Generating patterns on clothing for seamless design

by

Clara Kang

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

Waterloo, Ontario, Canada, 2021

Author's Declaration

Abstract

Symmetric patterns are used widely in clothing manufacture. However, the discontinuity of patterns at seams can disrupt the visual appeal of clothing. While it is possible to align patterns to conceal such pattern breaks, it is hard create a completely seamless garment in terms of pattern continuity. In this thesis, we explore computational methods to parameterize the clothing pieces relative to a pattern's coordinate system to achieve pattern continuity over garments. We review previous work related to pattern alignment on clothing. We also review surface quadrangulation methods. With a suitable quadrangulation, we can map any planar pattern with fourfold rotations into each quad, and achieve a seamless design.

With an understanding of previous work, we approached the problems from three angles. First, we mapped patterns with sixfold rotations onto clothing by triangulating the clothing pieces and ensuring consistency of triangle vertices on both sides of a seam. We also mapped patterns with fourfold rotations onto clothing by optimizing the shape of each clothing piece in the texture domain. Lastly, we performed quadrangulation guided by cross fields, and mapped fourfold pattern units into each quad. We assembled and simulated the texture mapped clothing in Blender to visualize the results.

Acknowledgements

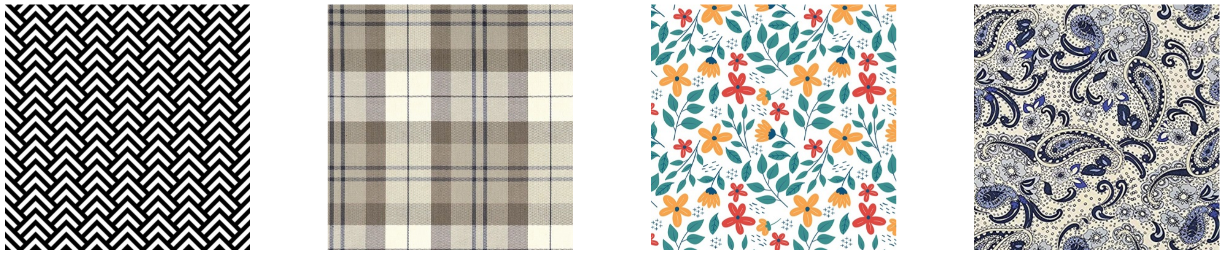
I would like to thank my supervisor Professor Kaplan for his guidance during my studies and the completion of this thesis. His insights and reading recommendations provided a lot of inspirations for my thesis, and greatly enriched my learning experience. I would also like to thank Professor Christopher Batty for providing help regarding finite element method and solving partial differential equations on a surface. I am also grateful for sharing graduate school experience with Yu Gu. The discussions we had was very helpful. Finally I would like to thank my parents for their support during my studies.

Table of Contents

List of Figures

Chapter 1

Introduction



(a) Herringbone (b) Plaid (c) Floral (d) Paisley

Figure 1.1: Some common fabric patterns

In clothing manufacture, garments are often assembled from patterned fabrics. Many traditional patterns are geometric, such as polka dots, herringbone and plaid. There are also more free-form patterns such as florals and paisleys. Some patterns are shown in Figure 1.1.

One common feature of traditional patterns is that they are fairly repetitive. There are several reasons for this repetition. First, the repetition reduces the complexity of the pattern design process. The designer creates a relatively small fragment of the overall pattern, which is then repeated to cover as much fabric as is needed. Repetition is also an integral part of the manufacturing process for fabrics. Entire rolls of fabric are printed from a few metal cylinders. The pattern must therefore repeat at a distance equal to the circumference of the cylinders. Besides manufacturing limitations, repeated patterns conform to what Gombrich called the "sense of order" in human aesthetics [11]: repetition can produce visual interest without overwhelming the eye.

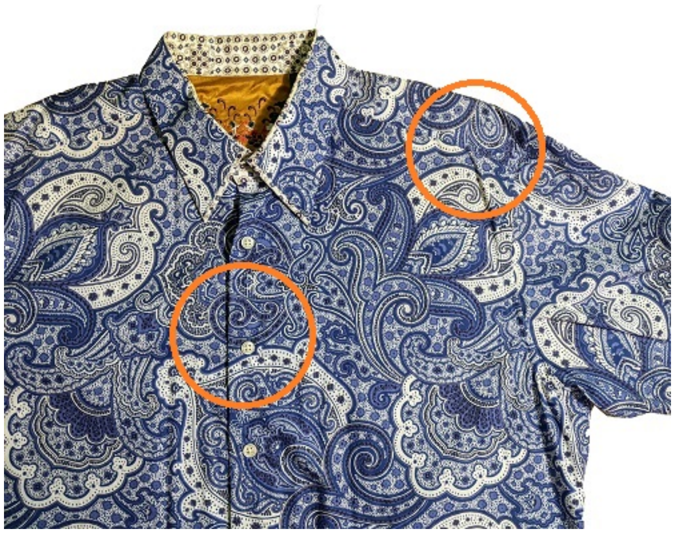


Figure 1.2: A men's dress shirt by Robert Graham
The fabric pieces are oriented so that the pattern is seamless across the placket. But no choice of orientation can completely eliminate the discontinuities at the shoulder.

A garment is constructed by cutting a set of shapes out of fabric and sewing them together. We will refer to a *garment design* as a collection of fabric *pieces*, each one a planar topological disk. The boundary of each piece can be decomposed into a circular sequence of *boundary segments*, which are a mixture of straight lines and curves. A subset of these boundary segments are paired with segments elsewhere in the garment design (possibly on the same fabric piece) and will be sewn together to create seams. We refer to these segments as *seam segments*, and assume that they are labelled to indicate the pairing relationships. Seam segments need not be congruent, but they must have the same arclength.

When fabric pieces are printed with a repeating texture, it is difficult or impossible to position and orient the pieces in texture space so that the texture passes continuously across all seams. If two seam segments are congruent, then the texture can be made continuous across that seam, as in the placket (the row of buttons) in Figure 1.2. But discontinuities are rarely avoidable across the whole garment. In places where multiple pieces of fabric meet in a loop, as in the shoulder in Figure 1.2, the fabric orientations can never be fully reconciled with the pattern. In this case, the complexity of the paisley texture helps to conceal the discontinuity. In general, the designer's best option is to choose "natural" orientations for the pieces. For example, a plaid pattern is typically oriented horizontally and vertically on the front and back of a shirt, and flows along the sleeves from shoulder to wrist.

In computer graphics, research on geometry processing has yielded a number of algorithms for seamless parameterization of surfaces [4], which might be used as a basis for constructing continuous textures for garments. Furthermore, modern computer-aided manufacturing, and web-based services like Spoonflower (spoonflower.com), make it easier than ever to imagine designing a pattern that exactly matches the pieces of fabric that will ultimately be assembled into a garment.

Given the pieces of fabric with the seams identified, and the repeating pattern, we want to find textures that can be mapped to the fabric pieces that evoke the original pattern and that flow continuously across all seams. In this thesis, we will demonstrate a few different techniques for solving this problem, by manipulating the repeating pattern so that it conforms to the shapes of the fabric pieces.

We will explore two main approaches to fitting patterns to fabric pieces. In the first, we subdivide fabric pieces into small units, and warp fragments of the original pattern into those units. Every periodic pattern can be decomposed into repeated copies of a *fundamental region*. For certain pattern types, we can define a square or equilateral triangle that is made up of a union of fundamental regions. By subdividing fabric pieces into shapes similar to one of these, we can accommodate patterns of these types.

In the second approach, we imagine warping the fabric pieces in the mathematical domain of the pattern, in such a way that the two sides of a seam are made to be equivalent under the pattern's symmetries. These new piece shapes are then used to define a warp of the pattern back into the original pieces, producing a seamless garment.

Obviously, the pieces and seams that make up a garment are meant to be assembled into a 3D shape. But the precise geometry of that shape is not uniquely defined from the flat pieces. Algorithms exist for simulating a plausible 3D garment from its pieces [2], and given a 3D model of a garment, we could turn immediately to standard parameterization algorithms in order to decorate it with a regular pattern. However, in this thesis we adopt the stance that such approaches are overkill. A sewing pattern is a natural representation of a piecewise developable 3D surface. It should be possible to apply a pattern to a garment without using any additional information (such as curvature) that is not already present in the piece shapes and seams. The continuity of the pattern will emerge when the pieces are sewn together, allowing the garment to assume whatever 3D shape it wants.

Throughout this work, we will use the fabric pieces of a shirt in Figure 1.3 and a pair of pants in Figure 1.4 to test our procedures. We obtained the shirt pattern from www.allfreesewing.com and the pants pattern is from www.siemachtsewingblog.com.

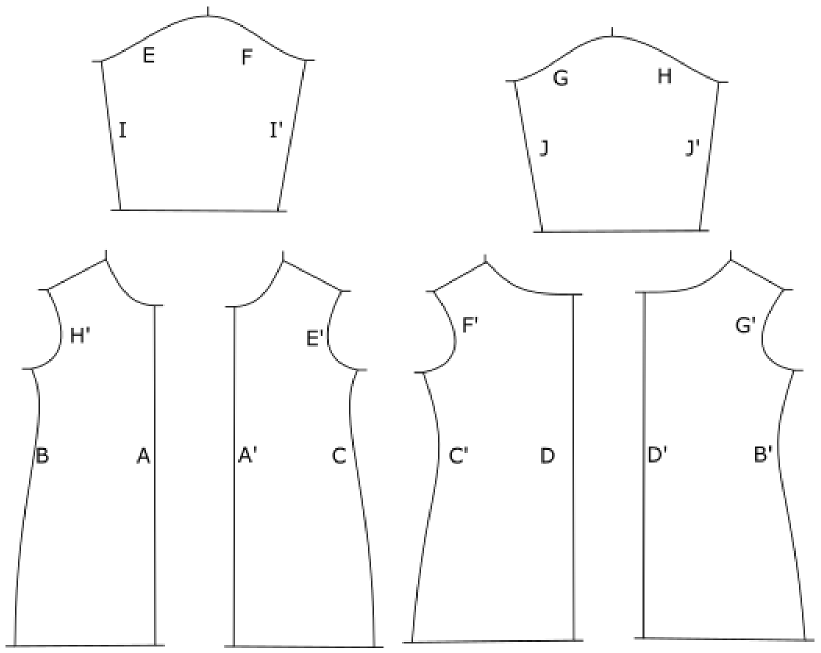


Figure 1.3: Shirt pattern
The letters indicate the seams.

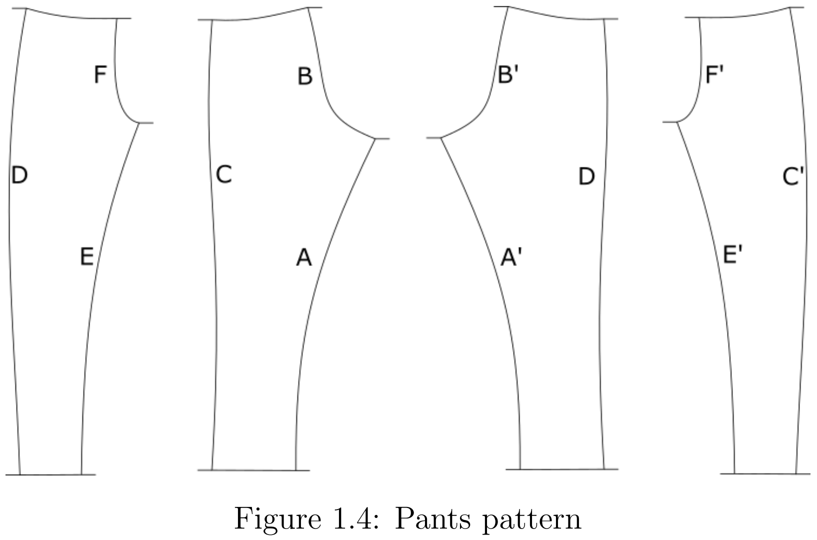


Figure 1.4: Pants pattern
The letters indicate the seams.

Our main contributions are the following.

- We present a method based on Delaunay triangulation, which permits any Euclidean pattern with sixfold rotational symmetries to be transferred to a garment.
- We present a method for patterns with fourfold rotational symmetries, which involves deforming fabric pieces in pattern space in order to derive a warp of the pattern to the garment.
- We present a method based on constructing a quadrangulation guided by a cross field, which permits any Euclidean pattern with fourfold rotational symmetries to be transferred to a garment.

1.1 Organization

Chapter 2 provides mathematical background on surface parameterization, cross fields and wallpaper groups, as they are the foundation of our work. Chapter 3 provides an overview of the relevant work about clothing pattern matching and mesh quadrangulation. Chapter 4 details our algorithm to map patterns with sixfold rotations onto clothing. Chapter 5 is about optimizing input pieces in pattern space for global parameterization. Chapter 6 describes our implementation of cross field guided quadrangulation.

Chapter 2

Mathematical background

This chapter provides an introduction to the mathematical concepts that are important to our purpose. We begin with the fundamentals of surface parameterization. We then introduce cross fields. Finally we briefly cover wallpaper groups.

2.1 Surface Parameterization

Viewed mathematically, a garment is a two-dimensional manifold surface S with one or more boundary curves. We will further regard a repeating pattern as being defined in a two-dimensional *pattern space* with axes u and v. A *parameterization* of the garment is a bijective mapping between the points of S and a subset U of the uv plane. This definition works equally well regardless of whether the garment is viewed as a single surface in 3D, or as a collection of disjoint 2D fabric pieces with labelled seams.

Surface parameterization has a long history in computer graphics. Floater and Hormann [10] offer an introduction and survey of techniques as of 2005; this section refers to that work.

In general, a surface has many possible parameterizations. We are usually interested in parameterizations that minimize some measure of distortion—the extent to which the pattern is scaled, stretched, or warped when mapped to the surface. A few especially well behaved types of parameterization can be understood via a function of a surface called the *first fundamental form* [8].

Let S be a surface, and let $x : U \to S$ be a parameterization of S. Fix a point

$p = x(u_0, v_0) \in S$, and let $\alpha(t) = x(u(t), v(t))$, $t \in (-\epsilon, \epsilon)$ be a curve on the surface, with $p = \alpha(0)$.

The first fundamental form is a quadratic form defined at every point on the surface. At each point p, it gives, for a vector $\vec{w}$ lying in the tangent plane T_p at p, the dot product of w with itself. It can be defined as a function $\mathrm{I}_p : T_p(S) \to \mathbb{R}$ such that $\mathrm{I}_p(\vec{w}) = \langle \vec{w}, \vec{w} \rangle = |\vec{w}|^2 > 0$.

Now let $x_u = \frac{\partial x}{\partial u}(p)$ and $x_v = \frac{\partial x}{\partial v}(p)$. The vectors x_u and x_v can be seen as the images of unit vectors u and v on the surface. The following equation expresses the first fundamental form in base x_u, x_v. Note that u' and v' indicate the derivative of u and v with respect to t.

$$\begin{aligned} \mathrm{I}_p(\alpha'(0)) = \langle \alpha'(0), \alpha'(0) \rangle_p &= \langle x_u u' + x_v v', x_u u' + x_v v' \rangle_p \\ &= \langle x_u, x_u \rangle_p (u')^2 + 2 \langle x_u, x_v \rangle_p u'v' + \langle x_v, x_v \rangle_p (v')^2 \\ &= E(u')^2 + 2Fu'v' + G(v')^2 \end{aligned} \tag{2.1}$$

where

$$\begin{aligned} E &= \langle x_u, x_u \rangle \\ F &= \langle x_u, x_v \rangle \\ G &= \langle x_v, x_v \rangle \end{aligned} \tag{2.2}$$

are the coefficients of the first fundamental form.

The coefficients above lead naturally to the expression of this quadratic form in a 2×2 symmetric matrix:

$$\mathrm{I} = \begin{pmatrix} E & F \\ F & G \end{pmatrix} \tag{2.3}$$

An *isometric* mapping preserves lengths, a *conformal* mapping preserves angles, and an *equiareal* mapping preserves areas. The following relations between I and the types of mapping exist.

1. x is isometric $\Leftrightarrow \mathrm{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
2. x is conformal $\Leftrightarrow \mathrm{I} = \begin{pmatrix} \eta & 0 \\ 0 & \eta \end{pmatrix}$ for some $\eta > 0$

3. x is equiareal $\Leftrightarrow \det(\mathrm{I}) = 1$

If $E = G = 1$, then the lengths of u and v are not changed by the mapping x, and if $F = 0$, the images of u and v are orthogonal to each other. Therefore, if I is an identity matrix, the mapping is an isometry.

If I is the identity matrix multiplied by a constant η, then the angle between local coordinates uv is still preserved, and the mapping is a conformal mapping.

For equiareal mappings, the relation between det(I) and the area is less intuitive. Let Q be a subset of U, corresponding under x to a subset of the surface S. The area of this subset of S is given by

$$\int_Q |x_u \wedge x_v| du dv$$

where $\wedge$ is the cross product operator.

And since

$$|x_u \wedge x_v|^2 + |x_u \cdot x_v|^2 = |x_u|^2 |x_v|^2,$$

$$|x_u \wedge x_v| = \sqrt{EG - F^2}$$

The element of the area is therefore the square root of the determinant of I, and having $det(\mathrm{I}) = 1$ means that the area is preserved under the mapping.

Most surfaces do not admit isometric planar parameterizations. Equiareal mapping is not unique, and may exhibit rotational behavior, which is unsuitable for our purpose. Conformal mapping is the most isometric mapping that one can get in practice. There are various ways to compute discrete conformal mapping on triangle meshes. However, most methods to compute conformal mapping do not take into account boundary conditions [13, 23, 16]. In this thesis we would like to have the ability to fix the boundary. For this reason, we find harmonic mapping more suitable in our case.

Harmonic mapping is a mapping that satisfies the Laplace equations

$$\triangle u = 0, \triangle v = 0 \tag{2.4}$$

One property of the harmonic mapping is that it minimizes the Dirichlet energy. Let $f(x, y, z) = (u(x, y, z), v(x, y, z))$ be a surface parameterization function.

$$E_D(f) = \frac{1}{2}\int_S ||\mathrm{grad} f||^2 = \frac{1}{2}\int_S ||\mathrm{grad} u||^2 + ||\mathrm{grad} v||^2 \tag{2.5}$$

Harmonic mapping is also convenient to compute. Once the boundary values are fixed, the texture coordinates of the interior of the shape can be solved with the Laplace equation.

For a discretized surface, the harmonic mapping can be calculated with the finite element method. The function value inside a triangle element is linearly interpolated from the function values at the vertices with barycentric coordinates.

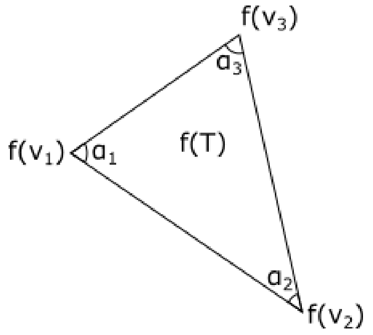


Figure 2.1: A triangular finite element

$$f(x, y) = N_1(x, y)f(v_1) + N_2(x, y)f(v_2) + N_3(x, y)f(v_3) \tag{2.6}$$

where the functions $N_i(x, y)$ give the barycentric coordinates of the vertex inside the triangle. These three functions are also sometimes known as the *shape functions*.

It can be shown that, with shape functions that equal to the barycentric weights of the vertices, the gradient of a function in a triangle element T, $\mathrm{grad}_T f$, satisfies the following equation:

$$2\int_T ||\mathrm{grad}_T f||^2 = \cot\alpha_3||f(v_1) - f(v_2)||^2 + \cot\alpha_2||f(v_1) - f(v_3)||^2 + \cot\alpha_1||f(v_2) - f(v_3)||^2 \tag{2.7}$$

The Laplacian equation for one vertex of the surface can then be written as

$$\sum_{j \in N_i} w_{ij}(f(v_j) - f(v_i)) = 0 \tag{2.8}$$

where N_i is the set of vertices that are neighbours of v_i, and $w_{ij} = \cot \alpha_{ij} + \cot \beta_{ij}$.

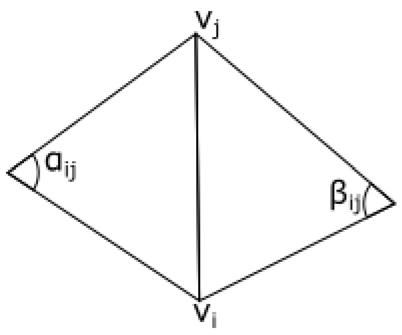


Figure 2.2: Weight of an edge

As long as the surface has at least one boundary vertex, the linear equations can be expressed with a symmetric matrix that has a unique solution. The number of rows in matrix M equals the number of internal vertices.

$$\begin{aligned} M_{ij} &= 1 && \text{if } i = j \\ M_{ij} &= 0 && \text{if } j \notin N_i \\ M_{ij} &= \overline{w}_{ij} = w_{ij} / \sum_{k \in N_i} w_{ik} && \text{if } j \in N_i \end{aligned}$$

The known values at the boundaries go to the right hand side of the equation. The matrix needs to be solved twice, once for u and once for v.

Methods that are similar to discrete harmonic mapping include shape-preserving methods and mean-value coordinates [10]. These methods produce similar-looking results, while the latter two behave more robustly with thin triangles.

2.2 Cross Fields

Some of the work in this thesis will involve computing continuous direction fields on fabric pieces in the *uv* plane. The literature on rotationally symmetric direction fields provides abstractions and algorithms for these computations.

A cross is an arrangement of four equal-length vectors, arranged at multiples of 90 degrees around a point [4]. A cross field assigns a cross to every point on a surface or in a 2D shape. Cross fields are special cases of n-fold rotationally symmetric direction fields, with $n = 4$.

Palacios and Zhang [20] provide a detailed explanation of the cross field. A cross s can be represented with

$$s = \left\{ \begin{pmatrix} R\cos(\theta + \frac{2k\pi}{4}) \\ R\sin(\theta + \frac{2k\pi}{4}) \end{pmatrix} | 0 \leq k \leq 3 \right\} \tag{2.9}$$

where R is the length of the arms of the cross, and θ is the angular component of one of the member vectors. To avoid ambiguities in the cross field calculation, a *representation field* can be used. A representation field is a vector field, where a representation vector is written as

$$\begin{pmatrix} R\cos(4\theta) \\ R\sin(4\theta) \end{pmatrix}. \tag{2.10}$$

A representation field can be calculated and then converted to a cross field. To calculate a representation field for a shape, we typically choose explicit field values for the boundary of the shape and then interpolate a smooth field in the interior. Most interpolation schemes aim to minimize an energy measure of the representation field. One such measurement is the Ginzburg-Landau energy [1] which is the Dirichlet energy plus an adjustable penalty term that aims to produce unit-length representation vectors:

$$E_\epsilon(f_1, f_2) = \frac{1}{2}\int_S (|\nabla f_1|^2 + |\nabla f_2|^2) dS + \frac{1}{4\epsilon^2}\int_S (f_1^2 + f_2^2 - 1)^2 dS \tag{2.11}$$

Here, f_1 and f_2 correspond to the two components of the representation field.

A *singularity* of a cross field is a point whose cross vanishes. *Hyperbolic sectors*, which categorize crosses with the same behavior, meet at singularities. We would like to find the *separatrices*, which are boundaries of the hyperbolic sectors, and thus get the quadrangular patches. We start with the singularity's Jacobian:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

where $a = \frac{\partial f_1}{\partial x}(p_0)$, $b = \frac{\partial f_1}{\partial y}(p_0)$, $c = \frac{\partial f_2}{\partial x}(p_0)$, $d = \frac{\partial f_2}{\partial y}(p_0)$

The local linearization at p_0 is

$$LV(p) = Jacobian(p_0)(p - p_0) \tag{2.12}$$

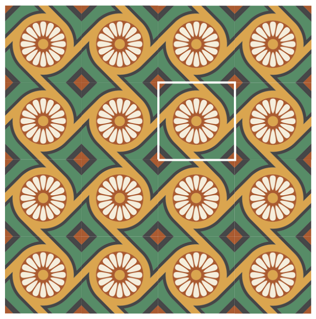
This gives the values of the representation field near a singularity. With local linearization, the directions of the separatrices that emanate from a singularity can be calculated. If p lies on a separatrix near a singularity p_0, then the vector from p_0 to p must align with the representation vector at p.

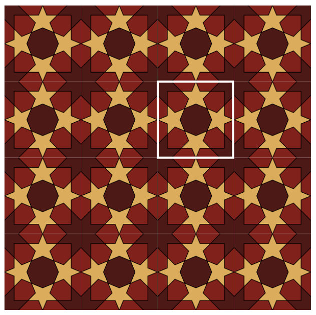
$$\frac{\sin(4\theta)}{\cos(4\theta)} = \frac{c\cos(\theta) + d\sin(\theta)}{a\cos(\theta) + b\sin(\theta)} \tag{2.13}$$

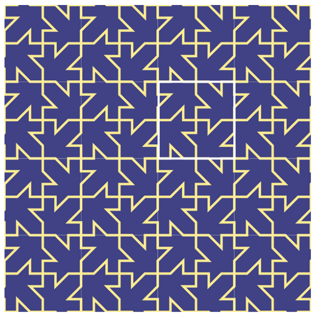
This equation can be transformed into a fifth order polynomial, yielding at most five separatrices at this singularity.

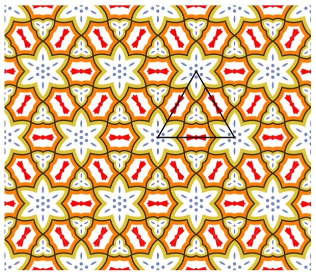
2.2.1 Wallpaper Groups

Although there are endless varieties of fabric patterns, they can be categorized into different wallpaper groups based on their symmetric properties. Kaplan gave an explanation of wallpaper groups [14]. A *symmetry* is a rigid motion that maps a figure onto itself. The symmetries of a figure form a group called the figure's *symmetry group*. A figure whose symmetries include translations in two non-parallel directions is called a *wallpaper pattern*, and its symmetry group is called a *wallpaper group*. There are precisely 17 wallpaper groups, which are based on different combinations of rotation, reflection, and glide reflection symmetries in addition to translations. A wallpaper pattern's base repetition unit is called a *tile*. The patterns in the same wallpaper group can be applied in the same way. We can view a garment as a surface that needs to be tiled. Different approaches are needed to tile the garment with patterns belonging to different wallpaper groups. In this thesis we are interested in patterns that include fourfold rotations (namely, patterns belonging to wallpaper groups $p4$, $p4m$, or $p4g$), and patterns that include sixfold rotations (and belong to groups $p6$ or $p6m$). Figure 2.3 shows sample patterns for each of these groups, with single tiles highlighted.


(a) *p4*


(b) *p4m*


(c) *p4g*


(d) *p6m*

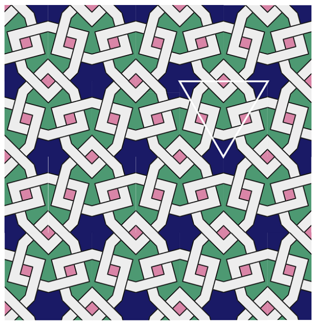

(e) *p6*

Figure 2.3: Some patterns
Examples of wallpaper patterns belonging to the five symmetry groups we will consider in this thesis. For each one, a tile is highlighted.

Chapter 3

Literature Review

This chapter gives an overview of the previous work that inspired this thesis. There are two major methods of mapping patterns onto clothing. The first one finds an optimal way to place a large region of the pattern onto each piece of fabric. This approach will be discussed in Section 3.1. The second option is to partition the clothing into cells that approximate simple shapes like triangle and squares, and then map a unit of texture into each cell. Section 3.2 surveys techniques for mesh quadrangulation, which can support patterns that can be decomposed into square units.

Mesh processing algorithms might not seem directly related to our goal, since our input is piecewise developable fabric pieces. However, most works on mesh quadrangulation only use the mesh as a discretization of the given geometry. It is the attributes of the underlying geometry that determine the result. As for us, we do not only have the freedom to subdivide the input clothing pieces, we are also given the seams. The seams are the only places where curvature is potentially nonzero. Therefore, we are able to use the ideas from mesh quadrangulation work, and make simplifications.

3.1 Pattern Alignment via Optimization

Wolff et al. [31] developed an algorithm that improves the alignment of texture patterns along clothing seams. The method uses the symmetries of patterns of different wallpaper groups, and finds an optimal way to fit patterns onto the pieces of fabric. The shapes of the clothing pieces are then adjusted to further improve the alignment. The input fabric pieces are first triangulated and turned into a 3D clothing mesh using cloth simulation.

All subsequent steps use the 3D clothing mesh directly. The optimization process involves three steps. Only pairs of triangles that will be joined along an edge to make a seam are involved in the first two pattern fitting steps. The first step is to find rotations of the pattern on each piece of fabric. The total rotational energy is the sum of the squared angle differences between orientations. The angle difference between pattern orientations is the minimum angular amount that a pattern needs to rotate to coincide with the orientation of the pattern on the adjacent fabric piece. To allow more freedom in aligning the patterns, reflections are permitted. The reflection energy is defined in the same way, but in reflection energy calculation, the angle is substituted with angle difference after performing reflection. The second step is to calculate the translations of patterns. In the translation energy calculation, the sum of distances between closest equivalent points on the patterns is minimized. An iterative algorithm is performed to keep adjusting the rotations and translations of fabric pieces within the pattern's coordinate system until convergence. The last step is to optimize the shapes of the fabric pieces in the world coordinates. According to the established texture pattern on each clothing piece, the ideal shapes of fabric piece boundaries are calculated, and moved to locations that are closest to their originals. This adjustment changes the shape of the final garment, thus some fidelity to the original clothing design is sacrificed for a better fit of the pattern. The above optimization steps are performed with multiple initial configurations of fabric piece rotations and translations, and the best candidate that minimizes the shape adjustment is selected. Figure 3.1 shows the intermediate results of the optimization process.

The authors also published another paper [30] which builds upon the initial work and takes the symmetry of the human body into account in the optimization process.

3.2 Quadrangulation

Remeshing is any process for converting a provided discretized surface, usually represented as a triangle mesh, into a new mesh that meets certain topological or geometric constraints. Mesh quadrangulation is a sub category of remeshing.

If we focus on patterns containing fourfold rotations, we can benefit from a range of mesh quadrangulation algorithms. With a quadrangulated mesh, we just need to map a repeat unit from the pattern trivially onto each quad. Bommes et al. [4] give a survey of the state of the art in mesh quadrangulation as of 2013. We adopt the terminology from their paper as follows:

The *valence* of a vertex is the number of outgoing edges.

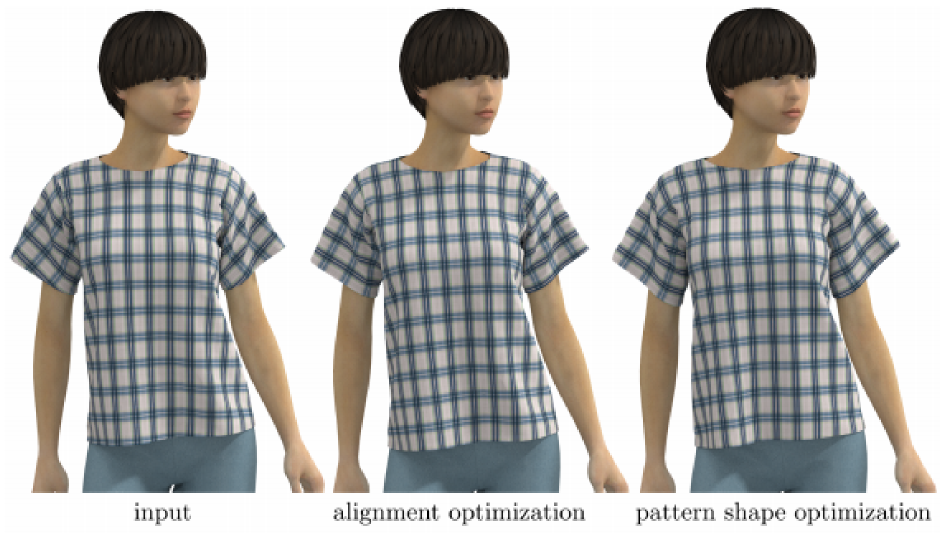


Figure 3.1: Pattern optimization steps [31]
The pattern near the shoulder area becomes increasingly aligned after each step.

A *regular vertex* has valence 4 if it is an internal vertex, valence 3 if it is on the boundary but not corner, valence 2 if it is on a corner.

A vertex is called *extraordinary* or *irregular* if it is not *regular*.

A *regular mesh* consists only of regular vertices.

A *semi-regular mesh* is created by refining a coarse base mesh made up of *patches.* The vertices where patches meet include all extraordinary vertices in the mesh.

A *valence semi-regular mesh* is a mesh that consists mostly of regular vertices.

A *unstructured mesh* is a mesh that consists mostly of irregular vertices.

3.2.1 Direct Quadrangulation

One way to construct a quad mesh is to start with a triangle mesh, and convert triangles directly into quads.

Owen et al. [19] create quadrilaterals from a triangular mesh by working inwards from the initial boundaries of the mesh. Neighbouring triangles of a front edge are modified to form a quad. The quad's quality is improved with local smoothing. Blossom Quad [22] does the conversion using perfect matching, and optimizes the shapes of all quads globally. Other methods [28, 27] identify a good matching and then process the leftover triangles with methods such as deleting and flipping edges. The performance of this type of method depends on the input mesh. They have limited range of resolution, since obtaining a good triangulation with large triangles is hard. Also, the resulting mesh is usually unstructured. In our work, we do not have any requirements for the regularity of the mesh, but we care about the quality of the quads. We do not want the p4 pattern tile to deform too much in terms of angles. Quadrilaterals constructed by joining pairs of triangles in a Delaunay triangulation are usually not close enough to squares.

In our case, however, we start with the shapes of fabric pieces in a garment design. we have the freedom to choose where to place the triangulation vertices in the input shapes. We find the following work more suitable to our purpose.

Mitchell et al. [18] use Delaunay triangulation and the two-color theorem [25] to quadrangulate 2D shapes. First, red and blue points are distributed within a shape using either Maximal Poisson Disk Sampling (MPS) or an advancing front method. A visualization of the quadrangulation by the two methods is shown in Figure 3.2. Each point is associated with a circle of the same colour and a circle of the other colour. Each circle indicates the region where other points of that colour cannot be. For MPS, points are generated randomly inside the region, while in advancing front, points are generated along the shape's boundary, then propagated inwards. The point generation process stops when no more points can fit in the region. Delaunay triangulation is then performed on the points. Next, the edges connecting points of the same color are removed. At this point, mostly quads are left. For polygons that are left, they can be triangulated by adding a Steiner vertex to the polygon. Quads with reflex angles can also be subdivided into smaller quads with smaller angles. The paper proves why such operations can always be performed, but we will not go into details. With a suitably chosen step size, the advancing front method of generating the vertices guarantees that most vertices will be regular, yielding a valence semi-regular mesh.

3.2.2 Patch-based Parameterization

A surface, typically represented by a triangle mesh, can be converted into a quad mesh by first partitioning the surface into coarse quadrilateral patches, and then subdividing each patch into a regular grid of quad faces. The resulting quad meshes are semiregular according to the definitions presented earlier. Compared to direct quad generation, patch-based quandrangulation offers more control over resolution. After obtaining the patches,

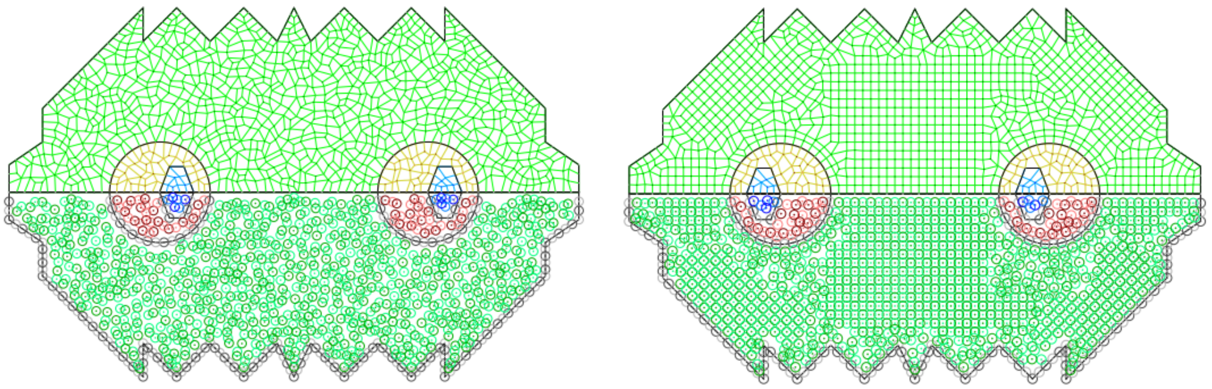


Figure 3.2: Quadrangulation with points generated with MPS (left) and advancing front (right) [18]

they can be subdivided and refined as desired by the user. The following works demonstrate various ways of generating the patches over a mesh.

Boier-Martin et al. [3] generate patches by clustering the mesh faces. First, an initial clustering is performed based on face normals, which gives an indication of flat regions. The cluster centres are then used as seeds to generate more refined clusters, this time using spatial distance. The new centers are calculated, and this process is repeated until convergence. All clusters must be homeomorphic to discs, roughly convex, and representable using an oriented height field. The second condition ensures the regions are well-shaped, and the last condition ensures that the parameterization does not distort the region's shape by too much. Any clusters that do not satisfy all three conditions are split until all clusters satisfy the conditions. Then cluster edges are cleaned up. The resulting cluster boundaries are polygons that can be split into quadrilateral patches without introducing Steiner points. The resulting base complex can be re-sampled robustly with the height field of the vertices. Catmull-Clark subdivision is applied to generate multiple resolution levels, and each level is re-sampled to achieve better quality before the next level is generated. Figure 3.3 shows an input mesh and the quadrangulated result.

Carr et al. [6] also use spatial clustering to obtain the patches. Instead of Euclidean distance, Chebyshev distance is used as a metric, which encourages the formation of square patches. Initial cluster centres (seeds) are placed on boundaries or at features with high curvature. In the cluster expansion process, mesh faces in each cluster are parameterized, and a potential new face's parameterization is evaluated based on its distance to its cluster's centre, the distortion that it will introduce, its potential to flip, and its contribution

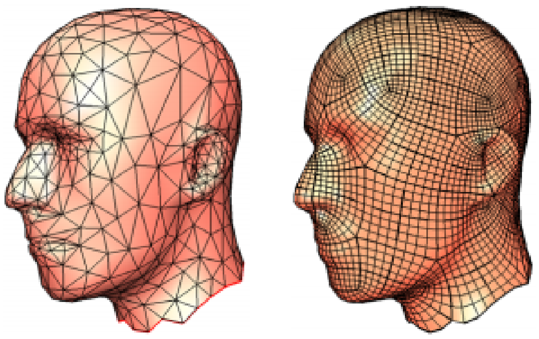


Figure 3.3: Mesh quadrangulation by Boier-Martin et al. [3]

to the smoothness of the cluster's boundary. Let us refer to the 2D space that the mesh maps to as the parameterization space. After expansion, the cluster centers in parameterization coordinates are calculated by taking a weighted average. The orientations of the parameterized clusters are also recalculated using principal component analysis. New seeds are placed at the frontiers of the existing clusters, and the process repeats until a satisfactory partition forms. Then the corners are identified for each patch, and the patch is remeshed to avoid degenerate triangles in the parameterization coordinates. Finally all patches are parameterized. Figure 3.4 shows the patches at each step.

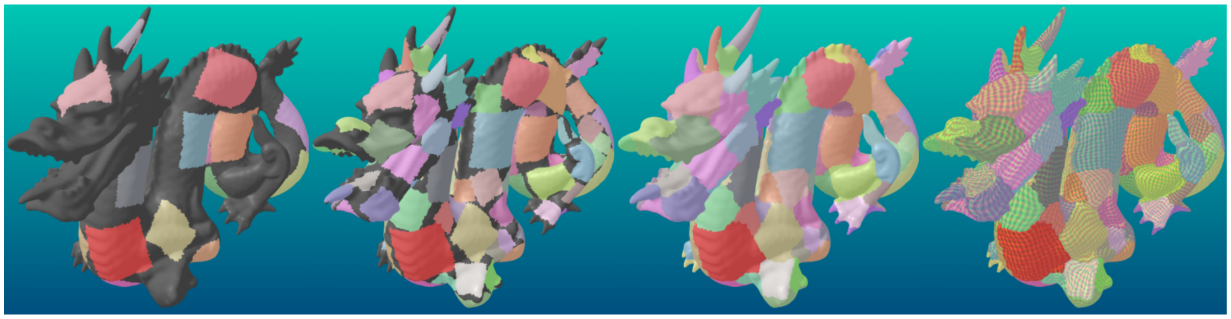


Figure 3.4: Patch generation by Carr et al. [6]
The first two images show the iterative patch generation process. The third picture shows the finalized patches. The last picture shows the parameterization of each patch with grid textures.

Building upon earlier work by Lee et al. [15], Daniels et al. [7] use a coarsening technique to obtain the patches. First, each triangle is subdivided using Catmull-Clark subdivision, resulting in an all-quad mesh. The mesh is then progressively coarsened with a simplification operation that preserves feature edge loops. Let us call the mesh before coarsening M, and let us call the coarsest mesh M^0. The mesh's forms at in-between steps are stored as keyframe meshes. For each keyframe mesh, its vertices are projected onto the next keyframe mesh using ray-casting. The mapping between two consecutive keyframe meshes is then relaxed until all inverted elements (neighboring faces whose projections overlap) are resolved. After all the projection functions are defined, they can be composed to yield a projection M to M^0. This mapping is bidirectional which allows backward projection. M^0 is then refined so that each face becomes a regular patch. The sampling on M^0 is guided by the surface area of M so that when the vertices are projected back onto M, the area of each face is similar. Figure 3.5 shows the quadrangulation process.

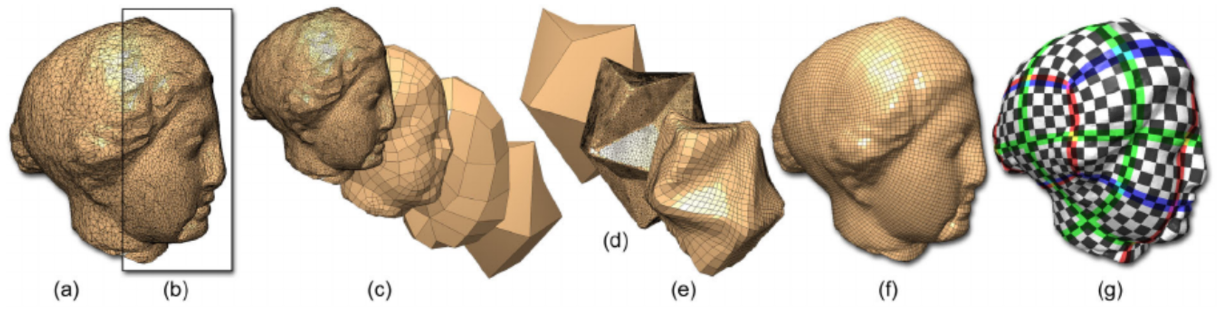


Figure 3.5: Semi-regular quadrangulation by Daniels et al. [7]
(a) is the input mesh. (b) is the mesh processed with Catmull-Clark subdivision. (c) shows the coarsening process. (d) is the mapping from the the mesh at step (b) to the base mesh. (e) the the refined base mesh. (f) shows the vertices on the refined base mesh projected back onto the original surface, which is the final result. (g) shows a grid texture mapped onto each patch.

PolyCube-Maps [26] are another technique that can yield patches for quadrangulation. The original mesh is approximated by a polycube (a collection of face-connected unit cubes). The faces making up the surface of the polycube are projected onto the model, and every vertex of the mesh is assigned to the polycube face whose projection contains it. Every vertex is then projected along its normal onto the polycube surface. The parameterization is optimized by minimizing the deformation energy. Figure 3.6 shows the quadrangulation process with polycube maps.

Dong et al. [9] apply discrete Morse theory to create a base quadrangular complex for a mesh. First, the discrete Laplacian operator L for the mesh is computed. The Laplacian

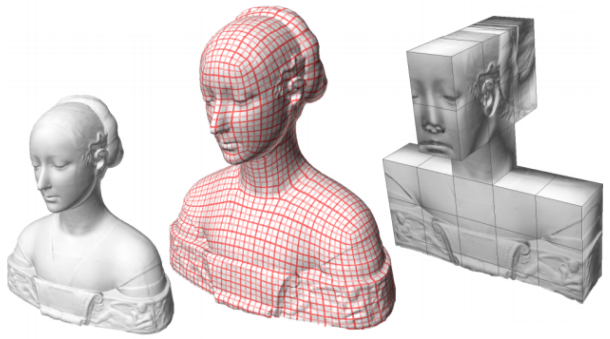


Figure 3.6: Quadrangulation with polycube map [26].
The first image is the input mesh, the second image is the quadrangulated mesh with each patch texture mapped with a square grid. The last image shows how the map is projected onto the polycube surface.

operator takes the form of a symmetric matrix with number of rows equals to the number of vertices in the mesh M.

$$L_{ij} = \begin{cases} \sum_k w_{ik} & \text{if } i = j \\ -w_{ik} & \text{if edge}(i,j) \in M \\ 0 & \text{otherwise} \end{cases}$$

An eigenvector of the Laplacian matrix, called a Laplacian eigenfunction, is chosen to be the surface function. The Laplacian eigenfunction assigns a value to each vertex, and distributes its critical values (saddles, maxima, and minima) evenly across the mesh. Let us call the vertices with the critical values *nodes*. The eigenfunction determines the number of critical values for a mesh. The steepest ascending and descending paths are traced starting from the saddles. The regions delineated by these paths are the patches. The nodes, paths and patches form a Morse-Smale complex. A property of the Laplacian eigenfunction is that all patches are quads. Topological noise can produce narrow patches, which are removed with a cancellation technique. For each patch, its nodes are assigned texture coordinates as the vertices of a unit length square. Mappings known as *transfer functions* are then computed between patches, so the texture coordinate of a vertex inside a patch can be expressed in terms of the reference frame of its neighbour patches. The texture coordinates of the vertices are calculated with discrete harmonic mapping with the values of the nodes as boundary values. In an iterative relaxation process, the boundaries of the patches are adjusted and the nodes relocated so that all the vertices inside the patch have texture coordinates between 0 and 1, and patch boundaries do not overlap. The process is repeated until no more adjustments can be made. With every patch parameterized, mesh quadrangulation can be performed by resampling the new vertices regularly. Figure 3.7 shows the spectral surface quadrangulation process.

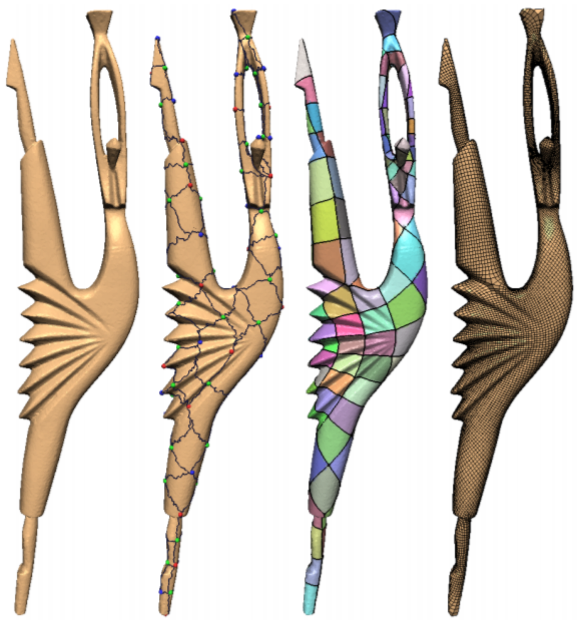


Figure 3.7: Spectral surface quadrangulation [9]
The first image shows the input mesh. The second image shows the initial complex. The third image shows the refined complex. The last image shows the quadrangulated mesh.

3.2.3 Field guided quadrangulation

Several methods use cross fields to guide the construction of patches for semiregular quadrangulation. Three major steps are needed in such methods: generation of the orientation field, specification of the sizing field, and construction of the quad mesh based on the two fields. For uniform remeshing a constant sizing field should be used, making the definition of the sizing field trivial. The steps are independent: different combinations of orientation generation methods and patch creation methods can be used. Some works that we include in this section are only about orientation field computation [12, 21], while others include both [5, 20, 29].

Hertzmann et al. [12] use cross fields to render 3D models with cross-hatching. A discretized cross field is defined by assigning a cross to every mesh face. The cross field is optimized to be as smooth as possible by defining an energy measure at each mesh edge, and minimizing the energy over all edges. The energy at an edge is based on the difference in cross orientations of the two faces that meet along that edge. Since each face has its own 2D coordinate system, two faces that share an edge need to be "flattened" (brought into the same coordinate system) before taking the difference of cross orientations. The cross is expressed as the angle of one arm plus $90k$ degrees where k is an integer. Let $E(i, j)$ denote the difference in cross orientations between neighbouring faces i and j. $E(i, j)$ is an expression that depends on k, but k can be eliminated by transforming $E(i, j)$ into $E_0(i, j) = -8E(i, j)^4 + 8E(i, j)^2 - 1$. The energy of the cross field is expressed as the sum of $E_0(i, j)$ over all edges. A cross field guided cross-hatching result is shown in Figure 3.8.

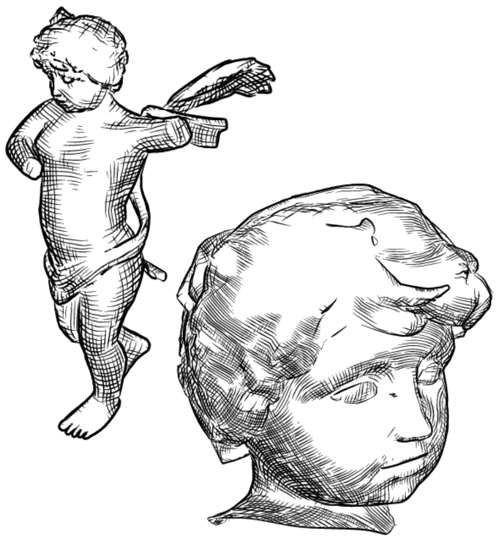


Figure 3.8: Cross hatching generated with cross field [12]

When a cross field is computed based only on mesh geometry and boundary conditions, the results can be highly sensitive to small variations in mesh geometry, and the field can contain many singularities. Ray et al. [21] provide a method to remove singularities by filtering geometry influence. The *index* of a point in the direction field reflects that type of singularity at that point. Zero index means that the point is not a singularity. The index is an expression of the direction field curvature and the geometry's curvature represented by angle defect at the corresponding vertex. The paper shows that the index can be precisely controlled by changing only the period jump of the direction field. With the target indices, the optimal direction field that minimizes the field's curvature can be calculated. Figure 3.9

shows a comparison of a cross field generated with full geometry influence and a cross field with partial geometry influence.

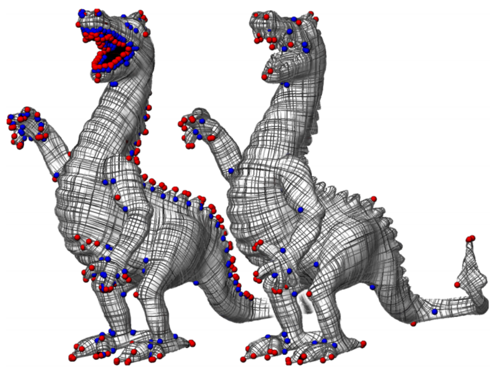


Figure 3.9: Direction field processing by Ray et al. [21].
The first image shows a crossfield generated with full geometry influence. The second picture shows the crossfield generated with filtered geometry influence.

Mixed-integer quadrangulation by Bommes et al. [5] bypasses the representation field, and calculates the cross field directly. The cross field in this work is represented by an angle field which is defined on the faces, and a *period jump* field that is defined on the edges. The energy is defined as the summation of the angle difference between all pairs of neighboring faces, similar to the energy defined by Hertzmann and Zorin [12]. The angle difference is calculated by translating the cross's angle on one face into its neighbour's coordinate system. Some orientation values are predefined to preserve sharp features on the mesh. A greedy heuristic is used to accelerate mixed-integer programming, which allows the computation of a high quality cross field in linear time. The patches can be derived from the cross field easily since period jump only happen on patch boundaries. Two steps are needed to parameterize the mesh. First the mesh is cut out to be homeomorphic to a disk to match the topology of the texture plane, and singularities are snapped to integer locations in texture coordinates. Next, the uv values across the mesh are also calculated by the solver to minimize the sum of the square of the difference between uv values and the field's gradient. Transfer functions identical to the ones in spectral surface quadrangulation [9] are used for cross boundary compatibility. With each patch of the mesh parameterized, quadrangulation can be performed like in spectral surface quadrangulation [9]. Figure 3.10 shows the intermediate results of mixed-integer quadrangulation.

Palacios and Zhang [20] provide a thorough explanation of fields with n-fold rotational symmetry (n-RoSy fields) and proposed a field editing algorithm to cancel out pairs of

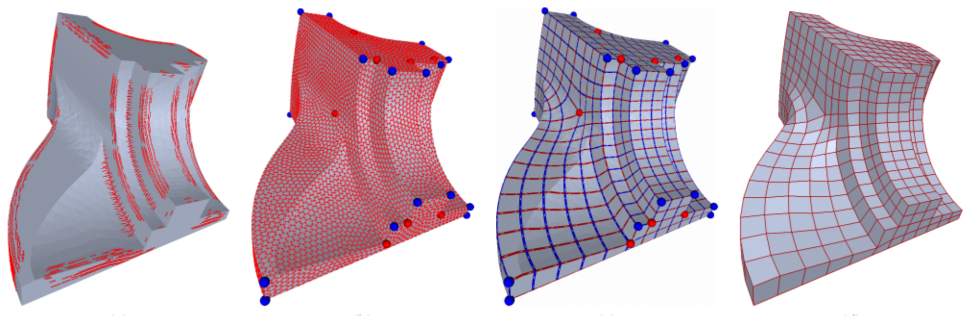


Figure 3.10: mixed integer quadrangulation by Bommes et al [5]
The first picture shows the constraint orientations. The second picture shows the computed cross field and the identified singularities. The third picture shows the parameterized mesh. The last picture shows the quadrangulation.

singularities. Their technique has proven useful for finding cross-hatching directions and quad remeshing.

Viertel and Osting [29] explain the concepts of cross fields, representation fields and separatrices in the context of complex analysis. They also prove that the procedure of first calculating a boundary-aligned cross field and then using separatrices to partition the domain always yields four-sided regions. Also, they propose to minimize the Ginzburg-Landau energy with the Merriman-Bence-Osher (MBO) algorithm [17], which initializes a random representation field, then solves the diffusion equation until convergence. They demonstrate the algorithm with 2D input shapes, where the cross field is defined to be perpendicular to a shape's boundary. Figure 3.11 shows their cross field generation and quadrangulation process.

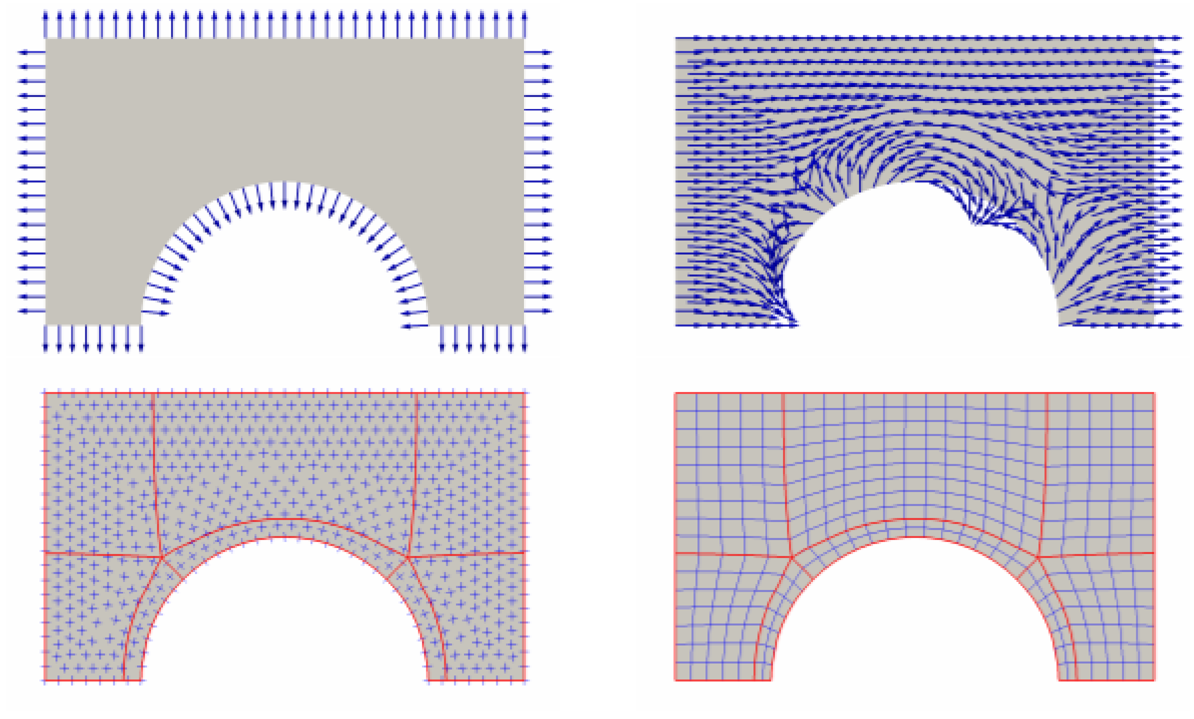


Figure 3.11: 2D Shape quadrangulation by Viertel and Osting [29]
The first image shows the normal directions of boundary vertices. The second image shows the solved representation field. The third image shows the cross field and the traced streamlines. The last image shows the quadrangulated mesh.

Chapter 4

Patterns with Sixfold Rotations

We came up with a simple way to map patterns with sixfold rotations onto garments. Any pattern with sixfold rotations, i.e, any pattern with symmetry group $p6$ or $p6m$, has tiles that take the shape of an equilateral triangle. If a tile is warped into any reasonably well behaved triangulation, like a Delaunay triangulation, we can get an approximately continuous representation of the pattern. Even though the pattern does not have a continuous derivative at the edges where two triangles meet, because the generated triangles are small compared to the scale of the fabric pieces, the distortion is spread out through the entire garment. This chapter describes the details of this approach, and some of its limitations, and solutions to overcome them.

4.1 Input

As discussed in section 1, the fabric pieces are given as input. The shapes of fabric pieces are described with either straight lines, or cubic Bézier curves, which will be called *boundary segments*. The connectivity between boundaries are identified in the input.

4.2 Procedure Description

The procedure is described as follows. First, we sample points on the boundaries of the fabric pieces. We assume that the clothing cannot be stretched at the seams, meaning that seam segments have the same length. For seam segments, we sample the same number of

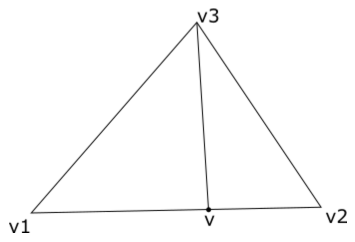


Figure 4.1: Annotated triangle

points on both boundaries. Second, Delaunay triangulation is performed. We used CGAL's 2D Conforming Triangulation library (https://www.cgal.org/), which is an implementation of Shewchuk's algorithm [24]. The triangulation library adds Steiner points inside the fabric pieces to ensure the triangulation does not contain triangles with angles or sizes outside a user specified range. Finally, the tile is mapped onto each triangle with affine mapping.

4.2.1 Continuity of Patterns

The patterns are C^0 continuous across connecting triangles. Suppose that we are warping the tile from the uv plane into a triangle with vertices v_1, v_2 and v_3 in the xy plane as indicated in Figure 4.1. Since $u(x, y)$ and $v(x, y)$ are continuous inside the triangles, we just need to prove that u and v are continuous on the seam segments.

With barycentric interpolation, we have

$$f(v) = \frac{||v - v_2||}{||v_1 - v_2||} f(v_1) + \frac{||v - v_1||}{||v_1 - v_2||} f(v_2)$$

For the other triangle that shares edge v_1v_2, $f(v)$ is calculated in the same way. Therefore, the patterns are C^0 continuous across neighbouring triangles.

With barycentric interpolation, the patterns are not C^1 continuous. However, as long as the two triangles that meet along an edge are reasonably close to equilateral, the derivative discontinuity will be small and unobtrusive. Furthermore, as we triangulate more finely, most triangles get closer to equilateral, which is good. Figure 4.2 shows two neighbouring triangles of different shapes texture mapped with a tile.

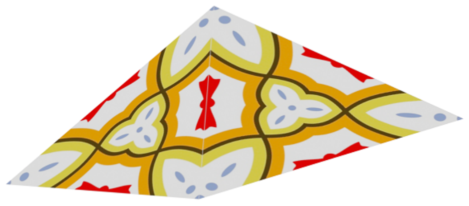
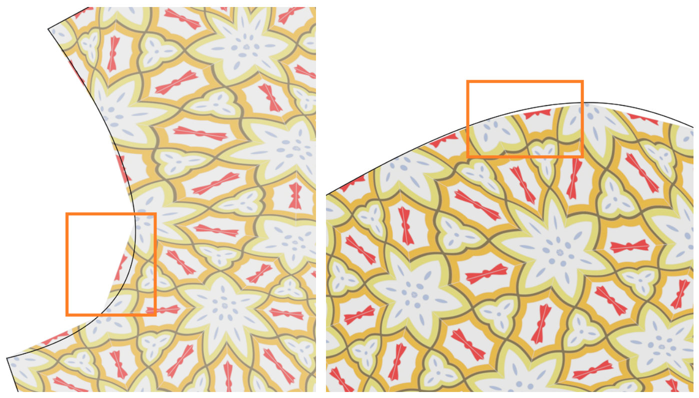


Figure 4.2: Two neighbouring triangles of different shapes mapped with a tile



(a) fabric piece with concave boundary (b) fabric piece with convex boundary

Figure 4.3: Discrepancy between fabric piece boundary and triangulation

4.2.2 Conform Pattern to Shape Boundary

When a fabric piece has a curved boundary, the sampling used to create vertices for the triangulation necessarily produces a discrete approximation of the piece's shape. If the goal is to simulate the appearance of a textured garment in a virtual environment, the discrete triangulation is likely to be sufficient as long as the triangles are small. But if we want to print and sew fabric pieces, the discrepancy may be visible: the texture might not reach all the way to convex parts of the boundary, or it might bleed over concave parts. Figure 4.3 shows examples of this behavior.

When a triangle contains two or more vertices from the boundary of the fabric piece, we construct a new shape where pairs of boundary vertices are joined by the boundary segment they delineate. We use harmonic mapping to warp the pattern non-linearly into this new shape. We sample the boundary of the shape at a fine scale and apply Shewchuk's triangulation algorithm again to generate a dense triangle mesh. With the fixed boundary condition, we then warp the original triangular tile harmonically into the new shape. Figure 4.4 shows the generated finite elements. While this doesn't completely eliminate the problem, it can make the error as small as we like.

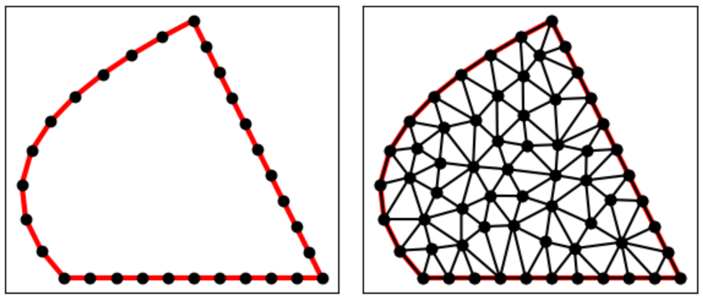


Figure 4.4: Creation of finite elements on a boundary element

The texture coordinates of the points on the straight boundaries of the original triangle are held constant. The points are sampled at equal distances on the boundary segments. For a point p on the curved boundary segment, let the fraction of the length of the path from the start of the curve to the point p over the length of the curved boundary segment be t, with $t \in [0, 1]$. Let the curve start at s and end at e, let s's texture coordinate be $(u(s), v(s))$, and let e's texture coordinate be $(u(e), v(e))$. The point p's texture coordinate is interpolated linearly from $(u(s), v(s))$ and $(u(e), v(e))$ by t: $u(p) = (1 - t)u(s) + tu(e)$, and $v(p) = (1 - t)v(s) + tv(e)$.

After the creation of finite elements and boundary texture coordinate values we can perform a harmonic mapping to get the texture coordinates of inner vertices. The result is shown in Figure 4.5.

4.3 Results

We control the scale of the texture pattern by specifying the minimal number of vertices on a shape boundary. For the shirt we experimented with a minimum of 4, 8, and 16 vertices on the boundaries, while for the pants we used 2, 4, and 8. We used a tile of group $p6m$ and a tile of group $p6$. The results are shown respectively in Figures 4.6 and 4.7. Figure 4.8 shows a close up rendering of the patterns mapped onto the clothing.

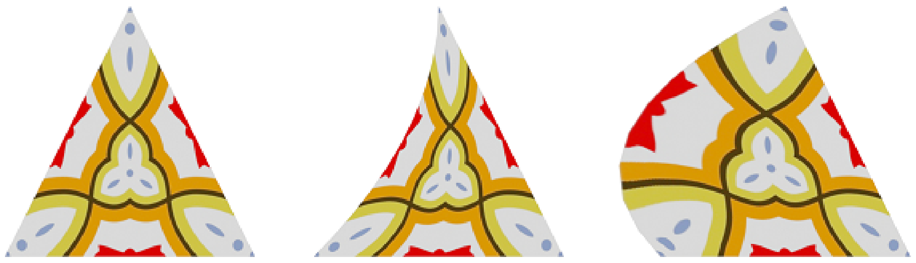
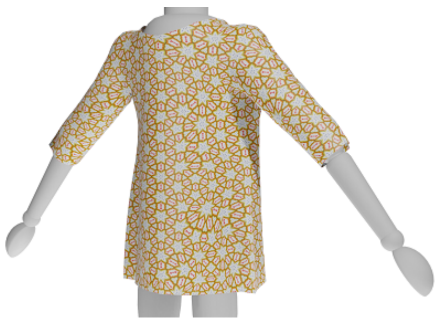


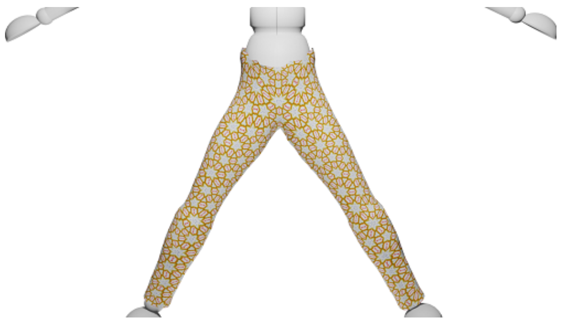
Figure 4.5: A triangle mapped harmonically to fill a patch of fabric that includes a boundary segment.

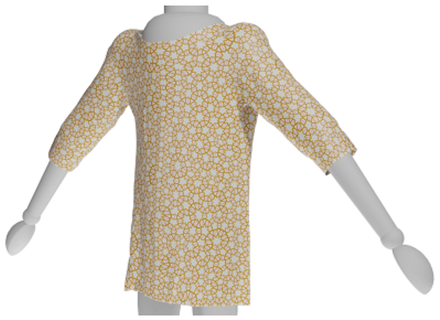
In these renderings, the garments are modelled and simulated based on the same triangulation used for texturing. This coupling imposes artificial upper and lower bounds on the scale of the triangles. If the triangles are too large, the simulated garment will not drape properly; if they are too small, the texture will be difficult to resolve. We have implemented harmonic mapping as a proof-of-concept, but we did not integrate it into the texturing pipeline for the production of real-world printed clothing.

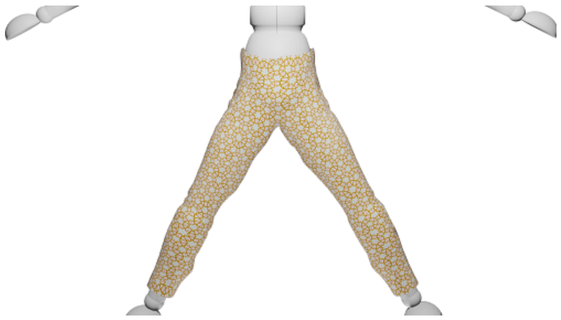
For all the sizes of patterns that we experimented with, the pattern is seamlessly and evenly mapped onto the clothing without obvious distortion. The pattern looks natural within fabric pieces. One can notice some large-scale alignment of the pattern of pattern due to the fact that vertices are sampled along the boundaries.

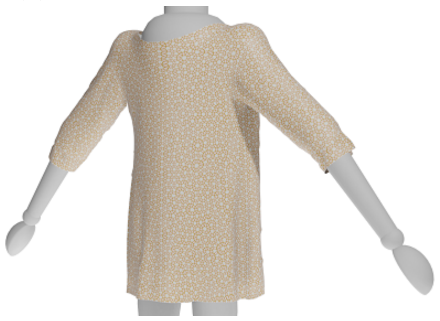
We managed to map patterns with sixfold rotations onto fabric pieces and align the patterns along seams. Patterns with fourfold rotations are also common in clothing design. The next two chapters detail our efforts to map patterns of fourfold rotations onto clothing.


(a) shirt with large pattern


(b) pants with large pattern


(c) shirt with medium pattern


(d) pants with medium pattern


(e) shirt with small pattern

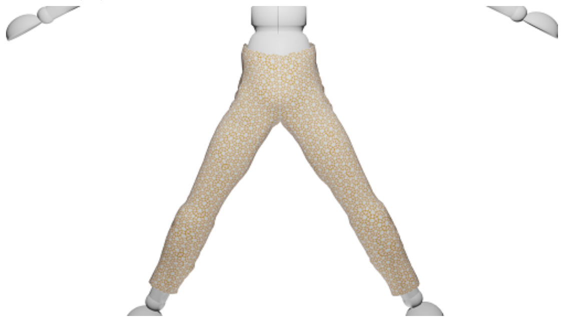
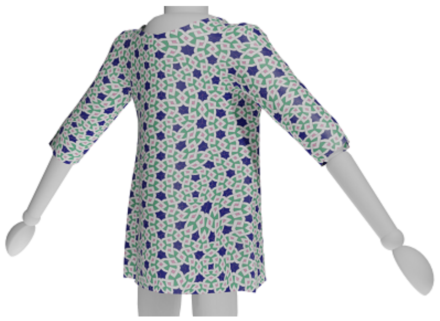
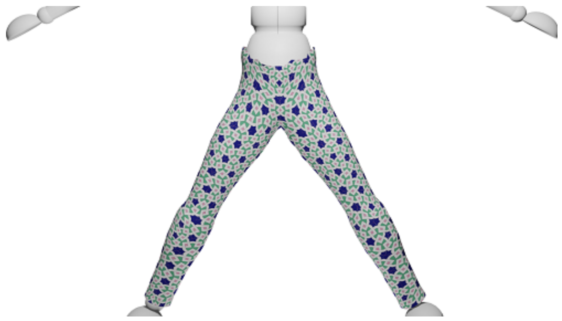
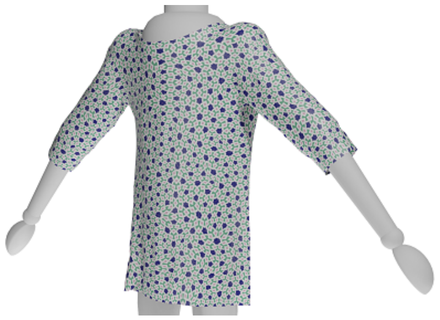
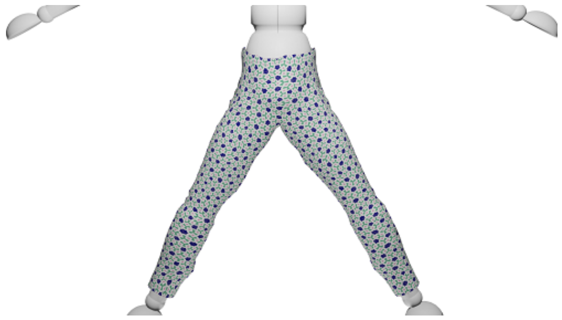

(f) pants with small pattern

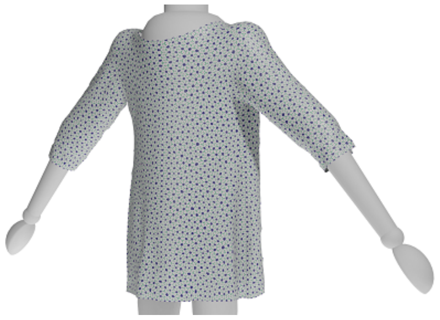
Figure 4.6: Clothing with seamless $p6m$ patterns


(a) shirt with large pattern


(b) pants with large pattern


(c) shirt with medium pattern


(d) pants with medium pattern


(e) shirt with small pattern

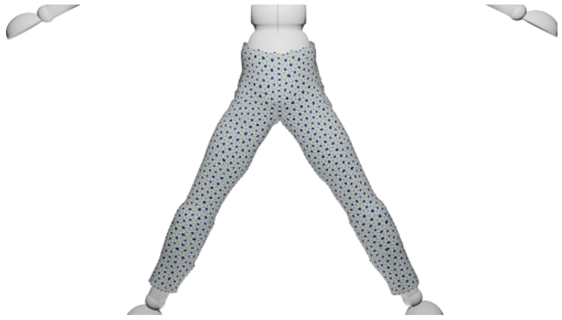
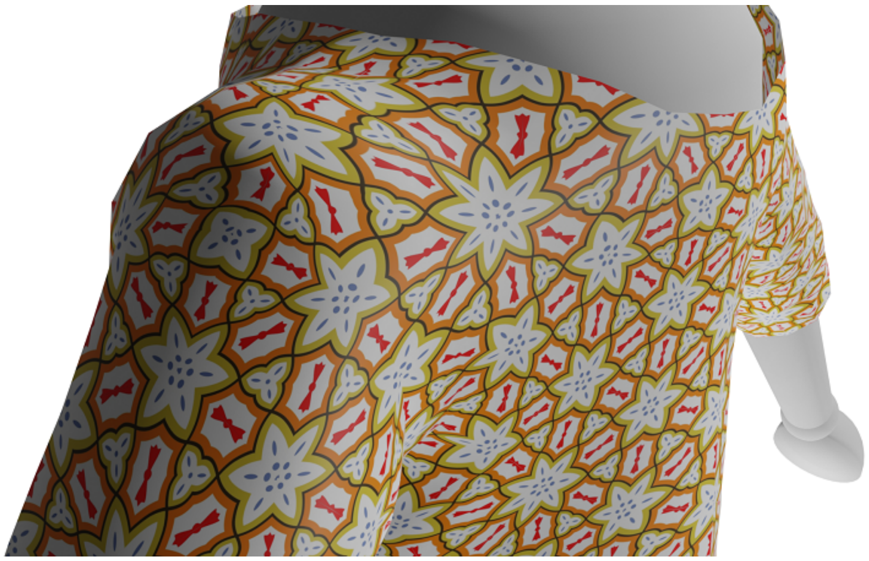

(f) pants with small pattern

Figure 4.7: Clothing with seamless $p6$ patterns


(a) clothing with $p6m$ pattern

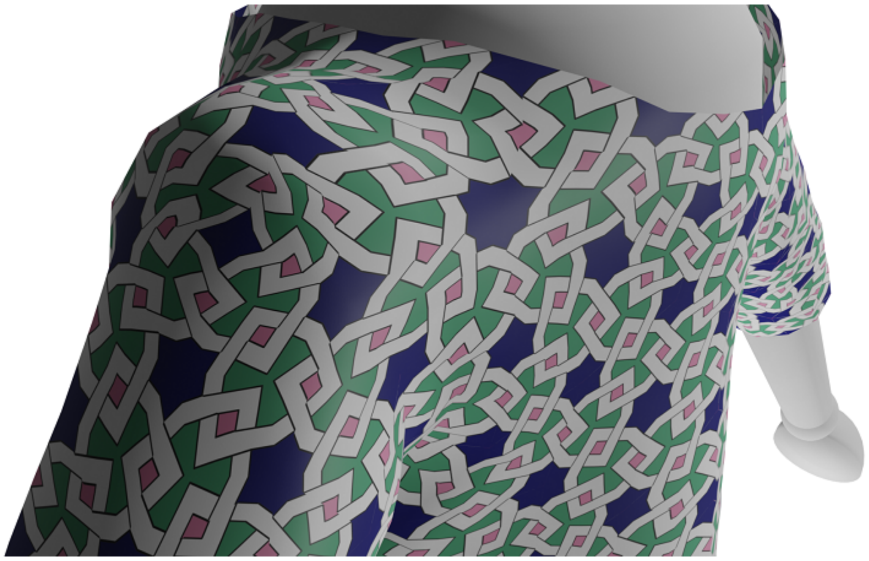

(b) clothing with $p6$ pattern

Figure 4.8: Close up rendering of clothing mapped with patterns of sixfold rotations

Chapter 5

Regular Parameterization with Shape Optimization

This chapter is inspired by the work of Wolff and Sorkine [31]. Given a fixed periodic texture and a sewing pattern for a garment, they translate, rotate, and deform the shapes of the fabric pieces to minimize the discontinuity in the texture across seams. This approach is reasonable when a garment must be designed based on a given roll of pre-printed fabric—the best we can do is to alter the garment to accommodate the texture.

If we are permitted to custom-print fabric pieces, it becomes possible to hold the garment design fixed and vary the texture instead. In this chapter we explore a variation on the technique of Wolff and Sorkine where we view their translation, rotation, and deformation as operating on the uv coordinates of the fabric pieces instead of on the geometry. We compute a set of seamless fabric pieces in texture space, which then induces a warp of the texture onto the original fabric pieces to produce a seamless garment. We demonstrate this approach for textures with fourfold rotations.

5.1 Boundary Matching Conditions

Let C_1 and C_2 be boundary segments in a garment pattern that are intended to be joined together at a seam. If we have no knowledge of the symmetries of a fabric texture, then the only way to make the garment seamless is to transform the fabric pieces in texture space so that C_1 and C_2 are made to coincide. If the texture is known to have periodic

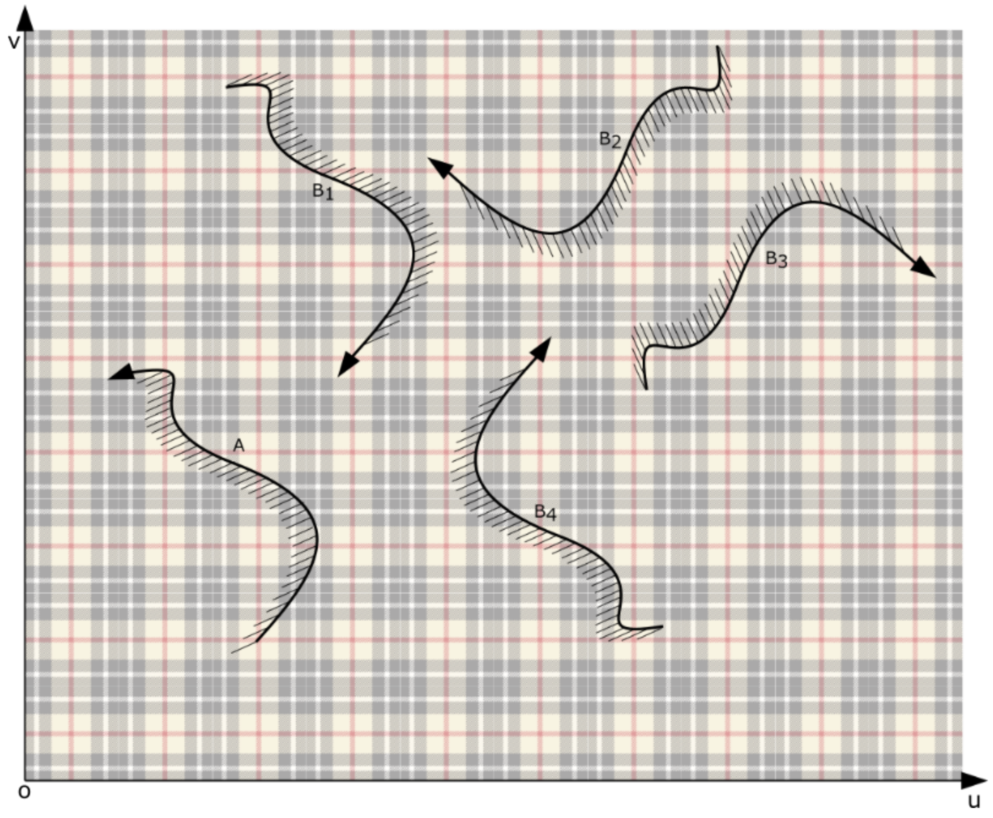


Figure 5.1: Equivalent matching boundaries

symmetry, though, then C_1 and C_2 need only be related by one of the texture's symmetry operations, and there will be infinitely many to choose from.

We focus here on textures with fourfold rotational symmetries, that is, textures belonging to wallpaper groups $p4$, $p4g$, or $p4m$. We know from symmetry theory that such patterns must have minimal-length translational symmetries that are orthogonal to each other. Scale and rotate the texture so that these translations are expressed using the vectors $(1, 0)$ and $(0, 1)$, and so that a centre of fourfold rotation lies at the origin. In that case, we can see that C_1 and C_2 can be joined without a visible seam if they are related through a rotation by a multiple of 90 degrees composed with an integer translation. That is, we seek integers $i \in \{0, 1, 2, 3\}$, j, and k such that

$$C_2 = T_{j,k}(R_{90i}(C_1)) \tag{5.1}$$

where $T_{x,y}$ represents a translation by vector (x, y), and R_θ represents a rotation about the origin by θ degrees. In Figure 5.1, B_1, B_2, B_3 and B_4 all match A.

5.2 Shape Simplification

If C_1 and C_2 are two boundary segments that make up a seam, then they are required only to have the same arclength. In general they can be arbitrary non-congruent curves. We will resolve these differences in texture space: we will deform C_1 and C_2 into two copies of the same curve C' that are equivalent under a symmetry of the texture. The deformation of the curves will cause the texture to be warped back onto the original fabric pieces, but the warped texture will be continuous across the seam.

For the purposes of optimization in texture space, we care only about the locations of the endpoints of C' due to the fact that they can potentially lie on two boundary segments. As long as two seam segments have endpoints that satisfy Equation 5.1, the endpoints can be connected by two congruent copies of any curve in texture space, and the resulting textured garment will be seamless. In our proof of concept approach, we simplify all boundary segments corresponding to seams by replacing them with line segments, as shown in Figure 5.2. A more general solution would further minimize texture distortion by constructing a new curve that interpolates between the two connecting boundary segments.

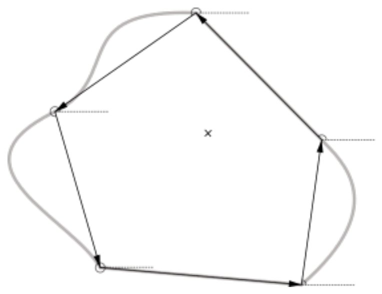


Figure 5.2: Simplified boundaries
The grey curves are the boundary segments. The vectors that start at an boundary segment's endpoint and end at the next one are the simplified boundaries. The angles that they make with the u axis are indicated.

5.3 Shape Optimization

Our goal is to make all connecting boundaries match, while minimizing the distortion to the original shapes. Let us refer to the whole process of adjusting the simplified shapes to satisfy Equation 5.1 for all connecting boundaries as the *shape optimization* process.

Since we would like the optimized shapes to stay as similar to their initial forms as possible, it is intuitive to use the difference in the shapes' angles before and after optimization as an indication of distortion. However, to represent angle, the dot product of two vectors is needed, and it would make the optimization problem too costly to compute. One way of avoiding this complexity is to separate the computation of boundaries' orientations from the locations of endpoints, as in Wolff and Sorkine's work [31].

5.4 Input Processing

Given a set of closed paths making up the boundaries of fabric pieces, we first compute their signed areas to determine their orientations. We ensure that all paths have clockwise orientation, reversing any paths that are oriented counter-clockwise. We then distribute evenly spaced samples along every boundary curve. For a Bézier curve, this is accomplished by the following steps. First we estimate the arclength of the curve by traversing it in small time steps. Then the curve's length is divided by a user defined minimum number of vertices on a boundary segment to get d_v. Finally, the curve is traversed one more time in small time steps, and a vertex is sampled every time we reach a distance of d_v. For connecting boundaries, we make sure that the number of sampled vertices on both boundary segments is the same. Because all boundary curves are oriented the same way, their sampled vertices will match with each other in reverse order. Figure 5.3 shows matching vertices.

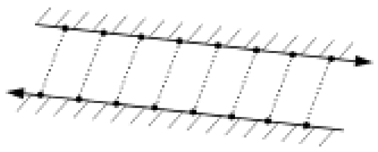


Figure 5.3: Matching boundary vertices

The shape's initial texture coordinates are calculated by multiplying every vertex's world coordinates by a global sizing factor. The sizing factor can be decided by the user. The larger the sizing factor, the smaller the texture appears on the garment.

5.5 Algorithm Description

As in the previous chapter, we first discretize the input shapes by triangulating them with Shewchuk's triangulation algorithm. We then proceed to optimize the shapes' boundaries, and finally use the *uv* coordinates of the vertices on the boundary and perform harmonic mapping to get the *uv* coordinates of the internal vertices.

5.5.1 Shape Boundary Optimization

The shape optimization process involves two major steps. First, the shapes rotate rigidly around the origin in order to optimize orientation differences between seam segments. Second, the endpoints of the boundary curve segments are modified so that they satisfy Equation 5.1.

Let C_1 and C_2 be two boundary segments that will be joined together into a seam. As discussed previously, we represent these two curves in terms of their endpoints. Let p_1 and q_1 be the start and end points of C_1, and likewise for p_2 and q_2. We seek to make p_1 equivalent to q_2 and q_1 equivalent to p_2 under a symmetry of the texture. We begin by optimizing the orientations of the fabric piece or pieces containing C_1 and C_2.

Let a_C be the angle that the vector from the start to the end of the curve C makes with the u axis. Let C_1 belong to shape S_1, and let C_2 belong to shape S_2.

We would like to rotate each shape such that the angle difference between boundary segment pairs are close to integer multiples to 90 degrees. Let θ_s be the rotation amount of a shape in degrees, where s is shape's identifier. Let d_P be an integer in $\{0, 1, 2, 3\}$ where $P = (C_1, C_2)$ is the pair ID of connecting boundaries. We would like to minimize diff_P, where

$$\text{diff}_P = (\theta_{S_1} + a_{C_1}) - (\theta_{S_2} + a_{C_2}) - 90d_P \tag{5.2}$$

Figure 5.4 shows an example of diff_P. The value diff_P measures how much the orientation difference between the simplified boundary segments deviates from their ideal orientation difference, which is an integer multiple of 90 degrees. It suffices to allow $d_P \leq 3$ because other angles differ from these by multiples of 360 degrees. Let U be the set of connecting boundaries in the input. The optimization's goal is to find values of θ_S that minimize the following sum:

$$\sum_{P \in U} \text{diff}_P^2$$

The optimization problem is a mixed-integer quadratic programming problem, which we solve using Gurobi (www.gurobi.com).

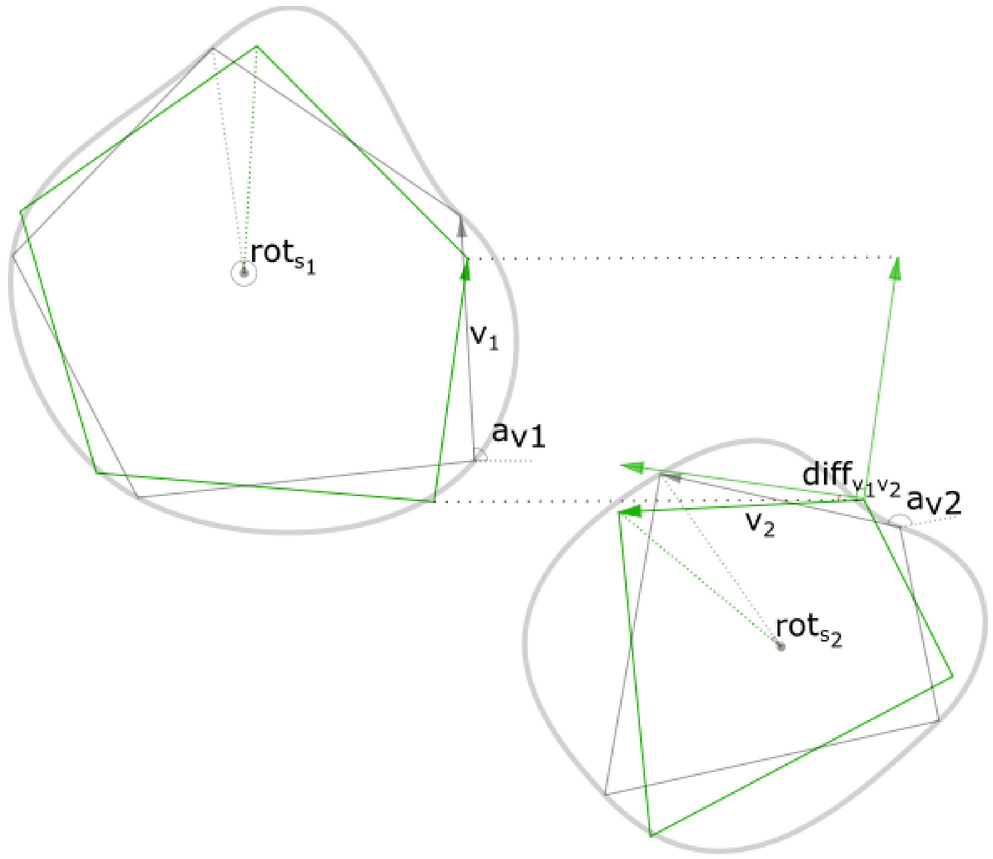


Figure 5.4: Orientation optimization
The grey shapes are before the rotation, the green shapes are from after the rotation. v_1 and v_2 are two simplified boundary segments. Their orientation difference after shape rotation is marked with $\text{diff}_{v_1 v_2}$. The orientation difference has integer multiples of 90 degrees removed.

After the calculations of θ_S, the shapes are rotated accordingly about the origin. They are then translated so that their new centres coincide with their old ones before rotation. However this step does not affect the result.

The second part of the optimization process is to modify the locations of the endpoints of the boundary segments, such that for a pair of connecting boundaries, their simplified forms satisfy Equation 5.1.

Even though the orientations of shapes are optimized, the orientation differences between pairs of connecting boundaries are not exactly integer multiples of 90 degrees. We need to enforce this constraint. First, the angle difference between connecting boundaries $\text{diff}_P = a_{C_1} - a_{C_2}$ is calculated, then the closest integer multiple of 90 degrees is chosen for the pair of boundaries. Let diff'_p be the final orientation difference between C_1 and C_2.

$$\text{diff}'_P = \text{round}(|\text{diff}_P|/90) \times 90 \tag{5.3}$$

For the pair of boundary segments C_1 and C_2 that has gone through orientation optimization, let p'_1, q'_1, p'_2, q'_2 be the new locations of the endpoints. Let the transformation from the final endpoints of C_1 to the final endpoints of C_2 be $\text{rot}(u, v) + (tu, tv)$. Depending on diff'_P, we have

$$\begin{cases} \text{rot}(u, v) = (u, v) & \text{if } \text{diff}'_p = 0 \\ \text{rot}(u, v) = (-v, u) & \text{if } \text{diff}'_p = 90 \\ \text{rot}(u, v) = (v, -u) & \text{if } \text{diff}'_p = -90 \\ \text{rot}(u, v) = (-v, -u) & \text{if } \text{diff}'_p = 180 \end{cases} \tag{5.4}$$

We would like to enforce Equation 5.1 while moving the boundary endpoints as little as possible. Let $\text{diff}u = u' - u$ and $\text{diff}v = v' - v$ be the differences between the final locations and the original locations in u and v dimension for an endpoint. We have:

$$\begin{cases} \text{rot}(u_{p_1} + \text{diff}u_{p_1}, v_{p_1} + \text{diff}v_{p_1}) + (tu_P, tv_P) = (u_{q_2}, v_{q_2}) + (\text{diff}u_{q_2}, \text{diff}v_{q_2}) \\ \text{rot}(u_{q_1} + \text{diff}u_{q_1}, v_{q_1} + \text{diff}v_{q_1}) + (tu_P, tv_P) = (u_{p_2}, v_{p_2}) + (\text{diff}u_{p_2}, \text{diff}v_{p_2}) \end{cases} \tag{5.5}$$

Figure 5.5 shows the transformation from the endpoints of one boundary segment to its mate. Let V be the set of endpoints belonging to boundary segments involved in seams. The objective of the optimization is to minimize

$$\sum_{pt \in V} \text{diff}u_{pt}^2 + \text{diff}v_{pt}^2 \tag{5.6}$$

with the constraint that tu_P and tv_P are integers. Again, we used Gurobi to solve this mixed-integer optimization problem.

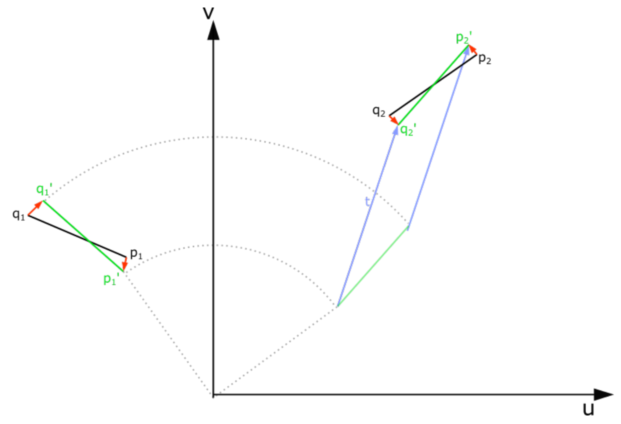


Figure 5.5: Vertices optimization
The red vectors show diff$u$ and diff$v$, and the blue vectors should tu_P and tv_P. The circular arcs indicate that the orientation difference is snapped to 90 degrees.

At this point, all endpoints of seam segments satisfy Equation 5.1. We can now determine the locations of the rest of the vertices on the boundaries. For connecting boundaries, even though they can take on any shape in uv space, we let them assume their simplified forms. The vertices' locations are interpolated linearly from the two endpoints. For the other boundaries, we interpolate the displacements instead of the locations of the vertices to preserve the shapes of the boundaries. For a boundary, let p_1, p_1', q_1, q_1' be the locations of the endpoints before and after location optimization. Let $d_{p_1} = p_1' - p_1$, $d_{q_1} = q_1' - q_1$ be the displacements of the endpoints. Let v_{num} be the total number of vertices on the boundary. The location of the ith vertex on the boundary is calculated as follows:

$$\begin{aligned} t &= i/v_{\text{num}} \\ d &= (1-t)d_{p_1} + td_{q_1} \\ v &= v + d \end{aligned}$$

5.5.2 Harmonic mapping with transfer functions

With the boundaries fixed, we calculate the uv coordinates of the interior vertices. We can use all the uv locations of boundary vertices as boundary conditions. However, knowing the connectivities among shapes, for a pair of connecting boundaries, we can represent one in terms of the other. After all, when the pieces are sewn together, the vertices on the seams appear only once. Dong et al. [9] use transfer functions to specify the relationship between vertices of different patches on the same mesh. We can adapt this technique to enforce continuity across seams. This section details the construction of the Laplacian matrix with transfer functions.

For each pair of connecting boundaries (C_1, C_2), let $v_1 \in C_1$, let $v_2 \in C_2$ be the vertex v_1 connects with. Let $f(v_1) = v_2$, where f is the transformation that brings v_1 to v_2, as determined in the previous section. We call f the transfer function in the context of harmonic mapping. The endpoints of all boundaries are fixed, and are not part of the linear system. Let us define all other vertices on C_1 as *explicit* and the ones on C_2 as *implicit*. We only put explicit and interior vertices in the linear system. Let the number of vertices in the system be N.

We need a Laplacian matrix of size $2N$, with one row for each of the u and v components of the explicit and interior vertices. This is because some transfer functions involve both components. We let the first N entries in the matrix correspond to the u components, and the last N entries to the v components. We have

$$Mx = b$$

where M is the coefficient matrix, $x = [u_0, u_1, \cdots, u_n, v_0, v_1, \cdots, v_n]^T$, and the constants of the equations are stored in b. The constants are from the translation part of transfer functions and the fixed uv values of the endpoints.

Every explicit vertex has two kinds of neighbours. The first are vertices belonging to the same fabric piece. The others are vertices to which it is connected across a seam. In Figure 5.6, the pink vertices are the neighbors that are in the same shape as the explicit vertex. The blue ones are its neighbors across the seam. The neighbors of v_{im} that are on the implicit boundary are not included, as they are duplicates of the neighbors of v_{ex} on the explicit boundary.

If a given vertex has an implicit neighbour, we replace that neighbour in the matrix with the implicit vertex's explicit twin. In Figure 5.7, the pink and aqua vertices are the neighbors of the blue vertex.

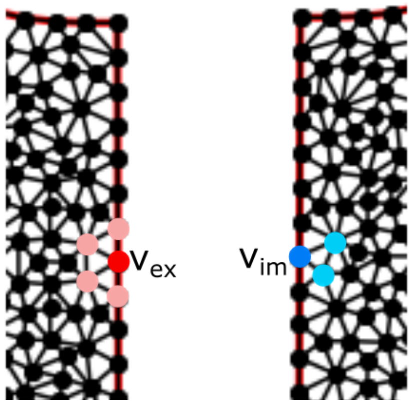


Figure 5.6: Neighbors of explicit vertex

To compute the harmonic mapping, for each vertex v_i, let $\mathbf{u_i} = (u_i, v_i)$ be its texture coordinates. Let vertex v_j be a neighbor of v_i, and let v_j belong to shape S_j, and let v_i belong to shape S_i. Let $\overline{w}_{ij}$ be the normalized cotangent weight introduced in Section 2.1. We have

$$\sum_{(j,S_j)\in N_i} \overline{w}_{ij}(\mathbf{u_i}^{S_i} - \phi_{S_j S_i}\mathbf{u_j}^{S_j}) = 0 \tag{5.7}$$

For example, let the transfer function from shape S_j to shape S_i across the corresponding boundary be

$$\phi_{\beta\alpha}(u,v) = (-v + tu, u + tv)$$

For v_j, its u value corresponds to the jth row of M and its v value corresponds to the $j+N$th row of M. We have:

$$M_{i(j+N)} = \overline{w}_{ij}$$

$$M_{(i+N)j} = -\overline{w}_{ij}$$

$$b_i = \overline{w}_{ij}tu$$

$$b_{i+N} = \overline{w}_{ij}tv$$

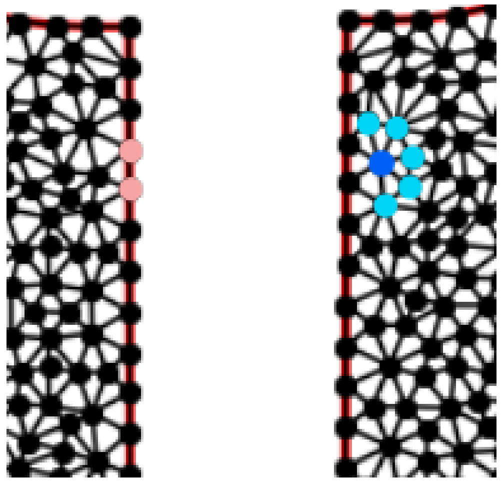


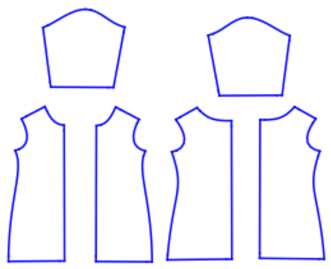
Figure 5.7: Neighbors of implicit vertex

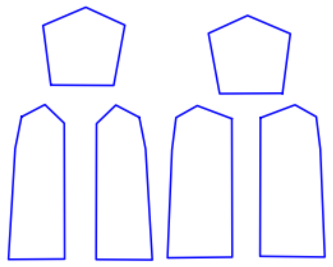
5.6 Results

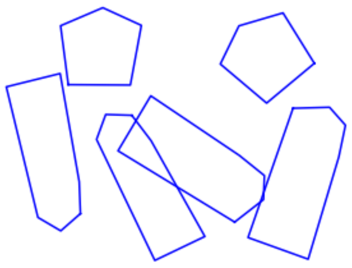
We perform the parameterization with different scales of texture patterns on the garment patterns for the shirt and pants.

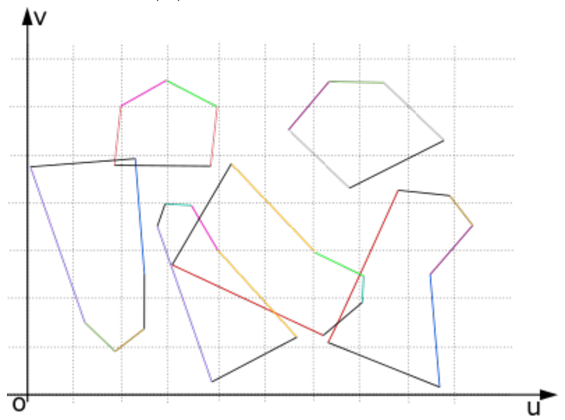
Figure 5.8 shows the shirt's shapes being optimized in the texture domain to permit the warped texture to flow seamlessly across the fabric pieces. When the size of the pattern in texture space is identical to its original size, we can perceive the most distortion. However, with transfer functions, we still get continuous patterns. In Figure 5.10, we notice that with transfer functions, we get C^1 continuous texture under the right sleeve while the discontinuity is obvious on the clothing parameterized with fixed boundary conditions.

An advantage of regular parameterization with shape warping is the flexibility of texture scale. Smaller texture scales usually give better results since the distortion can be distributed to more texture units. We scale the original shapes by a factor of 1, 5 and 20 respectively. The results for the shirt are in Figure 5.10. The results for the pants are in Figure 5.11. All results show patterns with balanced distortion. Figure 5.12 and Figure 5.13 show close up renderings of the garments.


(a) Original shapes


(b) Simplified shapes


(c) Rotation optimization



(d) Endpoint location optimization

Figure 5.8: Boundary optimization steps
The simplified boundary segments that belong to the same seam are coloured the same in (d) to show the relation between each pair of simplified boundary segments.

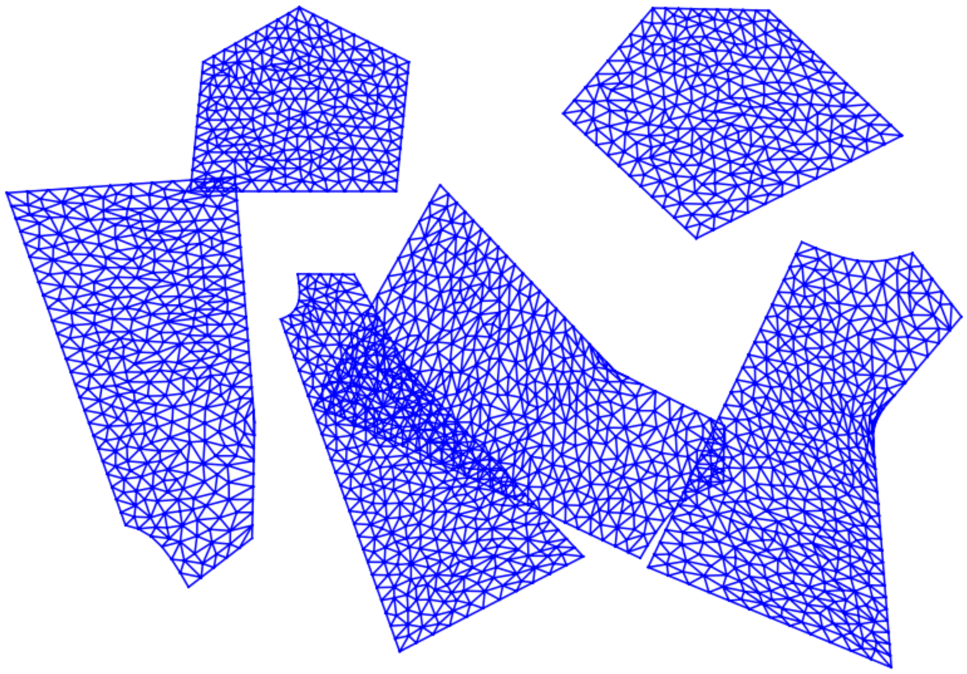
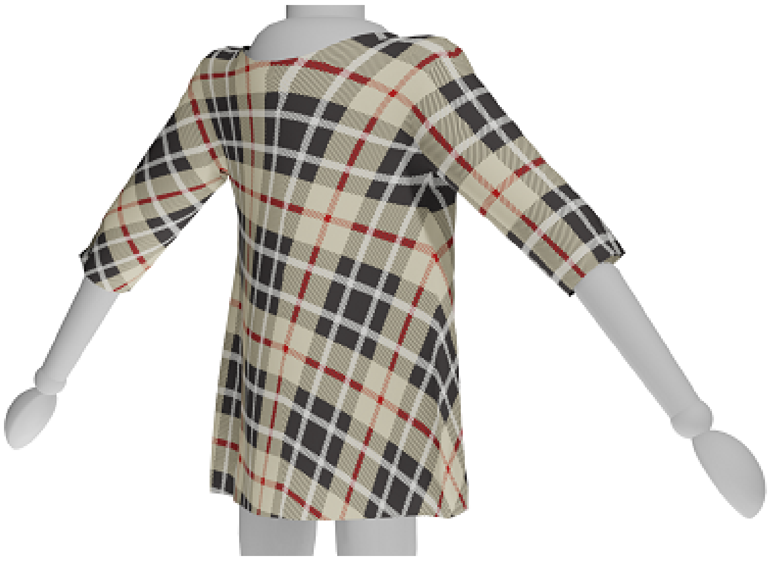
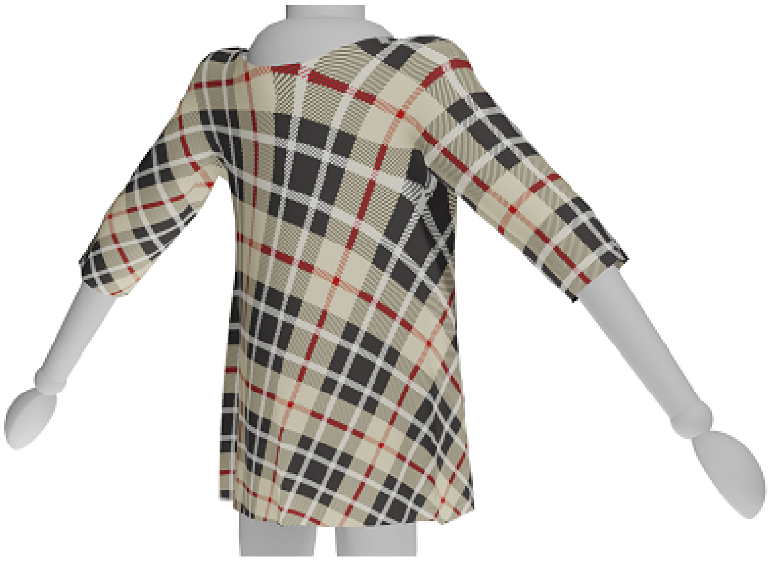


Figure 5.9: Shapes after harmonic mapping

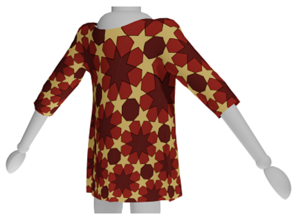


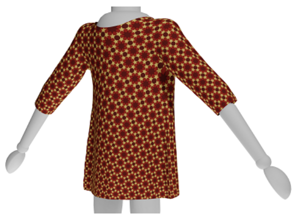
(a) Without transfer functions

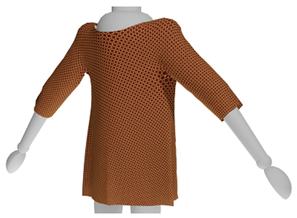


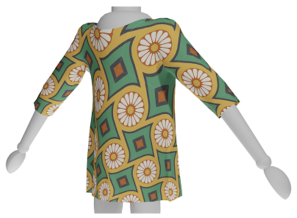
(b) With transfer functions

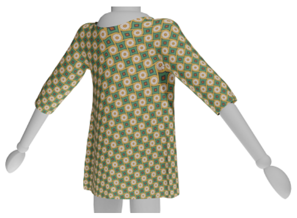
Figure 5.10: Comparison of parameterizations
With transfer function, we see a major improvement on the smoothness of the pattern near the seam under the right sleeve.

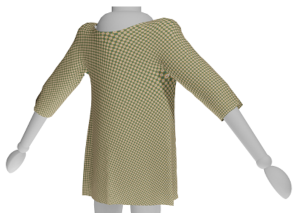

(c) shirt with large *p4m* pattern

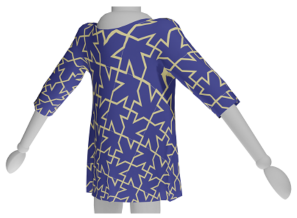

(d) shirt with medium *p4m* pattern

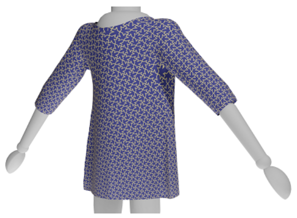

(e) shirt with small *p4m* pattern


(f) shirt with large *p*4 pattern


(g) shirt with medium *p*4 pattern


(h) shirt with small *p*4 pattern


(i) shirt with large *p4g* pattern


(j) shirt with medium *p4g* pattern

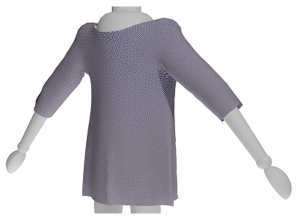
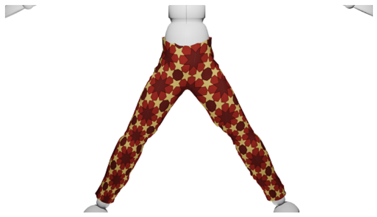
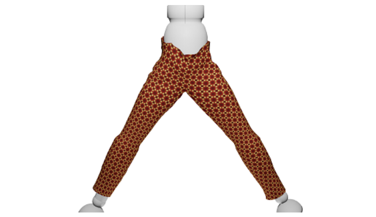
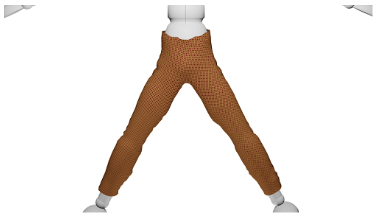
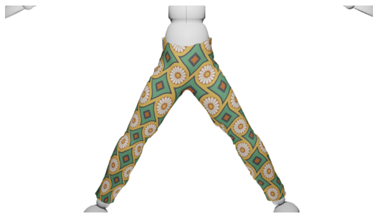

(k) shirt with small *p4g* pattern

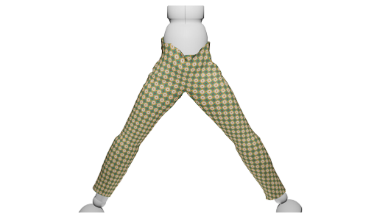
Figure 5.10: Regular parameterization with shape optimization for shirt

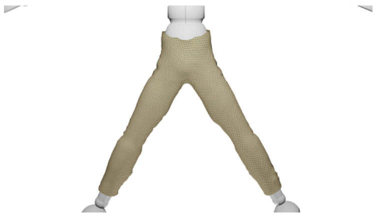

(a) pants with large *p4m* pattern

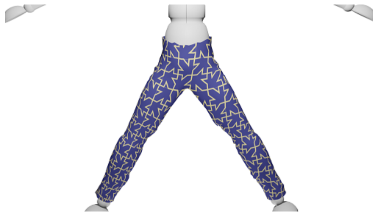

(b) pants with medium *p4m* pattern

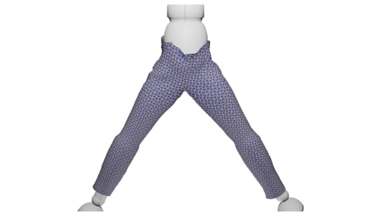

(c) pants with small *p4m* pattern


(d) pants with large *p4* pattern


(e) pants with medium *p4* pattern


(f) pants with small *p4* pattern


(g) pants with large *p4g* pattern


(h) pants with medium *p4g* pattern

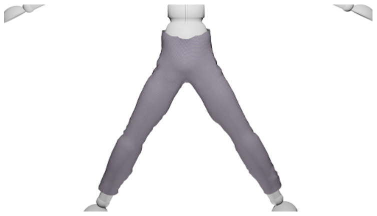

(i) pants with small *p4g* pattern

Figure 5.11: Regular parameterization with shape optimization for pants

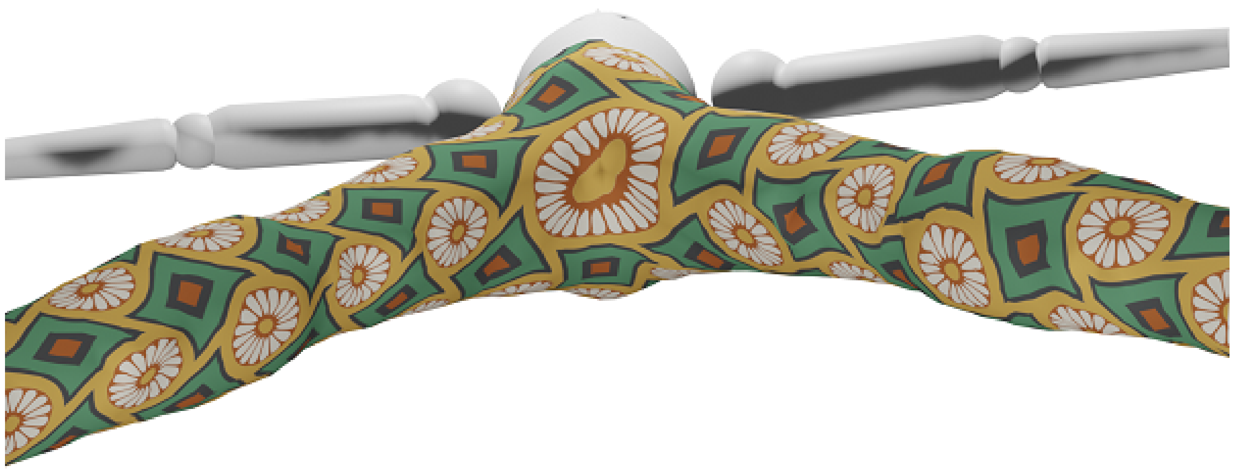


Figure 5.12: Bottom view of pants mapped with $p4$ pattern with regular parameterization

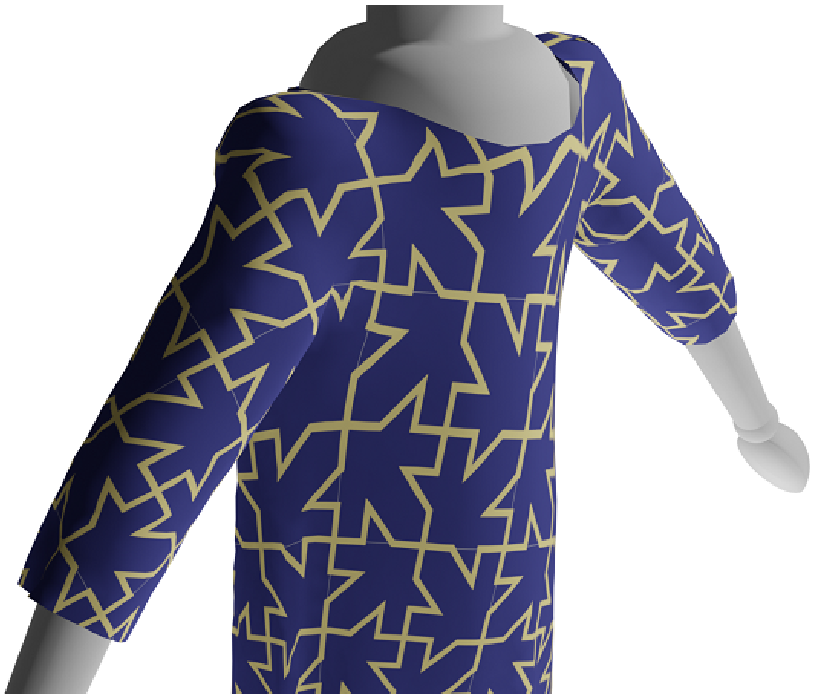


Figure 5.13: Close up rendering of shirt mapped with *p4g* pattern with regular parameterization

Chapter 6

Quadrangulation with Cross Fields

In Chapter 4, we explored the use of Delaunay triangulation as a means of applying textures with sixfold rotations to clothing. We subdivided each fabric piece into approximately equilateral triangles, into which we can map a unit taken from a wallpaper pattern. In this chapter, we propose a similar idea based on quadrangulation. If we can subdivide fabric pieces into approximately square regions, then we can map a unit taken from a pattern with fourfold rotations into each region. This process is more complex than Delaunay triangulation. We adapt ideas from the work of Palacios and Zhang [20] to construct cross fields within each fabric piece, from which we create a quadrangulation. We must also assume that the quads we create preserve the continuity of the texture across seams.

6.1 Cross field calculation

We use the algorithm of Palacios and Zhang [20] to compute the cross field. Because we use the finite element method, the first step as always is to triangulate the shapes so that we can use the triangulation as finite elements. Recall that a cross field is a function of space whose value at each point is a cross, which consists of four directions, and the angle difference between two consecutive direction is 90 degrees. We need to ensure that the cross fields generated on the individual pieces of clothing are continuous across seams when the pieces are sewn together. One straightforward way to do this is to orient crosses on the shapes' boundaries in such a way that two arms of the cross align with the tangent of the boundary at the cross's location. The only exception would be places where a set of seams meet in a closed loop around a point, yielding a location in the final garment where discrete curvature is potentially non-zero. At these singularities it may be impossible to choose a cross orientation that is continuous with the surrounding field. Instead, we set the cross field to zero at these points.

After deciding the boundary values, we use harmonic mapping to calculate the representation field in the interior of each shape. Recall that a representation field has the form

$$\begin{pmatrix} R\cos(4\theta) \\ R\sin(4\theta) \end{pmatrix}$$

Let us define scalar fields $F = R\cos(4\theta)$ and $G = R\sin(4\theta)$ for the representation field. Beaufort et al. state that the magnitude of the representation field will decrease quickly as one moves away from the boundary [1]. But for the fabric patterns that we experimented with, the input is small enough not to cause such instabilities.

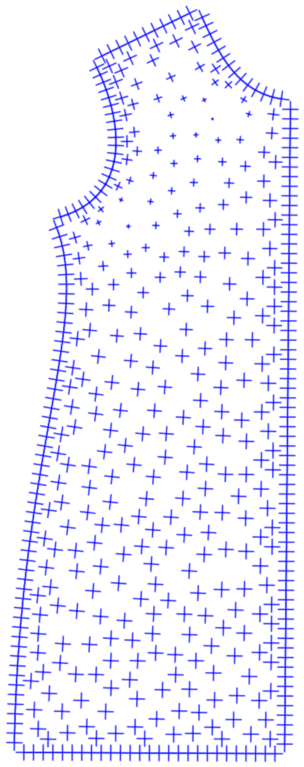


Figure 6.1: Cross field on a fabric piece
The orientation of each cross varies smoothly across the shape. The vanishing size of the cross indicates the presence of singularities. Note that crosses are oriented with tangents at the boundary of the shape.

6.2 Singularity and Separatrix Direction

A singularity point (x, y) has the representation vector $(F(x, y), G(x, y)) = (0, 0)$. We detect such singularities by calculating the barycentric coordinates of potential singularity points in each triangle. Given a triangle in the fabric piece bounded by vertices v_1, v_2, and v_3, let $(\alpha, \beta, 1 - \alpha - \beta)$ be the barycentric coordinates of a potential singularity within a given triangle. Then we can solve Equation 6.1 for (α, β).

$$\begin{cases} \alpha F(v_1) + \beta F(v_2) + (1 - \alpha - \beta)F(v_3) = 0 \\ \alpha G(v_1) + \beta G(v_2) + (1 - \alpha - \beta)G(v_3) = 0 \end{cases} \tag{6.1}$$

If both barycentric coordinate components are within $[0, 1]$, Then we can identify the singularity point within this triangle. We omit all singularities on the corners since they are already known.

For each singularity p_0, there will be a set of separatrices that leave in evenly spaced directions. We need to find those directions so that we can start tracing the separatrices. We calculate the Jacobian of the singularity, $DV(p_0)$. For a triangular finite element, there is a shape function associated with each vertex. A shape function $N_i(x, y)$ determines the influence of $F(v_i)$ at point (x, y). The output of a shape function is a scalar. Let $v_i = (x_i, y_i)$. We have:

$$N_i(x, y) = \frac{1}{2\Delta}(a_i + b_i x + c_i y), i = 1, 2, 3 \tag{6.2}$$

where

$$\begin{aligned} a_1 &= x_2y_3 - x_3y_2, b_1 = y_2 - y_3, c_1 = x_3 - x_2 \\ a_2 &= x_3y_1 - x_1y_3, b_2 = y_3 - y_1, c_2 = x_1 - x_3 \\ a_3 &= x_1y_2 - x_2y_1, b_3 = y_1 - y_2, c_3 = x_2 - x_1 \end{aligned} \tag{6.3}$$

and $\Delta = (x_1b_1 + x_2b_2 + x_3b_3)/2$ is the area of the triangle.

To interpolate the discrete values of F stored at the vertices of the mesh for this fabric piece, we use:

$$F(x, y) = N_1(x, y)F(v_1) + N_2(x, y)F(v_2) + N_3(x, y)F(v_3) \tag{6.4}$$

Note that the gradient of F is constant within a triangle.

$$\frac{\partial F}{\partial x}(p_0) = b_1 F(v_1) + b_2 F(v_2) + b_3 F(v_3) \tag{6.5}$$

$$\frac{\partial F}{\partial y}(p_0) = c_1 F(v_1) + c_2 F(v_2) + c_3 F(v_3) \tag{6.6}$$

The same equations apply for G.

Once we get the gradients of F and G at a singularity point, we have the linearization and can solve the following equation introduced in Chapter 2 for the separatrices' directions:

$$\frac{\sin(4\theta)}{\cos(4\theta)} = \frac{c\cos(\theta) + d\sin(\theta)}{a\cos(\theta) + b\sin(\theta)} \tag{6.7}$$

We also need to ensure that $\sin(4\theta)$ and $c\cos(\theta) + d\sin(\theta)$ have the same sign. Recall that for a cross field, there can be at most five roots for this equation. We use numerical methods to approximate the roots.

6.3 Streamline tracing

To get the separatrices, we trace the streamlines from the singularities. We used Heun's method with a step size that is much smaller than the triangle element's side lengths to do such tracing. The procedure can be summarized with the pseudocode shown in Algorithm 1. This tracing process will produce a collection of piecewise linear paths within each fabric piece, approximating the separatrices. Every path will begin at a singularity and end at another singularity or on the boundary of the piece, since its continuation is in another shape.

6.3.1 Aligning separatrices across shapes

Theoretically, the consistent boundary conditions should give us separatrices that meet at shape boundaries. However, factors such as inaccurate input shapes where seam segments are of different lengths, and the numerical errors in streamline tracing, can cause misalignment. To fix this problem, we manually snap connecting separatrices' endpoints together.

Algorithm 1 StreamLineTracing

```
1: procedure getArmAngle(repV, srcAngle)
2:     repAngle ← arctan(repV.y, repV.x)
3:     crossAngle ← repAngle / 4
4:     angleDiff ← $\infty$
5:     for $i \in \{0, 1, 2, 3\}$ do
6:         $angle \leftarrow crossAngle +$ iPI/2
7:         if $|angle - srcAngle| < angleDiff$ then
8:             angleDiff ← $|angle - srcAngle|$
9:             closestAngle ← $[crossAngle + i$PI/2]
       return closestAngle
10: procedure getArmV(v, Face, srcAngle)
11:     coords ← getBarycentricCoord(v, face)
12:     repV ← interpolate(F,G,face, coords)
13:     armAngle ← getArmAngle(repV, angle)
14:     armV ← norm(repV) * (cos(armAngle), sin(armAngle))
15: procedure Trace(singularityLoc, singularityAngle)
16:     streamLine ← []
17:     v ← singularityLoc
18:     face ← findFace(v)
19:     armAngle ← singularityAngle
20:     while true do
21:         armV ← getArmV(v, face, armAngle)
22:         vT ← v + dt * armV
23:         armVt ← getArmV(vT, face, armAngle)
24:         armVavg ← (armV + armVt) / 2
25:         armAngle ← arcTan2(armVavg.y, armVavg.x)
26:         v ← v + dt * armVavg
27:         face ← findFace(v)
28:         if face is None then
29:             Create boundary node
```

Let C_1 and C_2 be a pair of boundary segments that will be sewn together. In general, some number $k \geq 0$ of separatrices will end at both C_1 and C_2. On each curve, we sort these endpoints based on their positions as a fraction of the curve's arclength. We then identify corresponding pairs of endpoints (enumerating C_2's endpoints in reverse order), and snap the pair to the average of their respective arclength fractions, thereby ensuring continuity of the separatrices across the seam. After adjusting the endpoints, we use the displacement of the endpoints to interpolate the displacement of the vertices on the streamline quadratically. Let e be the endpoint of a separatrix that is on the boundary of the shape. Let e's displacement be d. Let p be a point on a separatrix whose distance to e is a fraction t of the arclength of that separatrix. The displacement at p is then $(1-t)^2d$. Figure 6.2 shows a visualization of the alignment step for two separatrices. steps.

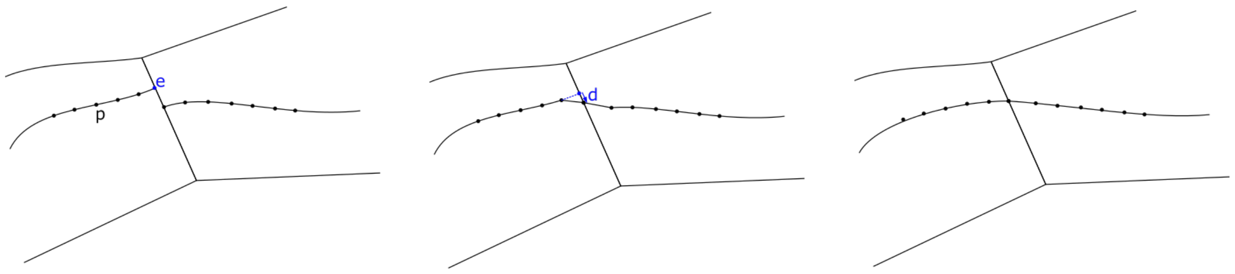


Figure 6.2: Stream line aligning steps
First the pairs of endpoints that should match are identified, and their fraction along the boundary segments are calculated. Second, the endpoints are snapped together. Last, the points on the separatrices are translated an interpolated distance

6.4 Getting the patches

After computing the separatrices, we can subdivide each fabric piece into patches. We always get quadrilateral patches from the separatrices of cross fields. Since separatrices intersect each other and the shapes' boundaries, the first step is to cut the shapes' boundaries into segments. In the previous section, we omitted the default singularities at the corners of the shapes. For a point in the vicinity of a singularity, if an arm of the cross coincides with the direction from the singularity to that point, the point must be on a separatrix. Therefore we know that the separatrices emanating from the singularities at shapes' corners follow the shapes boundaries, this will make all shape boundaries separatrices. Previously, we traced separatrices out from internal singularities, and recorded the locations on the shape's boundaries where these separatrices end. We refer to these points as *boundary nodes*. We now traverse the boundary of the shape beginning at any boundary node. Every time we encounter a corner of the shape or another boundary node, we store the segment we have traversed as a patch boundary. We continue this traversal until we return to the starting node.

Separatrices might cross inside a shape, and so we must also compute all intersections between separatrices as part of the subdivision into patches. We use a simple divide-and-conquer heuristic to find intersections. Given two separatrices, we split them into halves, and look for intersections between the line segments joining the endpoints of the halves. If we find any intersections, we continue recursively on those halves until the locations of the intersections have been fully determined. It is also possible to compute these intersections using a library for planar arrangements, such as the one found in CGAL (cgal.org).

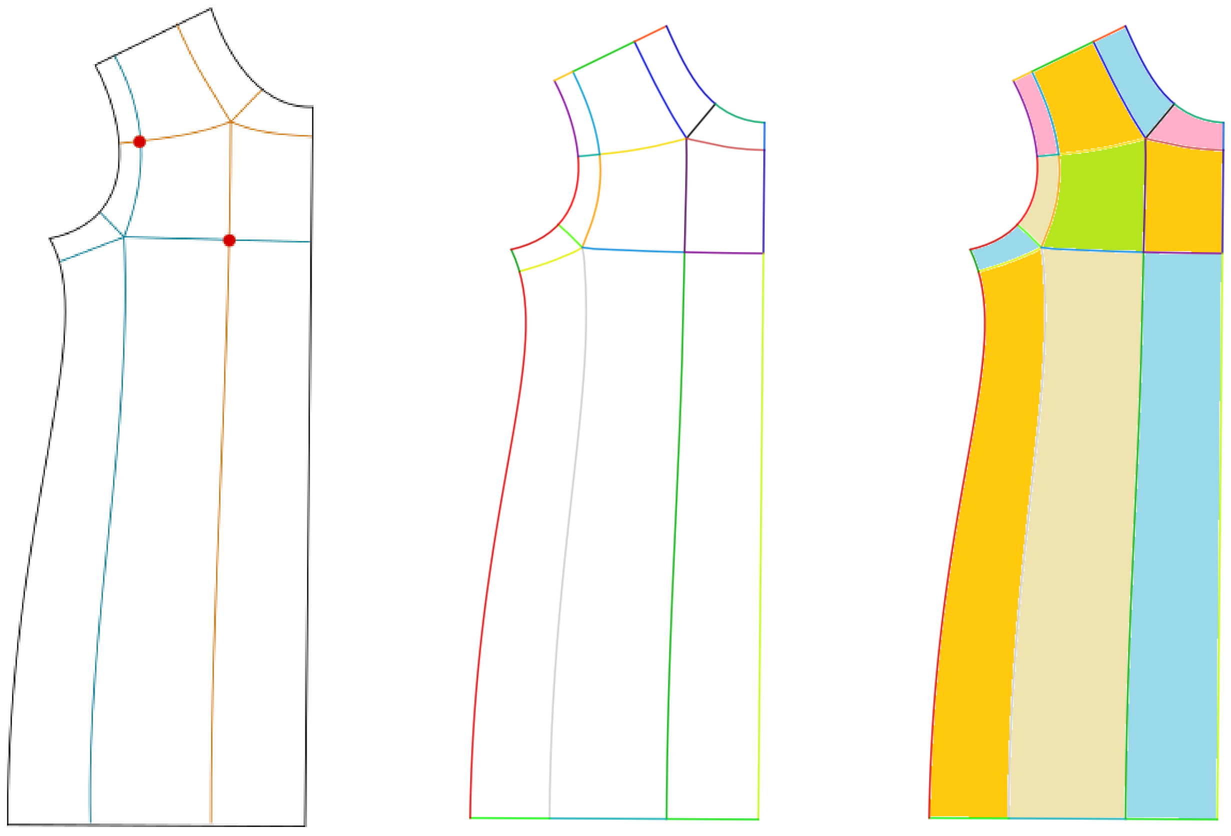

(a) Separatrices (b) Patch boundaries (c) Patches

Figure 6.3: Steps to getting the patches
(a) shows the separatrices emenating from singularities. the intersections are marked with a red dot. (b) shows the patch boundaries identified by colors. (c) shows the final patches.

We now construct a half-edge data structure to represent the subdivision of the fabric piece into patches. Each vertex of the subdivision is a corner of the shape, a singularity, a point where a separatrix ends on the shape boundary, or a point in the shape's interior where two separatrices intersect. Vertices are connected to each other by piecewise linear paths taken from segments of separatrices or the shape's boundary. We can find the faces of the subdivision by walking along edges of the subdivision; every time we arrive at a vertex along an edge, we leave by the next edge in a counterclockwise enumeration of the edges around that vertex. Figure 6.3 shows the steps towards getting the patches.

6.5 Parameterizing the patches

The subdivision produced above will consist in general of curvilinear quadrilateral patches. For the purposes of covering these quadrilaterals with a texture with fourfold rotations, we wish to subdivide them into shapes that approximate squares. We can then use standard texture mapping to apply a tile from the original texture to each square. The process of subdividing each patch into smaller quadrilaterals is equivalent to parameterizing the patches so that each patch is a rectangle with width and length that are integers in texture coordinates.

Since the two opposite boundaries of a patch need to be divided into the same number of segments, we can group all the boundaries that have the same length in uv coordinates. We do this by first iterating over all patches in the same shape, grouping the opposite boundaries into the same set, and joining the set with the group that it intersects with. Let us call the final sets *line groups*. Figure 6.4 shows the line groups within a shape. Next we join line groups across different shapes. If patch boundary P_1 is part of shape boundary C_1, patch boundary P_2 is part of shape boundary C_2, and C_1 connects with C_2, then the line group that P_1 belongs to and the line group that P_2 belongs to are joined into one set. The length in uv coordinates of the patch boundaries in the same line group must be the same integer. We have many options when choosing a length for a line group. We can set the length as the maximum or average of lengths of patch boundaries in the group divided by a user defined length parameter. Figure 6.5 shows a parameterized shape.

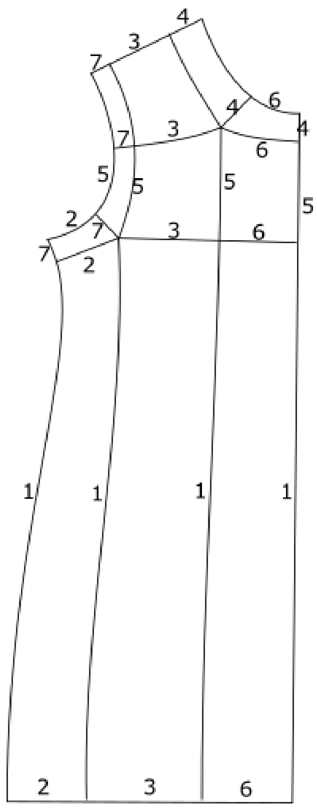


Figure 6.4: Line group
The patch boundaries are annotated with their line group indices.

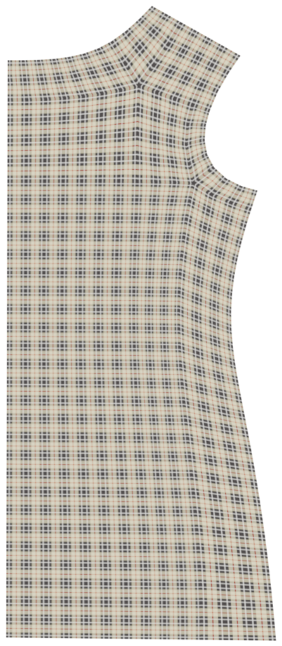
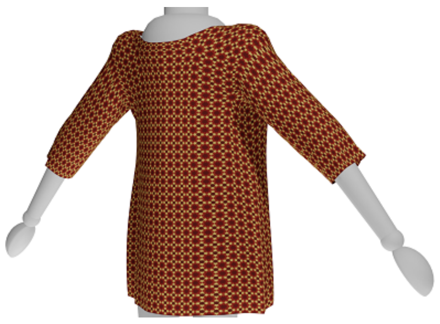


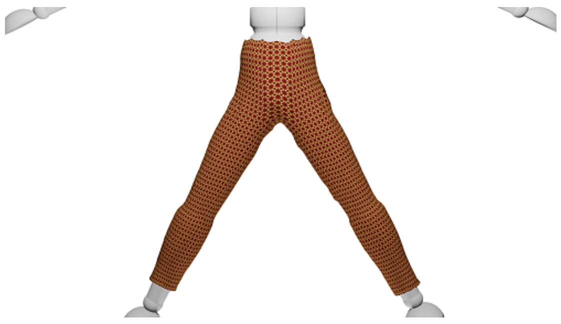
Figure 6.5: A parameterized piece of fabric

6.6 Results

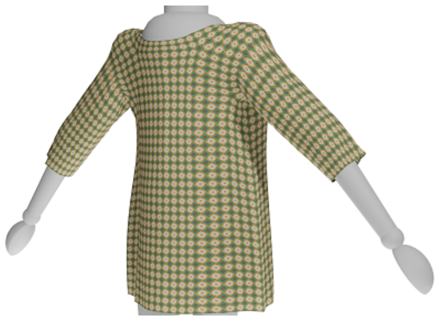
We applied the quadrangulation procedure on the shirt and pants model. Due to the size constraints on the quads, we cannot apply the texture more coarsely. It is possible to subdivide the quads, which would quadruple the number of copies of the texture tile on the clothing. These limitations will be further discussed in Section 7.1. Figure 6.6 shows the results of garments mapped with pattern of fourfold rotational symmetry after cross field guided quadrangulation. Figure 6.7 shows a close up rendering of the garments.



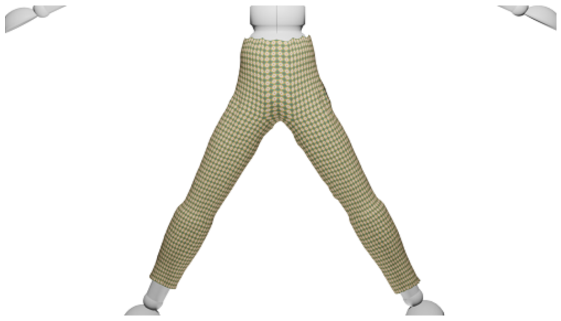
(a) shirt with p4m pattern



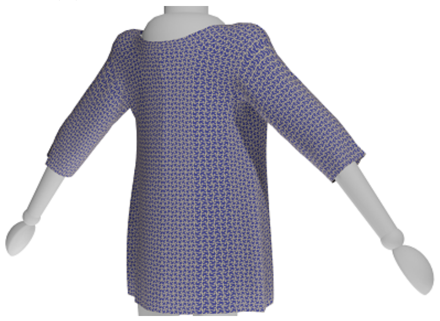
(b) pants with p4m pattern



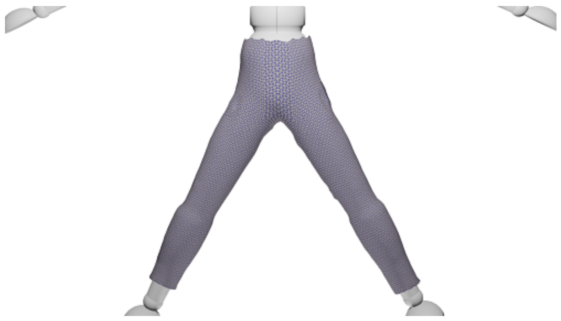
(c) shirt with p4 pattern



(d) pants with p4m pattern

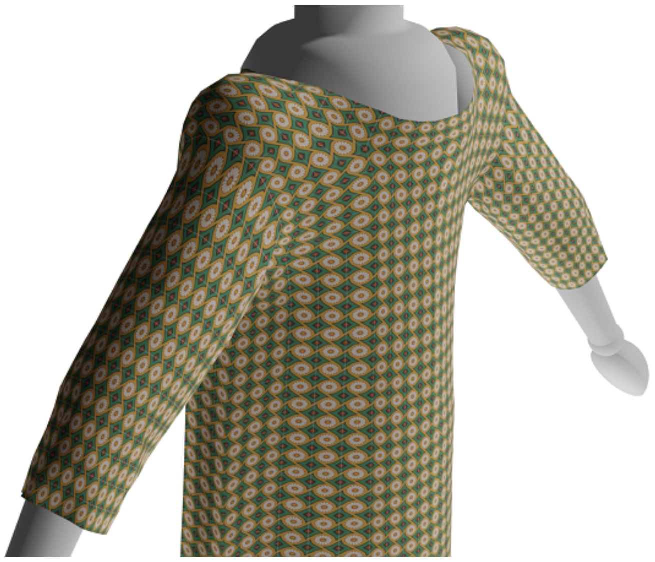


(e) shirt with p4g pattern

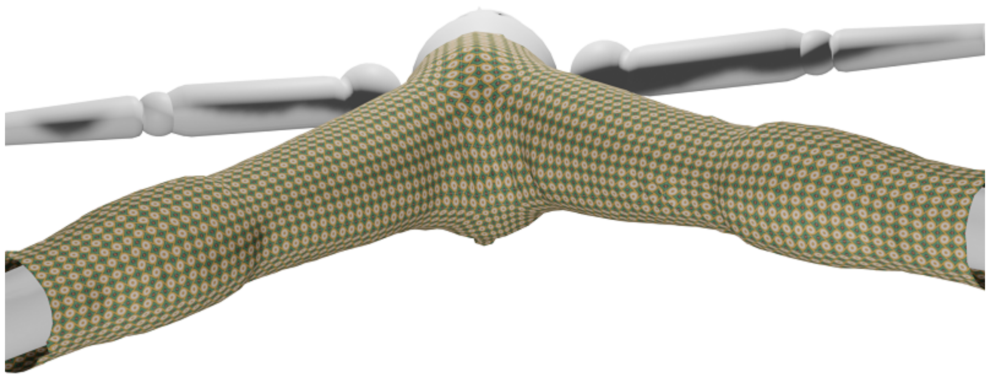


(f) pants with p4g pattern

Figure 6.6: Mapping p4 patterns with cross field assisted quadrangulation



(a) shirt with p4 pattern



(b) pants with p4 pattern

Figure 6.7: Close up rendering of garments mapped with p4 patterns with cross-field assisted quadrangulation

Chapter 7

Conclusion

In this thesis we reviewed surface parameterization and cross field design as tools to map textures seamlessly onto clothing. We explored the topic of seamless clothing textures from two directions. First, we subdivided pieces of fabric into small units into which texture tiles can be mapped. Second, we optimized the positions and shapes of fabric pieces in texture space, inducing a warp of the texture onto the original fabric pieces. We reviewed previous work about optimizing the placement of patterns, and the works related to surface quadrangulation, allowing periodic textures with fourfold rotations to be mapped seamlessly onto garments.

For textures with sixfold rotations (belonging to symmetry groups $p6$ or $p6m$), we used Delaunay triangulation to subdivide fabric pieces into triangular regions that support tiles from the texture. For textures with fourfold rotations (belonging to symmetry groups $p4$, $p4g$, or $p4m$), we subdivided fabric pieces into quadrilaterals using an algorithm guided by a smooth cross field, and mapped tiles from the original texture into these quadrilaterals. We also experimented with a variation of work by Wolff and Sorkine [31], in which we optimized the shapes of fabric pieces in texture space, and demonstrated its effectiveness on textures with fourfold rotations.

We demonstrated all of our techniques using sewing patterns for a shirt and a pair of pants. In all cases, the mapping of textures onto the pieces of fabric was carried out entirely based on the two-dimensional geometry of the sewing pattern. We visualized our results by virtually assembling the fabric into a full garment and simulating the drape of that garment on a mannequin model in Blender.

7.1 Limitations and Future Work

Triangulation produces seamless results for textures with sixfold rotations. However, every seam is a connected sequence of triangle edges, and when textured these long runs of aligned edges stand out against the overall irregularity of the triangulation. We might further suppress the visibility of the seams by allowing edges of the triangulation to cross seams. This change would require a new Delaunay triangulation algorithm that treats seams as "permeable", measuring distance continuously across them. Another limitation of our approach is that because we couple the triangulation used for texturing with the triangulation used for the 3D mesh representation of the garment, we are limited to a relatively narrow range of triangle sizes that trades off between the scale of the texture and the fidelity of the simulation. It would be straightforward to decouple these triangulations, or even just to subdivide the Delaunay triangles to produce a finer mesh for simulation. Section 4.2.2 accomplishes this goal to a degree, by subdividing individual triangular regions along the boundary and harmonically mapping texture tiles into these regions. More work is needed to ensure that these fine-scale triangles enforce continuity across seams.

For regular parameterization with shape optimization, we used two passes to optimize the shapes in texture space. The first pass optimizes the orientation of each shape, and the second pass fixes the endpoints of boundary segments that make up seams. It would be interesting to explore options that optimize the shapes in a single pass. Better descriptions of the shapes might be needed, so that the optimization process can be done in one pass with acceptable time complexity. Another flaw that we noticed is that our constraints do not enforce the shapes' convexity. When an angle that was convex became concave in texture coordinates, we noticed a high amount of the texture in that area. If we could ensure that convex angles do not become concave and vice versa, we might be able to improve our results.

Another area that might be interesting to explore is to apply image processing techniques to the shapes' texture coordinates. We used transfer functions to achieve C^1 continuity of the pattern across shapes. It would be possible to use fixed boundary conditions to map the internal texture coordinates of the shapes, and smooth the gradients of the texture coordinates so that the pattern is C^1 continuous near the seams.

Finally, in our work on shape optimization we experimented only with textures that have fourfold rotations. It is possible to apply the same technique to all N-RoSy patterns as in Wolff and Sorkine's work [31]. We would like to extend our technique to handle textures with sixfold rotations, and possibly experiment with textures that have lower orders of symmetry (i.e., where the highest degrees of rotation are 3, 2, or 1).

In our experimentation with cross fields in Section 6, we noticed that the main limitation is the size of the quads. A quad's length cannot exceed the shortest edge of any patch in any fabric piece of the garment. It may be worth investigating more sophisticated methods for creating and manipulating cross fields, in order to generate fields that support larger quads. For example, it may be possible to edit the topology of a cross field in order to remove unwanted singularities [21], leading to fewer, larger patches. Also, we traced the streamlines explicitly. Parameterization techniques from mixed-integer quadrangulation [5] might allow us to extract separatrices more reliably.

References

[1] Pierre-Alexandre Beaufort, Jonathan Lambrechts, François Henrotte, Christophe Geuzaine, and Jean-François Remacle. Computing cross fields: A PDE approach based on the Ginzburg-Landau theory. *Procedia engineering*, 203:219–231, 2017.

[2] Jan Bender, Matthias Müller, Miguel A Otaduy, Matthias Teschner, and Miles Macklin. A survey on position-based simulation methods in computer graphics. In *Computer graphics forum*, volume 33, pages 228–251. Wiley Online Library, 2014.

[3] Ioana Boier-Martin, Holly Rushmeier, and Jingyi Jin. Parameterization of triangle meshes over quadrilateral domains. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 193–203, 2004.

[4] David Bommes, Bruno Lévy, Nico Pietroni, Enrico Puppo, Claudio Silva, Marco Tarini, and Denis Zorin. Quad-mesh generation and processing: A survey. In *Computer Graphics Forum*, volume 32, pages 51–76. Wiley Online Library, 2013.

[5] David Bommes, Henrik Zimmer, and Leif Kobbelt. Mixed-integer quadrangulation. *ACM Transactions On Graphics (TOG)*, 28(3):1–10, 2009.

[6] Nathan. A Carr, Jared Hoberock, Keenan Crane, and John C. Hart. Rectangular multi-chart geometry images. In *Symposium on geometry processing*, pages 181–190, 2006.

[7] Joel Daniels, Claudio T. Silva, and Elaine Cohen. Semi-regular quadrilateral-only remeshing from simplified base domains. In *Computer Graphics Forum*, volume 28, pages 1427–1435. Wiley Online Library, 2009.

[8] Manfredo P. do Carmo. *Differential geometry of curves and surfaces: revised and updated second edition*. Courier Dover Publications, 2016.

[9] Shen Dong, Peer-Timo Bremer, Michael Garland, Valerio Pascucci, and John C. Hart. Spectral surface quadrangulation. In *ACM SIGGRAPH 2006 Papers*, pages 1057–1066. 2006.

[10] Michael S. Floater and Kai Hormann. Surface parameterization: a tutorial and survey. In *Advances in multiresolution for geometric modelling*, pages 157–186. Springer, 2005.

[11] E. H. (Ernst Hans) Gombrich. *The sense of order : a study in the psychology of decorative art*. Wrightsman lectures ; no. 9. Cornell University Press, Ithaca, N.Y, 1979.

[12] Aaron Hertzmann and Denis Zorin. Illustrating smooth surfaces. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 517–526, 2000.

[13] Kai Hormann, Ulf Labsik, and Günther Greiner. Remeshing triangulated surfaces with optimal parameterizations. *Computer-Aided Design*, 33(11):779–788, 2001.

[14] Craig S. Kaplan. Semiregular patterns on surfaces. In *Proceedings of the 7th International Symposium on Non-Photorealistic Animation and Rendering*, NPAR '09, page 35–39, New York, NY, USA, 2009. Association for Computing Machinery.

[15] Aaron W.F. Lee, Wim Sweldens, Peter Schröder, Lawrence Cowsar, and David Dobkin. MAPS: Multiresolution adaptive parameterization of surfaces. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 95–104, 1998.

[16] Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. Least squares conformal maps for automatic texture atlas generation. *ACM transactions on graphics (TOG)*, 21(3):362–371, 2002.

[17] Barry Merriman, James K. Bence, and Stanley J. Osher. Motion of multiple junctions: A level set approach. *Journal of computational physics*, 112(2):334–363, 1994.

[18] Scott A. Mitchell, Mohammed A. Mohammed, Ahmed H. Mahmoud, and Mohamed S. Ebeida. Delaunay quadrangulation by two-coloring vertices. *Procedia Engineering*, 82:364–376, 2014.

[19] Steven J. Owen, Matthew L. Staten, Scott A. Canann, and Sunil Saigal. Q-morph: an indirect approach to advancing front quad meshing. *International journal for numerical methods in engineering*, 44(9):1317–1340, 1999.

[20] Jonathan Palacios and Eugene Zhang. Rotational symmetry field design on surfaces. *ACM Transactions on Graphics (TOG)*, 26(3):55–es, 2007.

[21] Nicolas Ray, Bruno Vallet, Laurent Alonso, and Bruno Levy. Geometry-aware direction field processing. *ACM Transactions on Graphics (TOG)*, 29(1):1–11, 2009.

[22] J-F Remacle, Jonathan Lambrechts, Bruno Seny, Emilie Marchandise, Amaury Johnen, and C. Geuzainet. Blossom-quad: A non-uniform quadrilateral mesh generator using a minimum-cost perfect-matching algorithm. *International journal for numerical methods in engineering*, 89(9):1102–1119, 2012.

[23] Alla Sheffer and Eric de Sturler. Parameterization of faceted surfaces for meshing using angle-based flattening. *Engineering with computers*, 17(3):326–337, 2001.

[24] Jonathan Richard Shewchuk. Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In *Workshop on Applied Computational Geometry*, pages 203–222. Springer, 1996.

[25] Alexander Soifer. *The mathematical coloring book: Mathematics of coloring and the colorful life of its creators*. Springer Science & Business Media, 2008.

[26] Marco Tarini, Kai Hormann, Paolo Cignoni, and Claudio Montani. Polycube-maps. *ACM transactions on graphics (TOG)*, 23(3):853–860, 2004.

[27] Marco Tarini, Nico Pietroni, Paolo Cignoni, Daniele Panozzo, and Enrico Puppo. Practical quad mesh simplification. In *Computer Graphics Forum*, volume 29, pages 407–418. Wiley Online Library, 2010.

[28] Luiz Velho and Denis Zorin. 4–8 subdivision. *Computer Aided Geometric Design*, 18(5):397–427, 2001.

[29] Ryan Viertel and Braxton Osting. An approach to quad meshing based on harmonic cross-valued maps and the Ginzburg–Landau theory. *SIAM Journal on Scientific Computing*, 41(1):A452–A479, 2019.

[30] Katja Wolff, Philipp Herholz, and Olga Sorkine-Hornung. Reflection Symmetry in Textured Sewing Patterns. In Hans-Jörg Schulz, Matthias Teschner, and Michael

Wimmer, editors, *Vision, Modeling and Visualization*. The Eurographics Association, 2019.

[31] Katja Wolff and Olga Sorkine-Hornung. Wallpaper pattern alignment along garment seams. *ACM Trans. Graph.*, 38(4), July 2019.