Quadrilateral Mesh Generation for Computational Knitting

Rahul Mitra
Boston University
Boston, USA
rahulm@bu.edu

ABSTRACT

Computer controlled knitting machines are ubiquitous in the textile industry. However, a primary concern with their usage is the requirement of a low-level understanding of the machine and knitting details, on part of the user. In this course project, we attempt to streamline and simplify the process of computational knitting by presenting a novel construction of the vertices of a quad-dominant *stitch-mesh*. Knoppel et al.'s method of generating stripe patterns on 3D meshes is leveraged to inform the knitting the direction over the model. In particular, we use the intersections of two mutually orthogonal stripe patterns to generate the vertices of our quad-dominant *stitch-mesh*. It is our hope that the generated mesh can seamlessly integrate with extant methods for translating such meshes to machine instructions. We present the vertices of our *stitch-mesh* using three representative models, a simple prism, a uniform sphere and a bent cylinder.

KEYWORDS

computational knitting, stripe patterns, quad-dominant meshing

ACM Reference Format:
Rahul Mitra. 2018. Quadrilateral Mesh Generation for Computational Knitting. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Knitted clothes are ubiquitous in our daily lives. The two primary reasons knitting is favored over other its alternatives are that knitted fabrics easily stretch and also knitting allows for the production of complex 3 dimensional surfaces without seams [1].

However, designing these knitting patterns for shapes with arbitrary geometry is a complex problem. The design process requires a high level of domain expertise and numerous iterations of trial and error.

In this project, we present the first steps in improving the fully automatic method of Naryanan et al. [2] for converting an input 3D mesh to machine-readable knitting instructions. While Naryanan et al. [2] present a quad-dominant re-meshing pipeline, the generated meshes tend to be highly unstructured and lack quality. We use Knoppel et al.'s [3] method for generating stripe-patterns on surfaces to present the vertices of what will be a far more structured row-column *stitch-mesh* [2], an intermediate requirement towards the generation of knitting machine instructions.

Our algorithm takes as input a 3D mesh of genus 0 and 2 boundaries, a user specified knitting direction (realized as starting and ending cycles), and a stitch width. We then generate 2 orthogonal stripe patterns over the surface. Our *stitch-mesh*'s vertex locations are specified by the points of intersections of the two stripe patterns (fig. 1). Significantly, our approach assumes no domain knowledge on part of the user.

While the method implemented in this project was only able to handle topological cylinders ($g = 0, n = 2$), it is our hope that we will be able to generalize the generation of the vertices to arbitrary meshes.

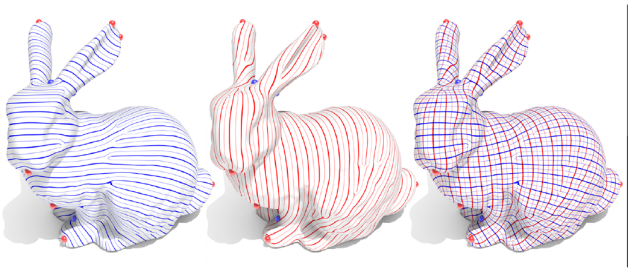


Figure 1: Two orthogonal stripe patterns (left, blue and middle, red) [3]. The points of intersection of these patterns are the vertices for our *stitch-mesh*.

2 BACKGROUND

Before we describe the details of our method, we briefly overview the concepts of computational knitting and stripe parameterization on surfaces.

2.1 Computational Knitting

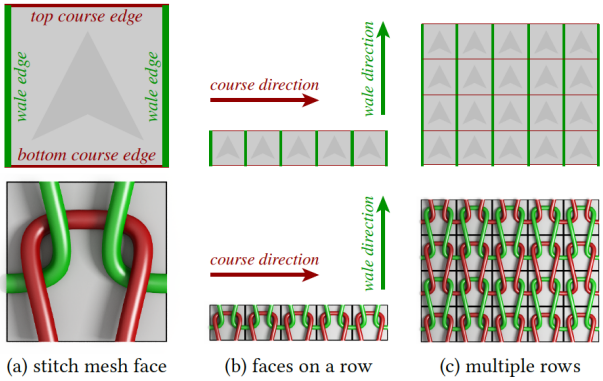
With the advent of automatic industrial knitting machines, there has been a growing interest in designing computational tools that can streamline their usage. From usability [4] to comfort [5] considerations, the field of computational knitting has received a plethora of varied research interests.

A significant step in the knitting pipeline involves the generation of a *stitch-mesh* [2]. This stitch-mesh is thought of as an abstraction for the yarn level geometry [6] and is highly useful for modelling knit structures. In particular, each face of this quad-dominant mesh corresponds to a knit in the final structure. The faces of this mesh are placed side-by-side forming rows, known as the *course* knitting

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

direction. The rows are stacked one on top the other, forming the *wale* knitting direction. The yarn used for knitting each face enters the face through one *wale* edge, forms the stitch, and then exits the face from the opposite *wale* edge [1]. See fig. 2.



(a) stitch mesh face (b) faces on a row (c) multiple rows

Figure 2: *(a)* A single *stitch-mesh* face, *(b)* Faces arranged along the *course* knitting direction, *(c)* Multiple faces stacked to form the *wale* knitting direction.[1]

Because we are interested in knitting arbitrary geometries, we also need to consider non-trivial 3D structures where the *course* rows are non-uniform. In particular, not all rows can have the same number of stitch-faces. The stitch-faces where the rows end prematurely are known as *short-rows* and their placement within the *stitch-mesh* is highly significant to guarantee knittability. See fig. 4

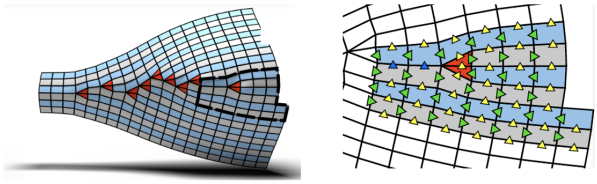


Figure 3: Green and yellow arrows represent the *course* and *wale* directions respectively. For non-trivial topologies, all our rows cannot be uniform length. Short rows (red faces) are introduced to guarantee knittability[5]

Finally, our *stitch-mesh* needs to be helix-free. The presence of helices makes it unsuitable for tracing because arbitrary helices hinder the row-structure construction [2].

In this project, we generate the vertices of the *stitch-mesh* on three models. It should be noted here that helices are born (and killed) at *short-rows*. Unfortunately, our method does not yet account for the existence of such *short-row* faces. As such, when we will eventually trace the vertices to form the row-column *stitch-mesh* of our three models, the graph is gauranteed to be helix-free.

2.2 Stripe Pattern Parameterization

Knoppel et al. use a per-vertex vector field to parameterize stripe patterns on surfaces [3]. Their approach takes a 3D model and a per-vertex vector field as input. Instead of directly specifying a vector at each vertex, we modify their pipeline to specify a 1-form in the space of edges. This 1-form serves as our parameterization coordinate for stripe generation.

To keep stripe-spacing uniform, "branch indices" are introduced, where the stripes bifurcate. These branch indices are realized as the non-integrability of the vector field over particular faces (i.e., non-zero curl). We model these branch faces as the *short-rows* described above (not yet considered by our vertex-generation pipeline).

To very simply summarize Knoppel et al.'s method, each vertex, v_i is assigned a scalar value, α_i. The stripes over a face, $\{v_i, v_j, v_k\}$ that does not contain a *short-row* is given by a linear function of the values, α_i, α_j and α_k. For a face that does contain a *short-row*, the stripes are given by a quadratic function involving the values α_i, α_j and α_k, together with an interpolant to ensure that the function (and by extension, the stripes) are continuous along the boundary of the face.

Our approach allows for changing stripe width which enables us to either increase or decrease the sampling rate of vertices. A lower width implies a higher stripe frequency which means more stripe intersections with an orthogonal stripe pattern and vice-versa. Physically, the stripe width corresponds directly to the user-specified stitch width as described in section 1.

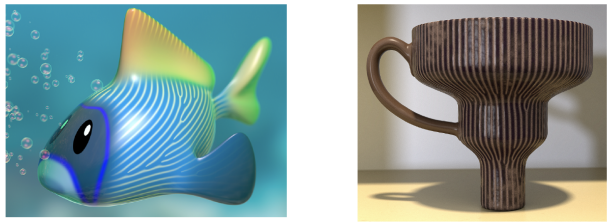


Figure 4: Knoppel et al.'s [3] method inserts stripe bifurcations, "branch indices" for uniform stripe-spacing over arbitrary geometries. We model these points of bifurcations as *short-row* faces in our pipeline towards *stitch-mesh* generation.

3 METHODS

Our pipeline consists of 3 main components. We first ask the user to input the knitting direction (realized as starting and ending cycles) and a stitch width. We then run an edge-based optimization to generate boundary-aligned stripes. We rotate the boundary-aligned stripes by an angle of $\frac{\pi}{2}$ with respect to the outward pointing face normal to get the orthogonal stripe pattern. Finally, we formulate how to find stripe intersections at each face, given the two orthogonal stripe patterns. Again, our method currently only accounts for faces where neither of the stripe patterns (boundary-aligned or orthogonal) have *short-rows*. We present results in section 4.

3.1 User input

Much like the method of Naryanan et al. [2], our approach takes as input a triangulated mesh. The user then selects a set vertices as the starting and ending cycles. We realize these starting cycle vertices as having a scalar value of 0 and the ending cycle vertices as having a scalar value of 1. These correspond to our boundary conditions. Given these boundary conditions we perform a harmonic interpolation (fig. 5) over all the vertices of the mesh.

$$\begin{aligned} \Delta z &= 0 \\ z_\partial &= z_{bc} \end{aligned} \tag{1}$$

where Δ corresponds to the standard vertex-based Laplace operator and z_{bc} are the specified boundary conditions. The iso-values of the harmonic function roughly align with the generated stripes (fig. 7) and are a good representation for the *course* knitting direction.

Further, our method also accepts a user-specified stitch width, which shows up as a period term in our optimization framework (described in more detail in section 3.2) since its value is directly proportional to stripe width. Recall that, as described in section 2.2, this period term influences the sampling rate for generating the vertices of the required *stitch-mesh*. The results of the interpolation on a bunny mesh is shown in fig. 5.

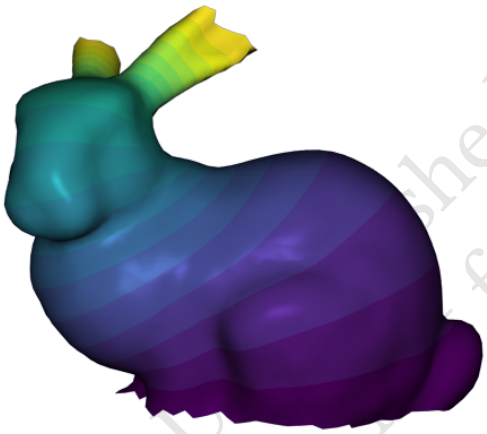


Figure 5: The user inputs the 3D model and then selects the starting cycle (the bunny base) and the ending cycle (bunny ears).

3.2 Edge-based optimization for stripe and orthogonal stripe generation

Now, we look for a 1-form that generates stripe patterns which are amenable for conversion to a row-column *stitch-mesh*. For our purposes, it suffices to specify a 1-form that provides principled placement of *short-rows* and is helix-free. Moreover, we are interested in finding a 1-form that appropriately maintains the correct directionality of stripes (with respect to the user-specified knitting direction) over each individual element and also maintains a good stripe width.

Let z be the result of our harmonic interpolation. Because z is a scalar function defined at each vertex, i.e., z is in $\mathbb{R}^{|\mathbb{V}|}$, we interpret it as a 0-form. The discrete differential of z, $p = d_0 z$ specifies a 1-form in $\mathbb{R}^{|\mathbb{E}|}$. Given p, we now define a gradient at each face, ∇f, $f \in F$. Then, a 1-form that has the properties defined above,

$$g = \frac{1}{2}\left\langle \frac{\nabla f_1}{||\nabla f_1||} + \frac{\nabla f_2}{||\nabla f_2||}, \mathbf{e} \right\rangle \tag{2}$$

where $\mathbf{e}$ is the intrinsic edge vector and f_1 and f_2 are the two faces shared by edge, e. For boundary edges, we consider only the non-boundary face.

The discrete differential of g, $d_1 g$ is a measure of its non-integrability and thus, provides insight into the placement of *short-rows*. We note that using g as a 1-form we are trying to match in the optimization leads to the good placement of *short-rows* for knitting purposes. Note that $d_1 g$ is in $\mathbb{R}^{|F|}$. See fig. 6.

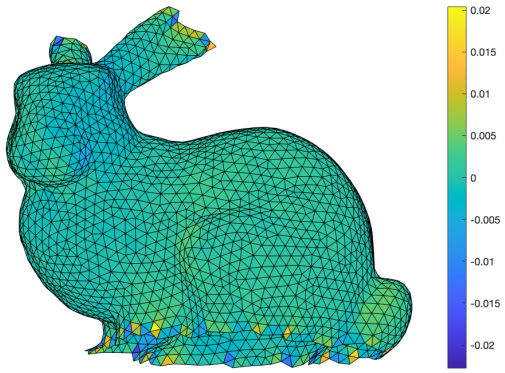


Figure 6: Given the harmonic function, z, a plot of $d_1 g$, as defined by equation (1). The highest regions of non-integrability (faces that are furthest from 0) correspond to the likely placement of *short-rows*.

Let θ be the 1-form we are solving for. We encapsulate the fact that the stripes should align with the user-specified boundary cycles by constraining θ at the boundary edges (i.e., edges where both endpoints are boundary vertices) to 0. Boundary edges are denoted by ∂. We also ask the integral of θ around each face i.e., a row in $d_1\theta$, be some integer multiple of the user-specified stripe width, say w. Physically, where the integral isn't 0 corresponds to *short-row* faces. To encourage the insertion of *short-rows*, we also add on a regularization term, $||d\theta - wk||^2$ to the objective function. Our final edge-based optimization is then,

$$\begin{aligned} &min_{\theta \in \mathbb{R}^{|E|}} ||\theta - g||^2 + ||d_1\theta - wk||^2 \\ &\text{subject to,} \\ &\theta_\partial = 0, \\ &d_1\theta = wk,\ k \in \mathbb{Z} \end{aligned} \tag{3}$$

We note that the regularization term does not change the objective value since any feasible solution will have, $||d\theta - wk||^2 = 0$,

due to the constraints. The result is a mixed-integer optimization with linear constraints for which we use the *gurobi* optimization software [7]. Note that our optimization automatically solves for k, i.e., locations of *short-rows*. However, the experienced user could also manually place *short-rows* by specifying particular k-values at faces, if so desired.

Figure 7: Stripes generated with our edge-based optimization. The stripes represent the *course* knitting direction. They adequately align with the boundaries and *short-rows* (faces where the stripes bifurcate) are appropriately placed. The thickness of the stripes correspond to the stitch width. However, these stripe patterns are not helix-free.

Now, we turn our attention to generating orthogonal stripes. To generate orthogonal stripes we begin by simply rotating the gradient, $\nabla f_i, f_i \in F$ at each face by $\frac{\pi}{2}$ with respect to the outward pointing normal. This is achieved using Rodrigues' formula [8]. Further, we no longer want the stripes to align with the boundaries but in fact ask that stripes are orthogonal to the boundary edges. As such we remove the constraint, $\theta_\partial = 0$ from our optimization. Now, re-running the optimization gives us the required orthogonal stripe patterns. We demonstrate two orthogonal stripe patterns in fig. 8.

3.3 Vertex Intersection Formulation

Knoppel et al.'s method [3] takes, as input, a vector field defined over the mesh and parameterizes stripe patterns according to the input-field. The method does so by finding a coordinate function that increases uniformly as one travels along the input vector field. However, not all vector fields are globally integrable. The faces at which the vector-field cannot be integrated is where the method introduces branch-indices (i.e., *short-rows* according to our terminology). As discussed in the introduction, the stripe parameterization coordinate is specified as a scalar at each vertex. However, instead of directly specifying a value at each vertex, we modify the pipeline to specify a 1-form in the space of edges. This 1-form is the integrated onto the vertices and serves as our parameterization coordinate for strip generation. The method also incorporates the

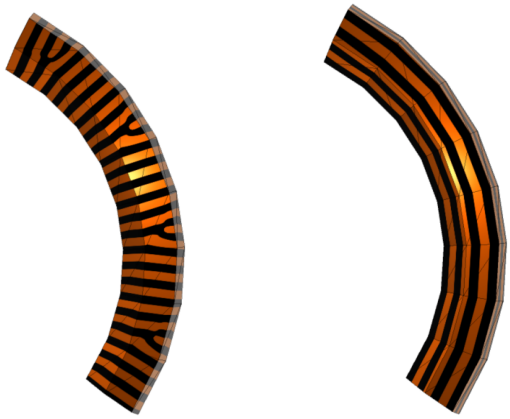


Figure 8: The original stripe pattern from our optimization (left) and the orthogonal stripe pattern by rotating face gradients and re-running the optimization (right). The intersections of the two stripe patterns are the vertices of our *stitch-mesh*.

notion of a "stripe period". One period is defined as the distance covered by a pair of black and orange stripes. Then, the number of stripes through an edge is given by the number of period differences between the scalar values at the edge's two endpoints.

Let us consider stripe intersection points for a single face, $f_{ijk} \in F$, where the subscripts signify vertex indices. Consider the simple case where neither of the stripe patterns (boundary-aligned or orthogonal) have a *short row* at this face. Then, we use barycentric coordinates to solve for the point of intersection of the two stripes. If p_i, p_j and p_k are the 3 vertex locations of f_{ijk} and p is the point of intersection then,

$$p = b_i p_i + b_j p_j + b_k p_j \tag{4}$$

where b_i, b_j and b_k are the barycentric coordinates. In the following, we use α for one stripe pattern and β for the other. To recover the barycentric coordinates, we simply solve the linear system,

$$\begin{aligned} b_i\alpha_i + b_j\alpha_j + b_k\alpha_k &= Z_\alpha \\ b_i\beta_i + b_j\beta_j + b_k\beta_k &= Z_\beta \end{aligned} \tag{5}$$

where Z_α and Z_β are iterated for every integer multiple of the period (or $\frac{\text{period}}{2}$ for higher sampling rate), within $[\alpha_{max}, \alpha_{min}]$ and $[\beta_{max}, \beta_{min}]$ in face, f_{ijk}. And, $\alpha_n, \beta_n, n \in [i, j, k]$ are the scalar values at the vertices of face, f_{ijk}. Also, $b_k = 1 - b_i - b_j$ so eqn. 5 is simply a linear system of 2 equations with 2 unknowns. We carry out this linear solve for every face in the mesh where neither of the two stripe patterns exhibit a *short-row*.

Next, we consider the case where f_{ijk} is a *short-row* face i.e., one of the stripe parameterizations exhibits a branch index. We will use α's to denote the scalar values that resulted in the *short-rows*. β's will be used to denote the linear stripes over f_{ijk}. Our method does not handle the case where both stripe parameterizations, given by α and β induce *short-rows* at the same face. [3] uses a quadratic function to generate *short-row* stripes. An interpolant is introduced

to keep the stripes continuous along the boundary. We let t be the stripe period as described earlier. Because [3] uses a different interpolants, depending on the barycentric region of the face, our intersection formulation must also consider different barycentric regions separately. In particular,

$$\text{Case } b_k \leq b_i \text{ and } b_k \leq b_j$$
$$b_i(\alpha_i) + b_j\left(\alpha_j - \frac{t\cdot n}{3}\right) + b_k\left(\alpha_k - \frac{2\cdot t\cdot n}{3}\right) + \frac{t\cdot n}{6}\left(1 + \frac{b_j - b_i}{1-3b_k}\right) = Z_\alpha$$
$$b_i\beta_i + b_j\beta_j + b_k\beta_k = Z_\beta$$

$$\text{Case } b_i \leq b_j \text{ and } b_i \leq b_k$$
$$b_i(\alpha_i) + b_j\left(\alpha_j - \frac{t\cdot n}{3}\right) + b_k\left(\alpha_k - \frac{2\cdot t\cdot n}{3}\right) + \frac{t\cdot n}{6}\left(3 + \frac{b_k - b_j}{1-3b_i}\right) = Z_\alpha$$
$$b_i\beta_i + b_j\beta_j + b_k\beta_k = Z_\beta$$

$$\text{Case } b_j \leq b_k \text{ and } b_j \leq b_i$$
$$b_i(\alpha_i) + b_j\left(\alpha_j - \frac{t\cdot n}{3}\right) + b_k\left(\alpha_k - \frac{2\cdot t\cdot n}{3}\right) + \frac{t\cdot n}{6}\left(5 + \frac{b_i - b_k}{1-3b_k}\right) = Z_\alpha$$
$$b_i\beta_i + b_j\beta_j + b_k\beta_k = Z_\beta \tag{6}$$

where again Z_α and Z_β are iterated for every integer multiple of the period (or $\frac{\text{period}}{2}$), within $[\alpha_{max}, \alpha_{min}]$ and $[\beta_{max}, \beta_{min}]$ in face, f_{ijk}. We solve all three cases in eqn. 6 for every *short-row* face in the mesh. Having done so, we only use the barycentric coordinates that satisfy the respective case conditions to find the points of intersection using eqn. 4.

4 RESULTS

Now, we present the vertices of our quadrilateral mesh on three simple models. First, let's consider the simple prism (fig. 9).

Figure 9: Two orthogonal stripe patterns on the prism model. We are interested in finding the locations of where these stripes intersect.

Using eqn. 5 to solve for stripe intersections over every face gives us the result in fig. 10.

Next, we consider the uniform sphere (fig. 11).

Again, we use eqn. 5 to solve for stripe intersections on every face of this mesh to arrive at fig. 12.

Finally, let's consider the bent cylinder shown in fig. 8 (results shown in fig. 13).

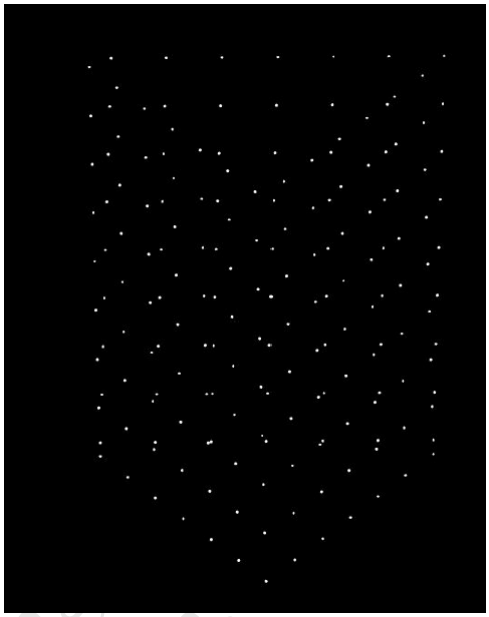


Figure 10: Vertices of the *stitch-mesh* of the prism model. (As a side note, I know the black background is alarming but the vertices are not very discernible if I plot them in black and use the standard white background as I have been for the rest of the paper.)

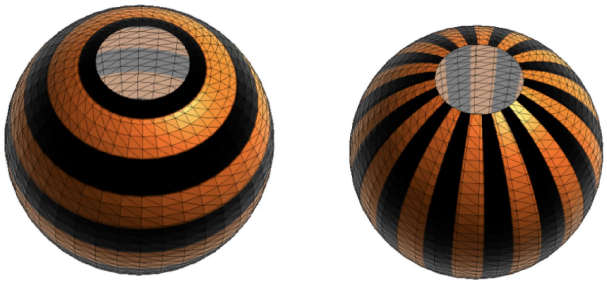


Figure 11: Two orthogonal stripe patterns on the sphere model. We are interested in finding the locations of where these stripes intersect.

Recall that our method also allows for sampling points at different stripe frequencies. For figures 10, 12 and 13, we have sampled vertices at every $\frac{\text{period}}{2}$ i.e., at every black (or correspondingly, orange) stripe. We can change the sampling rate as shown in fig. 14, where the vertices have been sampled at every period i.e., at every black + orange stripe.

5 CONCLUSION AND REFLECTION

We have implemented the first steps in the pipeline towards using stripe patterns for computational knitting. We have presented a novel mixed-integer optimization framework to generate stripe

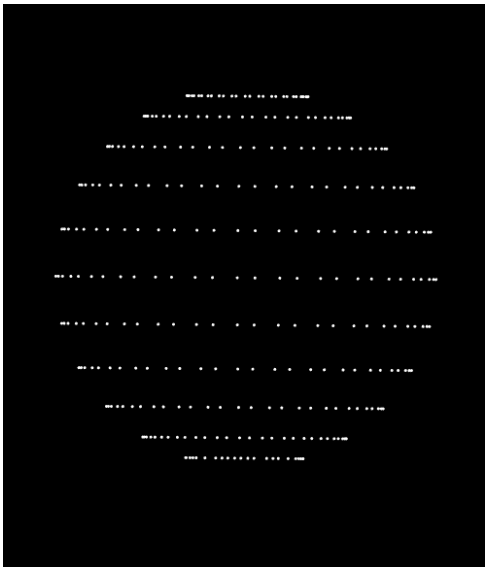


Figure 12: Vertices of the *stitch-mesh* of the sphere model.

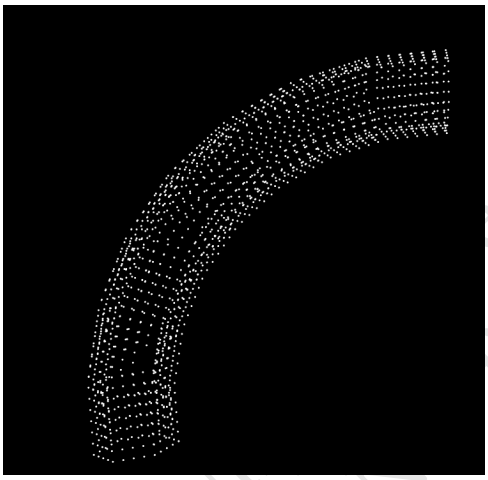


Figure 13: Vertices of the *stitch-mesh* of the bent cylinder model. Here, once again, we have used eqn. 5 to solve for stripe intersections at non-*short-row* faces. The holes in the mesh correspond to *short-row* faces, where our solve has not been adequately implemented.

patterns that are amenable for conversion to a *stitch-mesh*. Our method is flexible in that it accounts for both the automatic and manual insertion of *short-rows*. Further, our method assumes no domain knowledge on behalf of the user and only requires mesh, knitting direction and stitch width as input. Given these parameters, we are able to find the vertices of the required quadrilateral *stitch-mesh*, as shown by the results in section. 4.

In this article, we have steered away from the discussion of helix-removal from the stripe patterns. Going forward, it is an important aspect to consider as arbitrary helices hinder the row-structure construction of our *stitch-mesh*, as also mentioned in section 1.

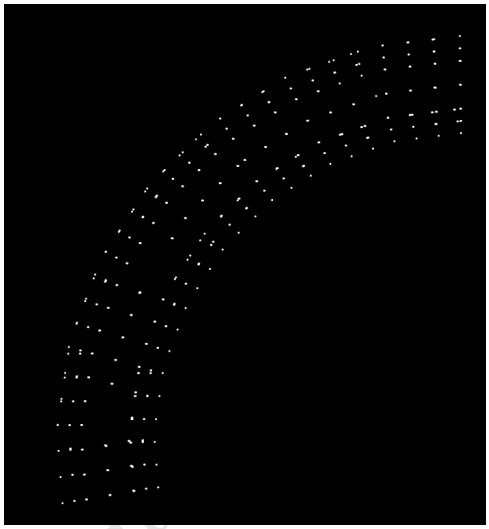


Figure 14: Our method allows for changing vertex sampling rate. Here, vertices have been sampled at every black + orange stripe as opposed to every black stripe as seen in fig. 13.

Further, we also need to consider vertex-generation at *short-row* faces. While the code for the solve of eqn. 6 has been written, it continues to be buggy and preliminary results seem incorrect. We are hopeful that the issues will be resolved soon.

Finally, we also need to address the question of tracing the generated vertices to actually form the rows and columns of our *stitch-mesh*. We have discussed a high-level approach and we're expecting it to be a really fun algorithmic and implementation challenge over the next few weeks.

REFERENCES

[1] Kui Wu, Xifeng Gao, Zachary Ferguson, Daniele Panozzo, and Cem Yuksel. Stitch meshing. *ACM Trans. Graph.*, 37(4), jul 2018.

[2] Vidya Narayanan, Lea Albaugh, Jessica Hodgins, Stelian Coros, and James Mccann. Automatic machine knitting of 3d meshes. *ACM Transactions on Graphics (TOG)*, 37(3):1–15, 2018.

[3] Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. Stripe patterns on surfaces. *ACM Transactions on Graphics (TOG)*, 34(4):1–11, 2015.

[4] Benjamin Jones, Yuxuan Mei, Haisen Zhao, Taylor Gotfrid, Jennifer Mankoff, and Adriana Schulz. Computational design of knit templates. *ACM Transactions on Graphics (TOG)*, 41(2):1–16, 2021.

[5] Zishun Liu, Xingjian Han, Yuchen Zhang, Xiangjia Chen, Yu-Kun Lai, Eugeni L Doubrovski, Emily Whiting, and Charlie CL Wang. Knitting 4d garments with elasticity controlled for body motion. *ACM Transactions on Graphics (TOG)*, 40(4):1–16, 2021.

[6] Cem Yuksel, Jonathan M Kaldor, Doug L James, and Steve Marschner. Stitch meshes for modeling knitted clothing with yarn-level detail. *ACM Transactions on Graphics (TOG)*, 31(4):1–12, 2012.

[7] Bob Bixby. The gurobi optimizer. *Transp. Re-search Part B*, 41(2):159–178, 2007.

[8] Jian S Dai. Euler–rodrigues formula variations, quaternion conjugation and intrinsic connections. *Mechanism and Machine Theory*, 92:144–152, 2015.