AmiGo: Computational Design of Amigurumi Crochet Patterns

Michal Edelstein
Technion - Israel Institute of Technology
Haifa, Israel
smichale@cs.technion.ac.il

Hila Peleg
Technion - Israel Institute of Technology
Haifa, Israel
hilap@cs.technion.ac.il

Shachar Itzhaky
Technion - Israel Institute of Technology
Haifa, Israel
shachari@technion.ac.il

Mirela Ben-Chen
Technion - Israel Institute of Technology
Haifa, Israel
mirela@cs.technion.ac.il

Segment 0
=========
1: 8sc in a ring [8]
2: 6inc,sc,inc [15]
3: sc,(inc,sc)*2,(sc,inc)*5 [22]
4: sc,inc,2sc,(inc,sc)*2,(2sc,inc)*2,
   (2sc,inc,sc)*2 [29]
5: 2sc,inc,sc,(2sc,inc,3sc,inc)*2,
   2sc,(sc,inc,sc)*3 [37]
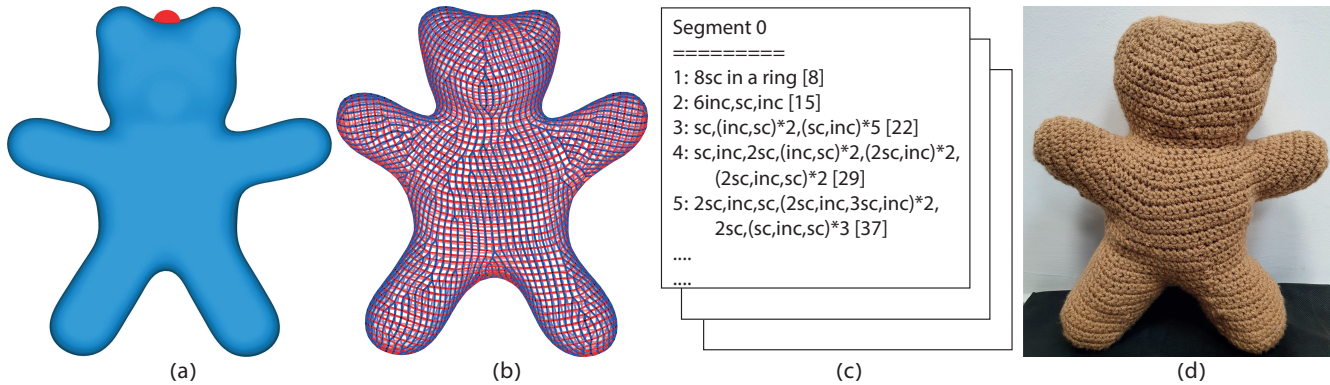....
....

(a)   (b)   (c)   (d)

Figure 1: Given an input 3D **model and a seed point (a) we automatically generate a *crochet graph* (b) which is translated into human-readable crochet instructions (c). When crocheted and stuffed, the output is a toy similar to the input shape (d).**

ABSTRACT

We propose an approach for generating crochet instructions (*patterns*) from an input 3D model. We focus on *Amigurumi*, which are knitted stuffed toys. Given a closed triangle mesh, and a single point specified by the user, we generate crochet instructions, which when knitted and stuffed result in a toy similar to the input geometry. Our approach relies on constructing the geometry and connectivity of a *Crochet Graph*, which is then translated into a crochet pattern. We segment the shape automatically into chrochetable components, which are connected using the join-as-you-go method, requiring no additional sewing. We demonstrate that our method is applicable to a large variety of shapes and geometries, and yields easily crochetable patterns.

CCS CONCEPTS

• **Computing methodologies** -> **Mesh geometry models**; • **Applied computing** -> **Computer-aided manufacturing**.

KEYWORDS

computational knitting, crochet, geometry processing

1 INTRODUCTION

Hand making toys is a popular and ancient craft. Knitting toys and stuffing them is one of the most popular approaches, as it allows for great flexibility in the color and texture of the final product, does not require sophisticated tools, and the technique can be easily taught, even to small children. Knitting approaches are mostly divided into methods which use a single hook or needle, usually classified as *Crochet*, and techniques which use two needles, which are generally called *Knitting*. Despite some superficial similarities, crochet and knitting yield very different fabrics, and the knitting patterns are different as well.

Computational knitting has been recently investigated in the graphics and fabrication communities [Yuksel et al. 2012], as knitting machines became available and popular. Crochet, on the other hand, cannot be as easily automated, and to-date there is no existing crochet-machine [Seitz et al. 2021]. Furthermore, the research on computational methods for generating crochet patterns is similarly

lacking. On the other hand, the crochet community has increased dramatically in recent years [Burns and Van Der Meer 2021], and *Amigurumi*, or crochet stuffed toys, became very popular. As the availability of computational techniques for pattern generation is quite limited, most designers rely on trial and error methods which are tedious and time consuming. Furthermore, novice crocheters are limited to producing existing patterns, and cannot easily express their creativity by generating patterns themselves.

Our goal in this paper is to address this gap, and suggest a computational method for generating a crochet pattern from an input 3D model. Given a single point on the model chosen by the user, denoted as the *seed*, and a user-selected stitch size, we automatically generate a *crochet graph*. The graph is then used for generating a crochet pattern, which after crocheting and stuffing, resembles the input model. We additionally provide a visualization of the expected crocheted shape, and thus the user can experiment with different seeds and different yarn sizes. We demonstrate that our algorithm is applicable to a large variety of shapes, and compares favorably to prior work.

1.1 Related Work

Knitting. The literature on computational *knitting* (as opposed to crochet) is quite large, and we mention some of it here for completeness. Note that, in general, crochet and knitting are two related, but very different, methods of fabric generation, and converting knitting patterns to crochet and vice versa is very challenging, even for human experts. Thus, a computational knitting pipeline cannot be used "as-is" for crochet.

The Stitch Meshes line of work [Yuksel et al. 2012] deals with representing 3D models at the yarn level using polygonal meshes whose faces represent different types of stitches, and on top of that considering knitability [Wu et al. 2019] and wearability [Wu et al. 2021]. Our crochet graph representation, on the other hand, is more abstract, representing only stitch heads/bases and stems, instead of the full yarn-level representation. However, for crochet this is sufficient. Specifically, the crochetability constraint of coupled rows, which directly translates into an algorithm for creating the instructions, can be easily enforced on our crochet graph.

Our representation has some resemblance to the representation of AutoKnit [2018], where the nodes represent two stacked stitches and the edges represent the connectivity and stitch size. They build the graph using a user specified *time function* and a set of constraints that guarantee machine knittability. They additionally propose tracing and scheduling algorithms to produce the machine knitting instructions. Popescu et al. [2018] build a similar graph by manually dividing the shape into patches and covering them with contours, which are later sampled to produce the instructions. Kaspaer at al. [2021] also generate machine-knitting instructions based on a stitch graph. The graph is computed by sampling 2D garment patterns according to a specified time function designed by the user. Other approaches focus on knitting compilers [McCann et al. 2016], interactive design of knit templates [Jones et al. 2021], and complex knit structures and multi-yarn [Nader et al. 2021].

When compared to machine knitting algorithms our approach is better tailored to hand-crocheting. First, the crocheted models are in many cases very low resolution (e.g., if they are aimed for beginners), and therefore we need to adapt our sampling rate accordingly (see Section 7.1.2). Furthermore, short rows are much less common in crochet, and we avoid them to generate patterns which appeal to beginner crocheters. The user only needs to supply an input seed point, instead of constraints on the time function as in AutoKnit, (e.g., two or more seed points), or a manual segmentation as in Popescu et al. [2018]. Finally, our simpler approach and the low resolution needed for crochet also leads to shorter processing times, where generating the instructions takes a few minutes, when compared to tens of minutes for AutoKnit. Hence opening the door to interactive editing and design of patterns in future work.

Crochet. A recent technical report [Seitz et al. 2021] provides an excellent background about the concept of computational crochet. The authors discuss pattern representation, as well as differences between crochet and knitting, and the lack of crochet machines. One of the first approaches to computational crochet was presented by Igarashi et al. [2008b], which provided an interactive tool for sketching a model and producing crochet instructions. Later, Nakjan et al. [2018] suggested a method that allows the user to design dolls using 2D sketches and generated from them crochet instructions.

Beyond sketching, Igarashi et al. [2008a] also allow the user to start from a 3D model. There, the mesh is covered with evenly spaced winding strips, which are then sampled at constant distances to compute the pattern. Their method requires a manual segmentation of the input, and the resulting knitted models are often very different visually from the input. More recently, Çapunaman et al. [2017] suggested a method that infers the stitch directions and connectivity from the (u, v) parameterization of a given surface. Thus, this approach requires a parameterized surface, whose parameter directions align with the required stitch directions. It is unclear how to achieve such a parameterization for a general surface. Finally, Guo et al. [2020] extended the Stitch Mesh framework to crochet by defining new faces that represent crochet stitches and new edge types that represent the current loop. They produce crochet instructions for 3D models and simulate the expected geometry. However, the crocheted item can differ greatly from the original model.

There are a few publicly available software projects that allow users to generate instructions from very simple geometries. The Crochet Sphere Calculator [Avtanski 2012b] generates instructions for crocheting spheres with a given number of rows. The Crochet Lathe [Avtanski 2012a] generates crochet instructions for surfaces of revolution, by allowing the user to design the profile curve. These are very basic, and do not allow the user to input a general 3D model.

1.2 Contribution

We propose an automatic method for generating crochet instructions (patterns) from a closed input 3D model, a seed point, and a stitch size. The generated instructions are crochetable, use only simple crochet stitches, and are based on the "join-as-you-go" method, and thus do not require any sewing. When crocheted and stuffed the models are similar to the input 3D shape, much more so than any previous approach for crochet instructions generation.
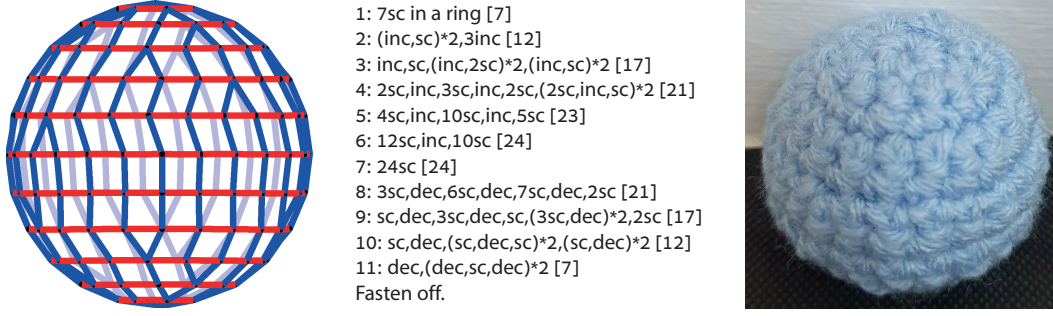
```
1: 7sc in a ring [7]
2: (inc,sc)*2,3inc [12]
3: inc,sc,(inc,2sc)*2,(inc,sc)*2 [17]
4: 2sc,inc,3sc,inc,2sc,(2sc,inc,sc)*2 [21]
5: 4sc,inc,10sc,inc,5sc [23]
6: 12sc,inc,10sc [24]
7: 24sc [24]
8: 3sc,dec,6sc,dec,7sc,dec,2sc [21]
9: sc,dec,3sc,dec,sc,(3sc,dec)*2,2sc [17]
10: sc,dec,(sc,dec,sc)*2,(sc,dec)*2 [12]
11: dec,(dec,sc,dec)*2 [7]
Fasten off.
```

Figure 2: (left) The *Crochet Graph* of a sphere, with row edges $\mathcal{R}$ in red, and column edges C in blue. (middle) The corresponding instructions (pattern) for crocheting the sphere. (right) The crocheted sphere.

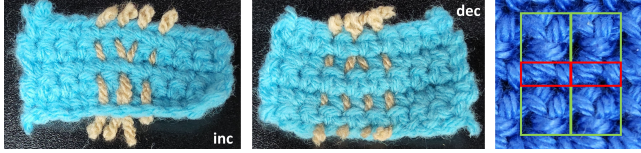**Figure 3: (left) Crochet stitches `inc`,`dec` marked using yarn on a crocheted patch. (right) The anatomy of crochet stitches, marked are the *top*/*bottom* of the stitch in red and the *stem* in green.**

2 REPRESENTATION

2.1 Background and Notations

Given a closed manifold triangle mesh $M = (\mathcal{V}, \mathcal{E})$, a *seed* vertex $s \in V$, and a stitch width $w \in \mathbb{R}$, our goal is to generate human-readable instructions $P(M, s, w)$ for crocheting M from the point s, with the given stitch width w.

Crochet has a wide variety of stitches, and we focus here on the simple stitch used for Amigurumi, named *single crochet* (`sc`). This is an approximately square stitch, thus covering M with `sc` stitches is equivalent to constructing a quad re-mesh of M, where each quad is a square, and all the edge lengths are constant. This is of course not possible unless the surface is developable, i.e. has zero Gaussian curvature. In practice, curved geometry is accommodated in crochet by introducing stitches which locally *increase* (`inc(x)`) or *decrease* (`dec(x)`) the amount of stitches by x. Figure 3 (left) shows an example of the `inc` and `dec` stitches on a crocheted patch. Crochet instructions for Amigurumi typically include *rows*, where each row is a series of `sc`, `inc`, `dec` stitches. Figure 2 (middle) shows the instructions (pattern) for crocheting the sphere in Figure 2 (right).

2.2 The Crochet Graph

A crochet stitch is composed of a *top*, a *base* and a *stem*, where the `inc`,`dec` stitches have multiple stems, see Figure 3 (right). The top of one stitch is always the base of some stitch on the next row, and similarly, each stitch has a base on the previous row. Therefore, a natural abstraction of the stitch pattern is to consider the stitches and their interconnections as a *graph*.

Specifically, we define the *Crochet Graph* $\mathcal{G} = (\mathcal{S}, \mathcal{R} \cup C)$, whose vertices $\mathcal{S}$ are tops/bases of stitches, where a vertex $(i, j) \in \mathcal{S}$ is the base of the j-th stitch in row i, and the vertices in each row are consecutively ordered. The *column* edges C are stems of stitches, and the connectivity between the bases in each row is represented by the *row* edges $\mathcal{R}$. We denote the total number of rows by N. Figure 2 (left) shows the crochet graph corresponding to the crocheted sphere in Figure 2 (right).

A crochet graph is an intermediate representation between the input triangle mesh M, and the output instructions P. Our goal is to generate a graph such that it is (1) translatable to valid crochet instructions P, and (2) when P is crocheted and stuffed, the result resembles the input mesh. Note that there exist multiple instructions P for the same graph $\mathcal{G}$, and within this space we aim for instructions which are *human-readable*.

We base our algorithm on the following observations.

DEFINITION 2.1. *A coupling [Gold and Sharir 2018] $C = (c_1, .., c_k)$ between two sequences $A = (p_1, .., p_n)$ and $B = (q_1, .., q_m)$ is an ordered sequence of distinct pairs of points from $A \times B$, such that $c_1 = (p_1, q_1), c_k = (p_n, q_m)$ and*

$$c_r = (p_s, q_t) \Rightarrow c_{r+1} \in \{(p_{s+1}, q_t), (p_s, q_{t+1}), (p_{s+1}, q_{t+1})\}, \quad \forall r < k. \tag{1}$$

DEFINITION 2.2. *Let $\mathcal{S}_i, \mathcal{S}_{i+1}, 1 \leq i < N$, be the vertices of two consecutive rows of $\mathcal{G} = (\mathcal{S}, \mathcal{R} \cup C)$, where $\mathcal{S}_i = ((i, 1), .., (i, n_i))$, where $(i, j) \in \mathcal{S}$, and n_i is the number of vertices in row i. If there exists a coupling C between $\mathcal{S}_i$ and $\mathcal{S}_{i+1}$ such that for all $p_s \in \mathcal{S}_i, q_t \in \mathcal{S}_{i+1}$ we have that $(p_s, q_t) \in C$ if and only if $(p_s, q_t) \in C$, then the two rows are* coupled.

OBSERVATION 2.3. *If all the pairs of consecutive rows of $\mathcal{G}$ are coupled, then there exist valid crochet instructions $P(\mathcal{G})$ that use only the instructions `sc`, `inc(x)` and `dec(x)`.*

DEFINITION 2.4. *Let $X_{\mathcal{G}} : \mathcal{S} \to M$ be an embedding of the vertices of $\mathcal{G}$ on M. An* embedded edge *of $\mathcal{G}$ is a shortest geodesic between the embedding of two vertices of $\mathcal{G}$ which share an edge, or between the embedding of the first and last vertices on the same row.*

DEFINITION 2.5. *Let $(p, q) \in M$ be two points whose geodesic distance is larger than some constant that depends on the stitch width w, and let $\gamma_{p,q}$ be the shortest geodesic between them. If $\gamma_{p,q}$ intersects*

some embedded edge of an embedding $X_\mathcal{G}$, for any two such points, then we say that $X_\mathcal{G}$ covers M.

OBSERVATION 2.6. *Let $X_\mathcal{G} : \mathcal{S} \to M$ be an embedding of the vertices of $\mathcal{G}$ which covers M, and $P(\mathcal{G})$ valid crochet instructions for $\mathcal{G}$. If all the edge lengths induced by $X_\mathcal{G}$ are equal to w, then when $P(\mathcal{G})$ will be crocheted and stuffed the result will be "similar" to M.*

We discuss Observation 2.3 in section 5.1, where we show how to translate the graph into valid instructions. The Observation 2.6 is in fact true only for a subset of meshes, as we discuss in the next section.

2.3 Crochetable Models

Curvature. Crocheting the patterns yields an empty flexible shell of fabric, which obtains its final shape by *stuffing*. Whether the stuffed model obtains the intended shape depends on how the model is stuffed (lightly, firmly, uniformly), as the yarn has some flexibility and will extend to accommodate if the model is over-stuffed. We will assume that the model is stuffed enough to attain maximal volume, but not too much to cause the yarn to stretch and generate gaps. Thus, we expect the resulting stitch size to be similar to the edge lengths induced by the embedding of the crochet graph. If for a given graph $\mathcal{G}$ its embedding in 3D with edge lengths w is *unique*, we expect the crocheted and stuffed shape to be similar to the input surface.

Importantly, unless the shape is convex, the edge lengths alone (i.e., the *metric*) do not contain enough information to uniquely determine the shape of a non-stuffed model. For example, a surface that has a "crater" leads to edge lengths which can be realized either as a crater or as a "hill". However, if we add the maximal volume assumption, only the hill is possible. This in fact implies that surfaces which have "craters", or more formally, regions of negative mean curvature with *positive* Gaussian curvature, cannot be realized by crocheting and stuffing alone. This is similar to the observation made by Konakovich et al. [2018], that surfaces which have negative mean curvature cannot be realized by maximizing the volume with a given conformal metric (i.e., only isotropic scaling is allowed relative to the flat configuration). We handle this case similarly, by preprocessing the model so that it does not contain "craters" (Section 7.1.1).

Furthermore, in our case, since we allow anisotropic scaling, negative mean curvature with *negative* Gaussian curvature is in fact possible, but requires a modified sampling rate, which we discuss in Section 7.1.2. To conclude, in terms of geometric obstructions to crochetability, the four possible curvature situations are summarized in Table 1. Note that, as with any sampling-dependent method, if the sampling rate (namely, the number of rows N) is too small compared to the feature size of the model, the crocheted output will lose some of the geometric detail.

Branching. Observation 2.6 requires that the crochet graph embedding $X_\mathcal{G}$ covers the input surface M. Because of the special structure of this graph, this induces additional constraints on the possible geometries. Intuitively, models that branch (see Figure 7) cannot be covered in this way. Mathematically, this means that the geodesic distance function on M from the embedding of the seed vertex s cannot have saddles. This is solved by segmenting

Table 1: Curvature obstructions to crochetability, see the text for details.

Gaussian Curvature	Mean Curvature Positive	Negative
Positive	Crochetable	Preprocessing
Negative	Crochetable	Sampling modification

the shape, and crocheting the segments in an iterative manner. We explain this in detail in Section 7.2.

We first explain the generation of the crochet pattern for a simple non-branching model with positive mean curvature, and then discuss how we handle negative mean curvature and branching.

3 OVERVIEW

Given a 3D mesh M, a seed point s and a stitch width w, we first compute a crochet graph $\mathcal{G}$ and its embedding $X_\mathcal{G}$ such that they adhere to Observations 2.3 and 2.6 (Section 4). Then we compute the crochet pattern $P(\mathcal{G})$ (Section 5).

To generate $\mathcal{G}$ and $X_\mathcal{G}$, we first compute the embedding of the vertices $\mathcal{S}$ on M (Section 4.1), and then derive from that the connectivity of $\mathcal{G}$, i.e. the row edges $\mathcal{R}$ and column edges C (Section 4.2). To compute the pattern $P(\mathcal{G})$, we first translate the graph into a program using standard code synthesis tools (Section 5.1), and then apply loop unrolling to make the pattern human-readable (Section 5.2). See Algorithm 1.

4 MESH TO CROCHET GRAPH

4.1 Geometry

Observation 2.3 implies that the vertices $\mathcal{S}$ should be grouped into ordered rows, where in each row the vertices have a well defined order. We address this requirement by computing two non-negative monotonically increasing, constant speed functions $f, g : M \to \mathbb{R}$ which define the row-order and column-order of every point on M. Furthermore, Observation 2.6 implies that the distance between embedded rows, and between embedded vertices in the same row should be w. We address this by sampling f, g appropriately.

Row order f. Our models are closed (so they can be stuffed), and therefore the first and last rows in the graph $\mathcal{G}$ contain a single
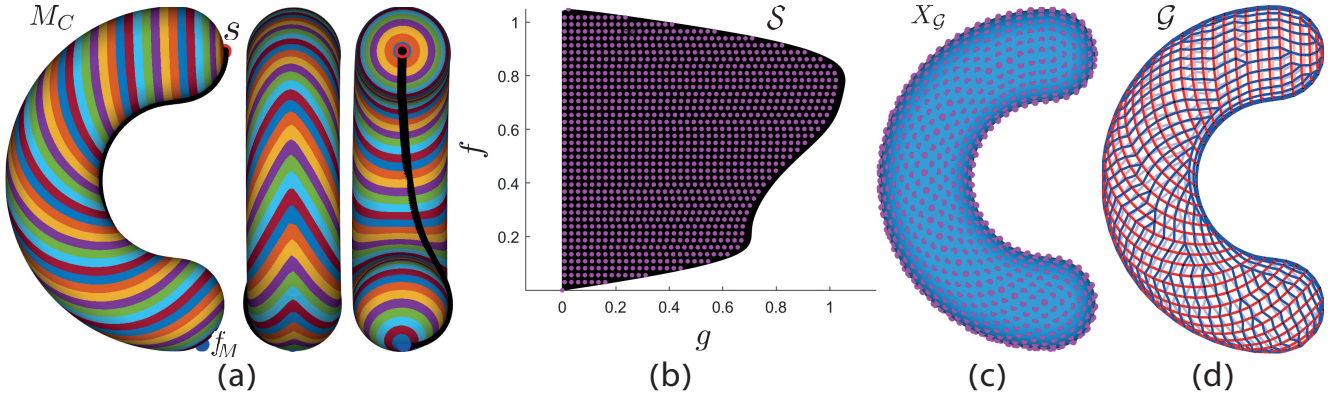

ALGORITHM 1: An outline of our algorithm

Input: A triangle mesh M, seed s, stitch width w
Output: Embedded crochet graph $\mathcal{G} = (\mathcal{S}, \mathcal{R} \cup C), X_\mathcal{G}$,
  crochet pattern $P(\mathcal{G})$
Mesh to Graph ; // Section 4
  Geometry $\mathcal{S}, X_\mathcal{G}$; // Section 4.1
  Connectivity $\mathcal{R}, C$; // Section 4.2
Graph to pattern ; // Section 5
  Graph to program ; // Section 5.1
  Program to pattern $P(\mathcal{G})$; // Section 5.2

Figure 4: (a) The isolines of f, with the seed (red), the maxima of f (blue), and the cut (black) marked. (b) The f, g parameterization, and the sampled grid $\mathcal{S}$, (c) The pushed forward points $X_{\mathcal{G}}$. (d) The output crochet graph $\mathcal{G}$, with red row edges $\mathcal{R}$ and blue column edges C.

vertex. The first row contains only the seed s, and its function value is $f(s) = 0$. We take $f(v), v \in \mathcal{V}$ to be $f(v) = d(v, s)$, where d is the geodesic distance. Thus, the isolines of f are rows, and two points $p, q \in M$ are on the same row if $f(p) = f(q)$. If f has more than one maximum, then we need to handle branching (see Section 7.2). Otherwise, the vertex that attains the maximum of f, denoted as f_M will be the single vertex on the last row.

Column order g. We first cut M along a geodesic from s to f_M, so that our model and the graph that we compute have the same topology, and denote the cut model by M_C. The requirements are that *within each row* the vertices of $\mathcal{G}$ have a well defined order. A row is an isoline of f, and therefore the rate of change along the isoline is given by the directional derivative of g in the direction of the tangent to the isoline. Specifically, the tangent to the isoline of f at a point $p \in M$ is given by $J \nabla f$, where J is the rotation by $\pi/2$ in the tangent plane of p. Thus to find g, we solve an optimization problem whose objective is to minimize $\int_{M_C} |\langle J \nabla f, \nabla g \rangle - 1|^2$, s.t. $g(\mathcal{B}) = 0$. Here, $\mathcal{B} \subset \mathcal{V}$ is the longest connected path of boundary vertices of M_C along which f is strictly monotone.

Sampling. The functions f, g define a parameterization of M_C to the plane. We conjecture that this parameterization is bijective (as it was in all of our experiments), but leave the proof to future work. The parameterization may have a large metric distortion, however, if $f(p) = f(q) = f_0$ for some two points $p, q \in M$, then $|g(p) - g(q)|$ is equal to the length of the isoline of f_0 between p and q. Therefore, we uniformly sample f, g on a 2D grid of width w, yielding the vertices of $\mathcal{S}$ with indices $(f/w, g/w)$. Pushing forward the sampled points to the mesh M_C yields the embedding of $\mathcal{S}$ on M_C (and therefore M), namely $X_{\mathcal{G}}$.

4.2 Connectivity

Row edges $\mathcal{R}$. Each two consecutive vertices of $\mathcal{S}$ on the same row are connected by a row edge. Namely, $\mathcal{R} = \bigcup_{i=1}^{N} \mathcal{R}_i$, and $\mathcal{R}_i = \{((i, j), (i, j+1)) \mid j \in \{1, .., n_i - 1\}\}$. Here $n_i = |\mathcal{S}_i| = |\{(i, j) \in \mathcal{S}\}|$, namely the number of vertices in the i-th row.

Let $x, y \in \mathcal{S}$ be two consecutive vertices on the i-th row. Then we have that $f(x) = f(y) = f_0$ and $|g(x) - g(y)| = w$. Therefore, $d_{\gamma(f_0)}(X_{\mathcal{G}}(x), X_{\mathcal{G}}(y)) = w$, where $\gamma(f_0)$ is the isoline of f_0 on M, and $d_{\gamma(f_0)}$ is the distance along the isoline. Hence, the Euclidean distance between the embedded vertices $||X_{\mathcal{G}}(x) - X_{\mathcal{G}}(y)|| \leq w$, and the distance tends to w for a "small enough" stitch size. Here, "small enough", means on the order of the square root of the radius of curvature of $\gamma(f_0)$, which is given in terms of the normal curvature in direction $J \nabla f$.

Column edges C. First, Observation 2.3 requires that all pairs of consecutive rows are coupled. Let C_i be the coupling corresponding to rows $\mathcal{S}_i, \mathcal{S}_{i+1}$, and let $(p_s, q_t) \in C_i$. Since p_s and q_t are on consecutive rows, and therefore embedded on isolines of f which differ by w, the minimal distance $||X_{\mathcal{G}}(p_s) - X_{\mathcal{G}}(q_t)||$ is close to w. Therefore, if among all couplings we seek the minimizer of:

$$\min_{C_i : coupling} \sum_{(p_s, q_t) \in C_i} ||X_{\mathcal{G}}(p_s) - X_{\mathcal{G}}(q_t)||, \qquad (2)$$

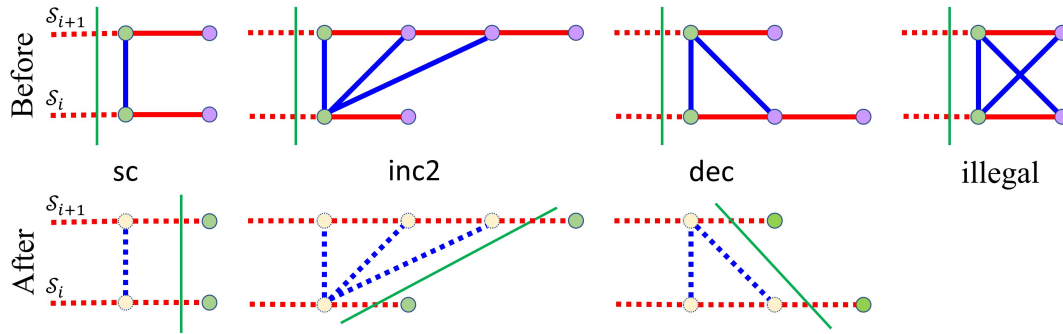then the length of the column edges will be close to w.

A minimal coupling between every pair of consecutive rows is found by Dynamic Time Warping (DTW) [Gold and Sharir 2018; Sakoe and Chiba 1978].

5 CROCHET GRAPH TO INSTRUCTIONS

5.1 Graph to Program

In order to turn the crochet graph into instructions, we rely on the following observation: crochet instructions (patterns) constitute an *instruction set*, and as such, crocheting is an *execution* and the finished object is an *execution result*. Moreover, because of the nature of a crocheted object, it is not only a result but a step-by-step *execution trace*.

Therefore, given a crochet object, or in this case its graph representation $\mathcal{G}$, deriving instructions constitutes a form of *execution reconstruction* [Zuo et al. 2021], a method for reconstructing the set of instructions that lead to an execution trace. While reconstructing an execution trace generally requires searching an exponential space of possible instruction sequences, the crochet instruction set

Figure 5: The state of the transducer before (top) and after (bottom) producing a stitch. Vertices at the head of the rows are marked in green, unconsumed vertices in purple, and consumed vertices in yellow. See the text for details.

is limited enough that reconstituting a trace is done in linear time using a transducer.

In order to reconstruct the trace, the degree of the graph vertices in both the current row S_i and next row S_{i+1} must be considered. Therefore, an execution reconstruction transducer accepts two consecutive rows and their connecting edges, and advances on the rows based on the vertex degrees in either row. For example, if at S_{i+1}, the transducer is at a vertex with one edge connecting it to the previous row S_i, and at the current head of S_i there is only a connection to the current head of S_{i+1}, the transducer will yield a sc. If there are $x > 1$ connected vertices in S_i to the head of S_{i+1}, the transducer will advance on S_i to consume them all, advance on a single vertex on S_{i+1}, and produce a dec(x). Analogously, for $x > 1$ connected vertices in S_{i+1} to the head of S_i, the transducer will advance on S_{i+1} to consume them, advance on a single vertex on S_i and then produce a inc(x). Figure 5 (left, middle) shows an illustration of the different situations. This is entirely analogous to the way crocheting the row is done.

Because the transducer chooses its transition based on which vertex at the current heads of the rows has a degree larger than 1, it is not technically deterministic. However, for a non-deterministic choice to exist, both the head of S_i should be connected to multiple vertices in S_{i+1} and vice versa. This is impossible, since all the pairs of consecutive rows of G are *coupled* (see Figure 5(right)). Thus, *under the constraints of the input*, i.e. a valid crochet graph G, all the transitions are mutually exclusive, rendering the transducer's behavior essentially deterministic.

5.2 Program to Human-readable Pattern

The instructions in a reconstituted trace can get quite repetitive. The raw output can contain the following rows:

```
row 2: sc, inc, sc, sc, sc, inc, sc, sc, sc, inc, sc, sc
row 3: sc, inc, sc, sc, sc, inc, sc, sc, sc, inc, sc, sc
```

indicating that row 2 is constructed via an sc instruction followed by an inc then another two sc, repeating three times, then row 3 is constructed the same way. To make the instructions both succinct and more human-readable, it is customary to convert this code to:

```
rows 2-3: (sc, inc, 2sc)*3
```

In order to do this, we employ a disassembly technique called *loop folding* [Lee et al. 1994], which finds maximal repetitions of instructions and turns them into loops. Loop folding of crochet instructions occurs at three different levels: repeating rows, repeating sequences of stitches, and repeating stitches. In order to find maximal repeating sequences, the order in which loops are folded must be: sequences first, and then repeating stitches. In the above example, if repeating stitches were folded first, row 2 after this initial folding would be sc, inc, 3sc, inc, 3sc, inc, 2sc, which means the repetition that can be identified within the row would have been smaller. Instead it is first folded to find the repeating sequence (sc, inc, sc, sc)*3, which is maximal, and then the internal sequence is re-folded to identify the 2sc. Finally, identical rows are folded together.
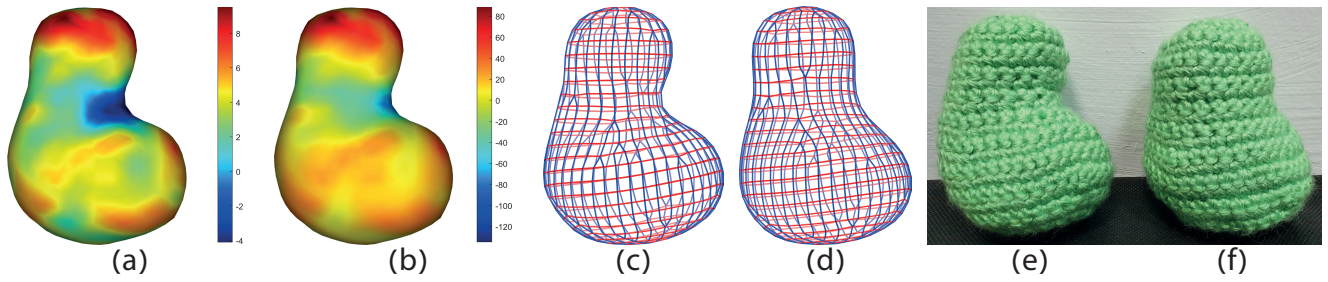
6 CROCHET GRAPH TO 3D EMBEDDING

We generate a 3D embedding Y_G of the crochet graph G using ShapeUp [Bouaziz et al. 2012], in order to obtain a visualization of the expected result. Interestingly, using purely geometric conditions the expected result is quite similar to the crocheted model in practice. We use the following constraints for ShapeUp: (1) The column edges which represent sc stitches, as well as the row edges are constrained to have length w, (2) the embedding of the seed point s is fixed to the position of the seed vertex, and (3) smoothness. We initialize Y_G using the sampled points X_G. Some of the figures (see for example Figure 6) show in addition to the embedded crochet graph X_G the ShapeUp result Y_G.

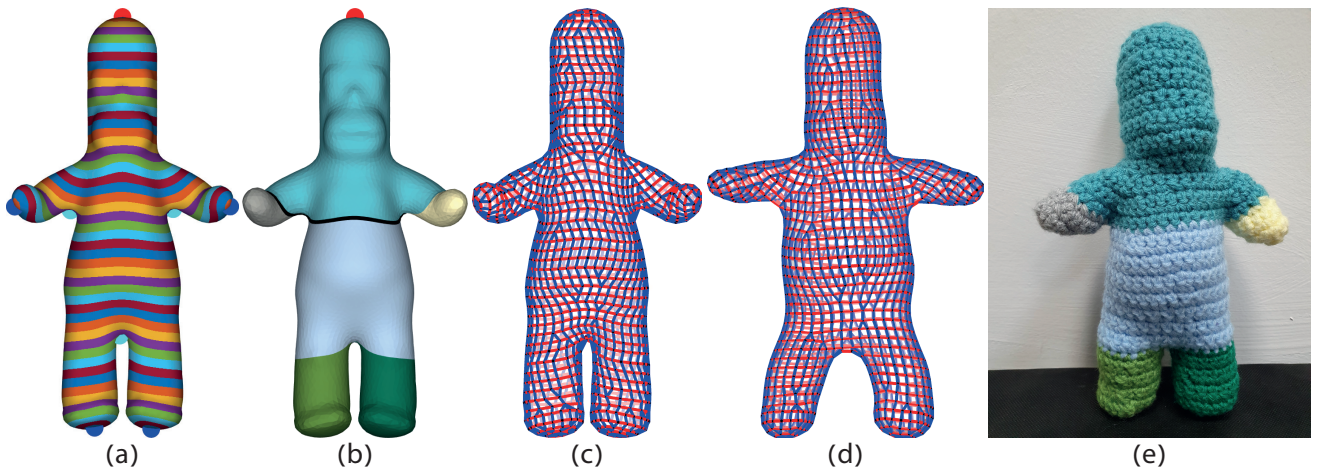## 7 OBSTRUCTIONS TO CROCHETABILITY

7.1 Negative mean curvature

As curvature computations are not scale invariant, all the models are normalized to have surface area equal to 1, so that we can use the same parameters across all the models.

7.1.1 Positive Gaussian curvature. Crocheting and stuffing a model which has "craters", i.e. regions of positive Gaussian curvature and negative mean curvature, will not yield a geometry that is similar to the input mesh. Thus, we apply a pre-processing step (similarly to Konakovic et al. [2018]) to smooth out the craters. Specifically, we apply Conformal Mean Curvature Flow [Kazhdan

Figure 6: A model with negative mean curvature (a) and negative Gaussian curvature (b) in the same region. The corresponding embeddings $Y_{\mathcal{G}}$ (c,d) and knitted objects (e,f), with curvature-adapted (c,e) and uniform (d,f) sampling rate of g. Note that the uniform sampling rate does not lead to a model that is similar to the input, whereas curvature-adapted sampling yields a better result.

Figure 7: (a) The isolines of f for the marked red seed point, and the saddles (cyan) and maxima (blue) of f. (b) The resulting segments. (c) The crochet graph $\mathcal{G}, X_{\mathcal{G}}$. (d) The embedding $Y_{\mathcal{G}}$ of the crochet graph. (e) The final knitted model. Different segments were crocheted in different colors for better visualization.

et al. 2012] localized to these areas until the mean curvature is positive everywhere.
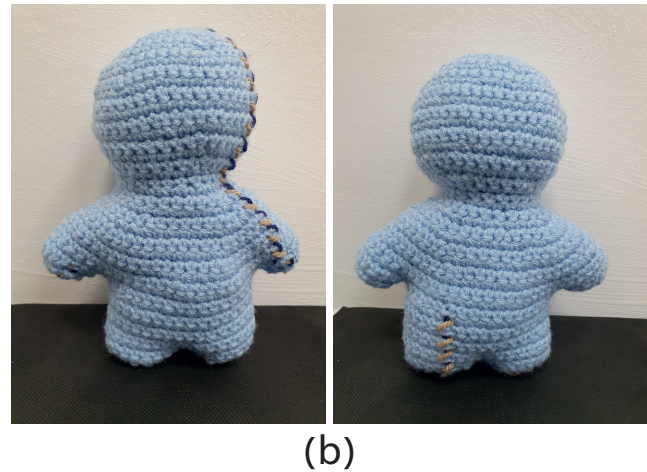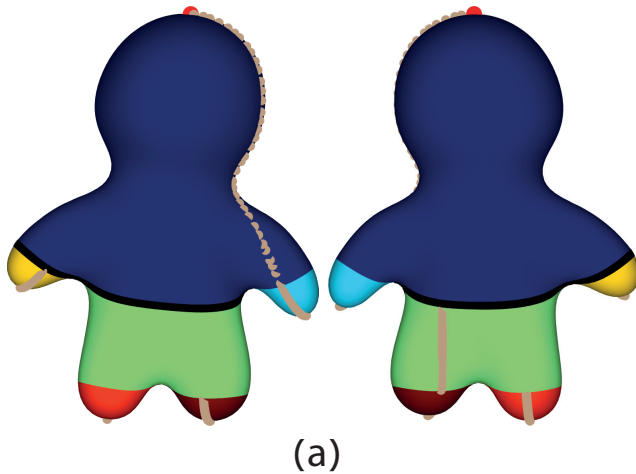
7.1.2 Negative Gaussian curvature. The sampling rate of the isolines of f is determined by the directional derivative of g w.r.t. the tangent to the isoline, namely by $\langle \nabla g, J \nabla f \rangle$. If the curvature in the direction of the isoline $k_{J\nabla f}$ is large compared to the stitch width w, a uniform sampling rate is inadequate, and does not result in a similar geometry. We therefore adjust the sampling rate in these regions by setting $\langle \nabla g, J \nabla f \rangle = h(k_{J\nabla f})$. We take $h(x) = \tanh(-x/\alpha)/2 + 1$, to avoid degenerate sampling rates, with $\alpha = 10$. Figure 6 shows an example of such a model. In the central region, the model has negative mean and Gaussian curvature (a,b). Using a uniform sampling rate does not fully reconstruct the negative curvature (d,f), whereas a curvature adapted sampling does (c,e).

7.2 Branching

If for a given seed s, the geodesic function $f(x) = d(s, x)$ has multiply-connected isolines, the graph $\mathcal{G}$ cannot cover the model.

In these cases, the model is automatically decomposed into multiple segments, each of which can be crocheted using the approach described in Sections 4 and 5. The segments are attached by a method called "join-as-you-go" [Bennett 2020], meaning each segment is crocheted onto the last row of the previous segment, and therefore no additional sewing is required. Furthermore, the segment boundaries are not visible in the crocheted object. The more common method, in contrast, involves sewing together closed segments, which requires accuracy to achieve geometric details such as symmetry. In this section we describe the modifications required to accommodate such objects.

7.2.1 Mesh to Graph. Given a seed s let $f(x) = d(s, x)$. Let $\Pi = (\sigma_1, .., \sigma_m)$ be the saddle points of f, sorted by $f_i = f(\sigma_i)$. Namely $f_1 \leq f_2 \leq ... \leq f_m$. For each σ_i in order, we compute the isoline of f_i, denoted by γ_i, and slice a new segment for each connected component of γ_i. Meaning, the segments are obtained by slicing along the isolines of the saddle points, ordered by increasing geodesic distance to the seed. Figure 7(a) shows the isolines of f for the seed

Figure 8: (a) The cut location (brown) on a segmented model. (b) The crocheted model with the cut marked using a piece of yarn.

point s marked in red, as well as the saddles (cyan), and maxima (blue).

In addition to the segmentation, we generate a directed graph G_σ, whose vertices are the segments, and where an edge (s, t) exists if the segments M_s and M_t share a boundary, and f values on M_s are smaller than f values on M_t. The crocheting order of the segments is determined by a topological sort of G_σ. Figure 7(b) shows the resulting segments. Very thin segments might not be sampled (marked in black in Figure 7(b) and other figures), and are skipped and not crocheted.

Geometry. Any resulting segment M_l is either a half sphere or a cylinder, and thus can be covered by a crochet graph $\mathcal{G}_l$. While f is computed for the whole model before segmentation, g is computed for each segment separately. The cut for g is made from the maxima of f in the segment to the closest point on the segment's boundary. If f attains its maximum on one of the boundaries of the segment (there are at most two boundaries), then cut is computed to the closest point on the other boundary.

Figure 8 shows an example of the location of the cut, where we show the front and the back of the model. We show the location of the cut in brown on the segmented model (a), as well as the location of the cut in the crocheted model (b). To mark the location of the cut during crocheting a piece of yarn was used to mark the beginning/end of the row.

Connectivity. For every two segments M_s, M_t which share an edge (s, t) in G_σ, we add an additional condition that the last row of M_s is coupled to the first row of M_t. Figure 7(c) shows the crochet graph for all the segments of the Homer model. Finally, (e) shows the crocheted model, where each segment was crocheted with a different color for better visualization.

7.2.2 Graph to instructions. The same simulation of the crochet operations is applied to the first row of a new segment, but the sequence of stitches that is used as its previous row is no longer a full row. Instead, the last rows of all attached segments are arranged

and filtered to include only vertices that have a connecting edge to the new segment's row, constituting a "joint" previous row. The transducer then takes stock of when its consumption of the previous row skips stitches, splits segments, skips segments, or spans multiple parent segments, and includes this information in the row instructions.

8 LIMITATIONS

Our method only handles closed surfaces, since we aim for Amigurumi models, which are stuffed. The segmentation approach may generate thin segments, which are harder to crochet. While we filter out very thin segments, we believe that small modifications to the singularity locations can yield a better segmentation without considerably affecting the shape. Our approach does not take the symmetry of the model into account, and thus discretization errors for low resolution patterns may lead to non-symmetric crocheted models. Our current setup also has limited design freedom. While using a single seed is simple, it does not give the user control of the knitting direction throughout the shape.

9 RESULTS

9.1 Implementation Details

We implemented our algorithm in Matlab and C++. We use Heat Geodesics [Crane et al. 2017] for computing geodesic distances, where we took the recommended time parameter t, namely the average edge length squared. In some cases, the mesh has too many geometric details, leading to neighboring saddles/extrema of the distance function. In this case, we take advantage of the tunability of heat geodesics, and repeated multiply t by an increasing power of 2 until there are no more neighboring saddles/extrema. For computing geodesic paths we use [Sharp and Crane 2020]. Table 2 provides the statistics for all the models. Algorithm running times were measured on a desktop machine with an Intel Core i7.

Table 2: Statistics for the crocheted models.

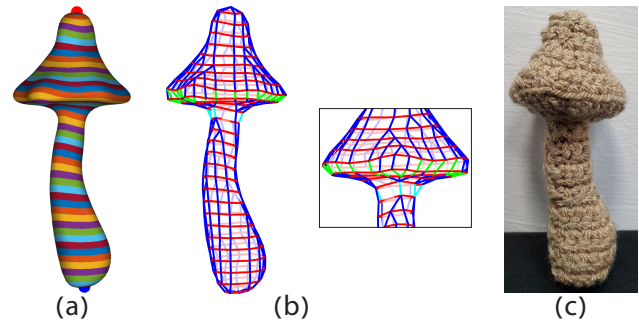| Model	Rows	Segments	Stitches	Time (min)
Teddy, Fig 1, Fig 10	60	6	3670	2.5
Teddy, Fig 10	30	6	880	2.1
Homer, Fig 7	50	6	1605	2.0
Mushroom, Fig 9	30	1	365	0.2
Man, Fig 8	45	6	1643	1.5
C, Fig 10	40	1	1118	0.3
Fish, Fig 10	40	1	1020	0.3
Bob, Fig 10	30	4	854	1.0
Bunny, Fig 10	45	5	2470	0.8
Moomoo, Fig 10	60	8	2491	6.9
Pretzel, Fig 10	30	7	586	1.5

9.2 Gallery

Figures 1, 7, 6 demonstrate our results. Figure 10 shows additional models, where we show the (a) segmented shape and seed, (b) the crochet graph $\mathcal{G}$ and its embedding $X_\mathcal{G}$ on the input mesh, (c) the embedding $Y_\mathcal{G}$ generated from the edge lengths, and (d) the final crocheted object. The sixth row of Figure 10 shows an example of a pretzel model, which cannot be scheduled for machine-knitting as discussed at AutoKnit [Narayanan et al. 2018] (see Figure 25 there). Our method, on the other hand, generates crochetable instructions. The last two rows of Figure 10 show the results for the same model, seed point, yarn type and hook but using different stitch width w. Note that while manually adapting the pattern to different sizes is a difficult task, our algorithm preforms it automatically. Note that the shapes are similar to the input, and the segment boundaries are not visible in the output crocheted models. Therefore, we can achieve varied geometries without visibly segmenting the shape, leading to more visually pleasing results than the approach by Igarashi et al. [2008a] (see Figure 11 there). Furthermore, our results have a closer resemblance to the input, compared to the method by Guo et al. [2020] (see Figure 12 there).

9.3 Creases

Since stuffed items tend to be smooth, there exist crochet shaping techniques that allow the generation of creases. Specifically, instead of inserting the hook under both loops of the stitch, the hook is inserted only in the front loop (denoted *Front Loop Only* FLO), or only in the back loop (denoted *Back Loop Only* BLO). The BLO (resp. FLO) stitch allows creases which are positively (resp. negatively) curved with respect to the columns direction (i.e. orthogonal to the knitting direction).

We define *crease vertices* as vertices that have large maximum absolute curvature, and their maximum absolute curvature direction is orthogonal to the knitting direction. For any two consecutive crease vertices on the same row, we mark all the stitches based on these vertices as BLO or FLO, depending on the sign of the curvature. We allow the user to choose whether to enable this option or not. Figure 9 shows an example of a model where this shaping technique was used. The BLO (resp. FLO) edges are marked in green (resp. cyan) in (b). Note the corresponding sharp crease in the knitted object (c). We note that creases have also been used in knitting. For example, [Wu et al. 2019] allow for creases, but they use a different technique.

Figure 9: (a) The isolines of f for the marked red seed point, and the maxima of f (marked blue). (b) The crochet graph with BLO edges (green) and FLO edges (cyan) (c) The final knitted model.

10 CONCLUSIONS AND FUTURE WORK

We presented a novel automatic approach for generating crochet knitting instructions for Amigurumi from an input triangle mesh. Given a single seed point and a stitch size, we generate human-readable instructions that use only simple crochet stitches (sc, inc, dec). Our method is applicable to a variety of geometries, and leads to crocheted models which are visually similar to the input shape. In the future we plan to incorporate our method within an interactive framework that allows the user to move the seed point, change the yarn and the gauge and see the expected shape of the output. Furthermore, we plan to add colors and texture, as well as support for additional types of stitches.

With the wide popularity of Amigurumi, and crochet in general, in recent years, we believe that tools that allow novice users, as well as pattern designers, to generate crochet instructions from 3D models would be quickly adopted and built upon by the crocheters and makers communities. Our approach provides an important stepping stone in this direction, and we expect that it will sow the seeds for further research in comptuational crochet.

REFERENCES

Alexander Avtanski. 2012a. *Crochet Lathe*. http://avtanski.net/projects/crochet/lathe
Alexander Avtanski. 2012b. *Crochet Sphere Calculator*. http://avtanski.net/projects/crochet
Cheryl Bennett. 2020. *Joining Amigurumi Limbs*. https://www.crochet365knittoo.com/joining-amigurumi-limbs-an-easy-technique
Sofien Bouaziz, Mario Deuss, Yuliy Schwartzburg, Thibaut Weise, and Mark Pauly. 2012. Shape-up: Shaping discrete geometry with projections. In *Computer Graphics Forum*, Vol. 31. Wiley Online Library, 1657–1667.
Pippa Burns and R Van Der Meer. 2021. Happy Hookers: findings from an international study exploring the effects of crochet on wellbeing. *Perspectives in Public Health* 141, 3 (2021), 149–157.
Özgüç Bertuğ Çapunaman, Cemal Koray Bingöl, and Benay Gürsoy. 2017. Computing stitches and crocheting geometry. In *International Conference on Computer-Aided*

Architectural Design Futures. Springer, 289–305.

Keenan Crane, Clarisse Weischedel, and Max Wardetzky. 2017. The heat method for distance computation. *Commun. ACM* 60, 11 (2017), 90–99.

Omer Gold and Micha Sharir. 2018. Dynamic time warping and geometric edit distance: Breaking the quadratic barrier. *ACM Transactions on Algorithms (TALG)* 14, 4 (2018), 1–17.

Runbo Guo, Jenny Lin, Vidya Narayanan, and James McCann. 2020. Representing crochet with stitch meshes. In *Symposium on Computational Fabrication.* 1–8.

Yuki Igarashi, Takeo Igarashi, and Hiromasa Suzuki. 2008a. Knitting a 3D model. In *Computer Graphics Forum*, Vol. 27. Wiley Online Library, 1737–1743.

Yuki Igarashi, Takeo Igarashi, and Hiromasa Suzuki. 2008b. Knitty: 3D Modeling of Knitted Animals with a Production Assistant Interface.. In *Eurographics (Short Papers).* 17–20.

Benjamin Jones, Yuxuan Mei, Haisen Zhao, Taylor Gotfrid, Jennifer Mankoff, and Adriana Schulz. 2021. Computational Design of Knit Templates. *ACM Transactions on Graphics (TOG)* 41, 2 (2021), 1–16.

Alexandre Kaspar, Kui Wu, Yiyue Luo, Liane Makatura, and Wojciech Matusik. 2021. Knit sketching: from cut & sew patterns to machine-knit garments. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–15.

Michael Kazhdan, Jake Solomon, and Mirela Ben-Chen. 2012. Can mean-curvature flow be modified to be non-singular?. In *Computer Graphics Forum*, Vol. 31. Wiley Online Library, 1745–1754.

Mina Konaković-Luković, Julian Panetta, Keenan Crane, and Mark Pauly. 2018. Rapid deployment of curved surfaces via programmable auxetics. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–13.

Tsing-Fa Lee, AC-H Wu, Youn-Long Lin, and Daniel D Gajski. 1994. A transformation-based method for loop folding. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 13, 4 (1994), 439–450.

James McCann, Lea Albaugh, Vidya Narayanan, April Grow, Wojciech Matusik, Jennifer Mankoff, and Jessica Hodgins. 2016. A compiler for 3D machine knitting. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 1–11.

Georges Nader, Yu Han Quek, Pei Zhi Chia, Oliver Weeger, and Sai-Kit Yeung. 2021. KnitKit: A flexible system for machine knitting of customizable textiles. *ACM Transactions on Graphics* (2021).

Pikanate Nakjan, Sukanya Ratanotayanon, and Natchayar Porwongsawang. 2018. Automatic Crochet Pattern Generation from 2D Sketching. In *2018 10th International Conference on Knowledge and Smart Technology (KST).* IEEE, 170–175.

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

Mariana Popescu, Matthias Rippmann, Tom Van Mele, and Philippe Block. 2018. Automated generation of knit patterns for non-developable surfaces. In *Humanizing Digital Reality.* Springer, 271–284.

Hiroaki Sakoe and Seibi Chiba. 1978. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing* 26, 1 (1978), 43–49.

Klara Seitz, Jens Lincke, Patrick Rein, and Robert Hirschfeld. 2021. *Language and tool support for 3D crochet patterns: virtual crochet with a graph structure.* Vol. 137. Universitätsverlag Potsdam.

Nicholas Sharp and Keenan Crane. 2020. You can find geodesic paths in triangle meshes by just flipping edges. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–15.

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

Kui Wu, Marco Tarini, Cem Yuksel, James Mccann, and Xifeng Gao. 2021. Wearable 3D machine knitting: automatic generation of shaped knit Sheets to cover real-world objects. *IEEE Transactions on Visualization and Computer Graphics* (2021).

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

Gefei Zuo, Jiacheng Ma, Andrew Quinn, Pramod Bhatotia, Pedro Fonseca, and Baris Kasikci. 2021. Execution reconstruction: Harnessing failure reoccurrences for failure reproduction. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation.* 1155–1170.
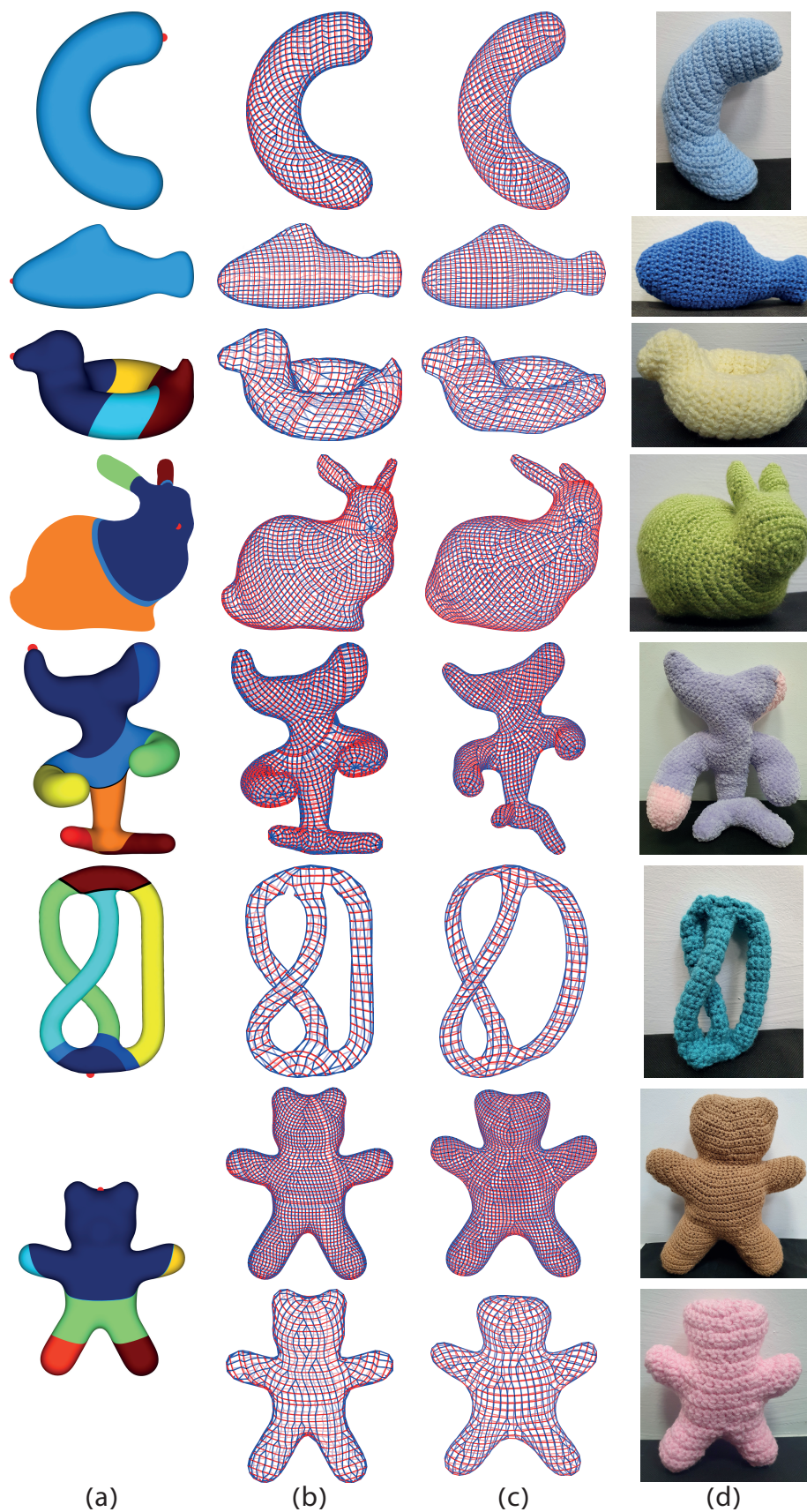
Figure 10: A gallery of our results. (a) The segmentation. (b) The crochet graph $\mathcal{G}, X_{\mathcal{G}}$ (c) The embedding $Y_{\mathcal{G}}$ of the crochet graph. (d) The crocheted model. Note that the locations of the segments is not visible in the final output.