TIMING-GUIDED DISTANCE-BASED HAND KNIT MESHING

by

CARMEN PARK

A THESIS

Presented to the Department of Computer Science
in partial fulfillment of the requirements for the degree of
Bachelor of Science.

JUNE 2026

An Abstract of the Thesis of

Carmen Park for the degree of Bachelor of Science
in the Department of Computer Science to be taken June 2026.

Title: Timing-Guided Distance-Based Hand Knit Meshing

Approved: Tao Hou, Ph.D.
Primary Thesis Advisor

This project explores a computational pipeline for converting a simple closed 3D mesh into a hand-knittable, quad-dominant mesh. Building on prior work by Yuksel et al. and McCann et al., we ask how well a simplified timing-guided, distance-based method approximates global quad remeshing and what requirements for a general pipeline it exposes. Our method assigns timing values via a linear Morse function, iteratively selects vertices using a euclidean distance metric, and constructs a stitch index map with a spacing heuristic for increases and decreases. The simplified approach succeeds on convex shapes but does not account for saddles or non-circular boundaries. This reveals the need for handling critical points and arbitrary start cycles as well as a more rigorous remeshing algorithm and additional contouring consideration during Reeb value assignment. The resulting stitch map can be translated into plain-text instructions and knit to validate results; extension to non-convex shapes are left for future work.

Acknowledgements

I would like to thank my primary thesis advisor, Tao Hou, for his continued guidance and support throughout this past term. Despite this topic lying outside his primary area of expertise, he consistently made time to review my work, listen to my ideas, and help direct the project toward its final form.

I am also grateful to Hank Childs for his support as a secondary thesis advisor. His deep knowledge of the research process, encouragement during the early stages of topic selection, his enthusiasm for scientific visualization, and being able to take computer graphics with him in the fall, all have been consistently inspiring and helpful over the past year.

I would additionally like to thank Daniele Panozzo for meeting with me last fall and first introducing me to this idea. Without seeing his contributions to "Stitch Meshing" [1], I would not have embarked on this work. While that paper represents only a small part of his broader body of research in computer graphics, it had a profound impact on me.

Finally, I thank my family and friends for their unwavering support, for listening to my thoughts and frustrations, and for being present throughout this process.

I have greatly enjoyed the past 9 months of research. Although much of my time was spent simply getting up to speed with the relevant literature, I am excited by the prospect of continuing this work in the future.

Contents

Supporting Materials
Git Repository: knit-mesh

List of Figures

Figure 1: Baby's Dress, Sarah Ann Cunliffe, 1851[2]

1 Introduction

Hand knitting has been a prominent part of human history for hundreds of years, with earliest "nålbinding" techniques[3] seen in a pair of socks from Egypt dating to the 3rd to 5th century AD. Earliest examples of double knitting in the Victoria and Albert Museum were made in North Africa around 1100 - 1300 AD during a period of Islamic rule. Knitting grew in popularity in the 1400s in Europe, and in the 1589 the stocking frame was the first knitting machine invented in Great Britain. Hand knitting has been an important part of historical textile industries. It was one of few acceptable ways for AFAB[4] people to make money in earlier societies, and while it is does not

[2]Hand knitted and beaded lace patterns that knitting machines could not replicate became very popular in the 19th century. They are made with extremely fine needles and tiny yarn. It is said this baby's dress had over a million stitches made with at least 6,300 yards of cotton. Cunliffe spent 5 months working 7 hours a day to complete it.

[3]Nålbinding was and is a precursor to knitting in the round where yarn is threaded through the eye of a needle and worked in round through a series of loops.

[4]Assigned female at birth

dominate the mainstream as it used to, knitting remains a relevant skill and means of creation.

Knitting patterns have been passed down through generations, emerging from a blend of trial and error, mathematical accuracy, and an understanding of shape and geometry. While there has been work towards automating or generating knitting patterns digitally, there are still no marketable programs that allow a user to design their own knitting pattern from a desired shape or form.

This difficulty stems partly from the discrete, loop-based nature of a knitted fabric. Unlike sheet materials like metal, paper or a weave, knitted stitches are interlocked loops that exhibit non-linear stretch or anisotropic behavior (with different properties along wales vs courses) and strong dependence on yarn type, tension, and needle size. Designing a knitting pattern for any given 3D shape requires solving the inverse problem of mapping the target geometry onto a grid of knit and purl stitches while strategically using increases, decreases, short rows, and stitch manipulations to achieve the correct curvature, drape, and edge finish. Trial and error remains dominant because even small changes in stitch placement or gauge can drastically alter the final object's size, fit, and stiffness. As a result, generating a reliable, printable knitting pattern from an arbitrary 3D model remains an open research problem that relates to the topology and geometry of the knitted loop network as well as the physical constraints of yarn under load. Building on the foundations of previous works by Yuksel et al. [1] and McCann et al. [3], this thesis introduces a lighter approach in order to gain a stronger understanding of existing work in this field and what is necessary to build a successful pipeline. We ask:

> **How well does a timing-guided, distance-based method for simple geometries approximate global quad remeshing for knittable meshes, and what requirements for a general pipeline does it expose?**

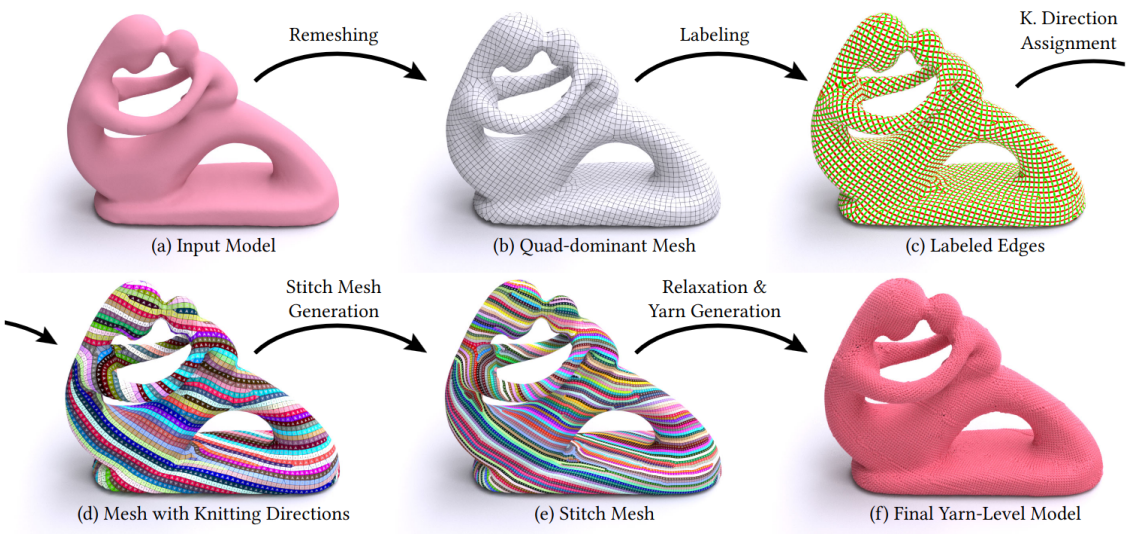


Figure 2: Yuksel et al. Pipeline

1.1 "Stitch Meshing" Yuksel et al. 2018:

"Stitch Meshing" [1] introduces the first fully automatic pipeline to convert an arbitrary 3D shape into a digitally knitted model. It is based on a globally parameterized remeshing method that produces an isotropic quad dominant mesh resulting from the mesh's 2-RoSy field. Knitting directions of each quad are assigned using a set of topological operations and a two-step global optimization over the mesh that minimizes the number of singularities. This mesh can then become a valid stitch mesh, where each quad represents a knit stitch and irregular faces represent knitted increases or decreases. The final yarn geometry is computed using a relaxation process. The resulting mesh is rendered using the physically based renderer Mitsuba, and it can be fabricated using 3D printing.

(a) Pipeline takes an arbitrary input of a 3D model

(b) Converts it to an isotropic quad dominant mesh with only quads and triangles

(c) Each edge of the mesh are labeled as wale or course edge

(d) Knitting directions over the mesh are assigned arrows to show orientation

(e) Stitch mesh is generated

(f) Final stitch mesh ”yarn-level model” is generated from stitch mesh via relaxation.

An important caveat of this paper is that the authors do not address the physical qualities of hand or machine knitting, nor can they guarantee that their pipeline's meshes are knittable. Because of this, my focus centers on their first step: the quad remeshing using 2-RoSy fields. The fact that the resulting mesh is not guaranteed to be knittable means that there may be invalid cycles of stitches, and short rows are not ensured to follow a consistent path. Similarly, the mesh may contain topological features that do not correspond to feasible knitting operations such as increases, decreases, or consistent loop connectivity.

Another feature of this paper that enables a cohesive digital knitting mesh is Yuksel et al.'s work in 2012 on digitally modeling knitted clothing [4]. In their stitch mesh representation, each unit covers one quad, where the wale edge connects stitches and the course edge connects rows. Figure 3 illustrates a single stitch, a row of stitches, and a set of rows.

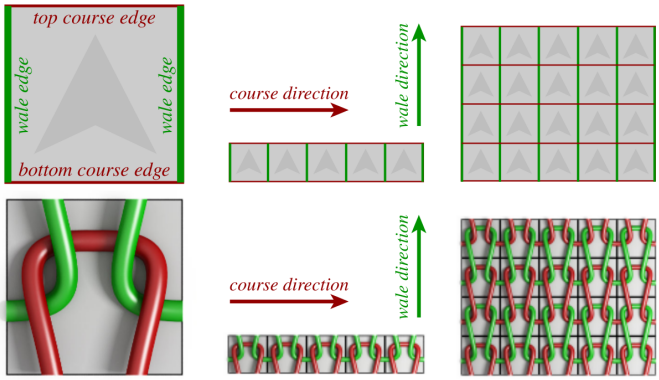


Figure 3: Yuksel et al. Stitch Mesh

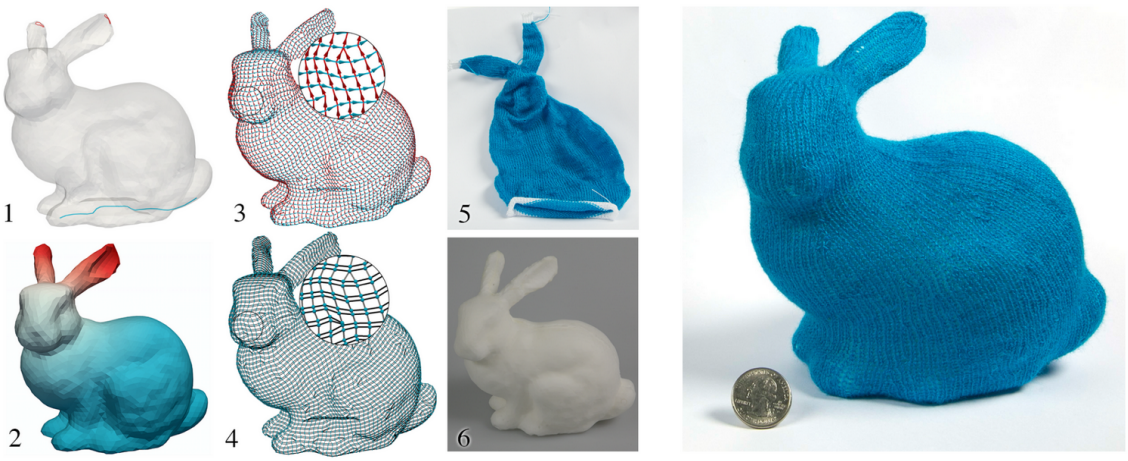


Figure 4: McCann et al. Pipeline

1.2 ”Automatic Machine Knitting of 3D Meshes” McCann et al. 2018

McCann el al. [3] presents the first computational approach to transform a 3D mesh directly into instructions for a computer-controlled knitting machine. Their pipeline takes user defined starting cycles on the mesh which are then:

1. interpolated to define a timing function from the beginning boundary to end;
2. the surface uses a harmonic quad remeshing strategy and create a row and collumn graph that represents a knit structure;
3. processed by a tracing algorithm that incrementally builds a helix free, uniform and regular quad dominant mesh;
4. used to define a graph-based knit structure with rows, columns, and nodes, subject to strict rules that enforce machine knittability;
5. the pattern is machine knit and fitted over a similar 3D model of the target shape.

this paper raises important questions regarding the actual knittability of a 3D object and the technical requirements involved. The knitting graph enforces strict directionality, and the Reeb

timing function directs the flow of knitting across the mesh. Many aspects of this work informed my own approach.

The authors specifically contribute: (a) a succinct criterion for whether a surface can be machine-knit; (b) a field-guided procedure that produces a knit-mesh-compliant graph structure; (c) a tracing algorithm that transforms the graph into knitting-level operations; and (d) a scheduling algorithm that assigns machine-knitting needle locations and operations.

Several elements of this work proved particularly helpful. The knit graph provided a clear abstraction and rules for stitch connectivity. The Reeb graph and Morse function offered a topological foundation for directing the knit path. The tracing algorithm helped with the construction of a similar iterative method used in building my own quad mesh. The re-ordering algorithm was informative but less critical for my purposes, as hand knitting does not require the same level of machine-level instruction detail.

2 Background

Over fall and winter term, research was conducted on these two leading papers by Naranyan et al[3] and Wu et al.[1] and the subsequent papers they cited. I will begin this section by describing knitting with greater technical depth. I similarly will address quad remeshing, and more specifically N-RoSy field generation. Finally I will look more closely at morse functions and how to construct a Reeb graph.

2.1 Knitting

(a) Stockinette (b) Garter Stitch (c) Reverse Stockinette

Figure 5: Three basic ways to make a knitted fabric

2.1.1 A Formal Definition

Knitting can be defined as the act of interlooping yarn with knitting needles to create a textile product. A stitch is defined as a single loop, with a knit or stocknette stitch being pulled forward through the previous loop and a purl or reverse stocknette stitch being pulled backward through the previous loop. Because of this structure, a stitch that is outward-facing on one side of the fabric appears as a purl on the opposite side when flipped over. A row is defined as a series of stitches that are connected to each other horizontally and a row increase is complete once you have looped through every stitch of the previous row.

Knitting is aided by knitting needles, which allow you to loop through the stitches. Knitting with two single-pointed needles creates a fabric with a clear beginning and end, where each row is disconnected from the next, and the resulting product is homeomorphic to a planar surface[5].

Another method is knitting with circular needles (or a set of three or more double-pointed needles); this allows the rows to be closed and connected in a continuous spiral, making a "tube" shape, and enabling you to create three-dimensional objects. Knitting in the round often means

[5]Garder stitch is the result of stocknette knitting with double needles since the piece is alternating knitting right sides (RS) and wrong sides (WS). Garder stitch and reverse stocknette stitch (texturing) will not be addressed in this thesis but are worth mentioning as ways of knitting and also can influence the shape and form of a knitted object.

that you are using stocknette stitch on the right side of the knit object[6].

Crocheting is not the same as knitting. It uses a single needle with a curved end and builds chains of stitches. since it operates at a single unit is much easier to build three-dimensional forms this way as there is no no required row of stitches and individual stitches are chained along the shape. I will not be examining crocheting in this thesis, but future comparisons of both methods could lead to interesting findings.

Similarly, machine knitting and hand knitting, while often producing similar results, have fundamentally different constraints. Industrial machine knitting machines typically use a bed with two rows of hooks to hold stitches. A sliding carraige moves across the bed and pulls yarn through the hooks to form new stitches. There are constraints such as

- Decreases and increases are limited to two stitches at a time.
- Knitting must be continuous; for example you cannot pick up stitches from already completed work which is common with hand knitting.
- cast on and cast off sizes are constrained to maintain machine efficiency.

There are many other differences I have not addressed. While I had hoped to make strides away from a machine knitting implementation, my implementation also stayed within the first two constraints. That being said I would like to look at how hand knitting techniques can be modeled as they break away from the shared capabilities of machine knitting.

I have decided to refine my focus to to 4 particular increase and decrease techniques, a left and right leaning increase, and a left and right leaning decrease. I will mention short rows, but they were not incorporated in my findings.

[6]There is a caveat to this with short rows but this is only important if you are knitting

2.1.2 Casting On and Casting Off

Casting on is the term for making a foundation row of stitches on your needle and is necessary to start any knitting project. Casting off similarly is the final step of binding your stitches so that you can take it off the needles without it unraveling. Going into detail of the cast on and off process is not necessary for this paper, but understanding that casting on and off (both notated as CO) are how a knitting project begins and ends could be helpful to someone unfamiliar with the knitting process.

2.1.3 Increasing

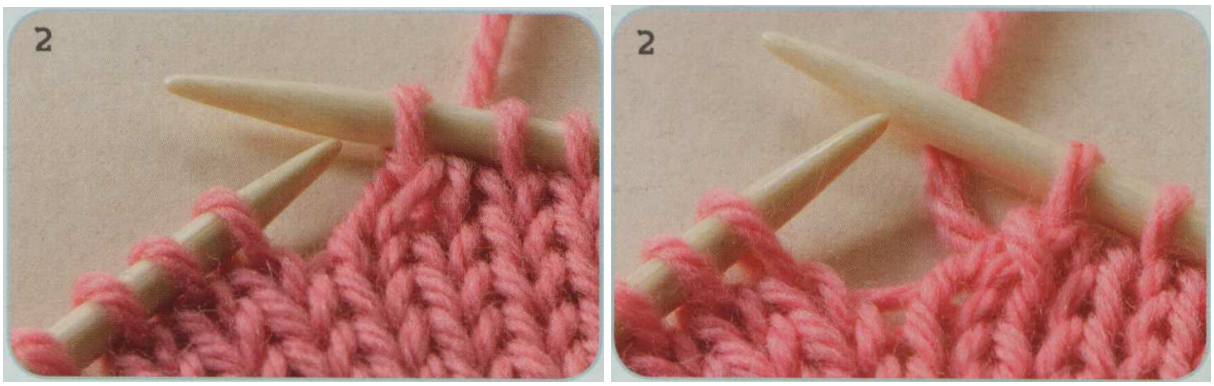
(a) Make One Right (M1R) (b) Make One Left (M1L) (c) M1R-M1L under needle

Figure 6: Make One Increases

The increasing stitch, make one stitch (M1), is the result of creating a loop between the yarn of two existing stitches and knitting it as a new stitch. Make one right (M1R) and make one left (M1L) are the right and left leaning versions of the make one increase respectively. They help make increase lines look more symmetrical, and result in a more polished final product.

2.1.4 Decreasing

Decreasing is the act of looping two stitches into one, leaving one less stitch in the current row than there was in the previous row. By default this folds one stitch underneath the other to

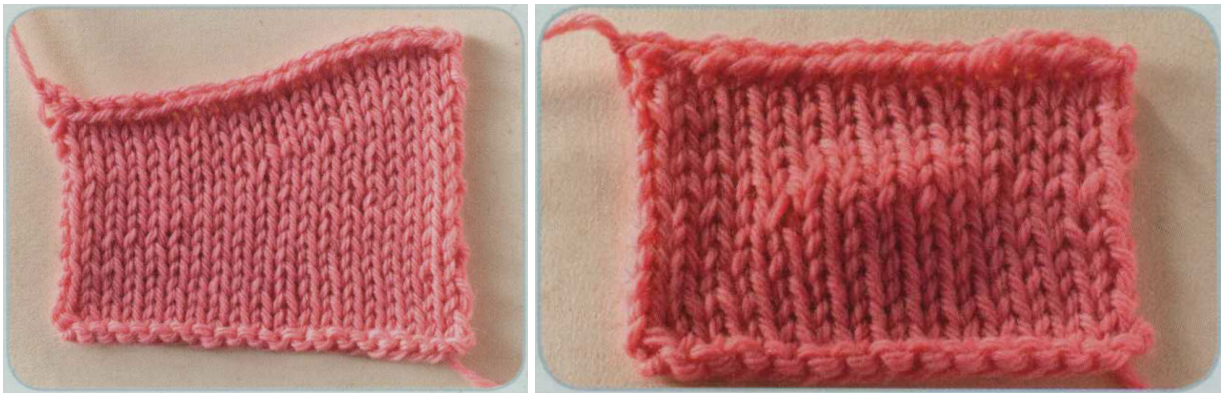


(a) Knit Two Together (K2tog) (b) Slip-slip-knit (SSK)

Figure 7: Right and left leaning decreases

create a sigle stitch. Knit two together (K2tog) is a common decrease technique that places the farthest of the two stitches on top, and it leans to the right. Slip-slip-knit (SSK) has the closest of the two stitches on top and means to the left.

2.1.5 Short Rows



(a) Short rows on one side (b) Short rows in the center

A short row is a knitting and crochet technique where you work partway across a row, turn around, and travel back. Insead of completeing one single row, short rows add two additional stitches[7] to a specific region of a knitted project. This process can be iterated over and over to

[7]One for traversal back, another for the traversal returning to the diverging point.

specifically change the curvature of a localized area of a knit object. Short rows are crucial for creating curves, 3D shapes, and adding length to targeted areas.

Some examples of conventional short row use is creating the heel of a sock, adding bust darts and raising the back in sweaters, making sculptural elements like ruffles, and so on. For hand knitting, wrap and turn or german short rows are necessary methods to avoid holes from making short rows.

2.1.6 Amigurumi

(a) Single arm construction (b) Connecting arms from above (c) Closing body from below

Figure 9: Knitting an amigurumi starfish

The word “amigurumi” literally translates in Japanese to “all encompassing knit”. It is a knit and crochet technique to create plush objects or any variety of shapes and forms. While amigurumi is very popular in the crocheting community, it is also slowly becoming popular in the knitting community. Singh constructs amigurumi patterns in the book “Amigrumi Knits”[5], with 20 patterns that all share similar strategies to the knitted starfish in figure 9.

In the instance of the starfish pattern, Singh first constructs the top of each arm, casting on at the base. She then decreases (alternating SSK, K2tog) so that the stitches come together along one center stitch or seam, and casts off at the point. The bottom half of the arm picks up stitches and purls perpendicular to the top half of the arm, casting off when both parts meet in the middle.

After five of these arms are constructed, she picks up stitches from the base of each arm to connect the starfish. This set of operations is not continuous or smooth, but rather she is patching many (in this case symmetrical) pieces into a closed surface.

while this is not how either papers I have looked at went about writing their knitting algorithms, it was most likely in the back of my mind when I wanted to approach this problem in the first place. It adds many more potential opportunities of construction, and is something to look at in future implementations of this project.

2.1.7 Future Aspirations

Left and right leaning decreases offer a mechanism for shaping regions of changing curvature on a 2-manifold. With careful consideration, the grain of a knit can be intentionally manipulated across an object. This observation makes the project particularly promising, and I hope to eventually develop it into a tool that enables average knitters to design their own patterns.

Additionally, finding how to patching together separate knitted pieces could raise another quad remeshing question, as definitions of simplices and separatrices could shape knit patches so that not only continuous knit projects are represented here.

2.2 Quad Meshing

Bommes et al. [6] describes polygonal meshes as fundamental representations with applications in computer graphics, geometric modeling, simulation, architecture, scientific visualization and other domains. A mesh is the representation of one large shape by many smaller polygonal cells, forming a cell complex.

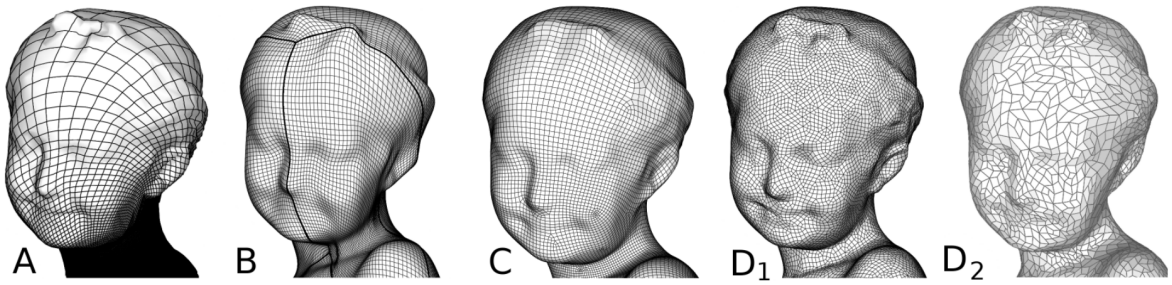


Figure 10: Quad Categories. A: regular; B: semi-regular; C:valence semi-regular; D_1, D_2: unstructured, D_2 more irregular than D_1.

2.2.1 Formal Definition

A two-dimensional polygonal mesh consists of vertices, edges, and facets[8]. The valence of a vertex is the number of its incident edges. The star of a vertex comprises its incident edges and facets; similarly, the star of an edge consists of its incident facets.

Meshes are typically conforming, meaning that any two faces share either a single vertex or a common edge. We assume meshes have a manifold configuration, i.e., there is no meeting at a single point [9]. An internal edge has a star of 2, while an external, or boundary edge has a star of 1. A boundary is composed of cycles of boundary edges. A mesh is watertight if all of its edges are internal.

Discrete functions (scalar, matrix, tensor) on a mesh can be defined by assigning values to vertices or facets. Continuous representations are obtained by interpolation, with bi-linear interpolation as a common method.

A *triangle mesh* is a mesh where all facets are triangles and a *quad mesh* is a mesh where all of ets facets are quadrilaterals. A *quad dominant mesh* has majority quadrilateral facets and a small number of triangular or pentagonal facets.

[8] points in space, ($\mathbb{R}^3$ written as $\mathbb{R}^4$ for rendering purposes); lines connecting said points; polygons bounded by cycles of edges or cycles of points

[9] no "bow tie" configuration

In a quad mesh, an internal vertex is called *regular* if it has a valence of 4, and *irregular* or *extraordinary* otherwise. Since this project addresses closed meshes, every vertex is considered an internal vertex.

There is a natural relation between quad meshes and cross fields. A cross field is the assignment of a pair of orthogonal directions, or fields, to each surface point. It is like a vector field but without magnitudes. This is like an N Rotational symmetry field.

Index theory classifies singularities in vector fields, there is similar work in cross fields. For quad meshes, singularities correspond to extraordinary vertices. *Seperatrices* are integral lines that start at singularities. When all separatricies start and end at singularities this creates natural partitions of a manifold.

2.2.2 Classification and Characteristics

There are several classes of quad meshes that can be seen in figure 10.

(A) A regular quad mesh is globally mapped from a planar tiling to a mesh.

(B) Semi regular quad meshes are when patches of quads are "glued" in accordance to its separatricies. In finite element method (FEM) this is called a multi block grid where shere blocks correspond to patches. This is the most important class of meshes.

(C) There is valence semi-regular where most of its vertices have a valence of 4, but not every valence semi regular mesh can be subdivided or partitioned into smaller patches. There are important algorithmic differences between valence and semi regular.

(D) Quad meshes can also be unstructured which is categorized by a large portion of its vertices being irregular.

2.2.3 Quad Quality

Triangles possess several favorable properties: they are always planar, convex, and support linear interpolation of functions defined at vertices.

Quadrilateral quality is more complex. Planar quads are not always convex, and interpolating attributes on quads requires an extended definition of barycentric coordinates. Rendering is also more difficult, and a quad mesh must remain nearly planar, a property referred to as *planar quad* (PQ). Facet corners are ideally close to 90 degrees, opposite sides should be similar in length, and in a uniform quad mesh, every edge should be approximately equal in length.

Quad orientation and size. The number of irregular vertices is intrinsically related to geometric properties.

1. **Regularity:** *Unstructured*, *valence semi-regular*, *semi-regular*, and *regular* meshes form a continuum of cases with increasing degrees of regularity.

2. **Placement of irregular vertices:** As a rule of thumb, irregular vertices should appear in regions of strong positive or negative Gaussian curvature rather than where resolution changes are required.

Connectivity is an important factor in knitting. For most stitching patterns, a vertex valence of four represents a standard knit. Irregular vertices with valence five or three correspond to increases and decreases. A key aspect of knitting is that these irregularities affect the form and geometry of the knitted object. While most quad meshing works to minimize singularities, knit meshing looks to arrange symmetries around curvature lines. A knit mesh must as well have uniform quad lengths for each facet.

A naive way to convert any polygonal mesh into a quad mesh is to apply topological Cat-

mull–Clark subdivision [7]. Yuksel et al.[1] focuses on N-RoSy field as means of quad generation because it more closely preserves uniform quad facets while considering the shape globally and focusing on primary curvatures.

2.3 N RoSy Fields

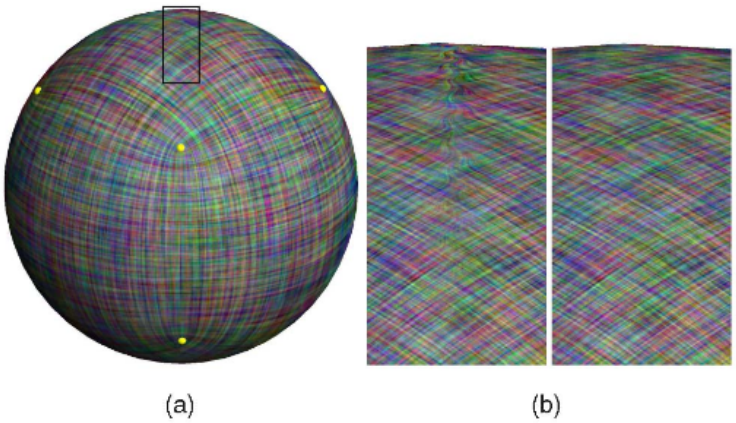


Figure 11: (a) A 4 RoSy field on a sphere forms a cube pattern. (b) A close up of the black box region from (a)[2]

Many objects in computer graphics and geometry processing can be described by *rotational symmetry fields*. N-way rotational symmetry (N-RoSy) fields are proposed as ways to model brush strokes, hatching, principal curvature directions, texture synthesis, and other related effects. N RoSy fields can be considered as a multivalued vector field, where at each point p there exist $n \in \mathbb{N}$ vectors that extend from p at angles $\frac{2\pi}{n}$. Common The singularities and separatricies of a quad mesh can have a direct impact on the quality of the results[8].

Most modern N-RoSy field generation follows 5 steps that are similar to those laid out by Bommes et al's "Mixed-Integer Quadragulation" paper[9].

1. **Define constraints:** This process starts by specifying directions of the field at certain points on a

mesh. Constrsints help guide the fields to align with specific geometric features or curvature.

2. **Represent field as an ”unrolled” vector field:** Each N-RoSy field is mapped to a ”direction” from the angles obtained by the number of rotational symmetries.

3. **Globally optimize for smoothness:** Within the constraints of this representation, solve a large optimization problem over the mesh. The end goal is to find the smoothest possible N-RoSy field.

4. **Identify and adjust singularities:** With a smooth field, majority vertices will be regular with a valence of 4. There will also be points where the field converges and diverges, or singularities. In finding a smooth field, one also can define optimal singularity location and number.

5. **Generate a global parameterization and extract the mesh:** Once these stages have been completed, a globally smooth parameterization is computed so that its iso-lines follow the directions of the quad field. A consistent quad mesh can then be extracted.

2.4 Morse function and Reeb Graphs

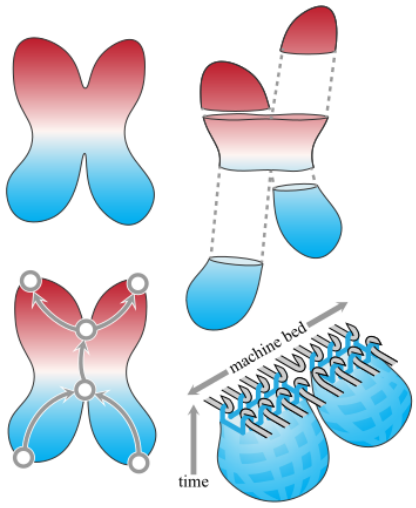


Figure 12: ”An oriented 2D manifold-with-boundary $\mathcal{M}$ is knittable iff there exists a Morse function $f : \mathcal{M} \to [-1, 1]$ such that the reeb graph of f has upward planar embedding.”[3]

Like McCann et al. [3] morse functions and Reeb graphs help answer the question of whether or not an arbitrary 2D manifold $\mathcal{M}$ can be knit continuously. This is the same as asking whether or not the manifold can be described as a set of generalized cylinders from beginning to end. In order for $\mathcal{M}$ to be knittable, it needs at least 2 boundaries, and it must be oriented so that each cycle follows the previous one. They define a timing function $f : \mathcal{M} \to [-1.1]$ that assigns steps from the starting boundary to the end boundary. Each step layers a one dimensional closed curve (row) or curve (short row). They slightly perturb the value of f so that it guarantees lines, circles, or figure 8 shapes, which can all be held in the bed of a knitting machine.

A morse function is defined by Edelsbrunner and Harer[10, 160–174] as a smooth function $f : M \to \mathbb{R}$ on a compact smooth manifold if all critical points are non-degenerate, meaning that the Hessian matrix at each critical point is invertable[10].

A Reeb graph of f is obtained by contracting each connected component of the level sets $f^{-1}(t)$ to a single point. It is the quotient space $M/\sim$ where $x \sim y$ if $f(x) = f(y)$ and x and y belong to the same connected component of $f^{-1}(f(x))$.

On a 2D surface, a morse funtion has 3 types of critical points:

- **Local minimum:** Represented by a source node in the Reeb graph. This is where new contours appear as f increases. In the context of knitting, the local minimum marks casting on.

- **Saddle point:** This is where contours split or merge as you pass over a col[11]

  In the Reeb graph, a saddle is a node of degree 3. Importantly, this is where a contour can split into two (or two merge into one)

[10] A Hessian matrix is a square matrix of all the second-order partial derivatives of a scalar-valued function. A critical point is a specific location on a geometric shape or a data field where the local topology, ie the number of connected pieces, holes or tunnels, changes.

[11] A col is another name for a saddle point, and appears when the gradient is zero in a Morse function but is neither a minimum nor a maximum. A contour is another name for a level set, which is the set of all points on the manifold where the Morse function f takes the same value t

- **Local maximum:** Represented by a sink node. This is where contours disappear as f increases. For knitting, this corresponds to casting off or the end of a piece.

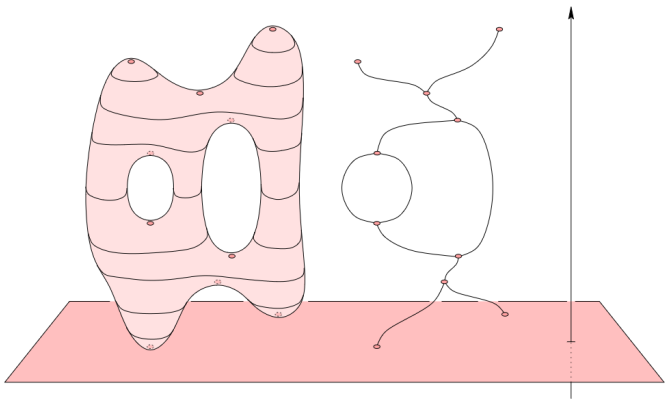


Figure 13: Level sets of the 2-manifold map to points on the real line and components of level sets map to points of the Reeb graph.

3 Method

Given that the majority of this work focused on understanding the existing body of research, implementing a full N-RoSy remeshing algorithm was beyond the scope of this thesis. Instead, a simplified algorithm was developed to demonstrate the core concepts.

The proposed algorithm takes as input a simple closed triangular mesh that is a sphere. It extracts the vertex, edge, and facet data, then assigns a scalar timing value to each vertex based on a user-defined starting point and ending point. The algorithm selects from the existing vertices those that lie within a specified distance threshold, then iteratively builds quads from the start boundary to the end boundary. Finally, it assembles these quads into a knittable stitch mesh and a knitting pattern to hand knit the mesh.

3.1 Setup and Build

The implementation was developed on a Linux system. Cross-platform compatibility has not yet been tested. The build process is managed with CMake.

3.1.1 Dependencies

Several external libraries were considered during development. The OpenGL Development (ogldev) library [11] is used for basic mesh import. GLM and GLAD were also tested, but both were ultimately replaced in favor of Qt. Qt was selected as the primary framework because it provides both a graphical user interface and robust support for the required backend functionality in C and C++.

3.1.2 Mesh Import and Data Structures

A basic mesh import is handled using a reader derived from ogldev [11], which supports loading arbitrary file formats. However, for consistency, the implementation primarily uses the Wavefront OBJ format, which was a familiar and straightforward choice. After importing, the mesh data is parsed and stored in a custom data structure called `knitMesh`. This object stores the vertices directly from the obj file and constructs edges from the facet definitions, providing a unified representation for subsequent processing.

3.1.3 Sourcing Meshes

Test meshes were obtained from several sources. Fertility meshes (OBJ format) were sourced from the Stitch Meshing repository by Yuksel et al. [12]. Additional triangular meshes were exported from Rhino the Autocad software. The Stanford Bunny and Dragon models were obtained from

the Stanford 3D Scanning Repository [13]. The Utah Teapot was sourced from the University of Utah Computer Graphics archive [14].

Several additional test meshes were created using Blender, including a torus and a stretched sphere. Blender was also used for its quad remeshing tool, QuadriFlow, with operations described below.

3.1.4 QuadriFlow

QuadriFlow [15] is a method for converting a triangle mesh into a clean, quad-dominant mesh. It improves upon the widely used Instant Field-Aligned Meshes (IFAM) algorithm by balancing local optimization speed with global solution quality. A common challenge with quad meshing is the correct placement of singularities (vertices where the mesh irregularly converges or diverges), and QuadriFlow addresses this through a three-stage process.

1. **Field initialization:** The algorithm first computes an initial orientation and position field for the quads using a fast local smoothing method (similar to IFAM). This provides a reasonable starting point without expensive global computation.

2. **Singularity reduction via network flow:** The initial field typically contains too many singularities. QuadriFlow reformulates the problem of reducing singularities as a *minimum-cost network flow* problem. By mapping the mesh onto a network, it calculates the adjustments needed to remove singularities with the least "cost", yielding a near-optimal global solution in an efficient way.

3. **Final re-optimization:** With the integer adjustments from the second stage fixed, the continuous variables of the position field are re-optimized to produce the final, smooth, all-quad mesh.

Unlike N-RoSy fields which look for global optimization and mathematical accuracy, IFAM

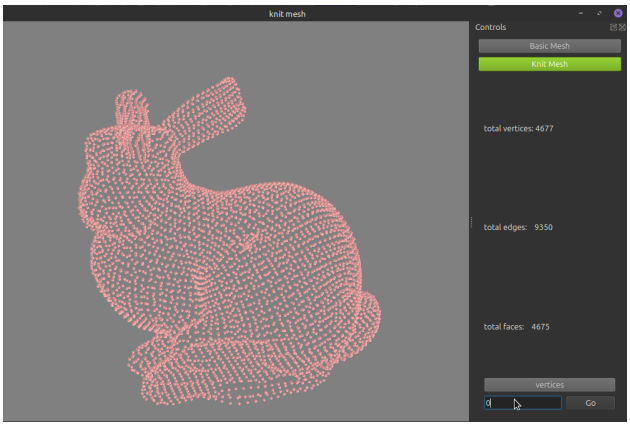


Figure 14: Early environment loading vertices from a quad dominant mesh created using blenders quadriflow algorithm.

prioritizes computational efficiency and practical usability. It is well suited as a fast and efficient foundation for Blender's quad remeshing algorithm.

3.1.5 Class `knitMesh`

The `knitMesh` class was constructed to store both the original mesh data (vertices, edges, and facets) from an OBJ file and the derived data (new vertices, edges, and facets) of the resulting knit mesh. Internally, the class computes a bounding box for the graphical user interface and, in the same step, it defines its own starting and ending boundaries based on the maximum and minimum vertex coordinates. This is currently along the z-axis (corresponding to the y-direction in OpenGL).

The class provides a multitude of methods, like sending vertex data to the Qt widget, as well as auxiliary functions like retrieving adjacent vertices and generating unique identifiers. Some of these methods were not ultimately required but were retained for potential future use.

3.2 Proposed Algorithm

After importing the original vertices of a base mesh, the `knitMesh` is generated through five main steps:

1. computing a morse function to assign a timing value to each vertex;
2. given a number of starting stitches creating the first boundary cycle;
3. iteratively finding new vertices and constructing row edges given each timing layer;
4. assigning quad, and triangle faces between each row, based off of increase and decrease algorithm.
5. outputting both a new mesh and a corresponding knitting pattern.

This forms a reasonable basis for trying to answer this thesis question.

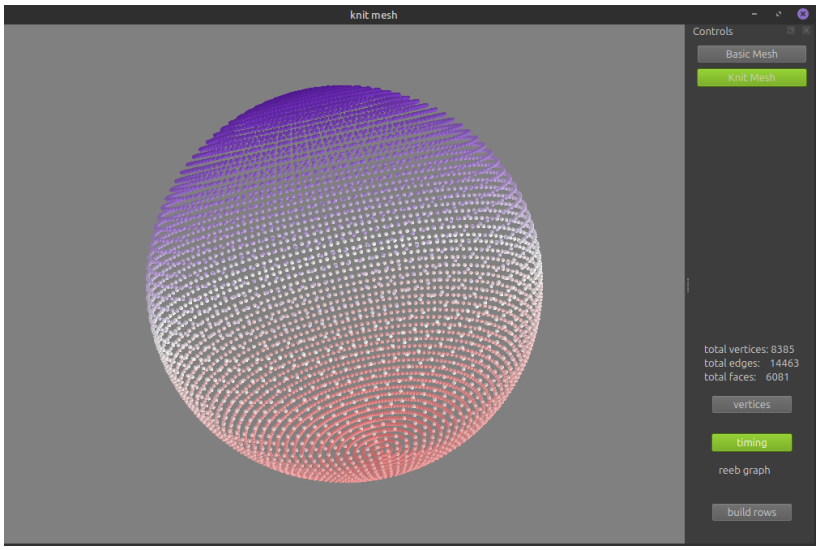


Figure 15: A simple reeb graph assignment to vertices on a sphere. Pink defines the starting boundary with indigo coloring at the end.

3.2.1 Morse Function Timing Assignment

For a simple shape like a sphere, a morse function $f : [0,1] \to M$ reduces to mapping each point on the surface of M to the interval $[0,1]$ based on a scalar function. I implemented a straightforward timing assignment that projects vertices onto the Reeb graph line segment from a user-defined start point v_{start} to an end point v_{end}. The timing value $t(v)$ for a vertex v is computed as the normalized scalar projection of $(v - v_{\text{start}})$ onto the direction vector $d = v_{\text{end}} - v_{\text{start}}$:

$$t(v) = \frac{(v - v_{\text{start}}) \cdot d}{\|d\|^2}$$

This assigns $t(v_{\text{start}}) = 0$ and $t(v_{\text{end}}) = 1$, with intermediate vertices receiving values in $[0,1]$ based on their linear position along the axis. The implementation assumes no saddles or critical points, which is sufficient for convex shapes. The pseudocode for this assignment is shown below.

Algorithm 1 Simple timing assignment (default Reeb)

Require: v_{start}, v_{end}, array of vertices $\{v_i\}$
Ensure: timing values t_i for each vertex

$d \leftarrow v_{\text{end}} - v_{\text{start}}$
mag $\leftarrow \|d\|$
$t(v_{\text{start}}) \leftarrow 0, t(v_{\text{end}}) \leftarrow 1$
for each vertex v_i **do**
    if $v_i \neq v_{\text{start}}$ **then**
        diff $\leftarrow v_i - v_{\text{start}}$
        $t_i \leftarrow (\text{diff} \cdot d)/(\text{mag}^2)$
    end if
end for

3.2.2 Starting Boundary

After obtaining the timing values, I implemented a method to define the first CO[12] cycle. This method takes a specified number of starting stitches and creates an edge for each stitch. It first locates the cycle of vertices whose timing values are closest to the starting boundary, identifies a center vertex among them, and then rotates from one start vertex to another to generate the desired number of edges in a cycle.

This approach is not universally applicable; it assumes that the vertices with the next timing values form a roughly circular arrangement. If they do not, the method will fail. For example, the starting boundary used by McCann et al. for the Stanford Bunny was a loop that had been stretched into a line in order to avoid knitting the base of the bunny and to better fit the constraints of an industrial knitting machine. For hand knitting however, the starting boundary should ideally be as small as possible, and this favors a circular cycle.

3.2.3 Extracting New Vertices and Building Rows

Once a starting boundary cycle has been constructed, the row building process begins. The method, called `build_rows`, is divided into several steps. In a loop, it performs the following operations:

1. Retrieve the vertices from the previous row.
2. Obtain the next cycle of vertices based on their timing values.
3. From the next cycle, select the vertices that best maintain uniform edge lengths.
4. Build column connections, handling decreases and increases while generating the next row cycle.

[12]Cast On

Steps one and two were straightforward. The previous rows' knit vertices were already stored, and retrieving the next cycle of vertices based on timing required only a simple search. Step three will be covered in this subsection and step four will be covered in the next subsection.

Building the next row took multiple iterations to achieve a usable set of "knit" vertices. Its final implementation assumes that within each timing bracket, the start of each cycle of vertices v_0 has the smallest z-coordinate and a constant x-coordinate throughout. The array is also ordered in a counterclockwise manner by using a center point to rotate through and re-order the array. This approach however does not generalize to all meshes, particularly those that are not well defined by a projection from a center point.

Vertices for the knit row are selected based on a euclidean metric

$$d_2(\vec{v_1}, \vec{v_2}) = \sqrt{(v_1.x - v_2.x)^2 + (v_1.y - v_2.y)^2 + (v_1.z - v_2.z)^2}$$

. If the distance falls within the stitch length, the vertex is accepted as a knit vertex, assigned as the new stitch, and the search proceeds to the next vertex. This method runs in $O(n)$ where n is the number of starting vertices in the timing cycle.

Algorithm 2 Select New Vertices and Create Row Cycle

Require: ε, stitch width, array of vertices $\{v_n\}$
Ensure: knit vertices $\{w_i\}$
1: $v_0 \leftarrow v[0]$
2: **for** each vertex v_n **do**
3: $d \leftarrow d_2(v_0, v_n)$
4: **if** $d \in [\text{stitch width} - \varepsilon, \text{stitch width} + \varepsilon]$ **then**
5: $\{w_i\} \leftarrow v_0$
6: $v_0 = v_n$
7: **end if**
8: **end for**

At the end of each cycle, if the euclidean distance between the last vertex and the first vertex $d_2(w_i, w_0) \notin [\text{stitch width} - \varepsilon, \text{stitch width} + \varepsilon]$, then each of the new knit vertices are shifted backward or forwards by the distance,

$$\text{shift dist} = \frac{|d_2(w_n, w_0) - \text{stitch width}|}{\text{length of } \{w_i\}}$$

to evenly space the difference between them. This process terminates once the next set of vertices is the ending boundary, v_{end} where $t(v_{end}) = 1$.

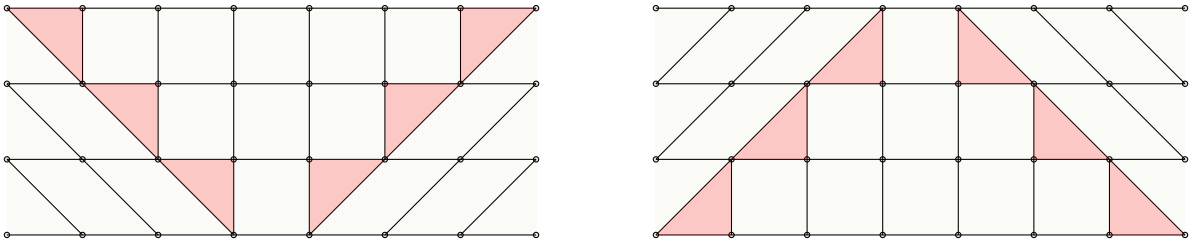
3.2.4 Extracting New Vertices and Building Rows

After the row building process completes, a separate method uses each rows' vertices to construct a stitch index array that tracks faces between rows, and builds an index map that plans where a knit stitch, increase or decrease will be placed over the mesh, assigning a quad, and triangle faces to knit and increase or decreases respectively.

(a) M1L and M1R (b) SSK and K2tog

Figure 16: Knitting increases and decreases in pink centered around a "main" stitch

Figures 16 and 17 describe different techniques for how to style a textile by increasing and decreasing. Figure 16a maintains an "increase line" with a pair of M1L, M1R on either side. For figure 16b a "decrease line" is created with K2tog and SSK on either side. Similarly, figure 17a opens around a grid with M1L, M1R on either side. Figure 17b closes around a grid with K2tog



(a) M1L and M1R (b) SSK and K2tog

Figure 17: Knitting increases and decreases in pink and opening and closing around a grid of stitches

and SSK on either side.

Because of the symmetry of both methods, these techiques create natural seams on a shape that scaffold directionality of stitches. These are a few of the many distinct ways that increases and decreases can style a textile. This upcoming stitch index algorithm most closely emulates figure 16, with increase and decrease lines described as "main" stitches.

For each row, an array is constructed that stores an integer value for faces built between a current cycle of knit vertices and the next cycle of knit vertices. A zero represents a normal stitch, and

1. represents M1R (make one right),
2. represents M1L (make one left),
3. represents K2tog (knit two together),
4. represents SSK (slip-slip-knit).
5. represents CO (cast on/off).
6. represents a "main" stitch.

Each stitch index array is iteratively built on top of a base case. The base case is the first array, where a boundary cycle of length k contains k entries of 5 (casting on).

For the first half of the subsequent rows, the algorithm follows this pattern. Given a previous stitch index array, previous vertices and current vertices:

1. Obtain the lengths of the previous and current rows. The total number of increases total id = curr size − prev size, the number of previous main stitches can be found iteratively through the previous stitch index array, the current main stitches, curr mains = $\left\lceil \frac{|\text{total id}|}{2} \right\rceil$.

2. initialize a new stitch index array as the current rows size with 0s.

3. If the previous main stitches are divisible by the current main stitches, then spacing is regular and you can assign mains and increases that follow mains and increases of the previous row given some regular spacing.

4. Else if the previous main stitches are not divisible by the current stitches, a smaller spacing is set with some remainder tacked on to the last spacing. Main stitches and increases are assigned based off this metric.

5. the new stitch index array is stored within the index map to be accessed for the next iteration.

Decreases are treated in a similar regard, but index arrays shift to assigning main stitch and decrease values to the previous array rather than the current array.

This algorithm does not account for significant changes in curvature where the number of stitch increases or decrecreases is greater than size of the current vertices. This could be addressed with an stitch index of only increases or decreases and could require double increases or decreases.

this algorithm also assumes consistent changes in curvature, and does not account for increases and decreases on the same row.

Initializing the stitch index array ensures that all positions that do not become increases, main stitches, or decreases will be treated as a regular stitch.

It is important to acknowledge that this approach does not prescribe a universal knitting method for all practitioners. Rather, it aims to achieve the desired effect of incorporating increases. This style of knitting has been observed and warrants further experimentation with respect to appearance, customizability, and other relevant factors.

After the row building process completes, a separate method uses the increase/decrease index map to construct the column edges. The knit vertices, column edges, and row edges are then exported to the graphical user interface, where the corresponding facets are also generated and delivered to the Qt widget for visualization.

3.2.5 Final Mesh and Pattern

After each step, all necessary information contained in the `knitMesh` object is exported to the Qt widget for the user to view. The final index map can be seen in Appendix A and represents an unparsed knitting pattern of the mesh. The selected knit vertices are saved in the Qt display, and faces are built from the new indices to show the quad and triangle faces determined by the index map.

4 Analysis

Through the methodology stated in the last section, I have established a timing-guided, distance based method for a simple 3D mesh to be reinterpteted as a quad dominant, knittable mesh, and a knitting pattern that can be hand knitted.

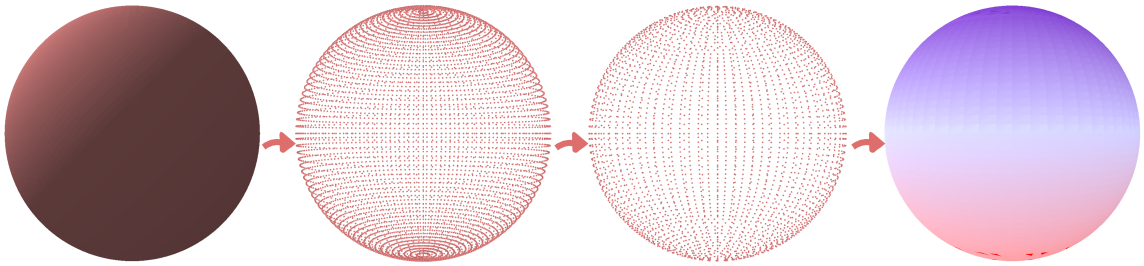


Figure 18: Pipeline: Simple 3D Mesh is parsed into its vertices, knit vertices are extracted, then quad and triangle faces are written given an index map.

4.1 Final Mesh

The output of the pipeline (Figure 18) is a quad-dominant mesh where each face is either a quadrilateral or a triangle, and the vertices are ordered to follow a continuous knitting path. The vertex extraction step identifies all points that will become knit stitches. Vertices are sorted iteratively by row, ensuring that adjacent vertices in the mesh correspond to directly connected stitches in the final pattern.

The final mesh's increases (colored red) and decreases (colored dark blue) did, to some extent, mimic the main stitch logic that determined which faces became knit stitches versus increase or decrease stitches. The pattern casts on 6 stitches and rows 1–13 contain spaced increases around a main stitch. Rows 51–63 contain decreases spaced around a main stitch, and the pattern casts off at row 64 with 5 stitches. This means that rows 14–50 each had 65 stitches, as there was no change in stitch count.

While the increase and decrease patterns were consistent with the existing method, this result was not what was anticipated. Looking at the final mesh, level of uniformity of each knit facet was

not expected. Quads grew in size through the middle of the mesh and were smaller and less regular in rows with increases and decreases.

One question is whether, because a Euclidean metric was used to compute distance, this affected the outcome. For example, this distortion of quads was most likely caused by how vertices were selected in each row. If a different metric, like the geodesic distance between two points on a sphere, had been used, it could have more accurately measured the change in curvature along the sphere's great circle contours and kept more uniform distances between vertices. The geodesic distance (on a sphere of radius r) is given by:

$$d = r \cdot \arccos\left(\sin\phi_1 \sin\phi_2 + \cos\phi_1 \cos\phi_2 \cos\Delta\lambda\right)$$

where ϕ_1, ϕ_2 are the latitudes and $\Delta\lambda$ is the difference in longitude. Using this instead of Euclidean distance would therefore have given more accurate row increases and decreases in the middle of the mesh.

It's worth wondering whether future implementations using geodesic distances would be more accurate. Similarly, since a sphere is easy to represent, a better approach could be to place equidistant vertices on circular contours of a given sphere object. That level of accuracy might also help fix this distortion.

Lastly, this existing method probably wouldn't be suitable for any arbitrary mesh. It ended up being a solution only for this specific case, and still had some areas that could be improved.

Taking the 2D RoSy field or other quad remeshing strategies of the sphere could help yield more accurate vertices to begin with, and at that point one could test a similar way of defining increases and decreases, or short rows to produce a knittable final mesh.

Figure 19: Physical Knit Model

4.2 Knit Sphere

The resulting physical model (Figure 19) matches the general shape of the input mesh even with unexpected shaping in the digital mesh. Kitting used meduim weight caooton yarn, with 4mm round knitting needles. The overall size is roughly 12 cm in diameter, and the pictured product has stuffing to keep its form.

Some distortion is visible around the beginning and end of the knit. The increase and decrease rows grow quickly and decrease quickly. This can be seen in divets on both ends of the sphere. This result suggests that while both ends accurately open and close the sphere shape, there exists a more optimal pattern for the sphere mesh.

The center of the pattern matched the input mesh very closely, which was surprising. While the results were not expected on the digital final mesh, they suited the actual physical knit model,

which could suggest that the digital mesh was more accurate than initially thought.

4.3 Reflection

Overall, the physical knit sphere demonstrates that the pipeline produces a valid, hand-knittable object, even if the digital mesh had unexpected distortions. The visible divets at the ends and the slight banding in the middle suggest room for improvement, but the close match at the center confirms that the method captures the core geometry reasonably well.

That said, this pipeline is most likely not generalizable to any arbitrary mesh. It cannot replace a proper quadmeshing strategy, which is necessary to space vertices evenly over an arbitrary surface. While I didn't use those methods here, I think starting with something like a Rosy quad remeshing seen in Yuksel et al. [1] or harmonic remeshing seen in McCann et al. [3] would help. I would like to look into how to let users define "main lines", for example, key contours or seam lines, to guide the remeshing and curvature choices.

Similarly, patching remeshing strategies could be useful because knitting naturally breaks down into patches (e.g., sleeves, panels) that can be worked continuously or separately and connected with knitting techniques.

One important realization is that my current approach does not try to minimize singularities in the mesh. Because of this, it might not fit well with existing remeshing strategies, which usually aim to reduce singularities for smoother surfaces. This in conjunction with a more contour aware morse function and reeeb graph could achieve truly accurate knittable meshes with hand knittable patterns.

Figure 20: Knit Cat from Yuksel et al.

5 Conclusion

This thesis explored a computational pipeline for converting a sphere triangular mesh into a quad-dominant representation related to hand knitting. A simplified Morse function (specifically, a linear projection of vertices onto an axis between user-defined start and end boundaries) was used to assign timing values. Based on this timing, a row-by-row construction algorithm was implemented. It selects vertices within a stitch-length distance, builds cycles of edges, and encodes increases (M1R, M1L) and decreases (K2tog, SSK) as integer labels (0–6) on edges.

The resulting `knitMesh` data structure stores vertices, edges, facets, and the stitch operation map. The pipeline was demonstrated successfully on convex shapes such as spheres. The stitch map can be translated into plain-text row instructions, providing a foundation for future hand-knitting pattern generation.

Several limitations must be acknowledged. The algorithm assumes that the next cycle of vertices forms a nearly circular arrangement and relies on a fixed projection axis; these assumptions do not hold for arbitrary manifolds. An attempt to use an explicit edge class introduced unnecessary complexity and was eventually simplified. Moreover, a full N-RoSy field was not implemented within the available time. QuadriFlow was examined as an alternative, but its vertex layout did not match the specific directionality of a hand-knitting graph. Future work should therefore focus on implementing a proper N-RoSy field to obtain a globally smooth cross field and optimal singularity placement, extending the row construction to arbitrary meshes, and finally validating the pipeline through hand-knitting of physical prototypes.

Despite these limitations, this work provides a concrete, working implementation of a timing-based row construction for knit-like meshes on simple geometries, and it clarifies the steps needed to move toward a fully general solution.

Knitting is a complex, multifaceted technique with many more aspects than have been introduced here. The continuation of this work could provide aid for knitters who want to design their own patterns without using traditional methods of constructing a knitting pattern.

A Unparsed Knitting Pattern

SPHERE KNITTING PATTERN:

1: [5, 5, 5, 5, 5, 5]

2: [6, 1, 6, 2, 6, 1, 6, 2, 6, 1, 6, 2]

3: [6, 1, 0, 0, 2, 6, 1, 0, 0, 0, 6, 1, 0, 0, 0, 6, 1]

4: [6, 1, 0, 0, 0, 0, 2, 6, 1, 0, 0, 0, 0, 6, 1, 0, 0, 0, 0, 0, 0, 2]

5: [6, 1, 0, 0, 0, 0, 0, 0, 2, 6, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2]

6: [6, 1, 0, 0, 0, 0, 0, 0, 0, 2, 6, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 6, 1, 0, 0, 0, 0, 0, 0, 0, 2]

7: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0]

8: [6, 1, 0, 0, 0, 0, 2, 6, 1, 0, 0, 0, 0, 2, 6, 1, 0, 0, 0, 0, 2, 6, 1, 0, 0, 0, 0, 2, 6, 1, 0, 0, 0, 0, 0, 0, 0, 6, 1, 0, 0, 0, 2]

9: [6, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0]

10: [6, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0]

11: [6, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

12: [6, 1, 0, 0, 0, 2, 6, 1, 0, 0, 0, 2, 6, 1, 0, 0, 0, 2, 6, 1, 0, 0, 0, 2, 6, 1, 0, 0, 0, 2, 6, 1, 0, 0, 0, 2, 6, 1, 0, 0, 0, 2, 6, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 6, 1, 0, 0, 0, 2]

13: [6, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2]

14: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

15: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

16: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

17: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

18: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

19: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

20: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

21: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

22: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

23: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

24: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

25: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

26: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

27: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

28: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

29: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

30: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

31: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

32: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

33: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

34: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

35: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

36: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

37: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

38: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

39: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

40: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

41: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

42: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

43: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

44: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

45: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

46: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

47: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

48: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

49: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

50: [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

51: [6, 3, 0, 0, 0, 0, 0, 0, 4, 6, 3, 0, 0, 0, 0, 0, 0, 4, 6, 3, 0, 0, 0, 0, 0, 0, 4, 6, 3, 0, 0, 0, 0, 0, 0, 4, 6, 3, 0, 0, 0, 0, 0, 0, 4, 6, 3, 0, 0, 0, 0, 0, 0, 0, 4, 6, 3, 0, 0, 0, 0, 0, 0, 4]

52: [6, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 6, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 6, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4]

53: [6, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

54: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

55: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

56: [6, 3, 0, 0, 0, 0, 0, 4, 6, 3, 0, 0, 0, 0, 0, 4, 6, 3, 0, 0, 0, 0, 0, 4, 6, 3, 0, 0, 0, 0, 0, 0, 0, 4, 6, 3, 0, 0, 0, 0, 0, 0, 4]

57: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

58: [6, 3, 0, 0, 0, 0, 0, 0, 0, 4, 6, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 6, 3, 0, 0, 0, 0, 0, 0, 0, 4]

59: [6, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 6, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4]

60: [6, 3, 0, 0, 0, 0, 4, 6, 3, 0, 0, 0, 0, 0, 4, 6, 3, 0, 0, 0, 0, 4]

61: [6, 3, 0, 0, 4, 6, 3, 0, 0, 0, 4, 6, 3, 0, 0, 4]

62: [6, 3, 0, 0, 4, 6, 3, 0, 0, 4]

63: [0, 0, 0, 0, 0]

64: [0]

References

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

[2] Jonathan Palacios and Eugene Zhang. Interactive visualization of rotational symmetry fields on surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 17(7):947–955, July 2011.

[3] Vidya Narayanan, Lea Albaugh, Jessica Hodgins, Stelian Coros, and James McCann. Automatic machine knitting of 3d meshes. *ACM Trans. Graph.*, 37(3):35:1–35:15, August 2018.

[4] Cem Yuksel, Jonathan M. Kaldor, Doug L. James, and Steve Marschner. Stitch meshes for modeling knitted clothing with yarn-level detail. *ACM Trans. Graph.*, 31(4), July 2012.

[5] Hansi Singh. *Amigurumi Knits: Patterns for 20 Cute Mini Knits*. Creative Publishing International, Minneapolis, 2009.

[6] David Bommes, Bruno Lévy, Nico Pietroni, Enrico Puppo, Claudio Silva, Marco Tarini, and Denis Zorin. Quad-mesh generation and processing: A survey. *Computer Graphics Forum*, 32(6):51–79, 2013.

[7] Edwin Catmull and James Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10(6):350–355, 1978.

[8] Jonathan Palacios and Eugene Zhang. Rotational symmetry field design on surfaces. *ACM Trans. Graph.*, 26(3):55–es, July 2007.

[9] David Bommes, Henrik Zimmer, and Leif Kobbelt. Mixed-integer quadrangulation. *ACM Transactions on Graphics*, 28(3):77:1–77:10, July 2009.

[10] Herbert Edelsbrunner and John Harer. *Computational Topology: An Introduction*. American Mathematical Society, Providence, RI, 2010. Pages 160–174.

[11] triplepointfive. ogldev: Opengl tutorials. https://github.com/triplepointfive/ogldev, 2018. Accessed: 2026-05-16.

[12] Cem Yuksel and Kui Wu. Stitch meshing, 2018.

[13] Stanford 3d scanning repository.

[14] Utah teapot.

[15] Jingwei Huang, Yichao Zhou, Matthias Niessner, Jonathan Richard Shewchuk, and Leonidas Guibas. Quadriflow: A scalable and robust method for quadrangulation. *ACM Transactions on Graphics*, 39(6), 2020.

[16] S. Dong, S. Kircher, and M. Garland. Harmonic functions for quadrilateral remeshing of arbitrary manifolds. *Computer Aided Geometric Design*, 22:392–423, 07 2005.

[17] Yu-Kun Lai, Miao Jin, Xuexiang Xie, Ying He, Jonathan Palacios, Eugene Zhang, Shi-Min Hu, and Xianfeng Gu. Metric-driven rosy field design and remeshing. *IEEE Transactions on Visualization and Computer Graphics*, 16(1):95–108, January 2010.

[18] Kui Wu, Hannah Swan, and Cem Yuksel. Knittable stitch meshes. *ACM Transactions on Graphics (TOG)*, 38(1):1–13, January 2019.

[19] The history of hand-knitting. Accessed: 2026-05-08.