Generating Virtual Wire Sculptural Art from 3D Models

Chih-Kuo Yeh, Zhi-Ying Hou, Tong-Yee Lee *Senior Member, IEEE*

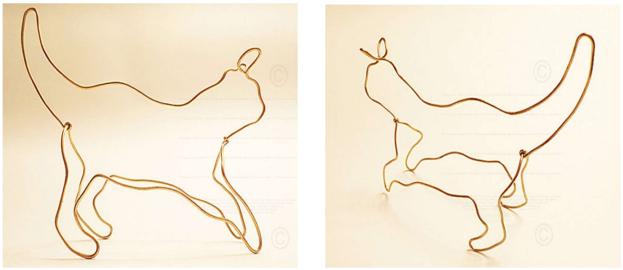
Abstract—Wire sculptures are sculpted objects by the use of wires. In this paper, we propose practical methods to create 3D virtual wire sculptural art from a given 3D model. In contrast, most of the previous wire art results are 3D wire reconstructed from input 2D wire art images. In our proposed methods, we first extract the important edges from a given 3D model to obtain the connected segments of edges which roughly represent the shape of an input 3D model. To generate 3D virtual wire art with the minimum number of continuous wire lines, we connect these segments by the order obtained by optimally solving a travelling salesman problem with some constraints. Then, we smooth the obtained 3D wires to simulate the real 3D wire results by artists. In addition, we provide a user interface to control the winding of wires by their design preference. Finally, we experimentally show our 3D virtual wire results, and quantitativly evaluate our created results. As a result, the proposed method is computed effectively and interactively, and results are appealing and comparable to real 3D wire art work.

Index Terms— Wire sculpture, edges, 3D wire art, edge segments, wire composition, smoothing

1 Introduction

Compact, abstract 3D shape representation are visually more appealing and interesting than the original models which seem to be visually cluttered. 3D wire as sculpture (see Fig. 1) is an interesting art created by 3D models using wire lines. Selecting suitable and small number of wire lines can show visually significant parts of a 3D object. It has been applied to many applications such as furniture design, wire wrapped jewelry and wire sculptural art etc. Usually, the artists require time-consuming manual procedure to design and adjust their wire sculptural art. Without expert experience and their time-consuming manual beautification, wire sculptural design may consist of too many short wires, and thus it easily lacks of appealing results. In this paper, our goal is to design a 3D virtual wire sculpture with less number of wires and without too time-consuming manual design and computational time. We introduce a novel methodology for abstracting three dimensional shapes using less number of wires. Our results are easily and efficiently created and comparable to arts' design. In contrast to our work, most of 3D wire art work is based on surface reconstruction of 2D wire art input images [3,4,5]. We propose a different way to create wire art from 3D models. In addition, we also quantitativly evaluate our created results from various input models.

Our contributions are listed as follows: (1) we create an automatic system to create 3D virtual wire sculpture art for a given 3D model and capable of efficiently creating appealing 3D virtual wire art, (2) we propose a travelling salesman optimization with some constraints to find minimum number of wire lines for creating our 3D virtual wire sculpture, and finally (3) we optimally smooth wire lines to make results more appealing and look like real 3D wire art.



(a) Side view **(b)** Side back view

Fig. 1. Wire line art by artist Theodore Huckins.

2 Related Work

In this session, we review the related works and the existing techniques that are related to create 2D /3D line representation from images and 3D model as follows.

Orzan et al. [1] proposed a vector-based diffusion curve method to draw a smooth-shaded image allowing a vector-based primitives, key-frame animation and stylization etc. Wong et al [2] introduced a method for automatically generating continuous line illustrations, drawings consisting of a single line, from a given input image. This approach can generate different amount of details drawn in their line illustrations, particularly usefully maintaining important features while simplifying the overall illustra-

- *C.K. Yeh is with the National Cheng-Kung University, Tainan, Twiwan. E-mail: simpson.ycq@gmail.com.*
- *Z.Y. Hou is with the National Cheng-Kung University, Tainan, Twiwan. E-mail: denny1994a@gmail.com*
- *T.Y. Lee is with the National Cheng-Kung University, Tainan, Twiwan. E-mail:.tonylee@mail.ncku.edu.tw*

tion of the input image.

Using spatial curves, Li et al. [6] proposed a method for reconstructing the fine and thin structure of an object. In this paper, Li et al. demonstrated that leveraging spatial curves can dramatically improve the thin structure reconstruction. They proposed a complete solution using spatial curves for surface reconstruction. Liu et al. [3] presented an image-based method to reconstruct wire objects from a few input images and their reconstructions faithfully capture both the 3D geometry and the topology of the wires. Later, Suzuki et al. [4] introduced a method for the generation of multi-view wire art by providing two line drawings and showed that a connected multi-view wire art always exists if each of line drawings is connected and they have the same height. Recently, Chu et al. [5] presented a computational framework for automatic creation of multi-view 3D wire sculpture and their system takes two or three user-specified line drawings and the associated viewpoints as inputs. These three above methods are based on 2D wire art images and reconstruct single [3] or multi-view 3D wire sculpture [4,5]. In contrast to their purpose, our method focuses on generating 3D virtual wire art models based on 3D input models rather than 3D reconstruction from 2D images [3,4,5]. Our approach generates 3D virtual wire art with the minimum number of continuous and smooths 3D wire lines to simulate the real 3D wire results by artists.

Mehra et al. [7] et al. proposed a novel method for abstracting 3D models using characteristic curves or contours as building blocks for the abstraction. Gal et al. [8] introduced iWIRES, a novel approach based on the argument that man-made models can be distilled using a few special 1D *wires* and their mutual relations. Their final shape of the object is determined by using the deformed wire scaffolding. Nealen et al. [9] presented a system for designing freeform surfaces with a collection of 3D curves. Wu et al. [10] presented a method to print arbitrary meshes on a 5DOF wireframe printer. Cohen-Steiner et al. [11] proposed a new strategy for the design of succinct and efficient shape approximations, and their approach is entirely error-driven and uses a novel discrete, variational method without resorting to any estimation of differential quantities or parameterization.

There are also some related work concentrated on modeling and fabricating the wire sculptures of a specific form. Iarussi et al. [12] proposed a framework to help the creation of wire wrapped jewelry. Miguel et al. [13] designed a computer-aided tool to facilitate converting a 3D model into a stable, self-supporting wire sculpture. Zehnder et al. [14] developed a computational design tool, aiming at the fabrication of ornamental curve networks defined on the 3D surfaces. Given an input 3D wire abstraction, Lira et al. [15] presented a fully automatic method that finds a small number of machine fabricable wires with minimal overlap to reproduce a wire sculpture design as a 3D shape abstraction. They considered non-planar wires, which can be fabricated by a wire bending machine, to enable efficient construction of complex 3D sculptures. Our work also belongs to this category [12,13,14,15] of 3D wire art design and modeling but applying fundamentally different design and approach. Our approach does not require either input 3D wire model or 2D wire images.

3 System Overview

Our input is a three-dimensional triangular mesh model containing information about points, edges, and faces. The output is a collection of 3D lines $W = W_1,...,W_k$, and each 3D line is made up of many consecutive points. As shown in Fig. 2, we present a system for allowing users to create 3D virtual wire art sculpture efficiently with less number of wire lines. In our system, given a 3D model, we first automatically extract its visually important edge segments to users. Furthermore, we allow users to select some extra edges if they prefer according to their personal interests. Then, we apply a travelling salesman optimization to find the minimum number of wire lines for creating our 3D virtual wire sculpture. Finally, we smooth wire lines to make it more appealing.

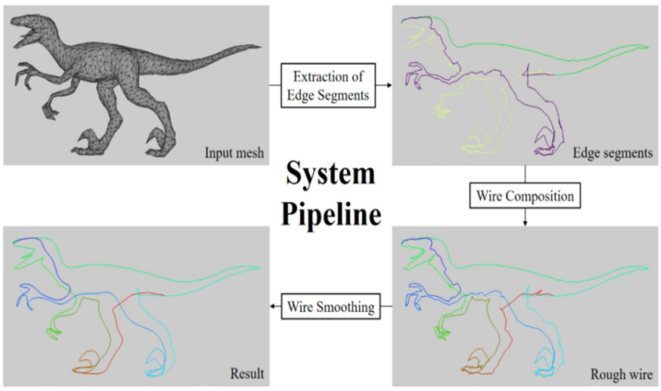


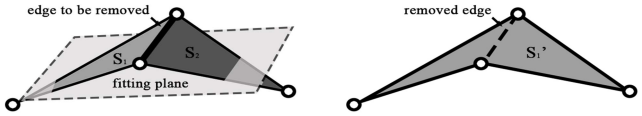
Fig. 2. System Overview.

Preprocessing. Our preprocessing remeshes each given 3D model in order to avoid the appearance of narrow triangles on the mesh. In the first step, we will directly extract the edges of the triangles as a rough wire. If the sizes or shapes of the triangles are very different, i.e., irregular, it may cause the extracted rough lines to be extremely uneven. Therefore, the results are not as expected. We will use the method proposed by Botsch et al. [16] to remesh the input model with the same number of the triangles as the original model. If the sizes and shapes of the triangles on the input model meshes are the same (or similar), we do not have to do this preprocessing. This remeshing work can be done by any other available remeshing work if necessary such as https://www.meshlab.net/. This preprocessing is not the focus of our work in this paper.

4 Wire Construction and Smoothing

4.1 Extraction of Edge Segments

Inspired by Garland et al. [**17**], we adopt a greedy approach to extract the important edges of a given 3D mesh. These extracted important edges will become the basis for the subsequent lines and are called as edge segments in our paper. Our goal is to remove the less important edges of the mesh. These removed edges are usually located in smoother areas of the mesh, and the remaining edges are more important edges, i.e., such as sharp edges, with the aim to preserve the shape of the model when we remove edges. In this manner, we create a rough 3D wire model for a given 3D mesh. We will describe how to measure importance later in Equation (1).



(a) Before merging **(b)** After merging

Fig. 3. The merging of two triangles S_1 and S_2 to S_1' by [**17**]

In our methodology, the basic idea is to merge two neighboring and smooth regions, remove the common edges between two neighboring faces during the merging process, and retain the more important edges. As shown an example in Fig. 3(a), S_1 and S_2 are two neighboring triangles to be merged. We calculate a fitting plane, and then measure the smoothness between the fitting plane and the triangles to determine whether to merge them or not. If S_1 and S_2 are merged, their common edges will be removed. The new result is shown in Fig. 3(b), where S_1' is the edge segment set after the merging of two triangles S_1 and S_2.

In addition to the above extraction of edge segments, we also want to remain the feature vertices, i.e., critical points, to our designed 3D wire art. For this purpose, we apply Zhou et al. [**18**] to find critical points of our input 3D model. In the process of extracting the edge segments, we make sure that these critical points on some extracted edge segments. Next, we generate n sets $\{S_1,S_2,...,S_n\}$ on the mesh where n is the number of triangles in the original model or the number of triangles after remeshing on the original model. At first, each set S_i contains only one corresponding triangle T_i. We refer to the two adjacent sets S_i and S_j as a pair, and then find all the vertices V_b on the boundary of the region formed by all the triangles in these two sets. With these vertices we calculate a fitting plane P_f by Emil [19] and then measure the difference between all triangles in (S_i,S_j) and P_f. We take the difference as the cost $C_{i,j}$ of merging S_i with S_j. Whenever a pair of sets is merged into one set, the common edges between the two sets are removed. The cost $C_{i,j}$ is computed as follows.

Given a pair of sets, we define T as all the triangles in this pair, i.e. $T=\{T_1, T_2, ... T_t\}$, where t is the number of triangles in T. In this merged region of T, we will find all the vertices V_b at outer boundary of this merged region. Then, we use these boundary vertices as input to calculate the fitting plane by using Emil [19]. There are exactly two directions perpendicular to this fitting plane and we select one with more similar direction to average normal vectors of all triangles in T. Then, we use following equation (1) to compute a cost C to measure the smoothness of this merged region.

$$C = \sum_{i=1}^{t} \frac{A_i}{A_{sum}} \sqrt{(a_f - a_i)^2 + (b_f - b_i)^2 + (c_f - c_i)^2 + (d_f - d_i)^2} \qquad (1)$$

,where A_i is the area of triangle T_i, A_{sum} is the total area sum of all triangles in T, a_f, b_f, c_f,d_f are the coefficients of the fitting plane, and a_i, b_i, c_i,$d_i$are the coefficients of the triangle T_i. Before computing C, we need to normalize all plane equations. If the cost C is small, it means merged region is smooth and vice versa.

We will sequentially remove these common edges of the pair according to the cost C from small to large until the retaining sets meet the user-specified number. In the process of merging pairs, we will also set some constraints to ensure that there will be extracted edges at these critical points, and all edge segments will be connected. In addition, we also provide a user interface to allow the user to pre-draw some edges where they wish to extract. In this manner, if the common edges between the merged sets contain user-specified edges, we will not merge this pair so that the user-specified edges can be retained.

Our algorithm extracts the edges from input mesh by iteratively merging two neighboring regions, which are two sets of triangles sharing some triangle edges, and removing the shared edges of these two regions from the mesh. We call these two regions "a pair of sets". On this bottom-up merging strategy, our merging execution is repeated until the retaining number of sets reaches the user specified number or meets all of the constraints. We will set some constraints to make sure that there are edges passing through the critical points and all the edges are connected, and also preserve user-selected edges not being deleted. We describe these constraints in detail in the later section. In preprocessing, we will remesh the input model into a regular mesh if necessary. If the model is a regular mesh, we can remove the shared edge or edges from the neighboring pair of sets starting from the smooth ones to the undulating ones to obtain better extracted edges. Our algorithm can be summarized as the following steps:

1. Find the critical points (see an example in Fig. 4(a)) on the mesh by Zhou et al. [18].
2. Place each triangle T_i in the corresponding set S_i and find all adjacent sets to make them a pair.
3. Find the boundary vertices V_b in each pair of sets, thereby obtaining the fitting plane P_f by [19] and then calculate the cost C (i.e., by equation (1)) of merging the pair.
4. Put all pairs of sets into a heap in order of small to large.
5. Take the pair with smallest cost C out of the heap and determine if the pair can be merged, and if so, the pair is merged and the common edge of the pair is removed from the mesh.
6. Update the cost of the set adjacent to the set in step 5, while also updating its ordering in the heap. Then returning to step 5, the execution is repeated until the retaining number of sets reaches the user specified number.

Note that in above merging processing, it is based on relative cost C on a heap ranked in order of small to large.

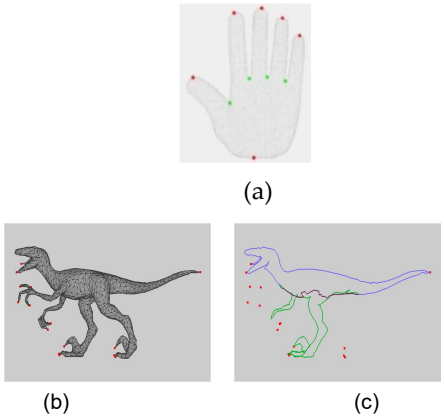


Fig. 4. (a) Critical vertices (red: local extrema, green: saddle vertices, gray: regular vertices), (b) Feature constraints selected from Zhou et al. [18] and user-specified, and (c) these extracted edges cannot capture the features of the model without feature constraints (2).

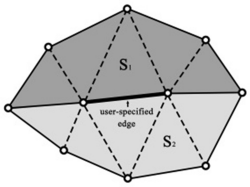


Fig. 5. The shared edges between S_1 and S_2 contain user-specified edge.

Constraints for the merging of pairs

In the process of merging pairs, we do not execute the merging if one of the following conditions is met.

1. **The shared edges contain user-specified edges.** As shown in Fig. 5, S_1 and S_2 are the sets to be merged and their shared edges contain user-specified edge. If they are merged, the user-specified edge is removed. In order to keep it, we do not merge this pair.
2. **A critical point is on the shared edges.** As shown in Fig. 6, S_1 and S_2 are the sets to be combined and there is a critical point on their shared edges. If they are combined, there is no other retained edge connected to this critical point. In order for the extracted edge segments to pass through the critical points and retain the object features, we do not combine this pair. Without this constraint, the bad result of extracted edge segments is shown in Figure 4 (c).

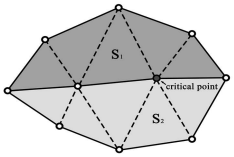
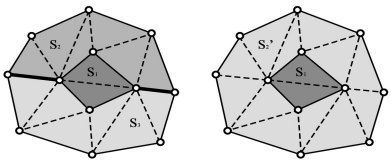


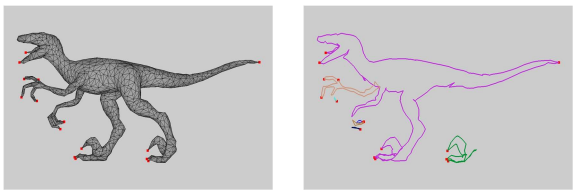
Fig. 6. A critical point is on the shared edges between S_1 and S_2.

3. **The shared edges of the pair are discontinuous.** As shown in Fig. 7(a), S_2 and S_3 are the sets to be merged, but their shared edges (bold edges) are discontinuous. If they are merged, the edges of S_1 and edges of merged set S_2' are disconnected, as shown in Fig. 7(b). However, we want the resulting edge segments to be connected, so in this case we do not merge this pair. Without this constraint, the result of extracted edge segments is shown in Fig. 8(b).



(a) **(b)**

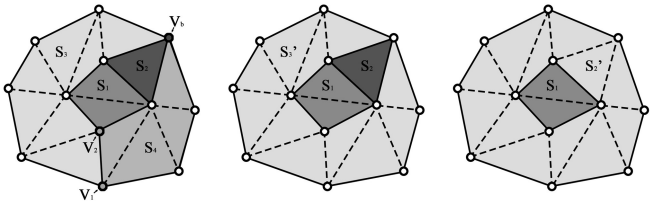
Fig. 7. The combination of S_2 and S_3 with discontinuous shared edges. This will separate the extracted edges. (a) The shared edges between S_2 and S_3 are not continuous. (b) S_2' is the combined set of S_2 and S_3.



(a) Input model **(b)** After edge extraction

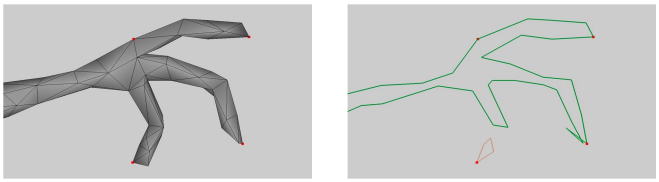
Fig. 8. The result without this constraint (3). The extracted edges are disconnected.

4. **Adjacent vertices of the sets are not on the shared edges.** As shown in Fig. 9(a), S_3 and S_4 are the sets to be merged. Among the triangle vertices contained in the sets, there are 3 vertices V_1,V_2 and V_b at the intersection of the sets, where V_1 and V_2 are on the shared edges, but V_b is not. In this case, if we combine S_3 with S_4, we will get the result as shown in Fig. 9(b), where S_3' is the merged set of S_3 and S_4. If we then merge S_2 with S_3', the result of edge separation will occur, as shown in Fig. 9(c), where S_2' is the merged set of S_2 and S_3'. In order to prevent this from happening, we must avoid merging S_3 with S_4 in advance. Without this constraint, the result of extracted edge segments is shown in Fig. 10.



(a) (b) (c)

Fig. 9. The merging of S_3 and S_4 with shared vertex V_b not on the merged edge. This could also separate the extracted edges. (a) V_1, V_2 and V_b are all the merged vertices between S_3 and S_4. (b) The result of merging S_3 and S_4. S_3' is the merged set. (c) After merging of S_2 and S_3'. S_2' is the merged set.



(a) Input model. **(b)** After edge extraction

Fig. 10. Without this constraint (4), the extracted edges can be disconnected.

Using these two constraints (3) and (4), the main purpose is to make sure extracted edges are connected. However, for example, in both Figure 8(b) and Figure 10(b), the extracted edges are disconnected if we do not have (3) and (4) constraints in these two examples.

4.2 Extraction of Edge Segments

After performing the combination of the sets and removing the edges on the smoothed regions, the retaining edges are extracted edges. We want to merge successive extracted edges into a single line segment. If there are intersections between the segments, then the edges adjacent to the intersections belong to different segments, as shown in Fig. 11. We call them edge segments.

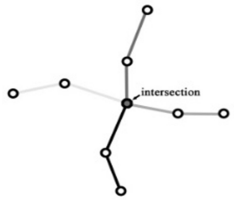
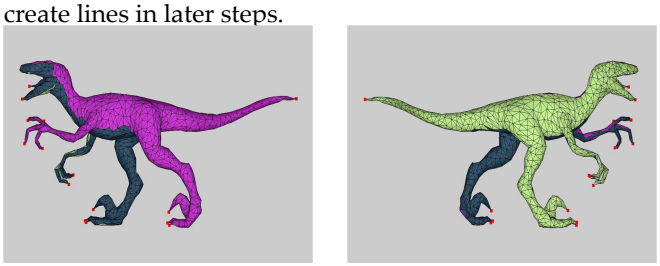
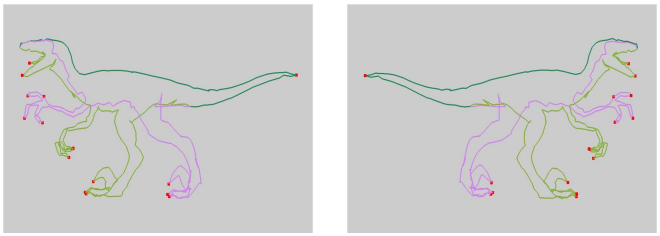


Fig. 11. Four edge segments (in different grayscale color) are separated at the intersection.

Fig. 12 is an example of the extraction of edge segments. There are 16 critical points, 3 retained sets, and 3 extracted edge segments. We will use these edge segments to create lines in later steps.



(a) (b)



(c) (d)

Fig. 12. An example of extracted edge segments. The result keeps the shape of original model. Note that all critical points (in red) are located on extracted edges. (a) The result has 3 retained sets of triangles. (b) Another view of retained sets. (c) The result has 3 extracted edge segments. (d) Another view of extracted edge segments.

5 Wire Composition

Liu et al. [**3**] proposed a method of connecting three-dimensional (3-D) line segments to minimize the number of lines. To achieve this, they solved a multiple traveling salesman problem (**mTSP**) to determine how the lines are connected. That is, the closer the lines are, the higher probability of connection is. Since Liu et al. [**3**] treat the line segments as nodes to calculate the costs and connect them lately, the resulting lines may not be continuous. In their work, the cost is formulated as:

$$\text{Pairwise cost} = \text{Distance term} + \mu * \text{Angle term} \qquad (2)$$

,where Distance term denotes the distance in 3D between the closest endpoints of two line segments, and Angle term is calculated as $(1 + cos \propto)/2$, i.e., $\propto$ is angle between the tangent directions of these closest endpoints on two line segments.

We depict their method in Fig. 13(a). In this example, the

line segments are numbered from 1 to 15. Assuming we start from segment 1, there are four candidates we take into account. They are the line segments: 2, 12, 14, and 15. Accordingly, we find the smallest cost from segment 1 to the four segments to define the next segment (i.e., it is segment 2). In this case, the distance term of segment 2, 12, 14, 15 is zero because their endpoints are directly connected to segment 1. If we calculate the angle term, we can clearly figure out that the angle term ($\propto = 180^{\circ}$) between segment 1 and segment 2 is the least because the larger the angle between the line segments is, the smaller the cost is. After tracing this calculation, we can see that the ending points of line segments are overlapped when we get to segment 4, as shown in the red circle of Fig. 13(a). This phenomenon produces a non-continuous line. Therefore, there is another wire needed for connection. Finally, the result of Liu's method, as shown in Fig. 13(b), needs five wires to connect a star. In contrast, our method just needs a wire to create a star in (d) from (c). Note that in Figure 13, the number in each line indicates the order of connection sequence after mTSP.

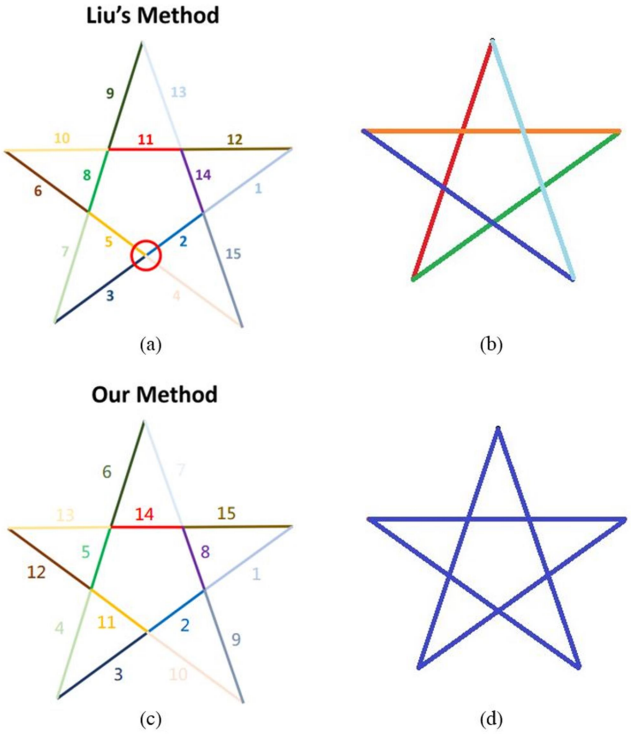


Fig. 13. (a),(c) The different color of line segments denote the number of line segments to be connected and the numbers on the figure denote the connecting sequence. (b) The result of Liu's method used five different wires to connect a star. (d) The result of our method used one wire to connect a star.

Basic concept. Motivated by the aforementioned limitation in Liu et al. [**3**], our proposed method improves **mTSP** to connect line segments in space. Contrast to [3], we use the ending points of the line segments as the nodes in the directed graph. Besides, we constrain these two nodes on the same line segment to have edge connected in the graph. This constraint represents the existence of the existing line segments at both ending points. By this way, all the connections between the segments can be considered, rather than only the nearest ending points. In Fig. 14, our varied **mTST** only requires a single wire (c) and Liu et al. [3] requires two wires (a) in this example. As shown in Fig. 14(d), there are three segments. Each of them composes of a pair of points ($P_{i,1}$, $P_{i,2}$), i=1..3. We consider these points as the nodes of the graph and constrain each two nodes on the same segment to have one edge connected. By using the optimized path to connect the ending points, we can get a continuous line (in Fig.14(c)).

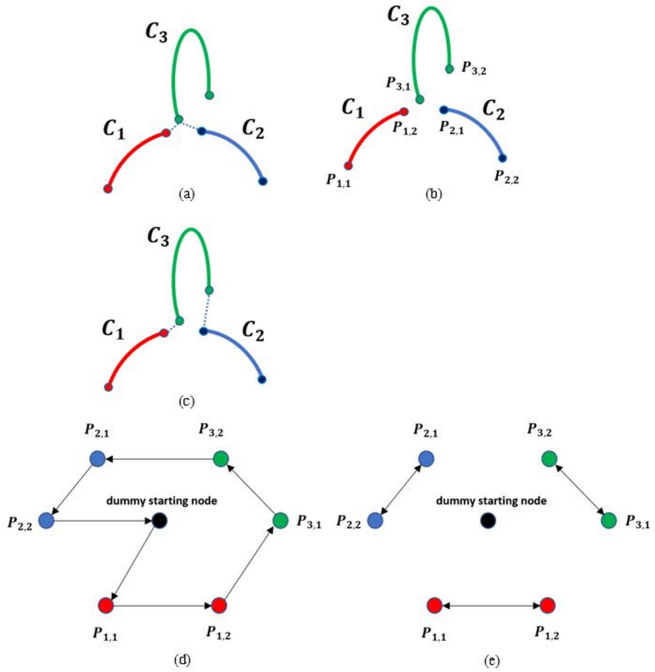


Fig. 14. (a) The nearest ending points between (C_1, C_3) and (C_2, C_3) are the same, resulting in non-continuous lines. (b) The two ending points $P_{i,1}$, $P_{i,2}$ on line segment C_i are the nodes of the directed graph. (c) The lines connected by our method are guaranteed to be continuous. (d) The directed graph by using the ending points as the nodes. The double-headed arrow in the figure indicates that the two ending points $P_{i,1}$, $P_{i,2}$ on the same line segment C_i must be connected; and the black node is the dummy starting node. (e) The path computed by our method. Notice that both $P_{i,1}$, $P_{i,2}$ have connected edges.

Later, in session 7, we will show our comparison with Liu et al. [3] for wire composition in Figure 23. Results show that two methods create similar wire art but our method requires less number of wires.

Method development and implementation. Given the set of 3D line segments $C_{opt} = \{C_t\}, t = 1,2,...,n$, where n is the number of segments, we can obtain a set including the endpoints of all segments $P_{opt} = \{P_i\}, i = 1,2,...,2n$, where the two endpoints of C_t are P_{2t-1},and P_{2t}. We use P_{opt} to create a directed graph $G = (\mathcal{V},\mathcal{E})$, where each node $V_i \in \mathcal{V}$ represents the endpoint P_i. We also define a dummy starting

node V_0 as the starting and ending node of each path. We assign the directed edge $e_{i,j} \in \mathcal{E}$ from V_i to V_j a cost $w_{i,j}$, namely, the cost between corresponding endpoint P_i and P_j. A pairwise cost $w_{i,j}$ is used to evaluate the pairwise relations between pairwise segments C_i and C_j, Liu et al [13] define a pairwise cost $w_{i,j}$ in equation (2) defined early.

The costs of directed edges are symmetrical, that is, $w_{i,j} = w_{j,i}$. Our goal is to find k paths in the directed graph G, each path starts and ends at V_0, and each node $V_i (i > 0)$ is included in exactly one of the path. Each path consists of the directed edges it passes through, and we assign the directed edge $e_{i,j} \in \mathcal{E}$ a binary variable $x_{i,j}$. If $x_{i,j} = 1$ means that $e_{i,j}$ is included in a path, and $x_{i,j} = 0$ is the opposite. In addition, we assign each node $V_i \in \mathcal{V}$ an integer variable $u_i \in \mathbb{N}$ to indicate the order in which V_i is accessed in the path, while $u_0 = 0$ corresponding to the dummy starting node V_0. Our objective function is the same as that of Liu et al. [3] and has same constraints (see more details in [3]). In our experiments, to ensure that the least number of lines is obtained, we set the extra contraint ξ as max $w_{i,j}$, leaving the variable k to a minimum. Liu et al. developed the following optimization function to calculate mTSP:

$$\{k^*, x_{i,j}^*\} := argmin_{\{k, x_{i,j}\}} \sum_{i,j} x_{i,j} w_{i.j} + \xi k \quad (3)$$

, where k is the minimum number of paths, namely, the number of connected lines. The constant ξ will affect the number of lines. If ξ is larger, k will be smaller, and vice versa. In their experiment, ξ = 1/10th of the maximum edge cost and $x_{i,j}$ is a binary variable. If $x_{i,j} = 1$ means that the node i to node j is included in one of the paths, if $x_{i,j} = 0$, otherwise. $w_{i,j}$ is the cost of node i to node j. In addition, they set following constraints associated with mTSP to solve the equation (3)

Constraints of mTSP. We use the same constraints in Liu et al. [3] to solve the equation (1). To ensure that each node Vi $(i > 0)$ is visited exactly once, one of the incoming and outgoing edges of a node needs to be selected:

$$\forall i > 0 : \sum_{j, j \neq i} x_{j,i} = 1, \forall i > 0 : \sum_{j, j \neq i} x_{i,j} = 1 \quad (4)$$

Each of the k paths is required to start and end at a dummy node, V_0.

$$\sum_{j, j>0} x_{j,0} = k, \quad \sum_{j, j>0} x_{0,j} = k \quad (5)$$

In order to avoid paths composed of disconnected cycles, we follow Liu et al [13] below to set up some sub-tour elimination constraints (i.e., equations (6)~(8)) as proposed by Kara and Bectas [20].

$$u_i + (L-2)x_{0,i} - x_{i,0} \leq L - 1, \forall i > 0 \quad (6)$$
$$u_i + x_{0,i} \geq 2, \forall i > 0 \quad (7)$$
$$u_i - u_j + L x_{i,j} + (L-2)x_{j,i} \leq L - 1, \forall i, j > 0, i \neq j \quad (8)$$

,where L is the maximum number of visited nodes in a path, i.e., $2n$+1. The inequalities (6) and (7) constrain the upper and lower bounds of the number of nodes visited by each path. The inequality (8) ensures that $u_j = u_i + 1 \equiv x_{i,j} = 1$.

In addition, we set up two additional constraints (9) and (10) in our design. First, to ensure that the two endpoints of the same line segment are connected, one of the directed edges between the endpoints must be selected.

$$x_{i,j} + x_{j,i} = 1, \text{for node } v_i, v_j \text{ on the same segment} \quad (9)$$

Since the edge segments generated in the previous step are all adjacent to each other and we only want to connect adjacent segments, we must ensure that non-adjacent segments are not connected. Second, we set another constraint as follows.

$$x_{i,j} + x_{j,i} = 0, \text{for disconnected } P_i, P_j \quad (10)$$

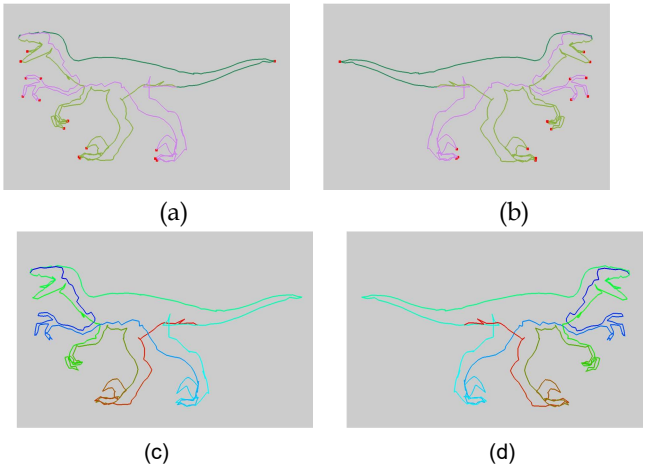


Fig. 15. The result of wire composition. (a) The three edge segments (in different colors) extracted in the previous step.(b) Another viewpoint of the edge segments.(c) After wire composition, only a single continuously rough wire is obtained. We use the red-green-blue gradient color to represent the detour of the wire. (d) Another viewpoint of the rough wire.

We use CPLEX Optimizer to calculate the objective function (3), a binary integer optimization problem, and use above constraints to solve the problem. Finally we get k continuous lines $W = \{W_1,...,W_k\}$, i.e., k is the minimum number of paths optimized from equation (3) to calculate mTSP, which we call "rough wires". These rough wires will be used to generate the final result. Fig. 15 is the result of wire composition.

6 Wire Art Smoothing and Creation

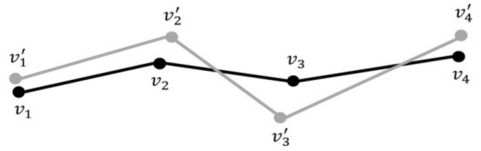


Fig. 16. The vertices v_i' of the rough wire (gray) and ones v_i of the smoothed wire (black).

Since the rough wires produced in the previous step are made up of edges on the mesh, the wires may look very uneven. Therefore, we want to smooth the wires while preserving the overall shape of the wires. We solve an optimization problem that uses an iterative approach to adjust the position of the vertices on the wires. As shown in Fig. 16, the gray wire is the original rough wire. We adjust the position of the vertex v_i' on the wire and calculate the corresponding new vertex v_i to form a smoother black wire for each rough wire:

$$argmin_{v_i} \sum_{i=3}^{n} \|(v_i - v_{i-1}) - (v_{i-1} - v_{i-2})\|^2 + \alpha \sum_{i=1}^{n} \|v_i - v_i'\|^2 \quad (11)$$

,where n is the number of vertices on the wire. The first term in the function is to make the vectors between the neighbor edges on the wire as same as possible, to achieve the purpose of smoothing the wire. The first term of Equation (11) is a kind of *Laplacian* energy for the polyline, which is used in geometry processing for smoothing. The second term is to hope that the newly calculated vertices on the wire are as close as possible to the original one, to retain the overall shape of the wire. α is a constant that measures the relative importance of these two terms, and in our experiment α =1.0. In addition, in order to preserve the features on the wire, we have the following constraints:

$$v_i = v_i', \text{ for each critical point } v_i \quad (12)$$

The purpose of this constraint is not to change the position of the critical points so that the wire can keep the features of the original model. We will iteratively calculate the equation (11), constantly changing the position of the wire vertices until the difference ratio between the function value of current iteration and the one of the previous iteration is less than the threshold s or it reaches the maximum iteration number t. In our experiment, s = 0.5, t = 3, and the user can adjust these two parameters based on the generated results. We also use CPLEX Optimizer to solve this optimization problem. Fig. 17 is an example of a smoothing wire. In Figure 18, we show the comparison of the results with different α values and experimentally find α = 1.0 can have smooth wires without creating overlaps or intersections in our examples.

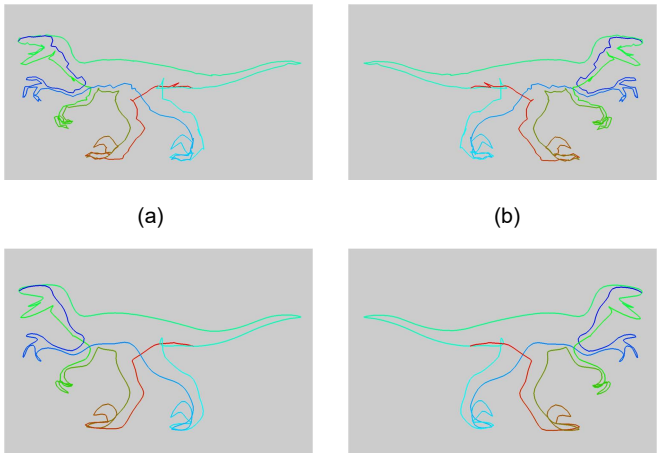


Fig. 17. An example of a smoothing wire. (a) Rough wire from previous step. Gradient color indicates the order of the detour. (b) Another view of the rough wire. (c) Smoothed wire after one iteration. (d) Another view of the smoothed wire.

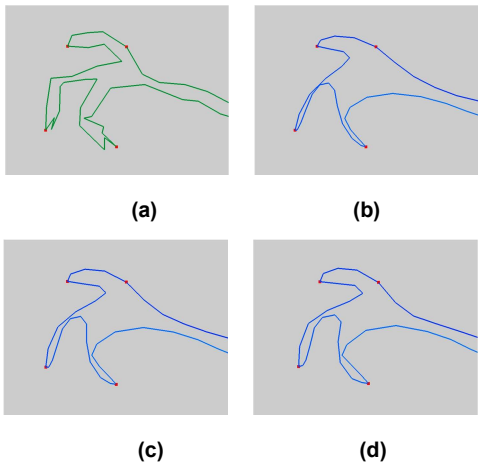


Fig. 18. The comparison of the results with different α values. (a) Rough wire; (b) Smoothed wire with α = 0.5; (c) Smoothed wire with α = 1.0; (d) Smoothed wire with α = 2.0.

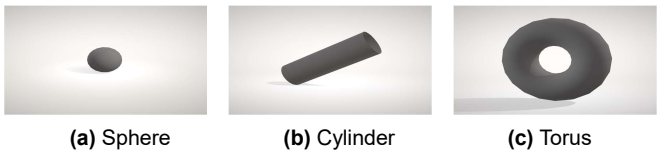


Fig. 19. The three kinds of surfaces used to construct the wire.

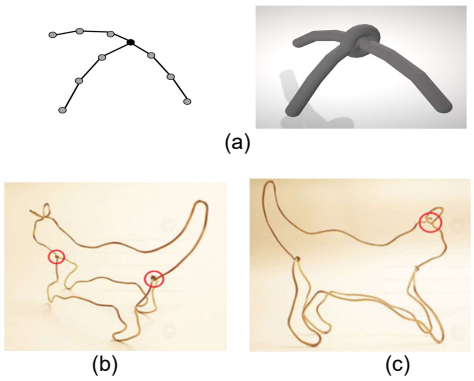


Fig. 20 illustrates how we constructed the wire model, which is the final output of our pipeline.

In order to make the results look like real wire arts, as shown in Fig. 19, we can also optionally use sphere, cylinder, and torus to construct wires, 1) sphere: constructs the vertices of the wire, 2) cylinder: constructs the edges of the wire and 3) torus: constructs the intersection between wires. Fig. 20 (a) shows an example. Our purpose to use the torus is to imitate the twisting effect by the real wire art. See a real artist's example in Figure 20(b). There is one wire forming the shape of a cat with the twisting connection (i.e., highlighted by red color) at the beginning and the end of it. We follow this rule to only create the torus at the beginning and the end of a wire. In addition, in our design, if there are two wires crossing each other, there is no torus on the intersection point because it is not the beginning or the end of the wire. For this case, in our design, they will directly cross each other like the artist's example (see Figure 20(c)) without any twisting connection.

7 Results and Discussion

Implementation Environment. We use Visual Studio 2013 to create a Windows Form project that provides a graphical interface for the user. In addition, we use OpenGL to implement the interactive interface function, which allows the user to view and draw the specified edges of the model, and finally present our results. In the extraction of edge segments, we implement the algorithm in C++. In the wire composition and smoothing part, we use CPLEX Optimizer (version 12.63) to solve the optimization problem. All results are generated on the same computer with Intel i7-6700 CPU (3.40GHz), 24G RAM, and we will show the performance of the program in the following section. All parameters are set the same for all our experiments.

Automatically Generated Results. In this section, we present 10 auto-generated results. We compare 2 results with the results with user-specified edges to prove that our method allows the user to produce a variety of results.

Table 1 and Table 2 are statistics for extracting edge segments. The more triangles of the model, the more iterations of pair combination, and the more execution time are needed to check the constraints. Table 3 and Table 4 are statistics for wire composition and how many wire lines is generated for each wire model. As shown in the table, the number of connected wires is extremely concise. The more number of edge segments, the longer the execution time will be. With user-specified lines, we can increase the number of wires as shown for an example in 11-Fertility. This is because we add more wire lines from our segmentation to increase wires to match user desire.

Currently, the computational timing is limited on edge segment extraction and this cost is related to how many number of polygons are processed in our system. The timing cost of the wire smoothing (Table 5 and Table 6) for our generated results is very fast, i.e., less than 0.104 second for all examples in our experimental results.

Extraction of edge segments (automatically)			
Model	# triangles	# edge segments	Duration (sec.)
1-Raptor	2500	3	1.666
2-Shark	2500	8	1.374
3-Butterfly	2500	3	1.360
4-Rabbit	4000	6	3.197
5-Deer Head	10000	3	19.707
6-Scorpion	5000	3	5.196
7-Stork	10000	2	19.79
8-Chair	10000	17	19.94
9-Fertility	3000	29	2.393
10-Elephant	3000	23	2.047

Table 1. The statistics of extraction of the edge segments for the automatically generated results.

Extraction of edge segments (with user-specified edges)			
Model	# triangles	# edge segments	Duration (sec.)
11-Fertility	3000	38	2.156
12-Elephant	3000	44	1.979

Table 2. The statistics of extraction of the edge segments for the results with user-specified edges.

Wire composition (automatically)			
Model	# edge segments	# connected wires	Duration (sec.)
1-Raptor	3	1	0.031
2-Shark	8	2	0.061
3-Butterfly	3	1	0.033
4-Rabbit	6	2	0.032
5-Deer Head	3	1	0.048
6-Scorpion	3	1	0.033
7-Stork	2	1	0.033
8-Chair	17	6	0.038
9-Fertility	29	10	0.050
10-Elephant	23	7	0.085

Table 3. The statistics of the wire composition for the automatically generated results.

Extraction of edge segments (with user-specified edges)			
Model	# triangles	# connected wires	Duration (sec.)
11-Fertility	3000	14	0.598
12-Elephant	3000	16	0.555

Table 4. The statistics of the wire composition for the results with user-specified edges.

Smooth wire (automatically)	
Model	Duration (sec.)
1-Raptor	0.053
2-Shark	0.032
3-Butterfly	0.023
4-Rabbit	0.043
5-Deer Head	0.094
6-Scorpion	0.069
7-Stork	0.076
8-Chair	0.093
9-Fertility	0.057
10-Elephant	0.063

Table 5. The statistics of the smoothing wires for the automatically generated results.

Smooth wire (wire user specified edges)	
Model	Duration (sec.)
11-Fertility	0.104
12-Elephant	0.072

Table 6. The statistics of the smooth wires for the results with user-specified edges.

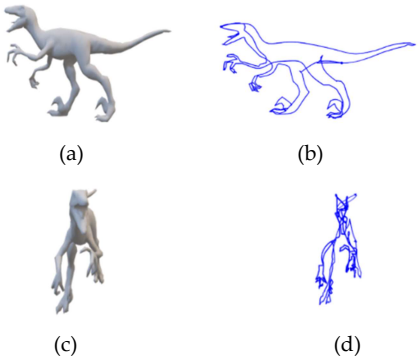


Fig 21. Comparsion between original model and our wire results. Distance error $D(a,b)$ = 0.000236924 and $D(c,d)$ = 0.00131612. Note we will define distance error in equation (13) later.

In this paper, our purpose is to try to find the minimum number of wires to create wire results. Our method can not guarantee good perception at all views of created results. For example, for the results of model 1-Rapter in Figure 21(d), the 1-Raptor result can not be well viewed due to that it does not contain details in this view, i.e., details are not preserved from our extraction of edge segments. Therefore, our method creates view-dependent results, i.e., better viewing results are not at all viewing directions. Figure 21(c) also shows rendered 1-Raptor result. This viewing also does not see its details well. In fact, our 21(d) also looks similar to 21(c).

In the part of the wire composition, we also implement the method of Liu et al. [**3**] and compare the generated results by our method. The method of Liu et al. [**3**] may connect the same endpoint of the edge segment to other different edge segments. In order to avoid this discontinuity, when this happens, we disconnect the edge segments. Fig. 22 is an example of the disconnection of the edge segments. In most cases, our improved method yields a smaller number of wires. In this example, assume that the connection order of the edge segments calculated by mTSP is 1-2-3, in which edge segment 2 is connected with edge segment 1 and edge segment 3 by the same endpoint, thus causing the edge to be discontinuous. So we break the connection of edge segment 2 and edge segment 3, eventually forming 2 continuous wires. Figure 23 shows some comparison with Liu et al. [3]. Our method can generate less number of wires than [3].

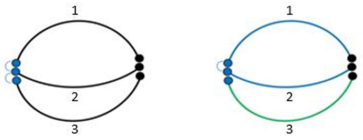


Fig. 22. (a) The three edge segments that were originally to be connected. (b) The result of disconnecting edge segment 2 and edge segment 3.

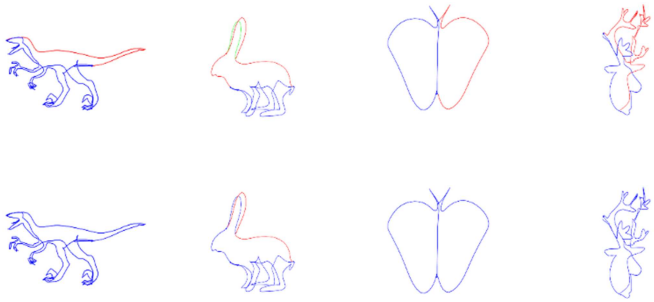


Fig. 23. Shows some comparison with Liu et al. [**3**]. For each pair comparison, top side is created by [3] and bottom side is created by ours. In each example, different wires are drawn with different colors.

To quantitativly evaluate our method to various input models, we evaluate it by using a unity function in OpenCV called matchShape that takes in two images, i.e., outer contours of both input mesh and its wire result and finds the distance between them using Hu Moments. If the distance is small, the shapes are close in appearance and if the distance is large, the shapes are farther apart in appearance. Let $D(A,B)$ be the distance between shapes A and B, and H_i^A and H_i^B be the i^{th} log transformed Hu Moments (https://www.learnopencv.com/shape-matching-using-hu-moments-c-python/) for shapes A and B. We compute the following distance defined as follows.

$$D(A,B) = \sum_{i=0}^{6} \left| \frac{1}{H_i^B} - \frac{1}{H_i^A} \right| \tag{13}$$

In our experiments, the distances in all examples approximately range from 0.001 to 0.01 Finally, we show our results in Figure 24 ~ 32. For each figure, we show a single view of a wire result and its $D(A,B)$. For different views of each wire result, please see our supplementary file.

Limitation and User-specified Edges. If the surface on the input model is too smooth, it may cause the automatically generated wires to be irregular. Because in the extraction of edge segments, we sort all the costs of the pairs and then merge sets from the pair with the smallest cost. If some regions of the model are too smooth (i.e., see cylinder examples in our supplementary video), the costs of the pairs will be very similar. So at the beginning of the combination, the sets of the smoothed regions are combined randomly. Once the important edges that should be extracted are removed, the result cannot be changed, resulting in the extracted edge segments not presenting the shape of the input model. Figure 31 is the result of extracting the edge segment using model 9-Fertility. If the program automatically generates lines, the lines on the smooth area will be irregular and the model features will not be presented. The result is shown in Figure 31 (b). Similarly, automatic method may not create user-like results. To improve the result, we must add the user-specified edges (see Figure 31 (c)) so that these edges are not removed during the extraction of the edge segments, as shown in Figure 31 (d). Another example without/with requiring user-specified edges is shown in Figure 32.

(A) (B)

Fig. 24. Result of model 1-Rapter. $D(A,B)$ =0.00304063.

Fig. 25. Result of model 2-Shark. D(A,B) = 0.00587722.

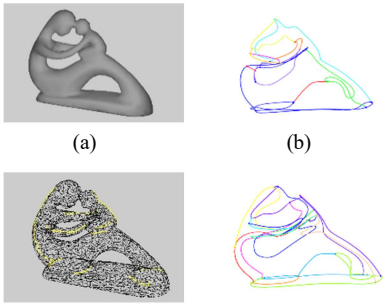
Fig. 26. Result of model 3-Butterfly. D(A,B) = 0.000857082.

Fig. 27. Result of model 4-Rabbit. D(A,B) = 0.00364158.

Fig. 28. Result of model 5-Deer Head. D(A,B) = 0.000249258.

Fig. 29. Result of model 6-Scorpion. D(A,B) = 0.000614232.

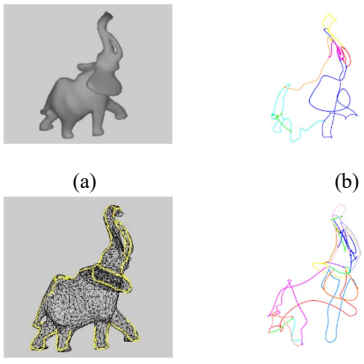
Fig. 30. Result of model 7-Stork. D(A,B) = 0.00169231.



(a) (b)

(c) (d)

Fig. 31. (a) 9-Fertility and (b) our automatically generated result with D(A,B) = 0.00239587 (c) 11-Fertility with user-specified edges and (d) its generated result with D(A,B) = 0.00197667.



(a) (b)

(c) (d)

Fig. 32: (a) 10-Elephant and (b) our automatically generated result with D(A,B) = 0.00365977 (c) 12-Elephant with user-specified edges and (d) its generated results with D(A,B) = 0.00188513.

8 Conclusion

We practically propose a method for generating 3D virtual wire art from a three-dimensional model. By removing

the edges on the smooth region, we use the retained edges as line segments which can present the shape of the input model. We then use an improved mTSP method to connect the segments and minimize the number of merged lines. Finally, we smooth the lines and use the information of the lines to construct 3D virtual wire art as the artist does. In our design, we also can allow the users to select some lines to yield their designed results. In addition, our segmentation method may not satisfy the users' desire. So, this interactive editing utility can be helpful to meet artists' desire. Currently, our computation timing is limited by segmentation cost. In future, we may find better idea and fast way to automatically segment models. Finally, we also plan to design/generate multiple view line art from two or three models to generate a virtual line art that can see the contours of different models from different perspectives by automatically finding out the best-viewing contours.

REFERENCES

[1] Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot, and David Salesin. Diffusion curves: A vector representation for smooth-shaded images. In *ACM SIGGRAPH 2008 Papers*, SIGGRAPH '08, pages 92:1–92:8, New York, NY, USA, 2008. ACM.

[2] Fernando J. Wong and Shigeo Takahashi. A graph-based approach to continuous line illustrations with variable levels of detail. In *Computer Graphics Forum (Proceedings of Pacific Graphics 2011)*, volume 30, pages 1931–1939, 2011.

[3] Lingjie Liu, Duygu Ceylan, Cheng Lin, Wenping Wang, and Niloy J. Mitra. Image based reconstruction of wire art. *ACM Trans. Graph.*, 36(4):63:1–63:11, July 2017.

[4] Ren Suzuki, Masaki Moriguchi, and Keiko Imai. Generation and optimization of multiviewwire art. Poster presented at the Pacific Graphics, October 2017.

[5] Kai-Wen Hsiao, Jia-Bin Huang, Hung-Kuo Chu, Multi-view wire art. *ACM Trans. Graph.*, 37(6):242:1-242:11

[6] Shiwei Li, Yao Yao, Tian Fang, and Long Quan. Reconstructing thin structures of manifold surfaces by integrating spatial curves. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2887–2896, June 2018.

[7] Ravish Mehra, Qingnan Zhou, Jeremy Long, Alla Sheffer, Amy Gooch, and Niloy J. Mitra. Abstraction of man-made shapes. In *ACM SIGGRAPH Asia 2009 Papers*, SIGGRAPH Asia '09, pages 137:1–137:10, New York, NY, USA, 2009. ACM.

[8] Ran Gal, Olga Sorkine, Niloy J. Mitra, and Daniel Cohen-Or. iwires: An analyze-andedit approach to shape manipulation. In *ACM SIGGRAPH 2009 Papers*, SIGGRAPH '09, pages 33:1–33:10, New York, NY, USA, 2009. ACM.

[9] Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. Fibermesh: Designing freeform surfaces with 3d curves. In *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[10] Rundong Wu, Huaishu Peng, François Guimbretière, and Steve Marschner. Printing arbitrary meshes with a 5dof wireframe printer. *ACM Trans. Graph.*, 35(4):101:1– 101:9, July 2016.

[11] D Cohen-Steiner, P Alliez, M Desbrun , Variational shape approximation, ACM TOG Vol.23 Issue 3, August 2004 (SIGGRAPH 2004),905-914

[12] Emmanuel Iarussi, Wilmot Li, and Adrien Bousseau. 2015. WrapIt: computer-assisted crafting of wire wrapped jewelry. ACM Trans. Graph. (Proc. of SIGGRAPH Asia) 34,6 (2015), 221.

[13] Eder Miguel, Mathias Lepoutre, and Bernd Bickel. 2016. Computational Design of Stable Planar-rod Structures. ACM Trans. Graph. (Proc. of SIGGRAPH) 35, 4, Article 86 (2016),

[14] Jonas Zehnder, Stelian Coros, and Bernhard Thomaszewski. 2016. Designing Structurally-sound Ornamental Curve Networks. ACM Trans. Graph. (Proc. Of SIGGRAPH) 35, 4, Article 99 (2016), 10 pages.

[15] Wallace Lira, *Chi-Wing Fu*, and Hao (Richard) Zhang. Fabricable Eulerian Wires for 3D Shape Abstraction, *ACM Transactions on Graphics* (*ACM TOG*), vol. 37, no. 6, article no. 240, SIGGRAPH Asia, 2018.

[16] Mario Botsch and Leif Kobbelt. A remeshing approach to multiresolution modeling. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, SGP '04, pages 185–192, New York, NY, USA, 2004. ACM.

[17] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 209–216, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[18] Yinan Zhou and Zhiyong Huang. Decomposing polygon meshes by means of critical points. In *10th International Multimedia Modelling Conference, 2004. Proceedings.*, pages 187–195, 2004.

[19] Emil Ernerfeldt. Fitting a plane to noisy points in 3d, September 2017, https://www.ilikebigbits.com/2017_09_25_plane_from_points_2.html

[20] Imdat Kara and Tolga Bektas. Integer linear programming formulations of multiple salesman problems and its variations. *European Journal of Operational Research*,174(3):1449 – 1458, 2006