Beady: Interactive Beadwork Design and Construction

Yuki Igarashi*
University of Tsukuba

Takeo Igarashi†
The University of Tokyo / JST ERATO

Jun Mitani‡
University of Tsukuba / JST ERATO

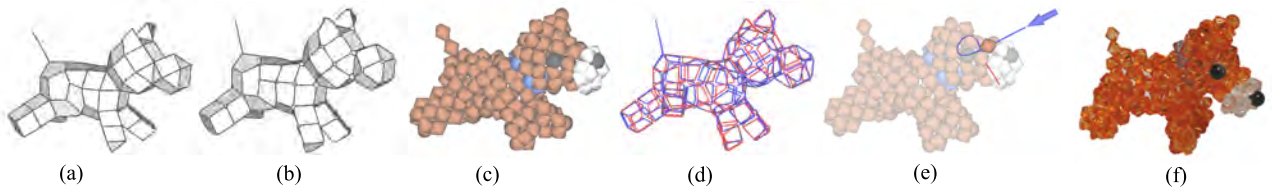

Figure 1: *System overview. (a) The user creates a design model either by modeling from scratch or converting an existing polygonal model. (b) The system runs a simulation to predict the resulting shape. (c) The beadwork model visualizes the expected result. (d) The system computes the wire path. (e) The system provides a step-by-step construction guide. (f) The user manually constructs the physical beadwork.*

Abstract

We introduce the interactive system "Beady" to assist the design and construction of customized 3D beadwork. The user first creates a polygonal mesh model called the design model that represents the overall structure of the beadwork. Each edge of the mesh model corresponds to a bead in the beadwork. We provide two methods to create the design model. One is interactive modeling from scratch. The user defines the mesh topology with gestural interaction and the system continuously adjusts edge lengths by considering the physical constraints among neighboring beads. The other is automatic conversion that takes an existing polygonal model as input and generates a near-hexagonal mesh model with a near-uniform edge length as output. The system then converts the design model into a beadwork model with the appropriate wiring. Computation of an appropriate wiring path requires careful consideration, and we present an algorithm based on face stripification of the mesh. The system also provides a visual step-by-step guide to assist the manual beadwork construction process. We show several beadwork designs constructed by the authors and by test users using the system.

CR Categories: I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction Techniques; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric Algorithms;

Keywords: polygonal meshes, user interface, simplification, face strip, construction guide, wire path planning, beadwork

Links: DL PDF

*e-mail: yukim@acm.org
†e-mail: takeo@acm.org
‡e-mail: mitani@cs.tsukuba.ac.jp

1 Introduction

Beadwork is the art of connecting beads together by wires. While common beadwork is two-dimensional (2D), three-dimensional (3D) beadwork [Maki 2004; Freese 2007] is also popular in oriental regions such as Japan and China. However, the design and construction of 3D beadwork is very difficult. The final shape is defined by the complicated three-dimensional interaction between beads and wires, thus making the beadwork very difficult to design manually. One also needs to specify an appropriate wire path to hold the beads together and to manually insert the wire into the beads one by one, following the path to construct the beadwork. Careful observation of existing beadwork structures shows several geometrically interesting problems, which make beadwork design a technical challenge.

Figure 2: *Example beadworks created from existing 3D models. These are CG shapes rendered using commercial software (e-frontier Shade8).*

This paper presents an interactive computational system to assist the design and construction of original beadwork. Figure 1 shows the overall process. The user first creates a polygonal mesh model, called a *design model*, which represents the overall structure of the beadwork (Figure 1(a)). A bead of the beadwork is represented as an edge (not a vertex) of the design model. The system then converts the design model into a beadwork model by placing beads on the edges with the appropriate wiring (Figure 1(c)). The user specifies the color and shape of individual beads in the beadwork model by using a painting interface. Finally, the user manually constructs the physical beadwork by following the step-by-step instruction generated by the system (Figures 1(e) and (f)).

We provide two methods to create the design model. First, the system provides a specialized modeling interface for the design of a simple polygonal mesh model with near-uniform edge length. The user first builds an approximate shape by combining predefined primitives. Next, the user modifies the shape by applying a basic mesh editing operation such as face extrusion, edge insertion, edge split, edge deletion, or vertex merger. We provide a modeless gestural interaction for applying these operations to support rapid exploration. The system also allows the user to add auxiliary parts (bead chains) to the model.

Second, the system provides an automatic method that converts an existing triangular mesh model to a design model. The system first applies mesh reduction to the model until the number of edges (i.e., number of beads) is equal to the user-defined target number. The system then makes a dual of the triangular mesh, yielding a polygonal mesh with all vertices with valence three and mostly hexagonal faces. Figure 2 shows examples of beadworks created from popular mesh models.

Conversion of the design model into a *beadwork model* consists of two main steps. The first step is geometry computation. The system generates another polygonal mesh model, called a *structure model*, by adding local wire connections between neighboring beads of the design model (Figure 1(b)). The system runs a physical simulation to compute the geometry of the resulting beadwork model by considering the physical interaction among beads and wires. This simulation runs during interactive modeling and updates the shape of the design model after each editing operation. The second step is wire path planning (Figure 1(d)). The wire path should be appropriately defined to efficiently connect the beads. We show that a valid wire path is given as an Eulerian cycle on a graph derived from the structure model. However, a single wire path often makes the manual construction process difficult. We therefore present an algorithm that covers the design model with multiple face strips with short branches, where each strip corresponds to a wire.

The contributions of this work are summarized as follows. (1) We present a mesh modeling user interface specialized for beadwork models by combining gestural operations and physical simulations. (2) We present an algorithm to convert an existing triangular mesh model into a design model appropriate for beadwork. The algorithm applies mesh reduction while applying mesh beautification to obtain a mesh whose edges have almost uniform length. It then makes a dual of the triangular mesh to obtain a near-hexagonal mesh model. (3) We present an algorithm based on face stripification to compute the wire paths for a beadwork model designed by the user. (4) We present a step-by-step construction guide to assist the user in the manual construction process. (5) We show the feasibility and effectiveness of our approach by presenting a real implementation by test users.

2 Related Work

Various interesting systems have been proposed recently to support the fabrication of physical objects using state-of-the-art graphics techniques. One approach is to take a given 3D model as input and generate a physical model as output, such as a 3D polyomino puzzle [Lo et al. 2009], a bas-relief [Weyrich et al. 2007], a paper craft [Mitani and Suzuki 2004; Shatz et al. 2006], a popup card [Li et al. 2010; Li et al. 2011; Iizuka et al. 2011], a stuffed animal [Julius et al. 2005], and a knitted animal [Igarashi et al. 2008]. Another approach is to support the interactive design of a physical model by sketching, such as a garment [Decaudin et al. 2006] or a plush toy [Mori and Igarashi 2007].

The design of a 3D model of a real-world object requires that certain physical constraints be satisfied. For example, a paper toy model has to be represented as a set of developable patches. This kind of restriction is often discussed in the design of architecture consisting of freeform surfaces [Liu et al. 2006; Schiftner et al. 2009]. Pottmann et al. [2010] called this new research area *architectural geometry*. Related to this trend, recent work presented methods to slightly modify the geometry of a polygonal mesh to reduce the number of unique polygons contained in the original model [Singh and Schaefer 2010; Eigensatz et al. 2010; Fu et al. 2010].

Resampling of a polygonal mesh is a well-studied area. Mesh reduction and simplification methods compute a low-resolution approximation of a high-resolution original mesh [Garland and Heckbert 1997; Cohen-Steiner et al. 2004]. However, these methods usually do not consider the uniform distribution of vertices. Retiling and beautification methods seek a uniform distribution of vertices on the original mesh surface [Turk 1992; Markosian et al. 1999]. These methods assume a reasonably high vertex density and can fail to cover a high-curvature region with a low-resolution mesh. Our reduction is closely related to the problems of isotropic remeshing and Voronoi surface tessellation [Yan et al. 2009; Lévy and Liu 2010; Nieser et al. 2010]. The resolution of the resampled mesh in their work is significantly higher than that in our work. Sampling with a low resolution mesh is difficult because the subject could easily loose sharp features that were in the original mesh, which we address in this work. Igarashi et al. [2003] extended the skin algorithm [Markosian et al. 1999] to handle high-curvature regions by adjusting the vertex density depending on the curvature. Our method is also an extension of the skin algorithm, but it uses a constant vertex density.

3 Observation of Existing Beadwork

We investigated existing beadworks in textbooks and in local shops as a basis for the user interface and the algorithm design. We observed that a 3D beadwork model can be seen as a closed manifold polygonal mesh model with mostly uniform edge lengths. Each edge of the polygonal mesh corresponds to a bead, and each face corresponds to a cycle formed by the beads. Faces in a low-curvature region are mostly hexagonal, and pentagons and quads appear occasionally in high-curvature regions. Non-manifold structures such as linear chains are used occasionally, but they can be seen as extra attachments to the main manifold mesh. Most beadwork models consist of a few separate parts, such as head, body, arms, and legs, and use one wire for each part. A wire passes through each bead two times to connect the bead to the left- and right-hand side face of the edge (bead).

As shown in Figure 3 [Maki 2004], textbooks use a 2D diagram to explain how to construct a beadwork model. This is seen as a

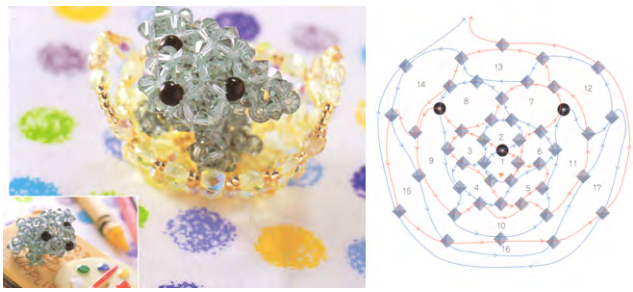

Figure 3: *A beadwork and a construction guide in a textbook. Cited from [Maki 2004] with permission.*

flattening of the polygonal mesh. The construction process starts by putting a bead in the middle of a long wire and then adding beads one by one to both ends of the wire. These two ends of the wire are color coded. One half of the wire is shown in blue and the other half is shown in red. Construction proceeds by forming the faces of the polygon one by one with the red and blue wires.

4 Workflow

This section describes the system from the user's point of view. The user designs a 3D beadwork as a simple polygonal mesh called a design model. An edge of the mesh corresponds to a bead, and thus the edges have mostly uniform lengths (exact length is determined by a physical simulation that considers the interaction between wires and beads). Each face is not necessarily planar. The user can use either an interactive modeling interface or an automatic conversion to obtain a design model. Next, the user defines the appearance of the model by choosing the shape and color of each bead. The system guides the manual construction of the physical beadwork by displaying a step-by-step guide to the user.

4.1 Interactive Modeling

The user creates the geometry of the beadwork model interactively. Starting with basic primitives, the user interactively edits the primitives by using gestural operations. The system runs a physical simulation (Section 5.3) after each editing operation and updates the mesh on the screen to enforce the constraint. The user only specifies the topology of the mesh and the vertex positions are automatically decided.

Combining Basic Primitives: The user starts a new model by combining the predefined primitives. Similar method was used in the modeling system presented by Leblanc et al. [2011], in which the user defines a shape as a collection of generalized cuboid primitives. We provide a regular polyhedron and some semi-regular polyhedrons as primitives. All primitives consist of edges of a unit length. A copy of a selected primitive appears at the center of the screen. The user can also append a primitive to the model by clicking on a face of the model. The system searches for a face with the same number of edges in the selected primitive and pastes it on the model by merging the corresponding faces. Nothing happens when the selected primitive does not have a face with the same number of edges. We chose this approach because we observed that most existing beadwork models can be seen as a combination of these simple primitives. Pasting a primitive is useful for designing the head, arms, and legs of animal models.

Mesh Editing Operations: The user modifies the model by using the mesh editing operations shown in Figure 4. These operations are designed to maintain the mesh as a manifold. The overall shape is already given as a combination of primitives and we expect that these editing operations are only used to adjust the shape locally. We implemented these editing operations as a modeless context-sensitive gestural interaction. The user either clicks on a component (face, edge, or vertex) or draws a short line by a dragging operation, and the system takes the appropriate action. The physical simulation adjusts the overall geometry after each of these operations. We show an example of a modeling sequence in Figure 5.

4.2 Importing Existing Polygonal Models

The user can also import an existing polygonal mesh model into the system. The current implementation requires the model to form a manifold surface and consists of triangular faces. It also expects the mesh to be reasonably well sampled. The system then converts the

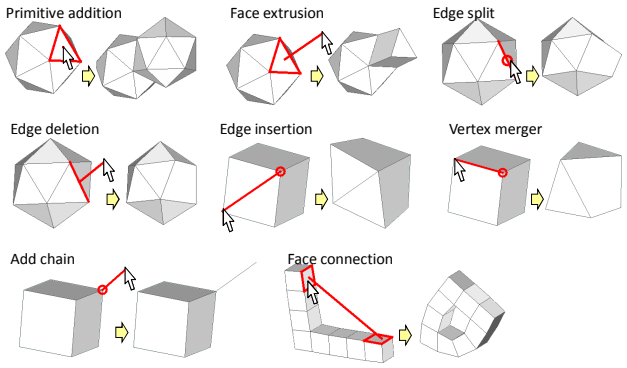

Figure 4: *Mesh editing operations.*

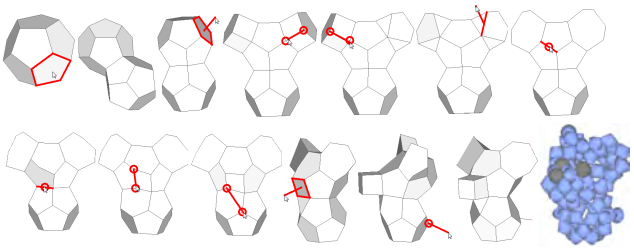

Figure 5: *Example of a modeling sequence using the gestural interface. The user only specifies the topology of the mesh by using the gestural interface. The vertex positions are automatically given by a physical simulation.*

triangular mesh into a near-hexagonal mesh with almost uniform edge lengths (Figure 6). The number of resulting edges is set by the user.

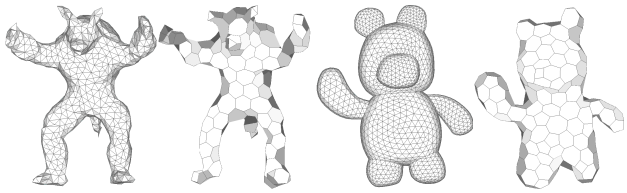

Figure 6: *Examples of automatic conversions from existing 3D models.*

We chose to use hexagonal meshes because we observed that existing beadworks, especially large ones, typically consist of hexagonal faces. This is probably because a hexagonal mesh (honeycomb lattice) is the most efficient structure to hold a flat surface with minimum support materials. We also found that a near-hexagonal mesh obtained as a dual of a triangular mesh yields a more aesthetically pleasing beadwork model after running the physical simulation, as shown in Figure 7. This is probably because wires make the surface as flat as possible in a hexagonal mesh, but it is not possible to do so in a triangular mesh due to insufficient flexibility.

4.3 Appearance Design and Construction Guide

The system shows the resulting beadwork model on the screen during interactive modeling. The user paints the bead model by using paint brush tools (Figure 8). Then, the user selects the bead shape and color, and paints the bead model by dragging the mouse. The

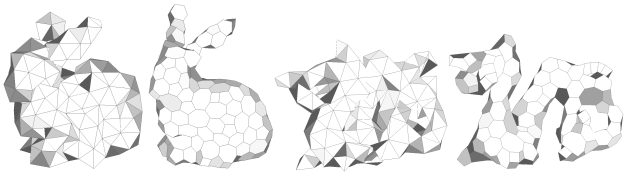

Figure 7: *Comparison of triangular and mostly hexagonal meshes after the physical simulation. The hexagonal mesh is obtained as a dual of the triangular mesh.*

shape and color of the beads designated by the mouse cursor are changed. We also provide a flood fill operation. Our current implementation uses pre-defined shapes and colors, but we plan to support customized bead shapes and colors in the future.

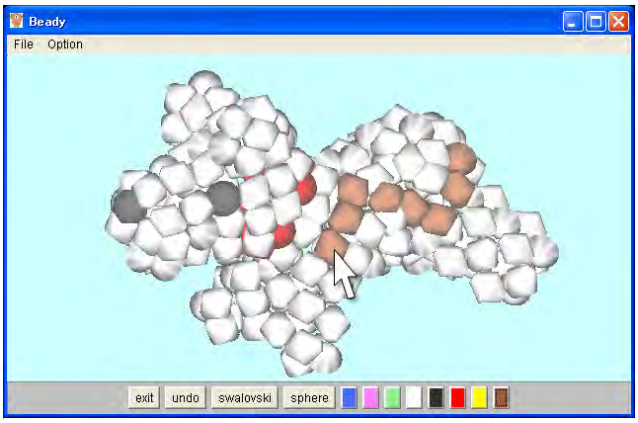

Figure 8: *Screenshot of our system in the painting mode.*

The system guides the manual construction of a physical beadwork by showing step-by-step instruction. The traditional printed beadwork instructions in textbooks use a specialized 2D diagram representation as the guide (Figure 3), but it is very tedious and difficult because the user needs to keep track of the relation between beads in the physical 3D beadwork and those in the 2D instruction. Our step-by-step instruction takes advantage of the expressiveness of interactive 3D graphics and makes it easier to understand the construction procedure. This interactive guide is inspired by recent work on assembly instruction [Agrawala et al. 2003], cutaway illustration [Li et al. 2007], and exploded diagrams [Li et al. 2008].

The construction guide shows which wire passes which bead in each step as 3D graphics. Figure 9 shows an example sequence. The user can view each step from an arbitrary viewing direction. The user presses the "next" button to proceed to the next step and presses the "prev" button to return to the previous step. The construction starts by placing a bead in the middle of a long wire (Figure 9(a)). One side of the wire is blue and the other side is red. The system shows the initial length of each side. The construction proceeds by inserting the blue or the red wire into an existing or new bead. A loop shows the wire that is used in the step (Figure 9(b) through (f)), and an arrow indicates that the wire passes a bead newly added to the beadwork (Figures 9(b), (c), (d), and (f)). Otherwise, the wire passes a bead already included in the beadwork (Figure 9(e)).

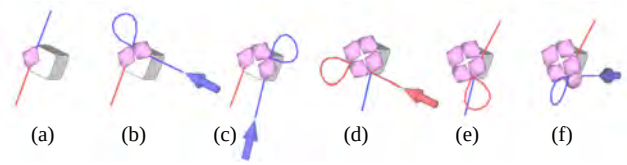

Figure 9: *An example of the visual construction guide. (a) Initial state. (b),(c),(f) Blue wire passes a newly added bead. (d) Red wire passes a newly added bead. (e) Red wire passes an existing bead.*

5 Algorithm

The system maintains three different model representations. The first one is the design model, which is a polygonal mesh model shown to the user during interactive modeling (Figure 1(a)). Each edge of the design model corresponds to a bead and each vertex corresponds to a set of wire segments connecting the surrounding beads. The second representation is the structure model, which is another polygonal mesh model that represents a more detailed structure (Figure 1(b)). This model is used for computing the appropriate shape of the beadwork by considering the physical constraints among beads. A structure model consists of *bead edges*, each of which represents a bead, and *wire edges*, each of which represents a wire fragment between two beads. The third representation is the beadwork model, which consists of beads connected together by wires with an appropriate wire path (Figures 1(c) and (d)). This section first describes the algorithm to convert an existing 3D model to a design model. It then describes the construction of the structure model and the physical simulation of the structure model. Finally, it describes the computation of the wire path necessary for construction of the beadwork model.

5.1 Converting a 3D Model to a Beadwork Model

Our goal is to convert a 3D triangular mesh model into a design model, which is a low-resolution near-hexagonal mesh with almost uniform edge length. The number of edges in the resulting design model is given by the user. The process consists of two steps (Figure 10). First, the system applies mesh reduction to the input mesh, while applying mesh beautification to make the edge length near-uniform. Second, the system constructs a near-hexagonal mesh by making a dual of the reduced triangular mesh.

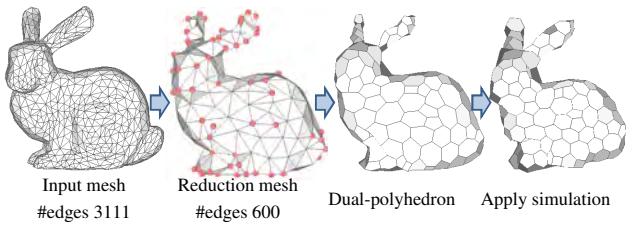

Figure 10: *Automatic conversion.*

The mesh reduction process iteratively removes the shortest edge while applying mesh beautification after each iteration. We use the algorithm described in [Markosian et al. 1999] with the offset set to zero for the beautification. The algorithm repeatedly adjusts the position and connectivity of the mesh to obtain a mesh with a near-uniform edge length and a near-uniform vertex distribution while preserving the original overall shape. Specifically, the system first makes a copy (called a *skin mesh*) of the input mesh (called a *skeleton mesh*) and moves the skin vertices to the centers of surrounding

vertices while updating its connectivity on the fixed skeleton mesh.

We make three changes to the original skin algorithm to satisfy our needs, First, we only apply edge swap and do not apply edge split and edge collapse in the beautification process so that it does not change the number of edges during beautification. This works well when the edges in the input mesh are sufficiently short. Otherwise, it is necessary to refine the mesh as a pre-process. Second, we move the skin vertices only on the skeleton vertices discretely for efficiency. This is also because the skin mesh becomes very coarse in our case and the center of neighboring skin vertices at high curvature region goes deep inside of the skeleton mesh, making the resulting skin mesh non-uniform after pushing the center to the skeleton surface (Figure 11). We therefore discretely evaluate nearby skeleton vertices and pick the one that produces the most uniform distribution of skin vertices on the skeleton surface. Specifically, for each nearby skeleton vertex we compute distances to the neighboring skin vertices and pick the maximum. We then take the skin vertex that returns the minimum of these maximums. This discrete method works for us because we only need final mesh connectivity and specific vertex positions are less important.

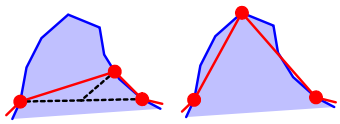

Figure 11: *Vertex relocation. The original Skin algorithm moves the vertex to the center of neighboring vertices and then pushes it to the skeleton surface (left). Our method discretely evaluates nearby skeleton vertices and picks the one that minimizes the largest distance to the neighboring skin vertices (right).*

Finally, we explicitly detect cases where a protrusion of the skeleton mesh sticks out of the skin mesh and pull the skin mesh to the tip of the protrusion to cover it. This problem occurs because the skin vertices always stay on the skeleton mesh but skin edges and faces can go inside of the skeleton mesh (Figure 12). To fix this efficiently, we monitor the distance from each skeleton vertex to the nearest skin vertex and constrain the skin vertex to the skeleton vertex position when the distance exceeds a predefined threshold α (2-5% of bounding box of the input mesh). The original paper used local search to keep track of a skeleton vertex nearest to each skin vertex. We use the same algorithm to keep track of a skin vertex nearest to each skeleton vertex.

Construction of the near-hexagonal mesh is straightforward. The system simply replaces vertices and faces of the triangular mesh with faces and vertices of the hexagonal mesh, respectively. Each vertex of the hexagonal mesh has valence three by construction, and each face typically has five to seven edges. The system then performs a physical simulation (Section 5.3) to obtain the final geometry. Strictly speaking, we want to have uniform edge lengths in the hexagonal dual mesh, so the edge length of the triangular mesh should vary depending on the vertex valence. However, we ignore this in the current implementation because the valences are typically 5 to 7 and the differences are small. In addition, uniformity of the edge length in the hexagonal mesh is not rigorous, because the length changes after the physical simulation.

5.2 Construction of the Structure Model

The structure model is generated by adding a wire edge to each corner of a face in the design model (Figure 13, left). Each valence n vertex is replaced by n wire edges and n new vertices between them. This construction naturally connects two wire edges to each

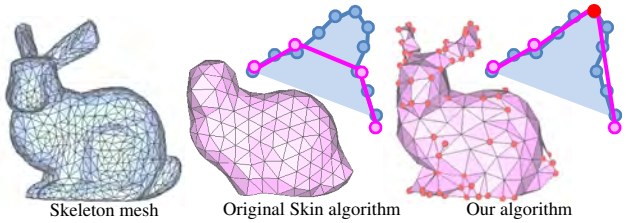

Figure 12: *The original Skin algorithm can fail to cover a protrusion in the skeleton mesh. We explicitly identify such a case and fix a skin mesh vertex to the tip of the protrusion.*

end of a bead edge so that a wire passes through each bead two times, as in existing beadworks. The initial position of a new vertex is given by moving the original vertex slightly toward the center of the corresponding edge. We handle linear chains separately after applying the above procedure to the design model (Figure 13, right). The system connects each linear chain to one of the wire edges that share a vertex in the design model, i.e., the wire edge is split and connected to an end of a linear chain.

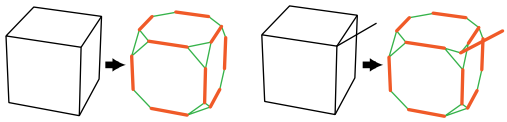

Figure 13: *Construction of the structure model without (left) and with (right) a chain. Orange and green edges are bead edges and wire edges, respectively.*

5.3 Physical Simulation of the Structure Model

The system adjusts the shape of the structure model to satisfy the physical constraints among neighboring beads: beads do not intersect each other and beads connected by a wire are as closely located as possible. We achieve this by running a simple mass-spring simulation cycle 500 times after each mesh editing operation. The system then updates the design model by moving its vertex positions to the center of corresponding vertices of the structure model.

We apply the following three forces to the vertices of the structure model in the simulation. The first force is edge springs to keep the edges to the desired lengths. The rest length of a bead edge is set to the length of the bead, and the rest length of a wire edge is set to zero. The second force is angular springs that keep the wires as straight as possible. The system examines the angles between each wire edge and the bead edges connected to it, and applies a force to keep them straight. The third force is a repulsion force to prevent the neighboring beads from penetrating each other.

5.4 Computing the Wire Paths

A wire path defines the order in which a wire connects the beads to hold them in place. Based on careful examination of existing beadwork designs, we made the following three goals for the algorithm. First, wires should pass through a bead for the minimum number of times necessary to hold the bead. In our case, it is always two, except for chains. Second, the minimum number of wires should be used to reduce the need for cutting and tying a wire. Third, the wire path should be designed so that the bead in the beadwork during the construction is as *stable* as possible. We consider a bead to be *stable* when the position of the bead is stably fixed to a specific location by the wires; that is, the bead belongs to a completed face (Figure 14).

An unstable bead makes manual construction extremely difficult because the user has to manually hold it. It is therefore crucial to design a wire path that minimizes the occurrence of unstable beads during construction. The second and third requirements conflict with each other, and so our method balances the two while always satisfying the first requirement.The following wire path planning algorithm works for an arbitrary manifold surface regardless of the vertex valence or the surface genus.

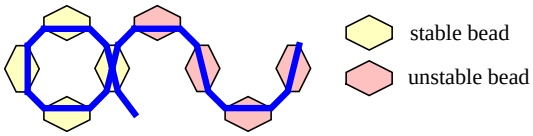

Figure 14: *Stable and unstable beads appearing during construction.*

Given the design model (Figure 15(a)), we start the process with the structure model (Figure 15(b)). The structure model defines the local wire connectivity. A global wire path (Figure 15(c)) is given as a loop that meets all wire edges once and all bead edges twice. This is the Eulerian cycle of the wire graph (Figure 15(d)), obtained by contracting the bead edges into vertices, with an additional constraint that a wire going into a bead should go out from the other side of the bead. The existence of an Eulerian cycle is guaranteed by the construction because each bead edge always has four wire edges. Various methods exist for obtaining such an Eulerian cycle.

However, an arbitrary Eulerian cycle (e.g., Figure 15(d)) can be inconvenient in the manual construction process because it can cause many unstable beads during construction (Figure 15(e)). We carefully examined existing beadwork designs and found that the problem of reducing unstable beads can be solved by using a face strip (Figures 15(f) and (g)). The design model is covered by a face strip, and then a wire path is placed so that it completes the faces in the strip one by one. This method makes all the beads in the previously visited faces stable during construction (Figure 15(h)). In cases that the model is not fully covered by a single strip, the system adds remaining faces as a branch to the strip, and places the wire path so that it strays off from the strip, visits the faces in the branch, and goes back to the strip (Figure 16). Such branches cause unstable beads, and so we restrict the length of a branch to one. We create independent face strips with branches for the remaining area until strips and their branches cover all the faces. Figure 17 shows examples of the stripification results. The first strip covers most of the surface and additional strips cover protrusions such as ears and arms.

The computation of a single face strip without any branch corresponds to finding a Hamiltonian path on the dual graph obtained by replacing the faces with vertices of the design model. However, it is not guaranteed that such a path exists. A practical alternative is constructing a spanning tree with some branches, and various heuristics have been presented for improving the rendering performance [Evans et al. 1996] and the data compression [Taubin and Rossignac 1998]. We also use a heuristic method specialized for our problem. Our method is a greedy expansion starting from a base face, followed by adding faces adjacent to the face at the end of the strip. We select one among the candidate faces so that the boundary length is minimized. The boundary length is given as the total length of the edges between the current face strip and the remaining faces. This keeps the intermediate beadwork model compact, thus making manual construction easier. We apply backtracking when the expansion is trapped into a dead end. We run this greedy expansion search starting from all faces and use the most successful result.

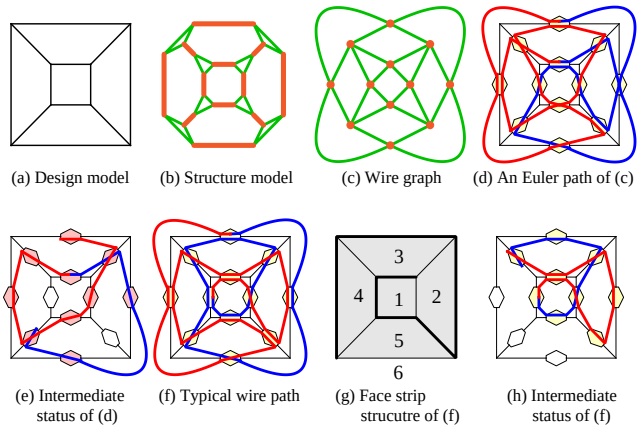

Figure 15: *Details of the wire path-planning algorithm.*

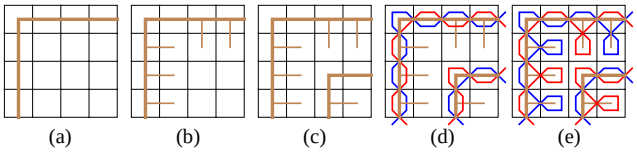

Figure 16: *Stripification process. (a) Initial strip. (b) Adding branches. (c) Second strip and its branches. (d) Wire paths for the strips. (e) Modified wire paths to cover the branches.*

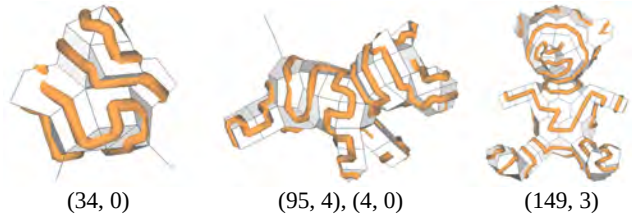

Figure 17: *Stripification results. The thick tan lines show the main strips and the thin lines show the branches. Each tuple corresponds to a strip (the number of faces in the strip and the number of its branches).*

6 Results

The prototype system is implemented using Java on a laptop (1.2 GHz CPU, 2 GB RAM). Figure 18 show some example beadwork models designed and constructed by using the system. Designing these models required 10 to 20 minutes, which included creative experimentation and exploration. Manual construction of each model by following the guide required a few hours. The beadwork bear is an exception: it was bought at a local shop. We created the model by referring to the product, which required approximately 90 minutes. Table 1 shows the performance measurements of the designed models in Figures 1 and 18. Modeling and wire path planning were completed almost instantly in most cases, whereas they lasted a few seconds in the complicated case.

Figure 2 shows example beadworks created from existing 3D models. These are CG models rendered using commercial software. We used 600 beads for these models to obtain reasonable results. Ideally we wanted to use fewer beads to make the manual construction easier, but it was difficult to obtain aesthetically pleasing results by using the mesh reduction approach. Extreme simplification re-

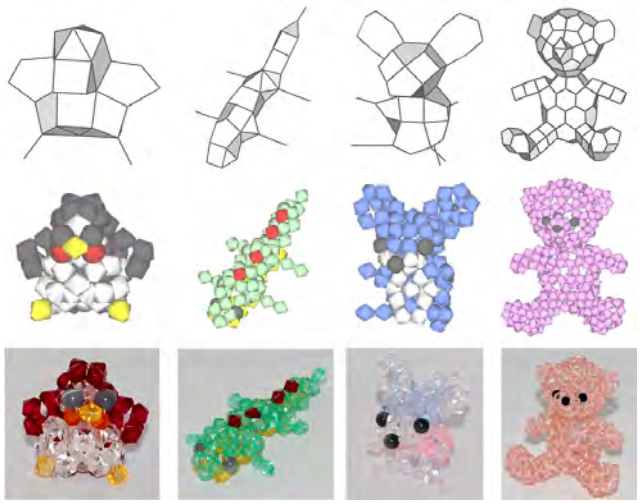

Figure 18: *Example models designed by the authors. Top: design models. Middle: beadwork models. Bottom: real beadworks.*

Table 1: *Performance measurements of the models in Figures 1 and 18.*

	Dog	Penguin	Lizard	Mouse	Bear
# beads	203	66	77	96	372
Simulation (sec.)	1.3	0.5	0.5	0.7	2.5
Wire (sec.)	0.7	0.2	0.1	0.3	5.6

quires artistic skill, as in the design of low-resolution bitmap images (pixel arts), and it is necessary to develop completely a different algorithm. Physical construction took approximately 15 hours each for the bunny and dragon (Figure 19). Table 2 shows the performance measurements and relevant statistics. Table 3 shows that the converted design models are mostly hexagonal.

7 User Experience

We asked five test users to try the interactive modeling system and obtained their feedback. They used interactive geometry modeling to create their original design. The users were university students in the Department of Computer Science who did not have any experience in making beadwork. The users were given a five-minute tutorial and free practice for another five minutes. All of the subjects learned the operations instantly without any difficulties. Then, we asked the users to design their own original beadwork until they were satisfied with the design. Figure 20 shows the results designed

Table 2: *Performance measurements and relevant statistics to convert existing 3D models into beadwork models in Figure 2.*

	Dragon	Armadillo	Horse	Squirrel	Bunny	Bear
Init #edges	2796	2992	3000	3063	3111	5964
# beads	600	600	600	600	600	600
distance α	0.02	0.05	0.05	0.05	0.02	0.03
fixed #vertices	102	23	11	3	81	20
Convert (ms)	2930	3086	3013	2943	3092	7939
· Reduction	1462	1585	1522	1426	1589	6396
· Dual	28	25	23	24	34	26
· Sim	1440	1476	1468	1493	1469	1517
Wire (ms)	7468	10833	14527	4677	14006	7768

Figure 19: *Photo of the real beadworks. The diameter of a bead is 4mm.*

by the subjects and the time required to design each: the first three are photos of the real beadwork and the others are CG models. Table 4 shows the gestural operations for each beadwork. The most used operation during the study was "Add primitive." This operation was used to design the rough forms at the beginning. Other operations were used to refine the details. The "Add chain" operation was used to add detailed parts at the finish. Although this operation generates a non-manifold mesh, it served an important role in designing the beadwork.

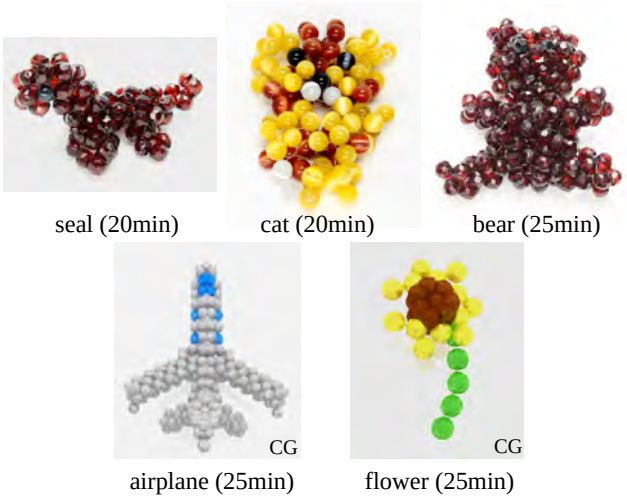

Figure 20: *3D models designed by the subjects and the time required to design each. All models were the subjects' original designs.*

Most comments from the subjects were positive, such as "It was very helpful that the system displayed the bead 3D model when I edited the mesh model," and "I could concentrate on designing shapes without thinking about the structure of the beadwork." We believe that our system successfully enabled these novice users to design their own original beadwork. Some users offered suggestions, such as "It would be nice if the system provided an operation to add flat planes," and "It was difficult to control the location of beads precisely while I completed my beadwork." These suggestions are possible enhancements for varying the modeling operations. Three users constructed the designed beadworks manually by referring to the step-by-step guide. We provided large beads (approximately 10 mm in diameter) to make the work easy. The beadworks of the first three models in Figure 20 were successfully constructed within a half day.

Table 3: *The ratio of hexagons and other polygons in the converted design models in Figure 2.*

Model \ # of sides per face	3	4	5	6	7	8	9	10
Bunny	3 (1%)	13 (6%)	36 (18%)	106 (52%)	33 (16%)	8 (4%)	2 (1%)	1 (0%)
Dragon	9 (5%)	20 (10%)	23 (12%)	98 (49%)	31 (16%)	10 (5%)	4 (2%)	1 (1%)
Bear	1 (0%)	6 (3%)	42 (21%)	114 (56%)	34 (17%)	4 (2%)	1 (0%)	0 (0%)
Armadillo	10 (5%)	12 (6%)	27 (13%)	100 (50%)	42 (21%)	7 (3%)	3 (1%)	1 (0%)
Horse	5 (2%)	5 (2%)	30 (15%)	127 (63%)	29 (14%)	4 (2%)	2 (1%)	0 (0%)
Squirrel	3 (1%)	5 (2%)	49 (24%)	107 (53%)	24 (12%)	10 (5%)	4 (2%)	0 (0%)

By interviewing professional beadwork designers, we confirmed that our system is very useful for designing beadwork, even for professionals. In their company, a 3D polygon model was first created using a standard 3D modeling software. A professional designer then designed a beadwork model based on the 3D model. In this way, it was confirmed that both methods (gestural modeling and automatic conversion) are compatible with current practice and are immediately applicable.

Table 4: *Gestural operations for each beadwork in Figure 20.*

Users	Seal	Cat	Bear	Airplane	Flower
Undo	27	21	47	19	39
Add primitive	3	21	10	25	67
Add chain	5	13	6	9	25
Edge deletion	31	0	5	16	2
Face extrusion	17	5	41	32	4
Edge split	2	0	2	21	17
Vertex merge	15	0	0	0	0
Edge insertion	0	8	16	0	3

8 Limitations and Future Work

Our current interactive modeling interface was designed for general polygonal meshes and not specifically designed for hexagonal meshes. We did not find this very problematic because the users only designed small beadworks in interactive editing and the requirement for a hexagonal mesh is not very important in such small beadworks. A hexagonal mesh is important for larger beadworks and it might be useful to implement some mechanism to facilitate construction and editing of hexagonal meshes in the future.

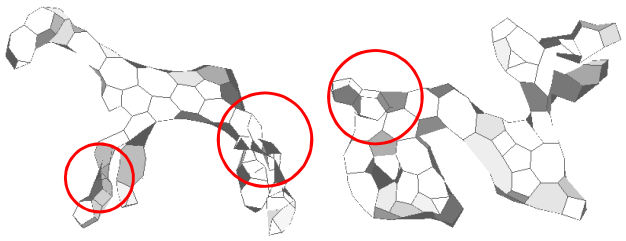

Figure 21: *Failure cases of simplification. The number of edges is* 200 *and the parameter* α *is* 0.02 *in both models.*

Our target in this work is very basic beadwork models. Many more techniques are used in real beadwork, such as combining beads of different sizes and creating non-manifold structures other than simple open chains. It is our future work to support these advanced techniques. Wire may need to visit some edges more than twice when there are non-manifold vertices in the mesh. Most beadwork is symmetric, so automatic symmetrization of the model during interactive editing could be very helpful. Symmetry could also be useful for computing better wire paths.

The final reduced mesh shape is determined by the combination of parameters and the total number of beads specified by the user. If the user specifies a very small number of beads, the final mesh can contain self-intersections, as shown in Figure 21, because we currently do not consider collisions between faces and edges. If the input model contains thin parts, such as the legs of a horse, it is better to use chains, as shown in the mouse model in Figure 18. We plan to extend the conversion algorithm to produce chains automatically by considering the skeletal structure of the input mesh.

Creating complicated objects by assembling simple primitives with wires is a very general idea and the method presented in this paper could be useful for designing physical objects other than beadwork. For example, the same user interface and algorithm can be used for the design of a model consisting of equal-length straws tied together by a string, as shown in Figure 22. An important difference is that quads and pentagons are not stable in a straw model. Nevertheless, we plan to extend our techniques to support the design of such physical objects, including a potential form of architecture.

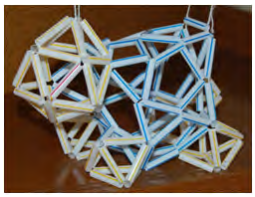

Figure 22: *Straw model.*

Acknowledgments

We are grateful to Maneesh Agrawala for his helpful advice. We would also like to thank the professional beadwork designers, Meiko Aikawa, Katsuko Uchida, Kimiko Nara and Yumiko Satou of Tougenkyou, Inc., for their insightful comments. Finally, we wish to express our appreciation to the test users for their cooperation and to the anonymous reviewers for their valuable feedback. This work was supported in part by a Grant-in-Aid for JSPS Fellows.

References

AGRAWALA, M., PHAN, D., HEISER, J., HAYMAKER, J., KLINGNER, J., HANRAHAN, P., AND TVERSKY, B. 2003. Designing effective step-by-step assembly instructions. *ACM Trans. Graph. 22*, 3, 828–837.

COHEN-STEINER, D., ALLIEZ, P., AND DESBRUN, M. 2004. Variational shape approximation. *ACM Trans. Graph. 23*, 3, 905–914.

DECAUDIN, P., JULIUS, D., WITHER, J., BOISSIEUX, L., SHEFFER, A., AND CANI, M.-P. 2006. Virtual Garments: A fully

geometric approach for clothing design. *Comput. Graph. Forum 25*, 3, 625–634.

EIGENSATZ, M., KILIAN, M., SCHIFTNER, A., MITRA, N. J., POTTMANN, H., AND PAULY, M. 2010. Paneling architectural freeform surfaces. *ACM Trans. Graph. 29*, 4, 45:1–45:10.

EVANS, F., SKIENA, S., AND VARSHNEY, A. 1996. Optimizing triangle strips for fast rendering. In *Proc. of the 7th Conference on Visualization '96*, 319–326.

FREESE, A. 2007. *Lovable Beaded Creatures*. Sterling Pub Co Inc.

FU, C.-W., LAI, C.-F., HE, Y., AND COHEN-OR, D. 2010. K-set tilable surfaces. *ACM Trans. Graph. 29*, 4, 44:1–44:6.

GARLAND, M., AND HECKBERT, P. S. 1997. Surface simplification using quadric error metrics. In *Proc. of SIGGRAPH '97*, 209–216.

IGARASHI, T., AND HUGHES, J. F. 2003. Smooth meshes for sketch-based freeform modeling. In *Proc. of the 2003 Symposium on Interactive 3D Graphics*, 139–142.

IGARASHI, Y., IGARASHI, T., AND SUZUKI, H. 2008. Knitting a 3D model. *Comput. Graph. Forum 27*, 7, 1737–1743.

IIZUKA, S., ENDO, Y., MITANI, J., KANAMORI, Y., AND FUKUI, Y. 2011. An interactive design system for pop-up cards with a physical simulation. *Vis. Comput. 27*, 6-8, 605–612.

JULIUS, D., KRAEVOY, V., AND SHEFFER, A. 2005. D-charts: Quasi-developablemesh segmentation. *Comput. Graph. Forum 24*, 3, 581–590.

LEBLANC, L., HOULE, J., AND POULIN, P. 2011. Modeling with blocks. *Vis. Comput. 27*, 6-8, 555–563.

LÉVY, B., AND LIU, Y. 2010. Lp centroidal voronoi tessellation and its applications. *ACM Trans. Graph. 29*, 4, 119:1–119:11.

LI, W., RITTER, L., AGRAWALA, M., CURLESS, B., AND SALESIN, D. 2007. Interactive cutaway illustrations of complex 3D models. *ACM Trans. Graph. 26*, 3.

LI, W., AGRAWALA, M., CURLESS, B., AND SALESIN, D. 2008. Automated generation of interactive 3D exploded view diagrams. *ACM Trans. Graph. 27*, 3, 101:1–101:7.

LI, X.-Y., SHEN, C.-H., HUANG, S.-S., JU, T., AND HU, S.-M. 2010. Popup: automatic paper architectures from 3D models. *ACM Trans. Graph. 29*, 4, 111:1–111:9.

LI, X.-Y., JU, T., GU, Y., AND HU, S.-M. 2011. A geometric study of v-style pop-ups: theories and algorithms. *ACM Trans. Graph. 30*, 4, 98:1–98:10.

LIU, Y., POTTMANN, H., WALLNER, J., YANG, Y.-L., AND WANG, W. 2006. Geometric modeling with conical meshes and developable surfaces. *ACM Trans. Graph. 25*, 3, 681–689.

LO, K.-Y., FU, C.-W., AND LI, H. 2009. 3D polyomino puzzle. *ACM Trans. Graph. 28*, 5, 157:1–157:8.

MAKI, Y. 2004. *The Motifs of Puppy*. Gakken (in Japanese).

MARKOSIAN, L., COHEN, J. M., CRULLI, T., AND HUGHES, J. 1999. Skin: a constructive approach to modeling free-form shapes. In *Proc. of SIGGRAPH '99*, 393–400.

MITANI, J., AND SUZUKI, H. 2004. Making papercraft toys from meshes using strip-based approximate unfolding. *ACM Trans. Graph. 23*, 3, 259–263.

MORI, Y., AND IGARASHI, T. 2007. Plushie: an interactive design system for plush toys. *ACM Trans. Graph. 26*, 3, 45:1–45:8.

NIESER, M., PALACIOS, J., POLTHIER, K., AND ZHANG, E. 2010. Hexagonal global parameterization of arbitrary surfaces. In *ACM SIGGRAPH ASIA 2010 Sketches*, 5:1–5:2.

POTTMANN, H., HUANG, Q., DENG, B., SCHIFTNER, A., KILIAN, M., GUIBAS, L., AND WALLNER, J. 2010. Geodesic patterns. *ACM Trans. Graph. 29*, 4, 43:1–43:10.

SCHIFTNER, A., HÖBINGER, M., WALLNER, J., AND POTTMANN, H. 2009. Packing circles and spheres on surfaces. *ACM Trans. Graph. 28*, 5, 139:1–139:8.

SHATZ, I., TAL, A., AND LEIFMAN, G. 2006. Paper craft models from meshes. *Vis. Comput. 22*, 9, 825–834.

SINGH, M., AND SCHAEFER, S. 2010. Triangle surfaces with discrete equivalence classes. *ACM Trans. Graph. 29*, 4, 46:1–46:7.

TAUBIN, G., AND ROSSIGNAC, J. 1998. Geometric compression through topological surgery. *ACM Trans. Graph. 17*, 2, 84–115.

TURK, G. 1992. Re-tiling polygonal surfaces. *SIGGRAPH Comput. Graph. 26*, 2, 55–64.

WEYRICH, T., DENG, J., BARNES, C., RUSINKIEWICZ, S., AND FINKELSTEIN, A. 2007. Digital bas-relief from 3D scenes. *ACM Trans. Graph. 26*, 3.

YAN, D.-M., LÉVY, B., LIU, Y., SUN, F., AND WANG, W. 2009. Isotropic remeshing with fast and exact computation of restricted voronoi diagram. In *Proc. of the Symposium on Geometry Processing*, 1445–1454.