Generating Chinese Knots from Arbitrary Shapes

Andrew Lee and Brandon M. Wang
EECS, Computer Science Division
University of California, Berkeley, CA 94720
E-mail: {a.lee, brandonwang}@berkeley.edu

Abstract

Chinese knotting is a decorative art form where a single cord is looped and knotted to form interesting shapes and patterns. Most examples are abstract and symbolic, but there are also a few existing knots designed to resemble actual objects, such as animals. This paper presents a method for the automatic generation of a single-string Chinese-style knot that resembles an arbitrary input 3D model. We implemented our procedure as a program called `Knotty`.

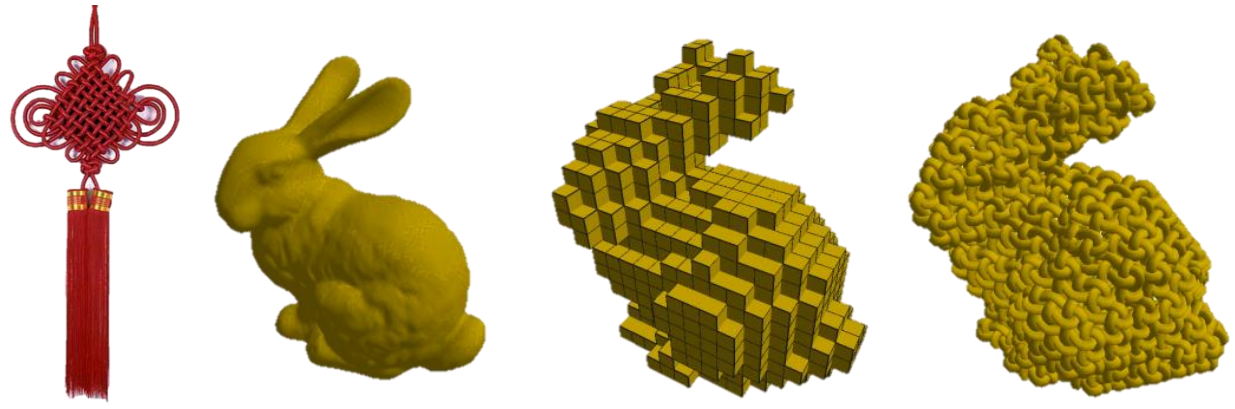
(a) (b) (c) (d)

Figure 1: *(a) An example of a common Chinese knot. The high-level workflow: (a) Input model, (b) voxelized shell, (c) final knot model.*

Introduction

Chinese knotting is a decorative folk art form that creates interesting shapes and patterns by knotting a single cord. Most examples of Chinese knots are abstract and symbolic. Figure 1(a) depicts a very common form of Chinese knot. However, there are a few examples of knots that are designed to resemble actual objects, like animals. The only widespread examples are of dragonflies, fish, and turtles.

We developed a workflow to procedurally generate a knotted, single-string representation of an arbitrary object (Figure 1(b-d)). The knot is exported as a model that can be fabricated using a Fused Deposition Modeling (FDM) machine or some other layered manufacturing technique. This allows us to provide physical examples of our extension of Chinese knotting. We implemented this workflow into a program called `Knotty`.

Workflow

To create our knot, we first create a voxelized shell that encloses the input geometry. Then, we use this shell to guide a single string to cover the whole surface with a knotted, woven pattern.

Voxelization. Our first step is to voxelize the model accepted as a boundary representation (in OBJ format) into axis-aligned cubes. From these voxels, we extract the outer shell by connecting together their outside-facing square facets. This results in a description with a uniform facet size, which allows us to easily create a uniform weave over the whole surface.

Graph. Using the aforementioned shell, we derive a graph simply by treating each face as a vertex, and then connecting the face through the midpoints of the shared edges. Figure 2(a-b) illustrates this process.

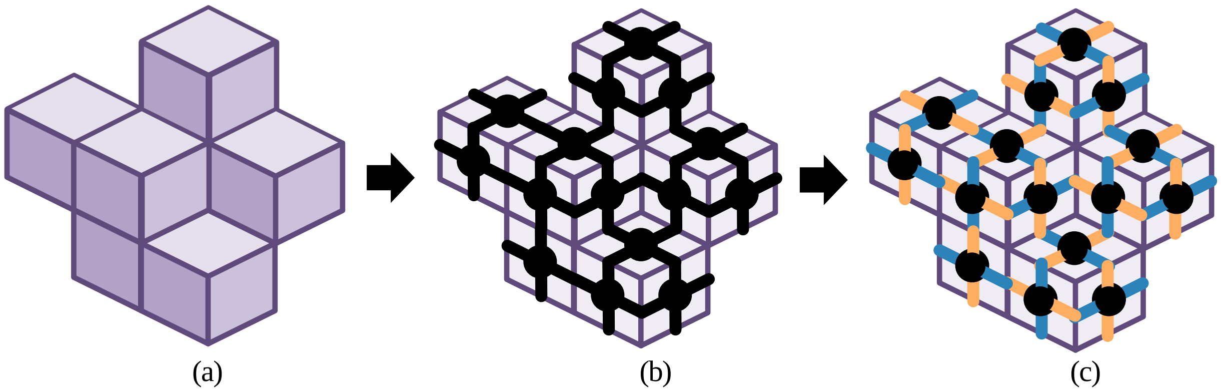

Figure 2: *(a) From voxels, (b) to a graph, and (c) to a graph with edges labeled "over" and "under".*

Eulerian Path. To generate a knot using a single string, we find an Eulerian cycle on the graph we have just generated. Every vertex on this graph has exactly 4 edges, so it is always possible to find such a cycle. We used Hierholzer's algorithm [1] to find such a path, outlined here:

```
Start with a vertex V on graph G
Find a cycle C by traversing G until arriving at V
While G has unused edges:
    Traverse C until arriving at V′ with unused edges
    Find a subcycle C′ starting and ending with V′
    Splice C′ into C at V′
```

We made one addition to the algorithm: At any vertex where we have a choice in which edge to take next, we preferentially select the edge that continues in the current direction. This will lead to fewer 90° turns in the path, which results in a more aesthetically pleasing knot.

Weaving. In order to have a good knot, the string should weave over and under itself in a consistent manner.

As illustrated in Figure 3(a-b), when we consider a surface facet, the Eulerian path on that face can be one of three cases. For each case, we have defined how the two string segments will interweave (Figure 3(c)) with a set of B-spline control points that yield appropriately curved tubes.

If we imagine cutting each string segment in half, and then labeling each piece either "over" or "under" (Figure 3(d)), it turns out that each case shows the same abstract pattern. Namely, the pieces starting at opposite edges are both "over" and the other two opposite edges are both "under" (Figure 3(e)).

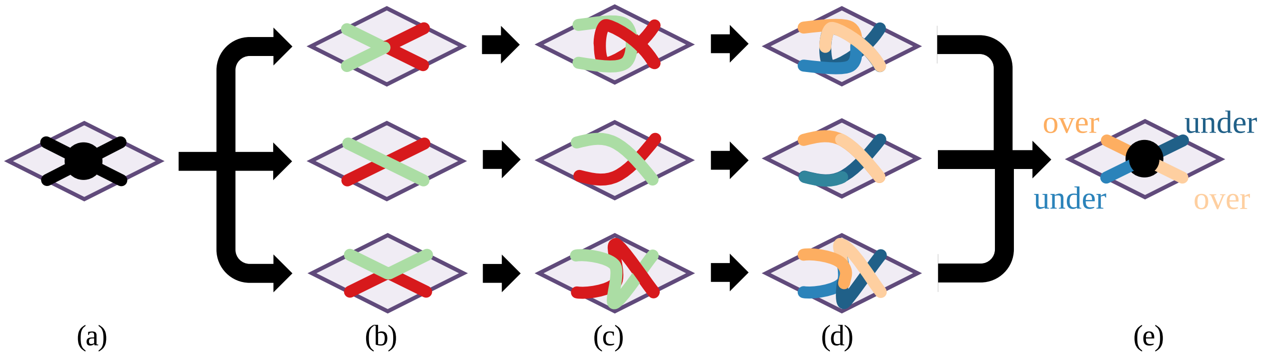

Figure 3: *The weaving cases: (a) portion of graph on a face, (b) the 3 cases that the Eulerian path yields, (c) the way each case is realized, (d) cutting each string segment in half to label which are "over" and which are "under." (e) No matter which case, we can label each piece "over" and "under" the same way.*

This property allows us to "texture" each face of the surface as shown in Figure 2(c), labeling which segments go over and under. Notice that when we follow a segment from one facet to the next, it always switches between "over" and "under". In fact, any surface composed of quadrilaterals can be textured in this way, where the label of a segment switches across an edge of the surface. Therefore, when we traverse the Eulerian path, the segments will consistently alternate between "over" and "under," leading to an aesthetic alternating weave.

At this point, we can concatenate together the sequence of B-spline control points associated with the facets traveled across to form complete the knot. We simply traverse the Eulerian path, and at each vertex, we determine which of the cases in Figure 3 is satisfied. We then append to our sequence the corresponding control points.

B-Spline Generation. Using the control points we just obtained, we generate a closed B-spline using de Casteljau's algorithm [2]. This B-spline is the final path of the string to form the knot.

Exporting. Because we ultimately fabricate our knot using layered manufacturing, we export our model as a STL or OBJ file. To generate this model, we simply sweep a circular cross section along the aforementioned B-spline, generating strips of quadrilaterals. A circular cross section was chosen because the shape of traditional Chinese knots is a circular cord. Finally, we split each quadrilateral into two triangles, and write the vertex information in the appropriate format.

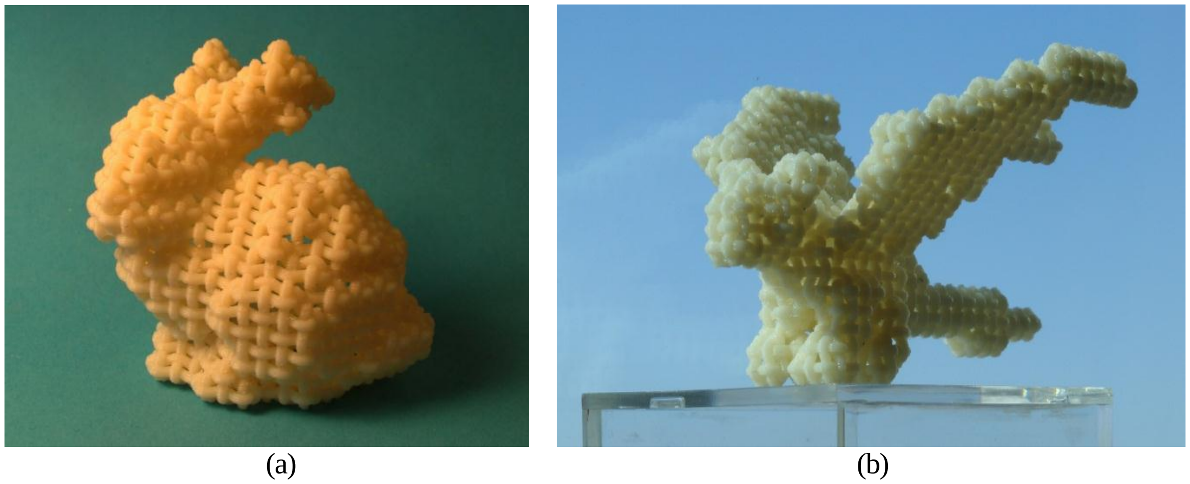

Figure 4: *Photographs of additional* `Knotty` *results printed using the FDM machine: (a) the Stanford bunny, and (b) a more complex dragon model.*

Results

Photographs of our results, fabricated using a FDM machine, appear in Figure 4, which depicts the output of `Knotty` for two different input models: the Stanford bunny and a dragon model. A video illustrating the `Knotty` workflow can be seen at [3], and a video illustrating the Eulerian cycle our algorithm finds can be seen at [4]. `Knotty` is open source, and the code can be seen on GitHub at [5].

Future Work

There are various features that we believe are natural extensions to our existing workflow.

Big Loops. Traditional Chinese knots often include big loops that flare out. In existing animal Chinese knots, these loops have been used to model the head and fins of a turtle and the wings of a dragonfly. Additional features these loops may model are feathers, horns, ears, and tails. This extension is conceptually simple: grab the section of string on a face, and pull out that length of string into a loop.

Surface Generation. Even though voxelization yields a simple surface to work with, it ultimately yields a “boxy” look. The axis-aligned voxels may not obey the “grain” of the input model. For example, in Figure 4(b), a more aesthetically pleasing knot would follow the figure’s body, instead of along the 3D lattice imposed by the voxels.

Path Finding. To find an Eulerian path for the knot, we simply took an existing algorithm that was simple and efficient. However, if we wanted to form this knot manually from an actual string, the chosen over-under weaving pattern would be very tedious to realize. An important aspect of the Chinese knot patterns is the associated set of moves that results in the desired knots. We suspect this results in knots that have a much lower crossing number than the knots generated by our algorithm. To generate such “feasible” knot patterns would require an entirely different algorithmic approach.

Acknowledgements

We thank Professor Carlo H. Séquin for teaching the UC Berkeley class, Procedural Solid Free-Form Modeling and Fabrication, and inspiring us to create `Knotty`, originally our final course project. We also thank Professors Paul K. Wright and Carlo H. Séquin for allowing us utilize UC Berkeley’s FDM machine to fabricate our models.

References

[1] Carl Hierholzer. Ueber die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen* 6 (1): 30–32. 1873.
[2] Hartmut Prautzsch. 1989. A round trip to B-splines via de Casteljau. ACM Trans. Graph. 8, 3 (July 1989), 243-254. DOI=10.1145/77055.77061 http://doi.acm.org/10.1145/77055.77061.
[3] Andrew Lee, Brandon Wang. CS285 Knotty Demo. http://www.youtube.com/watch?v=cKD7Vz54RK4.
[4] Andrew Lee, Brandon Wang. Knotty Path Finding. http://www.youtube.com/watch?v=vYV00L6PanE.
[5] Andrew Lee, Brandon Wang. Knotty. https://github.com/bmwang/knotty.