HAL
open science

WrapIt: Computer-Assisted Crafting of Wire Wrapped Jewelry

Emmanuel Iarussi, Wilmot Li, Adrien Bousseau

To cite this version:

Emmanuel Iarussi, Wilmot Li, Adrien Bousseau. WrapIt: Computer-Assisted Crafting of Wire Wrapped Jewelry. ACM Transactions on Graphics, 2015, Proceedings of ACM SIGGRAPH, 34 (6), 10.1145/2816795.2818118 . hal-01261459

HAL Id: hal-01261459

https://inria.hal.science/hal-01261459v1

Submitted on 25 Jan 2016

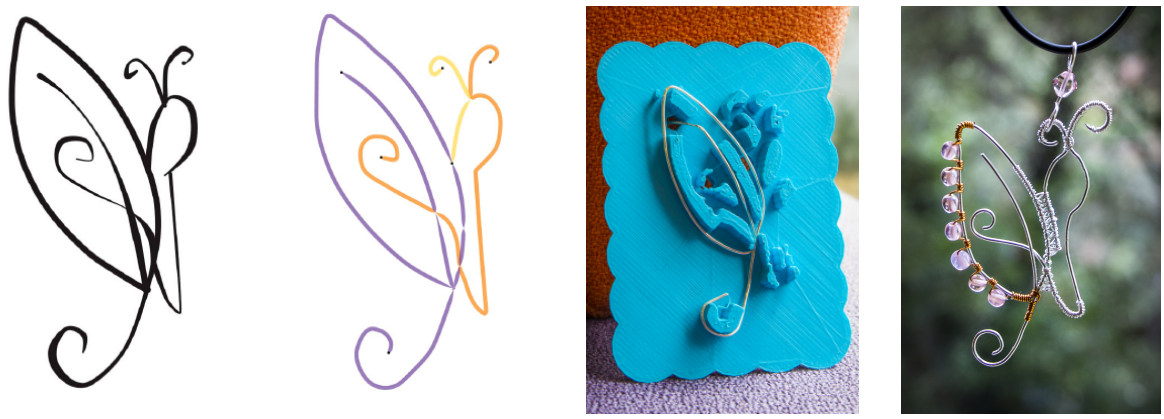
HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

WrapIt: Computer-Assisted Crafting of Wire Wrapped Jewelry

Emmanuel Iarussi[1] Wilmot Li[2] Adrien Bousseau[1]

[1] Inria [2] Adobe Research



(a) Input bitmap (b) Labeling (c) Printed jig (d) Fabricated piece

Figure 1: *Our system allows novices to create a variety of custom jewelry. Following aesthetic and fabrication principles, our algorithm decomposes an input line drawing (a) into smooth, well-connected paths that cover each line in the drawing exactly once (b). We extrude support walls inside the sharp turns of the paths to create a physical* jig *that guides wire wrapping (c). Assembling the three wires of this design yields a butterfly pendant (d).*

Abstract

Wire wrapping is a traditional form of handmade jewelry that involves bending metal wire to create intricate shapes. The technique appeals to novices and casual crafters because of its low cost, accessibility and unique aesthetic. We present a computational design tool that addresses the two main challenges of creating 2D wire-wrapped jewelry: decomposing an input drawing into a set of wires, and bending the wires to give them shape. Our main contribution is an automatic wire decomposition algorithm that segments a drawing into a small number of wires based on aesthetic and fabrication principles. We formulate the task as a constrained graph labeling problem and present a stochastic optimization approach that produces good results for a variety of inputs.

Given a decomposition, our system generates a 3D-printed custom support structure, or *jig*, that helps users bend the wire into the appropriate shape. We validated our wire decomposition algorithm against existing wire-wrapped designs, and used our end-to-end system to create new jewelry from clipart drawings. We also evaluated our approach with novice users, who were able to create various pieces of jewelry in less than half an hour.

© Iarussi et al. | ACM 2015. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in ACM TOG, http://dx.doi.org/10.1145/2816795.2818118.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations; I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques;

Keywords: computational design, line drawing segmentation, Eulerian path, jewelry, wire wrapping, jig

1 Introduction

Jewelry is one of the oldest and most prevailing forms of crafting, with the earliest known examples dating from almost 100,000 years ago. Traditionally, jewelry making has been a largely manual process that involves a range of techniques, such as mold-making, metal working, painting, etc. Today, while most commercial jewelry production leverages computer-aided design and manufacturing technology, there remains a large class of jewelry that continues to be made by hand. In recent years, interest in hand crafted jewelery has increased significantly with the growth of popular online crafting marketplaces such as Etsy and ArtFire, which allow independent artists to sell their work directly to consumers. The goal of our work is to enable a broader range of users to create their own, customized handmade jewelry.

We focus on a specific style of jewelry making called *wire wrapping* that involves bending and connecting metal wires to create complex shapes (Figure 1d). Wire wrapping is one of the most popular forms of hand crafted jewelry; e.g., Etsy.com returns over 220,000 “wire wrapped jewelry” results. Moreover, since it involves affordable materials and does not require melting or soldering, wire wrapping is particularly appealing to casual crafters. Crafting sites like Instructables.com include hundreds of wire wrapping tutorials for creating a variety of jewelry and other ornaments. However, since most tutorials provide instructions for a given piece of jew-

elry and are hard to generalize, novices are limited to creating a fixed set of designs. We present a computational design tool that empowers novices to create wire-wrapped jewelry from their own designs. Since wire wrapping can be viewed as a form of line drawing (where wire takes the place of ink or graphite), we allow users to specify their target designs as line drawings.

There are two major challenges in designing and fabricating wire-wrapped jewelry from line drawings.

Wire decomposition. The first and most critical step is to create a *wire decomposition* of the drawing into the appropriate set of wires. Since pieces made of many wires are unstable and difficult to assemble, good decompositions usually consist of few wires. Yet, many shapes cannot be represented with a single wire without doubling back over parts of the drawing, which often detracts from the aesthetics of the resulting jewelry. Moreover, it is hard to bend a single piece of wire to create sharp angles. Effective decompositions require balancing these constraints and objectives.

Wire bending. Given a wire decomposition, the next step is to bend each piece of wire to match the shape of the path specified in the design. To help create smooth curves, jewelry makers often wrap wires around tools called *jigs*. Several companies like WigJig [2015] sell generic boards and cylindrical pegs that support custom jig configurations. However, such pre-defined kits are not flexible enough to create arbitrary shapes. For instance, cylinders can only constrain bends of constant curvature, and target shapes with multiple nearby bends can result in collisions between the pegs.

We propose an end-to-end design and fabrication system to help novices address the above challenges. Our main contribution is an algorithm to automatically decompose an input drawing into wires. We formulate the decomposition task as a graph labeling problem, where the labels define groups of line segments, each of which should be "drawn" with a single wire (Figure 1b). Our aesthetic and fabrication objectives result in both soft and hard constraints, some of which have a global impact on the labeling. We describe a stochastic optimization that handles these constraints.

Another key feature of our system is the automatic generation of a custom 3D-printed jig from a given wire decomposition. Our jigs include a set of support walls that constrain the shape of the wire to match the design. We generate support walls strategically only on curvy portions of each wire path (Figure 1c) so that there is enough free space to easily manipulate the wire. Minimizing the amount of support geometry also reduces the physical material required to 3D print the jig. As extra guidance, our system generates instructions to help plan and execute the wire wrapping and assembly. Note that our approach differs from most recent work on computer-aided fabrication since our goal is not to 3D print the final jewelry piece, but rather to create an intermediate support structure that facilitates the hand wrapping process while preserving the joy of crafting.

We evaluated our approach with novice users with little or no training in jewelry design and crafting. Participants took several minutes to segment a design by hand. While they obtained similar solutions to ours on simple designs, our algorithm yields solutions with fewer wires or increased robustness on more complex examples. All participants were able to create a piece of jewelry in less than half an hour using a wire decomposition and custom jig generated with our system. Participants commented that our system provides clear instructions and that the jig is very helpful in creating the final wire-wrapped jewelry.

2 Related Work

Professional jewelry is a major industry for which dedicated CAD systems exist, such as GemVision Matrix and ArtCAM JewelSmith to name a few. These systems provide advanced features to model common types of jewelry (rings, bracelets, pendants, etc), to decorate shapes with relief and gems, as well as to render the design realistically. Our work targets a different audience: novice crafters who wish to create their own unique jewelry with affordable materials and hands-on techniques. Studies on the *Do-It-Yourself* community suggest that manipulating materials by hand contributes to the pleasure and pride of crafting [Tanenbaum et al. 2013].

Traditional crafting, such as wood working, sewing or metal smithing, require significant expertise both to manipulate physical materials and anticipate their behavior. Computer-aided design has the potential of making such crafting accessible to novices by simulating the end artifact as well as guiding its fabrication [Schmidt and Ratto 2013]. A typical example is *Plushie* [Mori and Igarashi 2007], an interactive system to design plush toys that combines inflation simulation and geometric constraints to generate developable patches that reproduce a target shape after sewing. Skouras et al. [2012; 2014] follow a similar workflow to assist the design of inflatable structures. *Holly* [Igarashi and Igarashi 2010] allows users to perform stencil design, automatically detecting islands and adding bridges to connect them to the main sheet to be cut. Other recent fabrication-oriented design systems assist the creation of furniture [Umetani et al. 2012], pop-up cards [Li et al. 2011], paper airplanes [Umetani et al. 2014], mechanical characters [Coros et al. 2013]. Closer to our application domain, Beady [Igarashi et al. 2012] assists the construction of customized 3D beadwork by decomposing a 3D model into strips of beads while simulating physical interactions between neighboring beads. We adopt a similar methodology to this family of work but apply it to the different domain of wire-wrapped jewelry. Given a line drawing, our tool automatically generates a decomposition into metal wires that satisfies artistic and fabrication constraints.

Computational tools have also been proposed to assist the assembly of physical objects. While early work focuses on the generation of instructions for existing models [Agrawala et al. 2003], recent work integrates assembly constraints as part of the design goals, for instance to create sculptures made of planar slices [Hildebrand et al. 2012] and interlocking furniture [Fu et al. 2015]. Craftsmen also often rely on intermediate support structures, or scaffolds, to assist assembly. Inspired by masonry techniques, Deuss et al. [2014] rely on temporary chains to guaranty stability during the assembly sequence of self-supporting structures. Temporary wooden structures have also been used for the fabrication of wire mesh sculptures [Garg et al. 2014]. Taking inspiration from traditional wire-wrapping techniques, our system automatically generates custom support structures, called *jigs*, to guide the bending of wires.

One of the objectives of our algorithm is to wrap the wire over each line of the input drawing exactly once. Similar objectives appear in the computational design of continuous line drawings [Wong and Takahashi 2011] and related travelling-salesman art [Kaplan and Bosch 2005]. However, these algorithms aim to find a *single* path that traverses all edges or visit all vertices of a graph, at the cost of adding new lines to the drawing if needed. In contrast, our algorithm allows the use of a variable number of paths to cope with shapes that cannot be covered by a single path.

3 Wire-Wrapping Principles

As discussed previously, the main challenge in creating wire-wrapped jewelry is to convert an input design, often represented

as a line drawing, into a wire decomposition that can be fabricated. Books [McIntosh 2007; Dismore 2011; DeField 2015] and online tutorials [WigJig 2015; Instructables 2015] provide many examples and recommendations for creating such wire decompositions. We studied this literature and identified three key characteristics of good decompositions.

Low complexity. Wire-wrapped jewelry should be made with a small number of wires because it is hard to join multiple wires in a robust, stable way. In addition, part of the beauty of wire-wrapped jewelry comes from the intricate loops created by a long wire following a complex path. Ideally, a piece of wire-wrapped jewelry should be made from a single wire. However, many input designs cannot be reproduced using a single wire, unless the wire doubles back over parts of the design, which artists tend to avoid for aeasthetic reasons. Good decompositions use the minimum number of wires such that each part of the design is traversed exactly once.

Smoothness. While jewelry wire is made to be malleable, the physical resistance of metal prevents the creation of sharp bends. The path of each wire in the decomposition should thus be as smooth as possible with sharp angles in the input design represented by wire crossings. In the inset, the wire follows the blue direction rather than the red one as it results in a smoother trajectory.

Robustness. A piece of jewelry is most robust when it is composed of a single wire. In cases where several wires are needed to create a shape, craftsmen try to connect each wire at least twice to other wires to avoid weak dangling segments. At each connection, thin wire is wrapped around the various wire segments to hold them together without soldering.

4 Wire Decomposition

4.1 Line-Drawing Vectorization

Our system takes as input a bitmap line drawing, which we assume to be made of a single connected component. We first convert this line drawing into a vector representation by applying morphological thinning [Soille 2003], chaining pixels between junctions and fitting Bezier curves on the resulting pixel chains. Note that more advanced algorithms could be used for this purpose [Noris et al. 2013]. The output of this vectorization is an undirected graph where each vertex corresponds to a junction (i.e., endpoint or intersection) and each edge corresponds to a Bezier segment.

4.2 Energy Formulation

We denote the graph extracted from the input drawing as $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ are the vertices and $\mathcal{E}$ the edges. Our goal is to decompose the graph into N sub-graphs, each sub-graph corresponding to a single wire of the final design. We express this problem as assigning a label $l_{n\in[0,N-1]}$ to each edge of the graph. We evaluate the quality of a given assignment $l \in \{l_{n\in[0,N-1]}\}^{card(\mathcal{E})}$ with three energy terms that correspond to the three design principles identified in Section 3. We now describe each of our energy terms and later provide details about the optimization procedure. Figure 2 illustrates the effect of each of the terms on the solution.

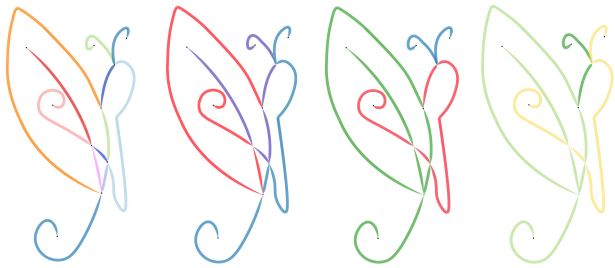


Figure 2: *Effect of each energy term. A trivial solution assigns a different label to each line segment (a). This line drawing cannot be represented by less than three Eulerian paths (b). In (b) the blue and purple paths contain sharp turns that would be hard to create with metal wire. Our smoothness term avoids sharp angles (c) but results in a dangling blue path for the antennas. Our robustness term encourages a solution where each path has at least two connections to other paths.*

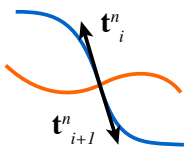
Low complexity. The first and foremost objective of our algorithm is to segment the graph into a small number of sub-graphs, such that within each sub-graph, a piece of wire can traverse every edge exactly once. In other words, the wire should form an *Eulerian path* through each sub-graph. The necessary condition for a graph to admit an Eulerian path is that it has either no odd degree vertices (for a closed path) or exactly two odd degree vertices that correspond to the two endpoints of an open path. This Eulerian path requirement makes our optimization particularly challenging as it introduces a complex, non-local hard constraint in our graph labeling problem. To address this challenge, we design our optimization procedure to only consider candidate decompositions where every sub-graph has an Eulerian path, as described in Section 4.3. Thus, our complexity term only needs to penalize solutions with a large number of labels

$$E_{complex}(l) = N. \tag{1}$$

Note that N is ultimately bounded by the number of edges in the graph, since the worst solution would be to assign a different label to each edge.

Smoothness. Our second term favors smooth paths by measuring the angle between consecutive curve segments at their junctions

$$E_{smooth}(l) = \sum_{n\in[0,N-1]} \sum_{i} \left| \pi - \arctan \frac{\mathbf{t}_i^n \times \mathbf{t}_{i+1}^n}{\mathbf{t}_i^n \cdot \mathbf{t}_{i+1}^n} \right|, \tag{2}$$



where we loop over the Eulerian paths that correspond to the N labels and measure at each intersection the angle formed by the tangent $\mathbf{t}_i^n$ of the incoming curve segment and the tangent $\mathbf{t}_{i+1}^n$ of the subsequent outgoing curve segment [1]. The inset illustrates our notation.

Robustness. We adopt a simple robustness heuristic which avoids dangling pieces by penalizing sub-graphs with less than two

[1] In practice, we compute $\arctan(y/x)$ using the $atan2(y, x)$ function from the C++ library "math.h".

connections to other sub-graphs

$$E_{robust}(l) = \sum_{n \in [0, N-1]} \delta_n(l) \quad (3)$$

$$\delta_n(l) = \begin{cases} 1, & \text{if sub-graph } l_n \text{ has less than 2 connections} \\ 0, & \text{otherwise} \end{cases}$$

4.3 Optimization Method

Our optimization minimizes a weighted sum of the terms

$$\underset{l}{\operatorname{argmin}}\, E(l) = E_{complex}(l) + \lambda_s E_{smooth}(l) + \lambda_r E_{robust}(l) \quad (4)$$

$$\text{s.t. each sub-graph } l_n \text{ admits a Eulerian path}$$

where λ_s and λ_r balance the contribution of the terms. The complexity and robustness terms have a global impact on the label configuration and as such prevent the use of standard graph-cut techniques. An exhaustive evaluation of the $N^{card(\mathcal{E})}$ configurations is also not feasible for any problem of reasonable size. Finally, as discussed previously, the Eulerian path requirement imposes a hard constraint on the labeling.

In light of these challenges, we adopt simulated annealing [Kirkpatrick et al. 1983] to explore the space of configurations, as summarized in Algorithm 1. Our datastructure represents each sub-graph as an Eulerian path. To satisfy the Eulerian constraint, we initialize the optimization by assigning a different label to each edge of the graph, i.e. we have $N = card(\mathcal{E})$ sub-graphs of size 1, each being an Eulerian path by construction. At each iteration, the algorithm generates a new configuration l' by perturbing the current configuration l. We carefully designed the perturbation operators to preserve the Eulerian property of the paths, as detailed in the next paragraph. The new configuration is always accepted if it decreases the energy. Configurations that increase the energy can also be accepted with a probability that depends on the energy variation between l and l' and a relaxation parameter T that geometrically decreases at a rate $T \leftarrow T - T \times C$. This acceptance strategy prevents the algorithm from getting trapped early in local minima. The algorithm stops when the relaxation parameter becomes smaller than a threshold T_{end}. The algorithm returns the configuration with the minimum energy overall. We initialize T to 100, C to 0.0001 and T_{end} to 0.0001.

Algorithm 1 Optimization procedure

Generate initial configuration l (one path per edge)
Initialize solution $l_{min} = l$
Initialize relaxation parameter $T = T_{init}$
repeat
  Generate l' from l by applying a random operator (join, split or permute) on a random path
  Compute acceptance probability $R = \exp\left(\frac{E(l) - E(l')}{T}\right)$
  Draw a random value $p \in [0, 1]$
  if $p < R$ **then** update $l \leftarrow l'$
  else update $l \leftarrow l$
  Update relaxation parameter $T \leftarrow T - T \times C$
  if $E(l) < E(l_{min})$ **then** update $l_{min} \leftarrow l$
until $T < T_{end}$
return l_{min}

Perturbation operators. We define three local perturbation operators to generate new configurations: *join*, *split* and *permute*. The *split* operator selects a random path and splits it into two paths at a random vertex (Figure 3a). The *join* operator selects two random paths that share an end point and appends them to create a single path (Figure 3b). The *permute* operator selects two random paths sharing a vertex and swaps the labels on either side of the vertex (Figure 3c). We randomly apply one of the three operators at each iteration. Note that a *join* can cancel the effect of a *split* and that applying *permute* twice on the same vertex in sequence has no effect, which is critical to allow the algorithm to escape bad configurations. Since our perturbation operators never create new internal vertices of odd degree, our optimization is guaranteed to maintain the Eulerian property of the sub-graphs.

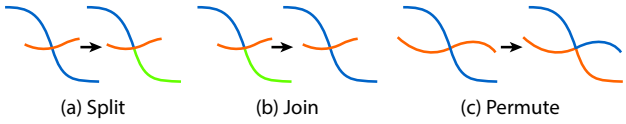


Figure 3: *Our stochastic optimization relies on three operators to generate new configurations of labels. Colors represent different labels, i.e. different sub-graphs. Split creates two sub-graphs from a path (a). Join merges two paths in one (b). Permute exchanges half-paths between two sub-graphs (c).*

Performance. Figure 4 shows the evolution of the configurations during the optimization. The energy makes large oscillations during the first iterations as the algorithm explores the solution space. It then makes more subtle adjustments and converges to a plateau as the relaxation parameter becomes selective. Since full enumeration of all possible decompositions is not practical for most designs of reasonable complexity, we cannot objectively evaluate how close our solutions are to the global optimum. Nevertheless, we performed an exhaustive search on a simple design (bee in Figure 6) using two labels and bounding the maximum length of the paths, which gave 6 242 730 configurations. Over 1000 runs, our algorithm found the best solution each time. We also performed 1000 runs on Figure 1 and obtained the same solution 999 times. Table 1 provides timings[2] for representative drawings, ranging from 3 seconds for the smallest design to 32 seconds for the biggest one. Although the final number of wires in the decomposition depends on the complexity of the input drawing, our algorithm reduced the number of segments by a factor of four in these examples.

Drawing	Input segments	Output wires	Time (sec.)
Butterfly (Fig. 1)	12	3	2.7
Horse (Fig. 8)	24	5	12
Tiger (Fig. 5)	54	16	32

Table 1: *Timing and number of input and output paths for a few representative input drawings.*

4.4 Pre- and Post-Processing

We complement the algorithm described above with two optional features that expand the space of solutions and increase robustness.

Bridges. Complex shapes are sometimes impossible to represent with few Eulerian paths. Experienced jewelery makers often reduce the number of wires in the final design by adding short extra segments, or *bridges*, between nearby paths. To incorporate bridges, we pre-process the vectorized input drawing and detect pairs of vertices that are closer than $0.3\overline{\mathcal{E}}$, where $\overline{\mathcal{E}}$ is the average edge length.

[2] Timings in Table 1 were measured on a MacBookPro with OS X Yosemite, 16GB RAM and a 2.3GHz processor.

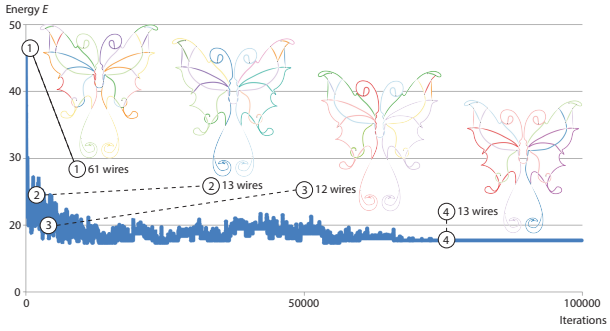
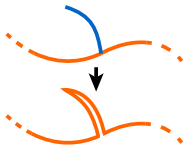


Figure 4: *The energy decreases quickly during the first iterations as edges get connected into longer paths. The optimization then makes subtle updates to the configuration until it converges to a stable solution after* 75000 *iterations.*

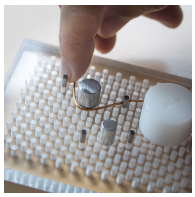
We only retain the pairs that are not connected by an edge and that are not separated by a line in the drawing. We insert these bridges as optional edges in the graph and augment our energy function with a term that penalizes their use. Denoting as $\mathcal{B}(l)$ the set of active bridges for a given label configuration, the penalty term is $E_{bridges}(l) = card(\mathcal{B}(l))$, weighted by $\lambda_b = 0.1$. We include the bridges in our optimization via two additional perturbation operators. The first operator enables a bridge, which allows subsequent iterations of the algorithm to join the bridge to another path. The second operator disables a bridge and creates two paths if the bridge was in the middle of a longer path.



Junction refinement. While our robustness term favors well-connected sub-graphs, it is sometimes impossible to avoid dangling wires that are connected to one other sub-graph at a single point. Inspired by traditional practice, we strengthen such configurations as a post-process by merging the dangling segment with its connected segment and then doubling the wire over it, as illustrated in the inset. However, the double wire and sharp bends required by this solution often make the design less aesthetically pleasing and harder to fabricate, which is why our optimization tries to avoid such cases in the first place.

5 Assisting Wire Bending

In addition to automating the decomposition of a design, our system also assists in its physical realization. The main challenge is in shaping the metal wire into the curves described by the design. A common technique to create smooth curves of a prescribed curvature is to wrap the wire around a curved tool, the so-called *jig*. We experimented with several alternatives to create jigs that best support the creation of a target path.



Pegs. Current practice for creating custom jigs involves fixing cylinders, called pegs, into a board [WigJig 2015]. Pegs of different radii constrain the wire to form loops of different curvature. The main advantage of this technique is that a generic set of pegs is sufficient to create a variety of shapes. However, this approach also imposes strong constraints on the design. First, cylindrical pegs can only produce turns of constant curvature. Second, a design with closely-spaced turns often results in colliding pegs. We originally considered moving pegs during fabrication to avoid collisions. However, optimizing peg placement and wrapping ordering is computationally complex. Our initial tests also revealed that the assembly sequence is too hard to follow while keeping the wire in place.

Naive extrusion. Given the increasing accessibility of 3D printing, an alternative solution is to generate and print jigs that are customized for a specific design. We first attempted to simply extrude the negative space around the design, which effectively produces a channel along the trajectory of the wire. Users thus need to *push* the wire into the channel, rather than *wrapping* the wire around pegs. However, our initial tests with this technique showed that pushing the wire creates jaggy curves because the wire isn't straightened by the longitudinal tension of wrapping. A naive extrusion also requires printing support all along the wire, even in areas where no support is needed such as straight lines or the exterior side of a turn.

Extruding at curvature maxima. Based on the above experiments, our final solution is to extrude support material only on the interior side of each turn. We define turns as portions of curves for which the curvature exceeds a threshold, fixed to 0.01 in our implementation. We use a wall thickness of 3.5mm, which we empirically found to be a good tradeoff between material usage and robustness. We additionally create walls for segments longer than 2cm even if their curvature is below the threshold. To prevent collisions between the walls and wire, we use a boolean operation to subtract the wire path from the extruded wall geometry. We also add small holes to the jig to mark the starting points of each path and to hold the wire in place. Unlike the peg board approach, this solution does not suffer from collisions and allows users to create curves of arbitrary shape. Moreover, the empty space around the extruded walls gives room to wrap the wire with tension to obtain a smooth result. We provide the jigs for all the results in this paper as supplemental materials.

Finally, our interface integrates several convenient features to guide users during fabrication. For each path, we display the wrapping sequence step-by-step by highlighting successive curve segments. We also display the length of each wire, to which we add a few centimeters of margin to ease manipulation. We also show how to attach each wire to its neighbors.

6 Evaluation

We evaluate our approach in three ways: comparing our wire decompositions to ground truth designs from existing wire-wrapped jewelry; gathering feedback from novice users; and using our system to create several pieces of jewelry from input clipart drawings.

Decomposing existing designs. Our wire decomposition algorithm implements principles we deduced from the wire-wrapping literature and example designs. To assess the effectiveness of this algorithm, we created three ground truth decompositions by tracing line drawings over images of existing wire-wrapped jewelry, creating one curve per wire of the jewelry (Figure 5). We then gave rasterized versions of the traced drawings as input to our system. Our results are near-identitical to the original decompositions.

User experience. We recruited five novice crafters to use our system and provide feedback on their experience. Three partici-

Figure 5: *We use existing jewelry (a) as inspiration to test our algorithm. . Our solution (c) closely matches the Artist's original decomposition (b). The decompositions are identical for the lion and the owl, while the left ear of the tiger results in a slightly different configuration than the original, although with the same number of wires.*

pants were female and two were male, aged between 21 and 33. Only participant P1 had any prior experience in jewelry making, while P1, P2 and P4 had some experience with soldering. None had done wire wrapping before.

To start, participants performed a warm-up fabrication task to gain familiarity with the challenges of bending and attaching wires. The task involved wrapping a butterfly design on an existing jig with one wire attachment in the center of the design (see inset). We provided participants with the jig, metal wire, pliers and a ruler. To facilitate the attachment step, we also provided a *third-hand*, an inexpensive crafting tool with two clips for holding different pieces of wire in place.

We then gave participants a line drawing and asked them to create a wire decomposition by hand according to our principles of few, smooth and well-connected wires. We provided a different input drawing for each participant to help us gain a broader range of feedback. The left two columns of Figure 6 compare our automatic decompositions with the user-generated results. Even for these relatively simple designs, our system produced different decompositions for three of the drawings. Our decompositions for the lotus (P3) and dolphin (P5) require fewer segments than the user segmentations. For the butterfly, P4 found a solution with the same number of wires as ours, but the purple antennae segment is not robust, as it only has one connection to the rest of the piece. We note that P3's two-wire solution for the lotus may have better preserved the rotational symmetry of the design after fabrication, which our algorithm doesn't model. We disabled bridges for this comparison since participants did not have the option of creating bridges in their manual decompositions.

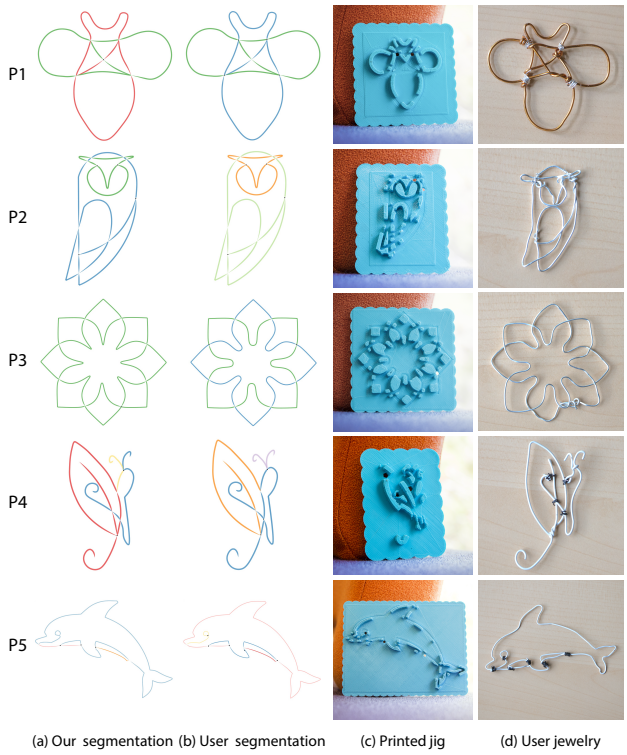


Figure 6: *Results of our user experiment. For simple designs (P1, P2), participants found the same solution as our algorithm (a,b). For more complex designs, our algorithm finds a more robust solution (P4) or a solution with less wires (P5). All participants managed to create their jewelry in less than half an hour (d).*

Finally, we asked participants to build the design from our decomposition and then conducted a post-study interview to gather feedback. Figure 6(c) shows the user-fabricated pieces of jewelry, which took about 20 minutes to create. Participants reported that the step-by-step visualization helped them understand the wrapping sequence of each path. On a Likert scale from 1 (strongly disagree) to 5 (strongly agree), three participants strongly agreed that the automatic decomposition helped in building the jewelry piece, one agreed, and one had no opinion. P2 commented that *"My decomposition was the same as the automatic one, but the automatic solution gave me confidence in my choices and about where to end each piece."* Participants unanimously appreciated the jig, agreeing or strongly agreeing that it helps in building the jewelry piece. P2 commented that *"It would have taken much, much longer to build the jewelry without the jig"*. P1 noticed that *"The wire tends to jump out of the jig when there are too many layers of wire passing at the same point."*, which may be addressed by higher support walls.

Clipart to Jewelry. As a final means of evaluation, we used our system to convert clipart drawings downloaded from the Internet into wire-wrapped jewelry. We applied our vectorization algorithm on the bitmap images and manually cleaned-up little segments that would be hard to build at this scale. We also made small edits to some of the input drawings to ensure that they were made of a single connected component. Figure 1 and 8 show several complex jewelry pieces that we created with this approach.

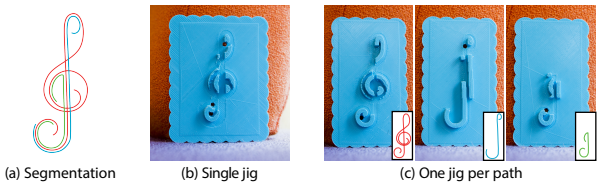


Figure 7: *Packed turns in (a) produce many collisions between the wire path and support walls. The resulting jig lacks support for some of the wires (b). To overcome this problem, users can print a separate jig for each wire (c).*

7 Limitations

As observed during our user study, our decomposition does not account for aesthetic criteria such as symmetry, and our support walls are sometimes not sufficient to keep the wire in place in the presence of many layers of wire.

In addition, complex drawings with densely packed turns can produce too many collisions between the support walls and wire path. In such cases, the resulting jig can lack support to achieve the desired shape or can contain many small pieces of walls that make the construction sequence harder to follow. Nevertheless, users can reduce complexity and collisions by printing a separate jig for each Eulerian path, as shown in Figure 7.

While our current implementation does not offer user control, our optimization could easily integrate user-provided constraints. For instance, users could impose that two segments should definitely share or not share the same label and the optimization could simply avoid any perturbations that violate such constraints. However, our approach does not explicitly consider the layering relationships between overlapping wires.

8 Conclusion and Future Work

We have presented an end-to-end system to assist the design and fabrication of wire-wrapped jewelry. From an algorithmic standpoint, our approach implements wire wrapping design principles in an optimization that segments an input drawing into a small number of wires. The optimization favors smooth wires that are well-connected to other wires to ease the fabrication of robust pieces. The optimization also strives to cover each line in the drawing exactly once to avoid double wires. From a fabrication standpoint, our system outputs a physical support structure that guides users in bending wires to obtain a desired shape. Our approach thus differs from completely automatic digital fabrication by preserving the hand manipulation of jewelry materials, which greatly contributes to the pleasure of craft.

We see several directions of future work to build on our contributions. Our segmentation algorithm finds Eulerian paths that exactly reproduce the input drawing, up to optional bridges. However, part of the art of wire wrapping involves *abstracting* a shape to make it more suitable to fabrication with smooth wires. While algorithms for shape abstraction exist [Mi et al. 2009], none account for the specific fabrication and aesthetic constraints of jewelry making. Jewelry makers also often combine wrapped wires with beads and stones. While we have included beads in some of our results, we have not considered the challenge of accounting for such additional components during segmentation. Our jigs could also be adapted to hold stones and beads in place during wrapping. Finally, we strongly believe that the computational design of jigs and other intermediate support structures has great potential for other forms of craft, such as clay modeling or 3D wire sculpting.

Acknowledgments

We thank Anastasiya Ivanova for letting us use her pictures of beautiful pieces of jewelry in Figure 5. This work was supported by ANR-12-JS02-003-01 DRAO and software and research donations from Adobe.

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 Transactions on Graphics (Proc. of SIGGRAPH) 22*, 3 (July), 828–837.

COROS, S., THOMASZEWSKI, B., NORIS, G., SUEDA, S., FORBERG, M., SUMNER, R. W., MATUSIK, W., AND BICKEL, B. 2013. Computational design of mechanical characters. *ACM Transactions on Graphics (Proc. of SIGGRAPH) 32*, 4 (July), 83:1–83:12.

DEFIELD, A. 2015. *Make Wire Wrap Jewelry: Basic Wire Wrapping Techniques and Jewelry Tutorials*.

DEUSS, M., PANOZZO, D., WHITING, E., LIU, Y., BLOCK, P., SOKRINE-HORNUNG, O., AND PAULY, M. 2014. Assembling self-supporting structures. *ACM Transactions on Graphics 33*, EPFL-ARTICLE-201940.

DISMORE, H. 2011. *Jewelry Making & Beading For Dummies*. –For dummies. Wiley.

FU, C.-W., SONG, P., YAN, X., YANG, L. W., JAYARAMAN, P. K., AND COHEN-OR, D. 2015. Computational interlocking furniture assembly. *ACM Transactions on Graphics (Proc. SIGGRAPH) 34*, 4, 091:1–091:11. ∗ joint first author.

GARG, A., SAGEMAN-FURNAS, A. O., DENG, B., YUE, Y., GRINSPUN, E., PAULY, M., AND WARDETZKY, M. 2014. Wire mesh design. *ACM Transactions on Graphics (TOG) 33*, 4, 66.

HILDEBRAND, K., BICKEL, B., AND ALEXA, M. 2012. crdbrd: Shape fabrication by sliding planar slices. In *Computer Graphics Forum*, vol. 31, Wiley Online Library, 583–592.

IGARASHI, Y., AND IGARASHI, T. 2010. Holly: A drawing editor for designing stencils. *Computer Graphics and Applications, IEEE 30*, 4, 8–14.

IGARASHI, Y., IGARASHI, T., AND MITANI, J. 2012. Beady: interactive beadwork design and construction. *ACM Transactions on Graphics (TOG) 31*, 4, 49.

INSTRUCTABLES, 2015. `http://www.instructables.com/howto/wire+wrapped+jewelry/`. Accessed: 2015-05-30.

KAPLAN, C. S., AND BOSCH, R. 2005. Tsp art. *Bridges: Mathematical Connections in Art, Music and Science*, 301–308.

KIRKPATRICK, S., GELATT, C. D., VECCHI, M. P., ET AL. 1983. Optimization by simmulated annealing. *science 220*, 4598, 671–680.

LI, X.-Y., JU, T., GU, Y., AND HU, S.-M. 2011. A geometric study of v-style pop-ups: Theories and algorithms. *ACM Transactions on Graphics (Proc. of SIGGRAPH) 30*, 4 (July), 98:1–98:10.

MCINTOSH, J. 2007. *Wire Wrapping: The Basics and Beyond*. CreateSpace Independent Publishing Platform.

Figure 8: *Clipart repositories provide a wide source of inputs for our approach. Note how our algorithm finds decompositions with few yet smooth and robust wires, such as the body of the bee and the legs of the horse.*

MI, X., DECARLO, D., AND STONE, M. 2009. Abstraction of 2d shapes in terms of parts. In *Proc. Symp. on Non-Photorealistic Animation and Rendering (NPAR)*, ACM.

MORI, Y., AND IGARASHI, T. 2007. Plushie: an interactive design system for plush toys. In *ACM Transactions on Graphics (TOG)*, vol. 26, ACM, 45.

NORIS, G., HORNUNG, A., SUMNER, R. W., SIMMONS, M., AND GROSS, M. 2013. Topology-driven vectorization of clean line drawings. *ACM Transactions on Graphics (TOG) 32*, 1, 4.

SCHMIDT, R., AND RATTO, M. 2013. Design-to-fabricate: Maker hardware requires maker software. *Computer Graphics and Applications, IEEE 33*, 6 (Nov), 26–34.

SKOURAS, M., THOMASZEWSKI, B., BICKEL, B., AND GROSS, M. 2012. Computational design of rubber balloons. *Computer Graphics Forum (proc. Eurographics) 31*, 2 (May), 835–844.

SKOURAS, M., THOMASZEWSKI, B., KAUFMANN, P., GARG, A., BICKEL, B., GRINSPUN, E., AND GROSS, M. 2014. Designing inflatable structures. *ACM Transactions on Graphics (Proc. SIGGRAPH) 33*, 4, 63:1–63:10.

SOILLE, P. 2003. *Morphological Image Analysis: Principles and Applications*, 2 ed. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

TANENBAUM, J. G., WILLIAMS, A. M., DESJARDINS, A., AND TANENBAUM, K. 2013. Democratizing technology: Pleasure, utility and expressiveness in diy and maker practice. In *Proc. of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, 2603–2612.

UMETANI, N., IGARASHI, T., AND MITRA, N. J. 2012. Guided exploration of physically valid shapes for furniture design. *ACM Transactions on Graphics (Proc. of SIGGRAPH) 31*, 4.

UMETANI, N., KOYAMA, Y., SCHMIDT, R., AND IGARASHI, T. 2014. Pteromys: Interactive design and optimization of free-formed free-flight model airplanes. *ACM Transactions on Graphics (Proc. of SIGGRAPH) 33*, 4 (July), 65:1–65:10.

WIGJIG, 2015. `www.wigjig.com`. Accessed: 2015-05-30.

WONG, F. J., AND TAKAHASHI, S. 2011. A graph-based approach to continuous line illustrations with variable levels of detail. *Computer Graphics Forum 30*, 7, 1931–1939.