Figure 3.15: In this design, the blue color strips began as copies of red color strips. Then a blue fabric was substituted and segments of the color strips were reflected repeatedly.

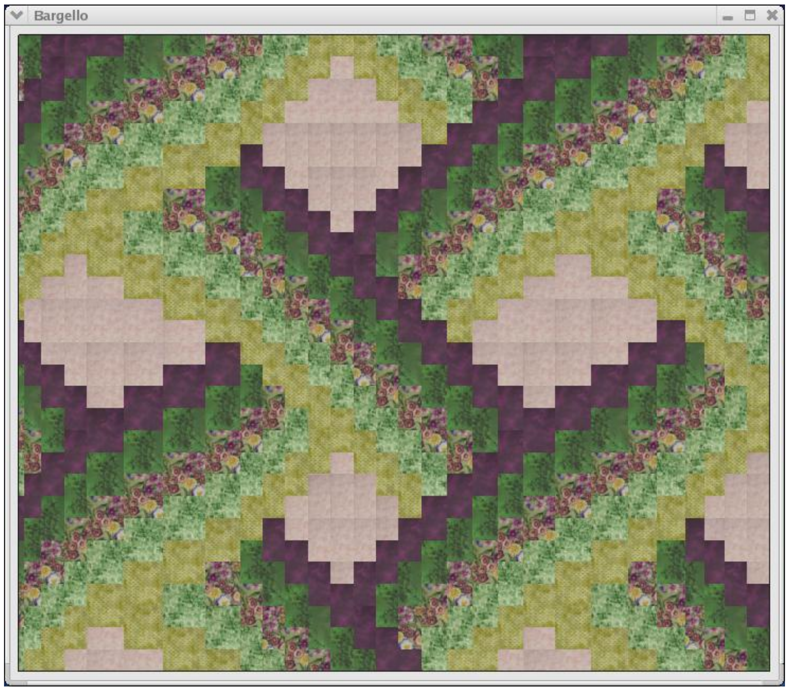


Figure 3.16: Only one curve was drawn to make this design: the others were copied and reflected. Then the depths of various tiles were modified to create a basketweave effect.

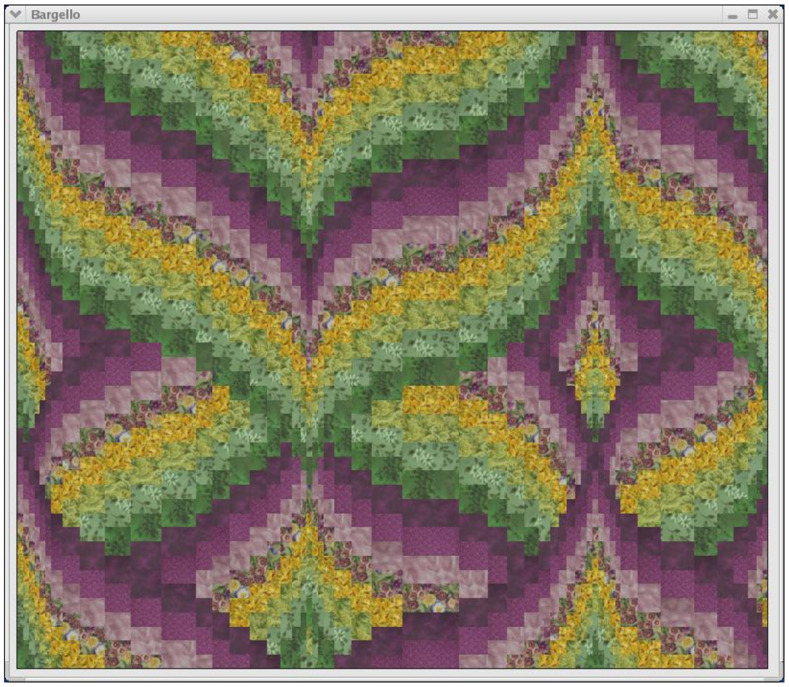


Figure 3.17: This design was also created by drawing a single design curve and then placing copies of the curve at various depths within the design.

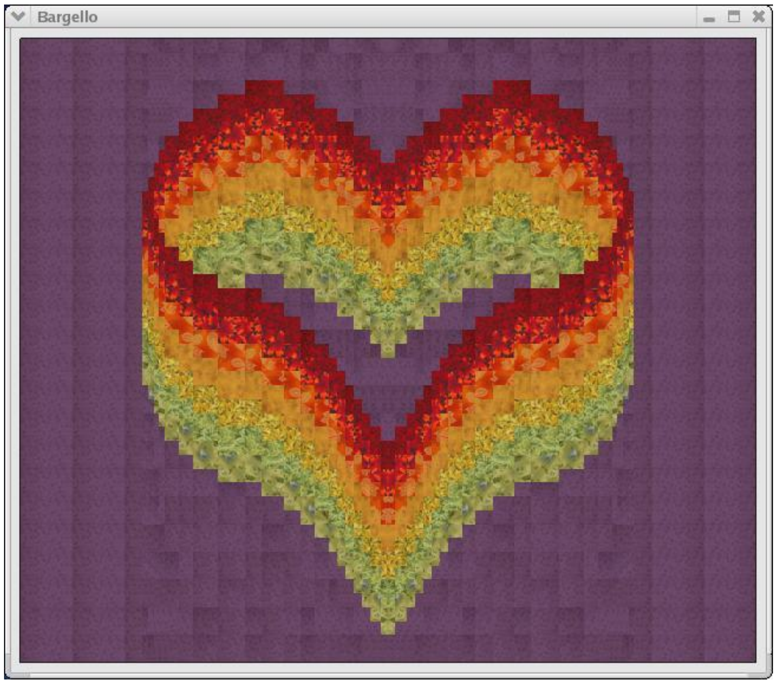


Figure 3.18: This design was drawn in a horizontally symmetric reflection template. Tiles in the top and bottom curve segments fit together in a mutual-compromise Bargello grid.

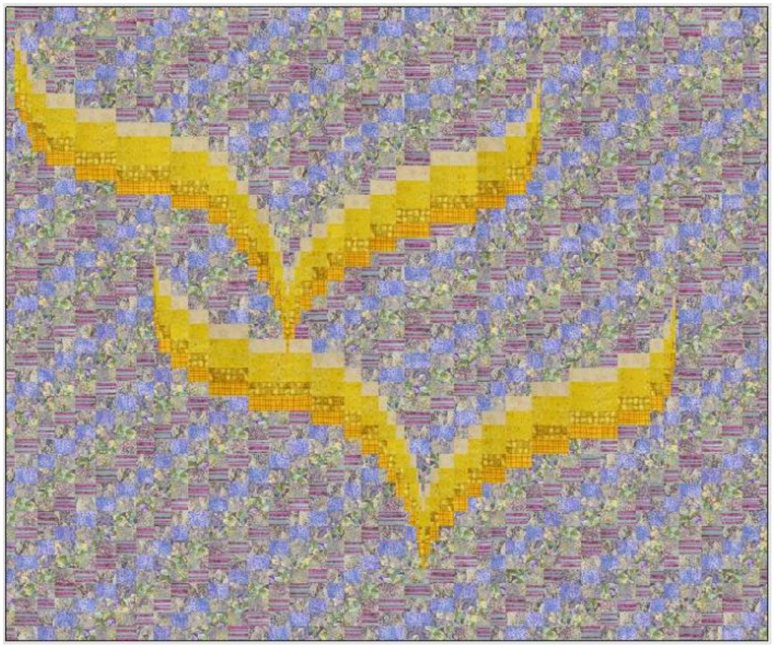


Figure 3.19: To make this design the background curve was drawn first, and a bird was drawn over it. The second bird was copied from the first and dragged into place.

Chapter 4

System Evaluation

When the system was nearly at its current development stage, we conducted a series of focus group sessions with quilters from the community. In each session, quilters saw a software demonstration, used the software themselves, and provided feedback on the system and on our ideas for future work. We adopted this evaluation approach because the system is a first-generation prototype [50]. Thus, we wanted to elicit open-ended discussions about the software among authentic potential users with various experience levels to better understand what system features various quilters found useful and what improvements could be made to better support the quilt design process. In this chapter, we present our methodology and findings from the focus group sessions.

4.1 Method

Modern-day *quilting guilds* are social organizations of amateur, but often highly experienced and accomplished, quilting enthusiasts. Guilds typically hold monthly meetings in which members organize activities, learn new techniques from invited speakers, and share their accomplishments with one another. Guilds may also host workshops and annual quilt shows. In addition, members may meet more frequently in smaller groups or sewing circles.

A web search was conducted to identify quilting guilds in Toronto and the surrounding area, and electronic mail messages were sent to ten guilds describing the project and inviting members to participate in an evaluation of the prototype software. Two of these messages were returned as undeliverable, and five received positive responses. Of these five, two did not culminate in focus group meetings: one guild failed to respond to a second communication, and the other responded with an invitation to speak for only fifteen minutes. However, focus group meetings were arranged with three guilds, and each meeting was attended by a small group of self-selected guild members. No attempt was made to tailor the demographics of the participants beyond stating that all interested members were invited and that prior Bargello experience was welcome, but not required. No payment was given for participation. The first two meetings were held on consecutive evenings, and the third was held on an afternoon two weeks later.

Each meeting began with a brief overview of the agenda, and participants were encouraged to ask questions and provide feedback, especially constructive criticism, throughout the meeting. A sign-in sheet was circulated, asking participants to describe their experience level with quilting in general, and Bargello in particular, as well as the frequency of their computer use (Table 4.1). Handouts were distributed with our contact information encouraging participants to send additional comments and questions after the meeting. Each meeting lasted between two and three hours, and all meetings were audio-recorded.

Following the administrative details, the focus group sessions began with a set of focus group questions and a software demonstration. Next, participants were given an opportunity to experiment with the software as they chose. No tasks were set for them, and no measurements were made; however, problems that occurred while participants used the software were recorded. Finally, to close the meeting, a second set of focus group questions was asked.

In every focus group, participants asked questions and made helpful suggestions throughout the demonstration and provided encouragement as their peers engaged with

Focus Group Demographics				
	Group 1	*Group 2*	*Group 3*	*Total*
Number of participants	9	7	5	21
Self-described experience level (quilting):				
beginner	0	0	1	1
intermediate	6	5	3	14
advanced, or advanced intermediate	2	0	1	3
expert	1	2	0	3
Number of Bargello quilts made:				
0	5	4	4	13
1	2	3	0	5
3 or more	2	0	1	3
Frequency of computer use:				
none	1	0	0	1
1-3 times per week	4	1	0	5
daily, or almost daily	4	6	5	15

Table 4.1: Focus group demographics

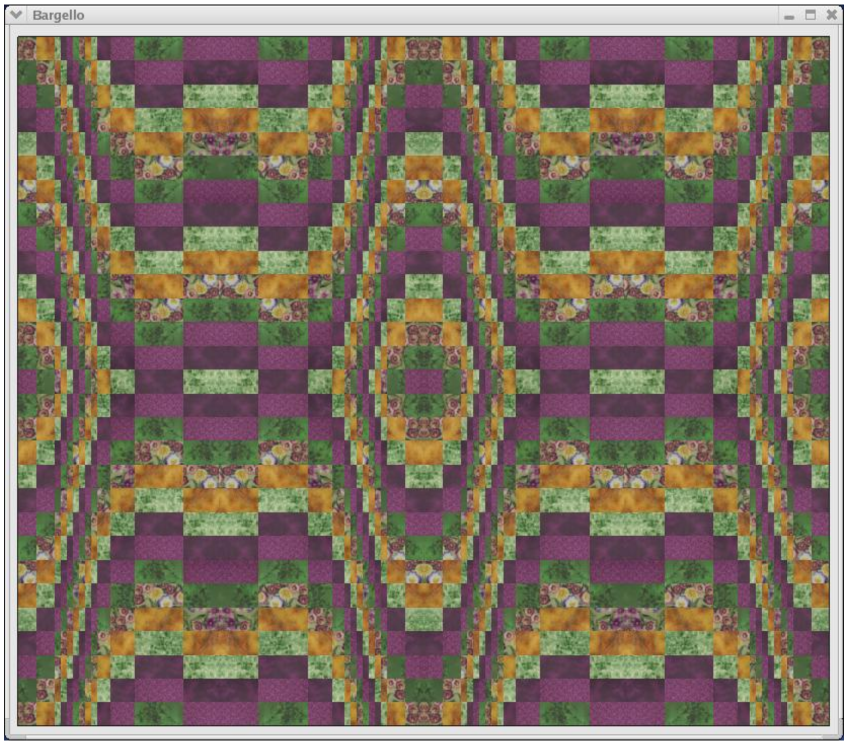


Figure 4.1: Focus group participant's Bargello design.

the software. While some participants were eager to try the software, others were more reticent: in one meeting every member participated in this aspect of the evaluation, but in another only one person was willing to do so. Yet participants were able to create designs almost immediately, and they were often pleased with their creations. Figure 4.1 shows a design created by an evaluation session participant.

Table 4.2 presents a list of questions used to guide the evaluation sessions; however, the discussions remained semi-structured and informal. In particular, not all questions were asked at every meeting. Typically, some questions became unnecessary when participants addressed them spontaneously, and other questions went unasked for lack of time. In addition, the questions evolved as the meetings progressed since some were found to elicit feedback more effectively than others and since we wanted to elicit feedback on a broad range of issues in a few time-limited meetings. However, all of the questions presented in Table 4.2 were addressed, either directly or indirectly, at least once over the course of the three meetings.

The software was not in its current form when the first two focus groups took place. Rather, in the two-week interval between the second and third meetings several improvements were made to the system in response to input we received during the first two sessions. Some of the improvements were recommended by participants, and others were made in response to problems that arose while participants engaged with the system. A list of the enhancements implemented during this period follows.

- By default, the tail end of a curve is now erased when the input device reverses direction horizontally. Previously, this action always resulted in a switchback being added to the curve.
- Multiple fabric palettes are now maintained concurrently.
- The Bargello design is now updated immediately as a result of updates made to the corresponding fabric palettes. (Previously, users needed to invoke an update operation in the design module to make this happen.)
- Users can now substitute new fabrics into specific tiles within a Bargello curve without modifying the default color strata of the entire curve.
- The semantics of the black outlines around individual color tiles in the main color wheel have been modified for clarity.

The first item in the list is particularly noteworthy. The system was designed to add a "switchback" to a curve in response to the input device reversing directions horizontally, so we were accustomed to this behavior. Thus, we were surprised when every user in the first two meetings drew curves with switchbacks and then became visibly disappointed with the results. Upon questioning, we learned that their unanimous intention had been to erase the latter portion of their curves. Clearly, curve erasure in this situation was deemed the intuitive program behavior. This notion was further corroborated in the

Focus Group Questions	
Topic	*Questions*
Current practice	Are there steps in your current design process that are difficult or time-consuming?
	Are there typical problems that result in disappointing outcomes?
	How could computers make the process easier or better?
	Do you currently use quilt design software? If so, what do you like and dislike about it?
Software demonstration and trial period	How is this process different from your current process? How is it similar?
	What did you like about the program? What did you dislike?
	Is there anything confusing or difficult about the program?
	What is missing that might be helpful?
	Do you have any other suggestions for its improvement?
Can you imagine wanting to. . .	edit the curves in a design?
	print instructions for constructing the quilt?
	start a design in a reflection template and then “turn off” the symmetry requirement for further changes?
	use a tool to explore the order fabrics should go in?
	select fabrics based on texture as well as color?
	import an image and select fabrics based on the colors in it?
	import an image and automatically fit Bargello curves to it?
	create hypothetical fabrics to work with?
	see the design rendered as realistically as possible ?

Table 4.2: Focus group questions

third meeting when, after the enhancement had been made, a quilter successfully erased a portion of her curve with this mechanism – without having been informed of the feature.

4.2 Findings

A summary of the findings compiled from the focus group meetings follows. Participants' comments are organized into the following categories: challenging aspects of quilt design, kudos, recommendations, and responses to proposed future work ideas. In addition, we describe problems that arose while users were operating the system.

4.2.1 Challenging aspects of quilt design

After an introduction, we began each focus group by asking participants what aspects of quilt design are challenging or time-consuming, and whether there are particular difficulties that frequently result in disappointing outcomes.

Resoundingly, quilters answered that fabric selection is the most challenging aspect of quilt design, and that this difficulty takes on many forms. The appearance of a fabric can change depending on lighting conditions and on what other fabrics it is placed next to. It can be difficult to visualize how a fabric will look from a distance, especially after being cut into small pieces. It can be difficult to look at a fabric and determine precisely what colors are in it. A set of fabrics that does not include sufficient value contrast can result in design curves that are difficult to see, and a set of fabrics that does not include sufficient texture contrast can look bland.

In addition, a few quilters mentioned other issues that can be challenging: modifying the size of an existing pattern, achieving the desired "flow" in a Bargello curve, and visualizing an imagined design for the quilting stitches.

4.2.2 Kudos

In general, participants were impressed with the software, and they made many complimentary comments regarding both program modules and the system as a whole.

For example, participants were impressed with how quickly and easily Bargello curves can be designed. When the first curve was drawn in each of the software demonstrations, participants typically responded with exclamations of surprise such as, “I want your program! Wow!” and, “What a feeling of power! Think how long that would take to design on graph paper.”

In two of the focus groups, participants especially enjoyed working in the fabric selection module. They liked the ability to select a set of fabrics and immediately view a design rendered in those fabrics. One participant said, “That would solve lots of quilters’ problems. Lots of times it happens that you’ve selected five fabrics, but later find that they don’t have the contrast to show the line that you want. So you just have to set those fabrics aside and go back to the drawing board.” Participants also found themselves trying unusual color combinations just for fun. They enjoyed the freedom to experiment quickly and easily, without having to purchase fabric. One quilter commented, “You know what I like about this is I can just try things that are totally not me, and I can just play! What I like about it is that you can try things you wouldn’t normally.”

Participants particularly liked the amount of color information that is provided by the system. One quilter commented about the color composition wheel, which displays the colors found in a given fabric, “I like the color wheel on the right. I find that really useful. Color always fascinates quilters, so the more information you can give us the better.” Another said, “What I like about your color wheel is that there are so many variations within each color.” A third added, “Because color does tend to intimidate a new quilter, and challenges all of us.”

Finally, participants told us that they found the program fun to use and entertaining in its own right. One quilter commented, “It also has an entertainment side to it. It’s not like playing a game, but you can play with it and feel like you are being frivolous for two hours. But you are still learning something.” Another went so far as to intimate that the system could supplant television: “I think the problem will be that people will buy the software and play with it, and never make the quilt! It’s relaxing. ‘Here’s my wall hanging for today.’ Forget television.”

4.2.3 Recommendations

Participants also suggested many ways the software could be improved. While some of their ideas were new to us, others provided healthy corroboration for work that was already planned and possibilities that were under consideration. Still others addressed practical issues that are outside the scope of prototype research software. A summary of the recommendations follows.

- *Curve editing.* Participants in every focus group requested the ability to edit curves, and the desired editing tasks fell into three categories. First, some participants suggested they would like to drag a particular point on the curve to a new location while the software automatically ensured the curve would retain a graceful appearance. Second, many participants requested the ability to manipulate individual strips in the design. In particular, they wanted to specify a particular width for a vertical strip, to specify a particular height for a color strip, and to translate particular vertical strips vertically without maintaining the connectivity of the curve. Finally, one participant requested the ability to modify curve tangents at the extrema, changing “peaks” into “waves”, and the ability to perform other beautification operations (e.g., specify that a curve segment should be straight or that adjacent segments should be symmetrical).

- *Support for additional design features.* Another request that arose in every focus group was the ability to create a particular advanced Bargello design style. In this style, two design curves co-exist on alternating vertical strips. These curves may be simple reflections of one another, or they may be independent. To allow viewers to perceive the two curves separately, one of the curves typically occupies a set of vertical strips that have a uniform narrow width, while the other curve occupies vertical strips whose widths vary as usual. Beyond this, quilters suggested the ability to add borders, to add applique figures, and to add quilting stitches to the design.

- *Issues of scale.* Participants also pointed out the need to create designs at different scales. Not surprisingly, they wanted the ability to specify the final size of the quilt and to specify the frequency of the design by specifying the height of each row of virtual fabric tiles. Participants also requested the ability to stretch or condense existing designs and to reproduce designs in various standard sizes.

- *Drawing with solid colors.* This request was less common, but two participants recommended the ability to draw designs in solid colors as well as fabric textures. This suggestion arose for two different reasons. One quilter felt that working with a design in gray scale would enable her to focus on its value contrast. Another quilter wanted to draw with a solid color in the event that the database of fabric images did not include a fabric in the desired color.

- *Zooming in on fabric swatches.* Many participants requested the ability to "zoom in on" individual fabric swatches displayed in the fabric browser. This need arises when a large number of fabrics are displayed in the browser concurrently since the swatches are automatically scaled down until they all fit within the window. Therefore, as more and more fabrics are displayed, it can become difficult to get an accurate impression of an individual fabric's appearance. Displaying a "zoomed" (i.e., correctly scaled) version elsewhere on the screen would alleviate the problem.

- *Color wheel.* Although participants liked the color wheel used in the system, they also offered suggestions for improvement. One quilter requested the ability to minimize the color wheel once a color scheme had been selected for the design, thereby removing the color distraction and also freeing screen space. Another quilter suggested the system should provide support for the various color combinations suggested by color theorists (e.g., analogous colors and complementary colors). Yet another quilter proposed presenting multiple color wheels, such that separate wheels could be dedicated to pure colors, tints, tones, and shades.

- *Browsing "focus fabrics" separately.* Participants in one group requested the ability to browse focus fabrics separately from other fabrics. They defined good candidates for focus fabrics as those that contain colors in four or more hues. However, it is likely that various quilters will have various definitions of what makes a good focus fabric.

- *Unified display.* In one group, participants requested that the fabric selection module and the design space be made visible simultaneously. They thought this would be less cumbersome than flipping back and forth between two windows, and that it would be helpful to see simultaneous updates to the design in response to changes in fabric and fabric orderings made in the fabric selection module.

- *Practicalities.* Not surprisingly, there were also several practical issues that arose in every focus group. These requests included the ability to import fabric images easily and the ability save designs and re-open them later to continue work. One quilter requested the ability to view several intermediate versions of a design simultaneously. Another unanimous request was for the ability to compute yardage requirements and to print detailed instructions for quilt construction. These features are necessary for all designs, but the need is even more pronounced for complex designs that require embellishments during the construction process.

- *User-friendliness.* Participants also pointed out the need for a more user-friendly interface. At present, many of the design operations and preference settings are accessed via keystrokes. These features should ultimately be presented in a more user-friendly fashion, probably through a WIMP-style interface since most users are familiar with that interface style. Finally, it was pointed out that a production application would need a help system, including a tutorial, to assist users in learning the software.

4.2.4 Problems and concerns

Inevitably, problems arose at each of the focus groups when participants engaged with the prototype software. As previously mentioned, in the two-week interval between the second and third meetings, several minor program modifications were made, and some of these were in response to problems that arose in the first two meetings. Those problems will not be addressed here. Instead, the issues described here are ones whose solutions will be less straightforward.

- *Users were sometimes confused by the program's response to curves they drew.* Due to the design constraints intrinsic to Bargello, and also the symmetry constraints specified for a given design, it is possible for the input device to trace a path that is ambiguous or cannot be represented with a simple Bargello curve. At present, these constraints are not made visible to the user, and as a result, we found that users were sometimes confused by the program's response to their input. This issue arose in two distinct places.

  First, the resolution of the Bargello grid determines the steepest curve that can be followed by a connected tile path. At present, when a user draws a curve that is too steep to follow, the program produces the steepest tile path allowed: a linear segment of minimum-width tiles. In general, if this segment is short, it can still approximate the sketch well, but if the segment is long, it does not. The endpoint of a linear tile path segment can get pushed arbitrarily far from the endpoint of the corresponding sketched segment as the offending segment becomes arbitrarily long. A more appropriate response might be to present users with multiple alternative Bargello curves to choose between. These could include one curve that does not meet the extrema of the input sketch and another in which continuity of the tile path is not maintained. These results may also surprise users, yet they may offer a visual explanation of the underlying source of the difficulty.

Second, when a design is made within a reflection template, a curve drawn in any one of the reflection regions is sufficient to define the entire design, and a curve which passes through multiple regions can be ambiguous. At present, once the user starts to draw in one region, the program ignores input received from another region during the same input stroke and waits for the stylus to return to the original drawing region. Although the region boundaries are visible, some users were confused by the program's lack of response when they drew out-of-region.

In both of these cases, providing additional guidance during the curve drawing process might be helpful. For example, dashed guidelines could be overlaid temporarily on the design to indicate the range of curves that can be followed by a connected tile path within the current grid. Similarly, dashed barrier lines could be used to indicate that the mouse will not be allowed to pass from one reflection region to another since doing so produces conflicting input under the current symmetry constraints.

- *Users sometimes had difficulty finding fabrics.* In the first two focus groups, participants spent a considerable amount of time in the fabric selection module, and in each group one participant became frustrated because she found it difficult to locate fabrics. In each case, the user selected a large number of color tiles in the main color wheel, but doing so did not result in additional fabrics being added to the fabric browser. This can occur when there are no fabrics associated with a given color tile or when the associated fabrics are already on display. One contributing factor was that the fabric image database contains approximately 450 fabrics, while fabric shops typically stock around 2000 fabrics. Therefore, the system simply does not have the range of fabrics that quilters are accustomed to. However, the ultimate question is: how can we help users locate the images that are available? We will propose some potential solutions for this problem in Chapter 5.

- *Linux locked up.* In one focus group meeting, a user was browsing fabrics by dragging the mouse within the main color wheel. This causes each of the visited color tiles to be selected and then de-selected in fairly quick succession. In response, several fabric images may be quickly inserted and then removed from the fabric browser. During this process, Linux locked up and the system became completely unresponsive. Fortunately, this only happened to one participant; however, the problem has also occurred during development, and it has not yet been resolved.

- *Color management.* Finally, participants in every focus group were concerned with the issue of color management. They were aware that fabric images displayed on a computer screen may look somewhat different from the corresponding physical fabrics. This raises a valid concern about how accurate the appearance of a quilt design displayed by this system can be. However, in each focus group, participants also spontaneously suggested that this imperfection could be overlooked. They felt that the system should give an approximate sense of the way fabrics appear individually and in combination, but that since the fabric images may be used as proxies for other physical fabrics anyway, reproducing their appearance exactly is not necessary.

  In addition, we may be able to take advantage of sophisticated color management tools to try to adjust the appearance of the screen image to that of the corresponding hardcopy image. Given that hardcopy images and physical fabrics are both viewed via reflective illumination, while computer screens are viewed via emissive illumination, a hardcopy image may match the appearance of a physical fabric more closely than does the corresponding unadjusted screen image.

4.2.5 Responses to proposed future work

We are also considering how this project might be extended in the future. It is fun to imagine tools that might be useful for quilt design, and we had several ideas in mind at the time of the focus group sessions. Perhaps not surprisingly, participants had mixed opinions about many of them. This may reflect the fact that different quilters envisioned using the system in different ways, and it may also reflect the fact that it is difficult to know in advance which as yet imaginary tools might prove useful in the future.

- *Generate and explore fabric orderings with computer assistance.* Fabric order is important in Bargello designs since the specific placement of fabrics within a color strata can make a remarkable difference in the overall impression made by a design. Participants had mixed opinions about computerized assistance for this task. Some liked the idea of having the computer generate many orderings for them to choose between, especially if the computer was able to learn from their previous choices and especially if they would not be obligated to look at inordinately many orderings. Other quilters felt that such a tool was not necessary and that they would prefer to explore fabric orderings on their own.

- *Categorize fabrics based on texture as well as color.* Participants also expressed mixed opinions about this idea. Some thought it would be helpful since quilters try to include a variety of textures in each quilt to give it a rich visual texture. Others felt they would prefer to browse all fabric textures together and noted that this is how fabric selection is done in fabric shops. One quilter expressed concern that separating fabrics into narrow categories might remove the serendipity that can occur when unexpected combinations are presented together.

In addition, participants who liked the idea of categorizing fabric textures had various ideas regarding the categories to use. Some quilters suggested a classification scheme based solely on texture size, while others suggested perceptual categories such as florals, geometric patterns, basketweaves, and so forth. However, given that computer algorithms are not conversant with humans' perceptual categories, a classification scheme based on image statistics, such as texture size, would be much more practical to implement.

- *Create hypothetical fabrics to draw with.* This idea drew an adamant yes from one quilter since it would allow her to create designs just as she imagined them, whether a desired fabric existed in the database or not. Other participants did not find it as important since they envisioned using the system to get a basic sense of the design, rather than a detailed one. These quilters felt they would be content to use proxy fabric images that existed in the database already.

- *Render designs photo-realistically.* Participants in one group felt that this could be useful for visualizing the effect of quilting stitches on the design, but they were less concerned with the need to accurately render seam lines and other depth cues, thinking such a tool might be used as a final step only. This question was not asked of the other two focus groups, but given that some participants expected to use proxy fabrics in their designs anyway, it is likely that they would not feel the need for a photo-realistic display. It is even possible that photo-realism could detract from the ability of such users to consider their designs in an abstract, inexact manner.

- *Import an arbitrary photograph and provide information on its color composition.*

We suggested that this feature could be used to create a quilt with colors reminiscent of a particular scene, or one that matched the color scheme in a given room. Although quilters were not immediately taken with the first idea, as soon as we suggested importing a photograph of the wallpaper the quilt would hang against, quilters gave the idea a resounding yes. This example serves to remind us that while users may not immediately see the value of a new tool, they may become enthusiastic once they recognize a concrete use for it.

- *Import an image and fit Bargello curves to it.* Unfortunately, this question was asked in only one focus group, and the participants in that group were not taken with it. However, given that quilters sometimes base quilt patterns on hearts, butterflies, and other objects, we still see merit in the idea. One difficulty that could arise, however, is that Bargello curves are inherently low-resolution. Therefore, it would be important to advise users of this tool against trying to fit Bargello curves to detailed geometry.

- *Start a design in a reflection template, and then "turn off" the symmetry requirement.* Once again, we regret that this question was asked at only one focus group. Members of that group were not particularly taken with the idea, saying only that "someone might" be interested in this option; however, we continue to see potential for the idea given that it would allow users to create designs that are nearly, but not completely, symmetrical.

4.3 Summary

Quilters resoundingly told us that fabric selection is the most challenging aspect of quilt design and that it is common for a finished quilt to look different than the quilter had imagined. For the most part, this occurs because fabrics look different in combination than they do individually and because a completed quilt takes on an holistic appearance that can be either more than, or less than, the sum of its parts.

Quilters were impressed with the prototype system and frequently asked whether it would be made available for public use. In particular, participants were dazzled by how quickly and easily Bargello curves can be designed, and they were fascinated by the color information provided during the fabric selection process. In addition, they thought the ability to experiment with fabric combinations and to view designs rendered in the selected fabrics would be helpful to quilt designers. Finally, quilters told us they found the system entertaining and fun to use.

At the same time, participants offered many suggestions for improvements to the system. They unanimously requested support for editing curves and for several practical operations such as importing fabric images, saving designs, computing yardage requirements, and printing instructions for quilt construction. In addition, we received many suggestions related to the fabric selection module, but quilters expressed a much wider range of opinions here. For example, some quilters suggested ways to augment the color wheels, but others liked them as they are. Similarly, some quilters wanted to view focus fabrics separately, but others liked viewing all fabrics together. Clearly, quilters were fascinated with the fabric selection process and wanted to explore ways that it could be enhanced; however, it is also clear there are no easy answers to this most challenging aspect of quilt design.

Inevitably, problems occurred while users were operating the system. Foremost among there were situations where users were confused by the program's behavior. For one, users were frequently surprised when they tried to erase a portion of a curve but ended up drawing a switchback instead. Because this happened consistently, we modified the default behavior in this situation. In addition, users were sometimes confused by the program's response to ambiguous input, and users sometimes had difficulty locating fabric images with the current fabric selection interface.

We found that quilters frequently had mixed opinions about the tools we proposed for future work. While some users expressed enthusiasm, others did not. Indeed, one quilter expressed concern about the inclusion of too many computerized tools, asking, “But if you have the computer do everything for you, where goes the fun?” This suggests that tools intended for the enhancement of the quilt design experience should be made optional. That way they are available to those who want to use them, and they do not intrude on the experience of those who wish to work more independently. Similarly, we found that users may develop enthusiasm for tools that they did not see value in initially. This suggests that user feedback based on a prototype demonstration may be more reliable than feedback based on a verbal description, and it allows us to retain some optimism about tools we envision but have not yet presented effectively.

Chapter 5

Conclusion

5.1 Accomplishments

We have presented a prototype computer-assisted design system for Bargello quilts. In this system, users can create traditional designs simply by sketching the desired curve with a mouse or stylus. Designs with specific symmetry patterns can also be created easily by sketching curves in one of a set of pre-defined reflection templates. More complex Bargello designs can be created by drawing multiple independent curves in the same design or by using a set of low-level design operations such as copying existing curves, dragging curves to new locations, modifying the apparent depth of curves or sets of tiles within a curve, and substituting new fabrics into sets of tiles within a curve. In addition, horizontal stripes can be added to fill background regions in a design, and designs can be created with either "matched-" or "staggered-seams."

At the core of the Bargello design system is an algorithm that transforms an input sketch drawn with a mouse or stylus into a graceful curve composed of corner-connected axis-aligned rectangles. The algorithm begins by generating a piecewise cubic Hermite curve from the input data, in which each Hermite segment is monotonic in both x and y. This ensures that the Hermite curve has the same frequency as the resulting Bargello curve. Next, the Hermite curve is "scan-converted" against a uniform grid of rectangular cells, and the geometry of the resulting rasterization is stored as a tile path. Finally, the initial tile path is "beautified" to produce a graceful Bargello curve that conforms to physical constraints which arise when working with fabric and to design constraints that arise from the Bargello construction process. When a design includes multiple independent curves or a curve that is not single-valued in x, the beautification algorithm adjusts each single-valued segment to fit with the others such that together they form a single set of shared vertical columns.

The system also includes a module that provides access to a database of fabric images and allows users to select fabric textures for use in their designs. The module contains a variety of widgets for this purpose. The main color wheel allows users to select colors, the fabric browser displays fabric textures that include those colors, the color composition wheel provides information about the colors contained in a given fabric texture, and three fabric palettes store and display fabrics that have been selected for use in a design. When Bargello curves are generated, they are automatically associated with the "current" fabric palette in the fabric selection module. Subsequently, changes to the fabrics or the order of fabrics in the fabric palette result in corresponding changes in the associated Bargello curves. Thus, users can explore fabric and color combinations, experiment with fabric orderings, and visualize their designs rendered with real fabric textures before committing to the expensive and time-consuming process of constructing a physical quilt.

Finally, we conducted a series of focus group sessions with quilters from the community to help us evaluate the system. In each session, participants watched a demonstration of the software, used the system themselves, and provided feedback on the system and on our ideas for future work. Participants in these sessions were quite enthusiastic about the system, and they also gave us many suggestions for its improvement. Some of their suggestions have already been implemented, and others remain as potential future enhancements.

5.2 Limitations

In general, the Bargello curve generation algorithm works well: it does indeed produce Bargello curves that conform to the constraints of the domain, follow the input sketch, and have a graceful appearance. However, there are situations in which the algorithm does not perform as well as we would like. For example, as mentioned in Section 4.2.4, when a segment of a sketched curve is too steep to follow with a corner-connected tile path, the resulting Bargello curve includes a linear segment of minimum width tiles. One result of this is that it is difficult to create attractive designs that depict 3D figures such as hearts or ribbons since Bargello curves that are not single-valued in x frequently contain such segments, and as a result, they tend to "bulge out" horizontally at x-extrema.

Similarly, the sketch-based approach to Bargello design has, in general, been quite successful. We have found that users enjoy drawing curves with the mouse and watching a complete Bargello quilt design emerge. However, we were not as successful as we would have liked at creating an environment and a tool set for designing complex contemporary patterns. We have not managed to create designs with the degree of intricacy seen in Figure 1.9, and this is somewhat disappointing given that these designs were the initial inspiration for the project. In Section 5.4 we discuss the issue further and consider possible approaches to solving the problem.

Our software design process could also have been improved. Given that our interest is in producing a system that fills an authentic need for real quilters, we probably would have done well to involve users more systematically throughout the design and development process. In fact, we did visit a regional quilt show and speak with vendors, who are typically experienced and enthusiastic quilters, to seek potential users to confer with during development. We initially thought we had found three interested participants from this exercise, but only one was ultimately able to meet with us to discuss the project. We met with this user three times over the course of development. However, Edie's book [19] was the main inspiration for the project and the main source from which we drew our system requirements. Our meetings with this participant were intended to elicit feedback on our chosen direction, rather than to engage her fully in the software design process. Nevertheless, her enthusiasm for the system and her eagerness for particular functionality, such as the ability to create designs with either matched- or staggered-seams, were helpful in guiding our progress.

Moreover, our three focus group sessions were one-time events in which individual users had only a short amount of time to engage with the system. Had we given a select set of enthusiastic users intermediate versions of the software to experiment with over the course of the development process, these users may have found time to explore the system thoroughly, new requirements may have emerged, and the system may have been strengthed as a result. Of course, we can not know to what extent the software would have evolved differently had this been done, but we surely would have learned a lot about the users' needs and the users' experience with our system.

5.3 Lessons learned

We found the focus group sessions fun and inspirational. Further, we learned a number of lessons from the sessions, some of which were surprising to us. For example, we noted that most participants who engaged with the prototype software spent more time working in the fabric selection module than in the Bargello design module. At first, we found this result surprising given that fabric selection was not the intended focus of our project. However, upon further reflection it makes perfect sense. Many of our focus group participants were experienced quilters, but not experienced Bargello quilters. Thus, it is not surprising that they responded more strongly to problems they were familiar with than problems they were not. Moreover, this result reflects the fact that color and fabric selection is a more general problem than Bargello quilt design: not only is it applicable to more quilting styles, but it is also more difficult to solve. As such, we predict that tools which support quilters in the fabric selection process and allow them to visualize designs in real fabric textures will be beneficial to and well received by quilters.

Another lesson that was impressed on us was that different participants wanted different levels and different kinds of software support. Some suggested we provide "kits" (i.e., ready-made designs), while others took a fairly rule-based approach to the design process and suggested we provide tools to support these rules, and still others reveled in pushing the boundaries and "breaking the rules" of a given quilt style. Given that our finest goal is to support the creative spirit and practice of individuals, we would like to support the artistry of all these quilters. Yet the type of support requested by some users may at times conflict with the needs of others. Therefore, we suggest that software intended to support quilt design (and probably any artistic endeavor) should be as flexible as possible, allowing tasks to be accomplished in multiple ways and providing tools that are optional.

5.4 Open questions

A number of issues were raised and areas for further study were identified in Chapter 4 and Section 5.2. In many cases, the previous discussion was sufficient for the issue at hand. However, we extend the discussion of a few of the larger issues in this section, devoting our attention to potential solutions for these problems.

Curve editing

The system currently does not provide support for editing existing Bargello curves. To refine the shape of a curve, users must clear the design and re-draw it. Given that sketching a new curve is quite easy in this system, we have found this method workable in the short term: a curve can be drawn repeatedly and improved over time. However, the ability to modify existing curves would be a marked improvement.

A curve-editing interface that provides widgets to modify the position of and tangents at extrema seems promising, given that it would leverage users' current conceptualization of Bargello curves as a series of "peaks" and "waves." In addition, such an approach would match the internal representation of Bargello curves well. Recall that the algorithms for Bargello curve beautification require that each internal curve segment be monotonic in both x and y. Therefore, the Bargello curve data structure maintains knowledge of the positions and classifications of each of the local extrema and inflection points.

However, several questions must be addressed, such as what extent of the curve should be modified when a local extremum is dragged, and what behavior is expected if a local extremum is dragged to a position where it no longer qualifies as an extremum? For example, suppose point B in Figure 5.1 is dragged to a position midway between points A and C. One reasonable response would be for both points B and C to be reclassified as inflection points of the curve, such that segment AB remains convex, while segment BC becomes concave. This scenario results in three inflection points between the extrema at points A and E, and raises the possibility that users may want to remove pairs of inflection points to smooth a curve. Further, the system may need to insert or remove individual inflection points when users modify the tangents at local extrema.

Finally, recall that focus group participants requested the ability to manipulate individual column widths as well as the ability to drag curves. There is an intrinsic tension between the two requests, given that when extrema of the curve are dragged we would like the system to re-fit the Bargello tile path to retain its gracefulness, yet when users modify tile widths manually they presumably would prefer not have them altered in subsequent beautification steps. Therefore, if we are to support both types of curve editing, this ambiguity of intent will need to be resolved.

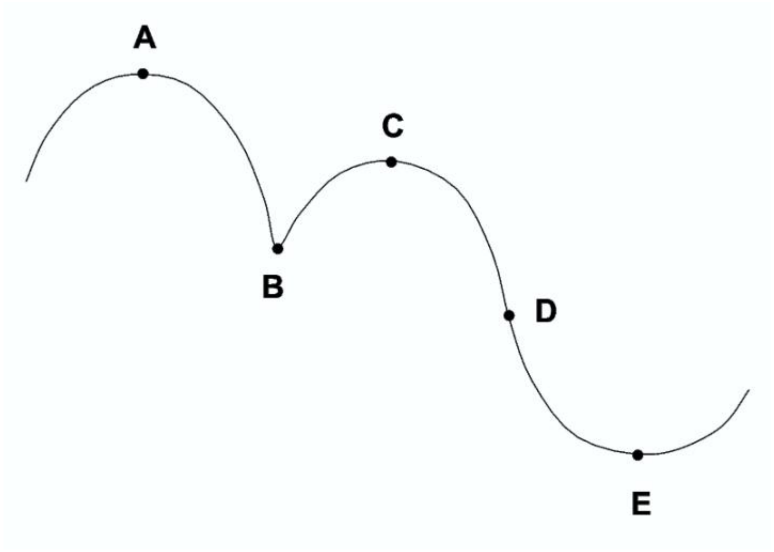


Figure 5.1: Curve editing

Contemporary Bargello designs

Although it is possible to create non-traditional Bargello designs in our system (e.g., Figures 3.13 through 3.19), it is not as easy to do so as we would like, and the resulting designs tend to have a "blocky" appearance. The system provides several low-level operations that allow users to add complexity to their designs, but these operations fall short of providing the intuitive interface we had hoped for. At this point, it is an open question what additional operations or enhancements would improve the situation, but some speculative suggestions follow.

Providing support for editing existing curves may go a long way toward providing an intuitive interface for creating complex designs. In designs such as the one in Figure 1.9, it is the intricate interconnections between various curves that give the design its appeal. For designers to ensure that curves interact just so, they must have explicit control over the placement of each curve segment as well as the number of tiles in each segment. For example, a user may want to ensure that instances of the same fabric within each of two opposing curves meet at a particular point in the design. Allowing users to drag extrema and place them precisely with respect to other curves should serve this purpose.

However, allowing users to edit curves will raise other issues. It may prove useful to allow users to specify explicit relationships to be maintained between curves. For example, if one curve is created as a reflection of another, a designer may wish to assert that the two should retain their symmetry when either one is modified. Alternatively, she may want to break the symmetry, allowing one to be altered without a corresponding alteration in the other. Similarly, a designer may wish to specify that one curve segment should remain parallel with, or maintain contact with, another. The number of constraints that might prove useful in some designs is probably unlimited; the trick will be to specify a small subset of the most salient among them to support.

Allowing users to select arbitrary tile sets in a single operation may also improve system usability when creating complex designs. Currently, users may select only "rectangular" tile sets (i.e., contiguous sets of tiles that extend from row a through row b in the color strata and from column c through column d in the design space). This strategy is sufficient since users can operate on arbitrary tile sets in multiple passes, but it can be cumbersome. However, it is an open question what interface mechanism would be appropriate for selecting arbitrary tile sets within a Bargello curve. A lasso tool is one possibility, but it may be difficult to lasso a specific set of tiles without including unwanted ones, given that individual tile paths within a Bargello curve are corner-connected. Similarly, users could click on individual tiles for inclusion or exclusion, but this approach would become tedious quickly. Perhaps a combination of these two approaches would work well.

Ultimately, more investigation is needed to determine the set of operations and interaction features that could provide an intuitive way to create complex contemporary designs. Collaboration with experienced Bargello quilt designers will be essential to the solution of this problem.

Quilt construction instructions

The current implementation of our system is useful for creating Bargello designs, but it is not yet useful for creating Bargello quilts. For the software to become useful in a production setting, it would need the additional ability to produce instructions for quilters to follow when constructing a physical quilt from a given design. For simple designs that consist of a single set of parallel tile paths, the instructions could simply reiterate the standard Bargello construction process, as described in Section 1.2, and provide the widths of each vertical strip in the design. However, complex designs are constructed by "embellishing" the standard construction process, which involves removing fabric pieces from vertical strips and replacing them with other pieces or with the same pieces in another order. Therefore, as designs become more complex, quilt construction becomes more complex as well. Given that one of our system goals is to allow users to create complex designs, the system should also produce step-by-step instructions that lead quilters through the construction of such designs. However, producing instructions for arbitrary Bargello designs may be challenging, especially given that we would like the resulting construction process to be efficient in both fabric usage and work flow.

We have not yet begun to explore solutions for this problem, but we present one nascent idea for future consideration. Although the geometry of each tile path in a design is known, if a design includes multiple tile paths, they may occlude one another in complicated ways. Thus, this problem may best be approached from the data stored in the bitmap rather than the geometry of individual tile paths. However, we know (at least in the current implementation) that every Bargello curve present in the design is associated with one of the fabric palettes in the fabric selection module. Therefore, we may begin by examining each column in the design for instances of the fabric sequences represented in each fabric palette as well as the reverse orderings of those sequences. Any tiles in the column that remain unaccounted for represent "embellishments," yet even they must be associated with some fabric palette. Therefore, a next step could be to search for any remaining tile sequences in the design as subsequences of each fabric palette. Eventually, we should be able to represent the sequence of fabrics present in each design column with a shorthand notation based on palette numbers and fabric sequences within each palette (e.g., palette1:[1-6], palette1:[1-6], palette2:[2-3], palette1:[1-6]).

This information may be sufficient to create a basic set of construction instructions. For example, in the example given, quilters could be instructed to produce three color strata from the fabric sequence in palette 1 and another from the sequence in palette 2, then stitch them together in the order given. However, additional efficiencies may be possible by extending the algorithm to then search the shorthand notations for similar sequences across adjacent design columns. Doing so would provide information about the length of the Bargello curve that could be produced from the same color strata, and this information could be used to plan for efficient material usage.

Fabric selection

Our fabric selection module addresses a very open-ended problem: provide a tool set that draws upon knowledge of the colors in each fabric to support the task of fabric selection. As it stands, the module provides the basic support necessary to find color matches between various fabrics and to select fabrics for use in quilt designs. However, it is difficult to assess how successful the module currently is in comparison to how successful it could become, given that the goal is to support users engaged in an ambiguous task. Ultimately, we would like to provide a tool set that conforms to the mental model quilters use for color and fabric selection; however, it is non-trivial to codify the mental models used by others, and it may be especially difficult in this arena given that quilters tend to speak of color in anthropomorphic, almost magical, terms. Needless to say, the module is still a work-in-progress, and it provides ample opportunity for further exploration.

Indeed, during the focus group sessions a number of difficulties with the module were identified. For example, users sometimes had difficulty locating fabrics in the system, yet at other times the system displayed so many fabrics that they became too small to see well. Further, the fabric browser currently displays fabrics that contain the specific colors (hue, saturation, and value) selected for the color scheme, yet quilters often include fabrics containing various colors of the same hue in their quilts.

It might be preferable for the system to present a wider range of fabrics in response to a specified color scheme. However, during development we tried presenting all fabrics that contain the selected hue, rather than a specific color within that hue, and we found that the resulting montage of fabrics was often cluttered and confusing. The trouble is that each fabric contains many colors, so the more fabrics that are included in the browser, the more disparate the set of included colors becomes. Our impression at the time was that the wider array of fabrics was more distracting than helpful, and our response was to tighten the inclusion criteria. The result has been mostly favorable in that the displayed fabrics now appear to be selected correctly. However, a more sophisticated fabric browser might provide a better alternative.

For example, a browser with an interface based on zooming images and image groupings could be used to display all fabrics at all times, unless instructed otherwise by the user. Such an approach would eliminate the difficulty users had in locating fabrics to display, and would also allow users to view fabric swatches clearly via zooming in on them. *PhotoMesa* [7] is a “zoomable” image browser that presents an entire directory tree of images simultaneously, yet provides easy access to each image. In this browser, images are organized in groups, which may initially be quite condensed, and the user zooms in on image groups and individual images with a few mouse clicks. Although PhotoMesa is freely available, it is not directly applicable to our problem; we can not require users to pre-sort fabric images, and we would not want to replicate image files in a directory structure in order to present them in multiple groups. Nevertheless, the concept of a zoomable image browser seems quite promising for our system. Images could be grouped by color, and users could be allowed to zoom in on a hue, a saturation level within a hue, a value within a saturation level, or a specific image.

While it may be useful to provide access to all fabric images simultaneously, it may also be useful to present subsets of fabrics, reducing the number of images users must consider at any given time. For example, the members of one focus group requested the ability to view candidates for focus fabrics independently of other fabrics. They also requested the ability to view fabrics that "support" a selected focus fabric (i.e, fabrics that contain the same colors or variants of the colors in the focus fabric). Both of these ideas could be supported in a browser that is otherwise prepared to display all fabrics: populace image categories could be expanded to fill space not needed by sparce categories.

Another avenue that could be explored is the use of more sophisticated methods for color quantization and the determination of color salience. After all, any tool developed for the module will rely on two fundamental questions: what colors are present or suggested in a given fabric, and which of them are salient for the purposes of quilt design? The simple approaches used to answer these questions in the current implementation leave room for improvement. For example, the images used in our system contain a significant amount of chromatic noise given that they contain 256 colors, while the corresponding physical fabrics contain many fewer colors. Perhaps a sophisticated color quantization method could recover the original colors used to print each fabric. More accurate color information might be helpful for identifying salient colors in each fabric: at present there are often so many colors in a given fabric image that the peaks in the color histogram are low and not well-defined.

Perhaps more importantly, more sophisticated heuristics for determining color salience may be possible. We currently use simple thresholds based on the percentage of the fabric image covered by the color in question, so it seems likely that a better approach could be developed. For example, we might explore heuristics that consider attributes of the colors themselves. It may be that saturated colors are more salient than muted colors in a given fabric image. Alternatively, a more complex approach might involve color segmentation of each image. Perhaps colors that occur in large connected regions are more salient than colors that are spread thinly throughout the texture, or perhaps colors in foreground regions are more salient than colors in background regions.

There are many possibilities left to explore regarding this challenging problem. It is not yet clear what tools and what information will ultimately prove most useful to quilters, but we anticipate that collaboration with quilters could help us untangle these issues.

Sketch-based interaction for other quilt styles

Finally, we envision extending this work to include other quilting styles. For example, some quilters use fabric as a medium to "paint" stunning landscapes, and to our knowledge no quilt design software currently supports the design of landscape quilts. Potentially, such a system could allow users to paint scenes electronically, rendering the input strokes with real fabric textures. The system would record the geometry of each stroke and subsequently use this information to create cutting patterns and instructions for constructing the physical quilt. In this case, since the shapes of each fabric piece will vary, it may also be useful for the system to compute an appropriate packing such that the pieces can be cut efficiently, reducing fabric wastage as much as possible.

Alternatively, quilt designers could use existing paint programs to produce initial scene descriptions, and the quilt design software could detect the "strokes" in the resulting images. In that case, users could subsequently select and substitute fabric textures for each color used in the scene. Conceivably, the system could automatically solve for an optimal set of fabric textures without user intervention, but this may not be desirable. We are wary of removing the artistic aspects of quilt design from the hands of human artists. Our goal is to provide computational tools that support human creativity, not to supplant it.

Bibliography

[1] J. Adamson. *Calico & Chintz: Antique Quilts from the Collection of Patricia S. Smith.* Renwick Gallery of the National Museum of American Art, Smithsonian Institution, Washington, DC, 1997.

[2] J. Van Aken and M. Novak. Curve-drawing algorithms for raster displays. *ACM Transactions on Graphics*, 4(2):147–169, 1985.

[3] J. Arvo and K. Novins. Fluid sketches: Continuous recognition and morphing of simple hand-drawn sketches. *UIST '00*, pages 73–80, 2000.

[4] Bargello designer. http://www.sheilawilliams.com/bargo/Bargello.html.

[5] R. Bartels, J.C. Beatty, and B.A. Barsky. *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling.* M. Kaufman Publishers, Los Altos, CA, 1987.

[6] B. Baxter, V. Scheib, M.C. Lin, and D. Manocha. Dab: Interactive haptic painting with 3D virtual brushes. *SIGGRAPH '01 Conference Proceedings*, pages 461–468, 2001.

[7] B. Bederson. PhotoMesa: A zoomable image browser using quantum treemaps and bubblemaps. *UIST '01*, pages 71–80, 2001.

[8] J.E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965.

[9] J.E. Bresenham. Run length slice algorithm for incremental lines. In R.A. Earnshaw, editor, *Fundamental Algorithms for Computer Graphics, vol. 17 of NATO ASI*, pages 59–104. Springer, Berlin, 1985.

[10] C.M.A. Castle and M.L.V. Pitteway. An efficient structural technique for encoding 'best-fit' straight lines. *The Computer Journal*, 30(2):168–175, 1987.

[11] G.M. Chaikin. An algorithm for high-speed curve generation. *Computer Graphics and Image Processing*, 3:346–349, 1974.

[12] M. Coahran and E. Fiume. Sketch-based design for bargello quilts. In T. Igarashi and J. Jorge, editors, *Eurographics Symposium Proceedings: Sketch-based Interfaces and Modeling, 2005*, pages 165–174. Eurographics Association, 2005.

[13] T. Cockshott, J. Patterson, and D. England. Modelling the texture of paint. *Eurographics '92*, 11(3):217–226, 1992.

[14] S.D. Conte and C. de Boor. *Elementary Numerical Analysis: An Algorithmic Approach, 2nd ed.* McGraw-Hill Book Co., New York, 1972.

[15] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms, 2nd ed.* MIT Press, Cambridge, MA, 2001.

[16] C.J. Curtis, S.E. Anderson, J.E. Seims, K.W. Fleischer, and D.H. Salesin. Computer-generated watercolor. *SIGGRAPH '97 Conference Proceedings*, pages 421–430, 1997.

[17] C. de Boor. *A Practical Guide to Splines.* Springer-Verlag, 1978.

[18] G. Dewaele and M.-P. Cani. Interactive global and local deformations for virtual clay. *Graphical Models*, 66:352–369, 2004.

[19] M. Edie. *Bargello Quilts.* Martingale and Company, Woodinville, WA, 1994.

[20] Electric quilt 5. http://www.electricquilt.com.

[21] E. Ferley, M.-P. Cani, and J.-D. Gascuel. Practical volumetric sculpting. *The Visual Computer*, 16:469–479, 2000.

[22] E.L. Fiume. *The Mathematical Structure of Raster Graphics.* Academic Press, Boston, 1989.

[23] J.D. Foley, A. Van Dam, S.K. Feiner, and J.F. Hughes. *Computer Graphics, Principles and Practice.* Addison-Wesley, 1990.

[24] T.A. Galyean and J.F. Hughes. Sculpting: An interactive volumetric modeling technique. *Computer Graphics*, 25(4):267–274, 1991.

[25] M.D. Gross and E. Y.-L. Do. Ambiguous intentions: A paper-like interface for creative design. *UIST '96*, pages 183–192, 1996.

[26] P. Haeberli. Paint by numbers: Abstract image representation. *Computer Graphics*, 24(4):207–214, 1990.

[27] A. Hausner. Simulating decorative mosaics. *SIGGRAPH '01 Conference Proceedings*, pages 573–580, 2001.

[28] R.D. Hersch. Character generation under grid contraints. *Computer Graphics*, 21(4):243–252, 1987.

[29] A. Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. *SIGGRAPH '98 Conference Proceedings*, pages 453–460, 1998.

[30] J.D. Hobby. Rasterization of nonparametric curves. *ACM Transactions on Graphics*, 9(3):262–277, 1990.

[31] J.D. Hobby. Generating automatically tuned bitmaps from outlines. *Journal of the Association for Computing Machinery*, 40(1):48–94, 1993.

[32] K. Ichida, T. Kiyono, and F. Yoshimoto. Curve fitting by a one-pass method with a piecewise cubic polynomial. *ACM Transactions on Mathematical Software*, 3(2):164–174, 1977.

[33] T. Igarashi, S. Matsuoka, S. Kawachiya, and H. Tanaka. Interactive beautification: A technique for rapid geometric design. *UIST '97*, pages 105–114, 1997.

[34] T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy: A sketching interface for 3D freeform design. *SIGGRAPH '99 Conference Proceedings*, pages 409–416, 1999.

[35] C.S. Kaplan and D.H. Salesin. Escherization. *SIGGRAPH '00 Conference Proceedings*, pages 499–510, 2000.

[36] A. Kaufman. Efficient algorithms for 3d scan-conversion of parametric curves, surfaces, and volumes. *Computer Graphics*, 21(4):171–179, 1987.

[37] R.V. Klassen. Drawing antialiased cubic spline curves. *ACM Transactions on Graphics*, 10(1):92–108, 1991.

[38] P. Lancaster and K. Salkauskas. *Curve and Surface Fitting: An Introduction*. Academic Press Inc., London, 1986.

[39] J.A. Landay and B.A. Myers. Interactive sketching for the early stages of user interface design. *CHI '95 ACM Press*, pages 43–50, 1995.

[40] J.M. Lane and R.F. Riesenfeld. A theoretical development for the computer generation and display of piecewise polynomial surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(1):35–46, 1980.

[41] J. Lee. Simulating oriental black-ink painting. *IEEE Computer Graphics and Applications*, 19(3):74–81, 1999.

[42] S.L. Lien, M. Shantz, and V. Pratt. Adaptive forward differencing for rendering curves and surfaces. *Computer Graphics*, 21(4):111–118, 1987.

[43] D. Mould. A stained glass image filter. *Proceedings of the 14th Eurographics workshop on Rendering*, pages 20–25, 2003.

[44] B. Naylor. SCULPT: An interactive solid modeling tool. *Graphics Interface '90*, pages 138–148, 1990.

[45] V. Ostromoukhov. Digital facial engraving. *SIGGRAPH '99 Conference Proceedings*, pages 417–424, 1999.

[46] C.A. Padgham and J.E. Saunders. *The perception of light and colour.* Academic Press, New York, 1975.

[47] T. Pavlidis and C.J. Van Wyk. An automatic beautifier for drawings and illustrations. *Computer Graphics*, 19(3):225–234, 1985.

[48] M.C. Penders. *Color and cloth: The quiltmaker's ultimate workbook.* The Quilt Digest Press, San Francisco, CA, 1989.

[49] M. Plass and M. Stone. Curve-fitting with piecewise parametric cubics. *Computer Graphics*, 17(3):229–239, 1983.

[50] J. Preece, Y. Rogers, and H. Sharp. *Interaction design: beyond human–computer interaction.* John Wiley & Sons, Inc., New York, NY, 2002.

[51] W.T. Reeves. *Quantitative Representations of Complex Dynamic Shape for Motion Analysis.* PhD Thesis, Department of Computer Science, University of Toronto, 1980.

[52] G.B. Reggiori. *Digital computer transformations for irregular line drawings.* Technical Report 403-22, New York University, 1972.

[53] M.P. Salisbury, M.T. Wong, J.F. Hughes, and D.H. Salesin. Orientable textures for image-based pen-and-ink illustration. *SIGGRAPH '97 Conference Proceedings*, pages 401–406, 1997.

[54] C.E. Shannon. Communications in the presence of noise. *Proc. IRE*, pages 10–21, 1949.

[55] A.R. Smith. Digital paint systems: An anecdotal and historical overview. *IEEE Annals of the History of Computing*, 23(2):4–30, 2001.

[56] A.R. Smith and E.R. Lyons. Hwb – a more intuitive hue-based color model. *Journal of Graphics Tools*, 1(1):3–17, 1996.

[57] P. Stephenson and B. Litow. Running the line: Line drawing using runs and runs of runs. *Computers and Graphics*, 25:681–690, 2001.

[58] S. Strassmann. Hairy brushes. *Computer Graphics*, 20(4):225–232, 1986.

[59] S.W. Wang and A.E. Kaufman. Volume sculpting. *1995 Symposium on Interactive 3D Graphics*, pages 151–156, 1995.

[60] E.V. Warren and S.L. Eisenstat. *Great American Quilts: The Quilt Collection of the Museum of American Folk Art.* Penguin Studio *in association with* Museum of American Folk Art, New York, NY, 1996.

[61] B.A. Williams. *Colorwash Bargello Quilts.* Martingale and Company, Woodinville, WA, 2001.

[62] J. Wolfram. *Color Play: Easy Steps to Imaginative Color in Quilts.* C&T Pubishing, Lafayette, CA, 2001.

[63] W. Wong. *Principles of Color Design.* Van Nostrand Reinhold, New York, 1987.

[64] B. Wyvill, K. van Overveld, and S. Carpendale. Rendering cracks in Batik. *NPAR '04: Proceedings of the 3rd International Symposium on Non-photorealistic Animation and Rendering*, pages 61–70, 2004.

[65] R.C. Zeleznik, K.P. Herndon, and J.F. Hughes. Sketch: An interface for sketching 3D scenes. *Computer Graphics*, 30(4):163–170, 1996.