Generating Virtual Wire Sculptural Art from 3D Models

CHIH-KUO YEH, School of Computer Science and Software, Zhaoqing University, China
THI-NGOC-HANH LE, ZHI-YING HOU, and TONG-YEE LEE, National Cheng-Kung University, Taiwan, Republic of China

Wire sculptures are objects sculpted by the use of wires. In this article, we propose practical methods to create 3D virtual wire sculptural art from a given 3D model. In contrast, most of the previous 3D wire art results are reconstructed from input 2D wire art images. Artists usually tend to design their wire art with a single wire if possible. If not possible, they try to create it with the least number of wires. To follow this general design trend, our proposed method generates 3D virtual wire art with the minimum number of continuous wire lines. To achieve this goal, we first adopt a greedy approach to extract important edges of a given 3D model. These extracted important edges become the basis for the subsequent lines to roughly represent the shape of the input model. Then, we connect them with the minimum number of continuous wire lines by the order obtained by optimally solving a traveling salesman problem with some constraints. Finally, we smooth the obtained 3D wires to simulate the real 3D wire results by artists. In addition, we also provide a user interface to control the winding of wires by their design preference. Finally, we experimentally show our 3D virtual wire results and evaluate these 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.

CCS Concepts: • **Computing methodologies → Image manipulation;**

Additional Key Words and Phrases: Wire sculpture, edges, 3D wire art, edge segments, wire composition, smoothing

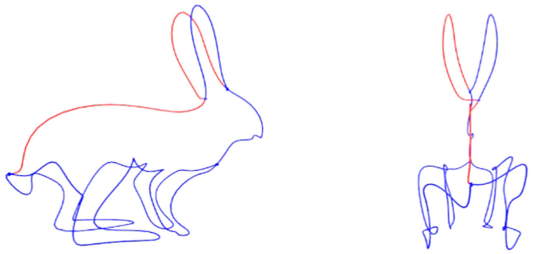
ACM Reference format:
Chih-Kuo Yeh, Thi-Ngoc-Hanh Le, Zhi-Ying Hou, and Tong-Yee Lee. 2022. Generating Virtual Wire Sculptural Art from 3D Models. *ACM Trans. Multimedia Comput. Commun. Appl.* 18, 2, Article 51 (February 2022), 23 pages.
https://doi.org/10.1145/3475798

1 INTRODUCTION

Compact, abstract 3D shape representations are visually more appealing and interesting than the original models, which seem to be visually cluttered. 3D wire as sculpture is an interesting art created by the artist using wire lines. Selecting suitable and a small number of wire lines can show visually significant parts of a 3D object. For instance, we show similar wire sculptural art forms of

This work was supported in part by the Ministry of Science and Technology (under nos. 108-2221-E-006-038-MY3 and 110-2221-E-006-135-MY3), Taiwan, Republic of China.
Authors' addresses: C.-K. Yeh, School of Computer Science and Software, Zhaoqing University, China; email: simpson.ycg@gmail.com; T.-N.-H. Le, Z.-Y. Hou, and T.-Y. Lee, National Cheng-Kung University, Tainan City 701, Taiwan, Republic of China; emails: ngochanh.le1987@gmail.com, denny1994a@gmail.com, tonylee@mail.ncku.edu.tw.

1551-6857/2022/02-ART51 $15.00
https://doi.org/10.1145/3475798



(a) Side view (b) Side back view

Fig. 1. The images are created by using our method in the wire sculptural art form of Theodore Huckins.

the artist Theodore Huckins that are created by our method. Intuitively, Figure 1(a) is easier to be recognized as a rabbit than Figure 1(b). In general, the artists' results are always view dependent.

Wire art has been applied to many applications such as furniture design, wire-wrapped jewelry, and wire sculptural art. Usually, the artists require a time-consuming manual procedure to design and adjust their wire sculptural art. Without expert experience and their time-consuming manual beautification, their wire sculptural results may consist of too many short wires, and thus it easily lacks appealing results. In this article, our goal is to design a 3D virtual wire sculpture with a smaller number of wires and without too time-consuming a manual design and computational time. In this article, we introduce a novel methodology for abstracting three-dimensional shapes using a smaller number of wires. Our results are easily and efficiently created and comparable to arts' design results. In contrast to our work, most of 3D wire art work is based on surface reconstruction of 2D wire art input images [8, 13, 20]. We propose a different way to create wire art from 3D models. We experimentally show our 3D virtual wire results and evaluate these created results. Our proposed method is computed effectively and interactively, and results are attractive and similar to real 3D wire art work.

In summary, our contributions are listed as follows:

- We create an automatic system to create 3D virtual wire sculpture art for a given 3D model by extracting its edge segments and capable of efficiently creating appealing 3D virtual wire art.
- Motivated by artists' design goal, we propose a traveling salesman optimization with some constraints to find the minimum number of wire lines for creating our 3D virtual wire sculpture.
- We optimally smooth wire lines to make results more appealing and similar to real 3D wire art.

2 RELATED WORK

In this section, we briefly review the related works to create 2D/3D line representation from images and 3D models.

Orzan et al. [17] proposed a vector-based diffusion curve method to draw a smooth-shaded image allowing vector-based primitives, key-frame animation and stylization, and so forth. Wong and Takahashi [21] introduced a method for automatically generating continuous line illustrations, and drawings consisting of a single line, from a given input image. This approach can generate different amounts of details drawn in their line illustrations, particularly usefully maintaining important features while simplifying the overall illustration of the input image.

Using spatial curves, Li et al. [11] proposed a method for reconstructing the fine and thin structure of an object. In this paper, Li et al. [11] 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. [13] presented an image-based method to reconstruct wire objects from a few input images, and their reconstructions faithfully capture both 3D geometry and the topology of the wires. Later, Suzuki et al. [20] 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 the line drawings is connected and they have the same height. Recently, Hsiao et al. [8] presented a computational framework for automatic creation of multi-view 3D wire sculpture. Their system takes two or three user-specified line drawings and the associated viewpoints as inputs. These three methods are based on 2D wire art images and reconstruct single [13] or multi-view 3D wire sculptures [8, 20]. In contrast to their purpose, our method focuses on generating 3D virtual wire art models based on 3D input models rather than 3D reconstructions from 2D images [8, 13, 20]. Our approach generates 3D virtual wire art with the minimum number of continuous and smooth 3D wire lines to simulate the real 3D wire results by artists.

Mehra et al. [14] proposed a novel method for abstracting 3D models using characteristic curves or contours as building blocks for the abstraction. Gal et al. [6] 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. [16] presented a system for designing free-form surfaces with a collection of 3D curves. Wu et al. [22] presented a method to print arbitrary meshes on a 5DOF wireframe printer. Cohen-Steiner et al. [2] 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 is also some related research concentrated on modeling and fabricating the wire sculptures of a specific form. Iarussi et al. [9] proposed a framework to help the creation of wire-wrapped jewelry. Miguel et al. [15] designed a computer-aided tool to facilitate converting a 3D model into a stable, self-supporting wire sculpture. Zehnder et al. [23] 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. [12] 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 [9, 12, 15, 23] of 3D wire art design and modeling but applies a fundamentally different design goal and approach. Our approach does not require either input 3D wire model or 2D wire images.

The other relative work to our research is the mesh segmentation techniques [18, 19], but our work/goal is quite different on the extracted consistent boundary segments. The mesh segmentation techniques [18, 19] always segment a mesh into several semantic components. In such components, they do not consider the connectivity between boundaries, while our goal is to generate a single or minimal number of continuous line(s) on the surface to represent the original surface. The other similar work consists of view- and scale-independent ridge-valley line techniques [4], but this approach usually generates too many lines, which is contrary to our design goal, i.e., create a single continuous line to present our wire art work.

3 SYSTEM OVERVIEW

As shown in Figure 2, we present our system that allows users to create appealing 3D virtual wire art efficiently with a smaller number of wire lines. Given a 3D model, we first automatically extract

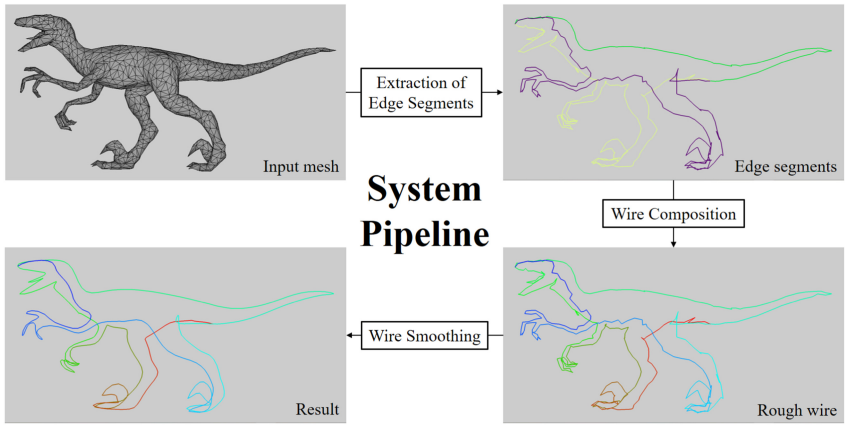


Fig. 2. Our system overview.

its visually important edge segments to users. Optionally, we allow users to select some extra edges according to their personal interests. Then, we apply a traveling salesman optimization with some constraints 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.

Preprocessing. Our preprocessing remeshes each given 3D model in order to avoid the appearance of narrow triangles on the mesh. In our 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 may be not as expected. To alleviate this issue, we adopt the method proposed by Botsch and Kobbelt [1] to remesh the input model with the same number of triangles as the original model. If the sizes and shapes of the triangles on the input model 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[1] if necessary. This preprocessing is not the focus of our work in this article.

4 WIRE CONSTRUCTION AND SMOOTHING

4.1 Extraction of Edge Segments

Inspired by Garland and Heckbert [7], 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. In our article, we use the term *"edge segment"* to mention such extracted edges. Our goal is to remove the less important edges of the input mesh. These removed edges are usually located in smoother areas of the mesh, and the remaining edges are more important, 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).

In the above manner, our 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 in the example in Figure 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,

[1] https://www.meshlab.net/.

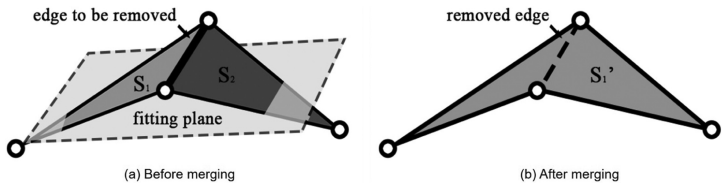


Fig. 3. The merging of two triangles S_1 and S_2 to S_1' by [7].

their common edges will be removed. The new result is shown in Figure 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 retain the feature vertices, i.e., critical points, on our designed 3D wire art. For this purpose, we apply the method of Zhou and Huang [24] to find critical points of our input 3D model. In the process of extracting the *edge segments*, we make sure that these critical points are located on some extracted edge segments. Next, we generate n sets $\{S_1, S_2, \ldots, S_n\}$ 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 boundary vertices we calculate a fitting plane P_f by Emil [5] and then measure the differences between all triangles in (S_i, S_j) and P_f. We treat the differences 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 $\mathbf{T}$ as all the triangles in this pair, i.e., $\mathbf{T} = \{T_1, T_2, \ldots, T_t\}$, where t is the number of triangles in $\mathbf{T}$. In this merged region of $\mathbf{T}$, we find all the vertices V_b at the outer boundary of this merged region. Then, we use these boundary vertices as input to calculate the fitting plane by using Emil [5]. There are exactly two directions perpendicular to this fitting plane, and we select the one with more similar direction to average normal vectors of all triangles in $\mathbf{T}$. Then, we use Equation (1) to compute a cost C to measure the smoothness of this merged region:

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

where A_i is the area of triangle T_i; A_{sum} is the total area sum of all triangles in $\mathbf{T}$; 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 the merged region is smooth and vice versa.

We 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 extra 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 draw some edges in advance where they wish to extract. In this manner, if the common edges between the merged sets contain user-specified edges, we do not merge this pair. Therefore, 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 describe these constraints in detail

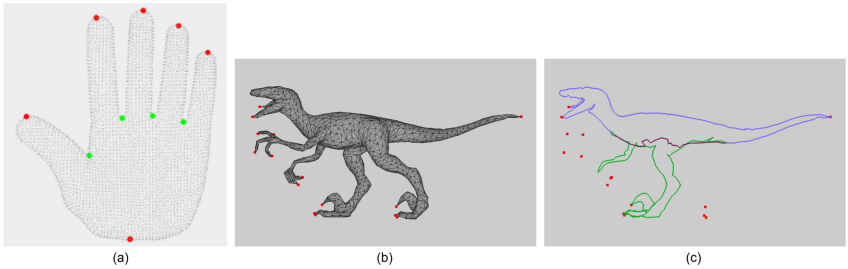


Fig. 4. (a) Critical vertices (red: local extrema, green: saddle vertices, gray: regular vertices). (b) Feature constraints selected from Zhou and Huang [24] and user specified. (c) These extracted edges cannot capture the features of the model without feature constraints (2).

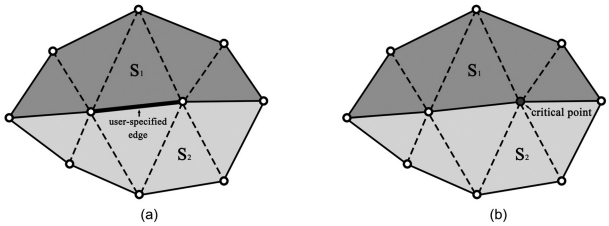


Fig. 5. (a) The shared edges between S_1 and S_2 contain a user-specified edge. (b) A critical point is on the shared edges between S_1 and S_2.

in a later section. In preprocessing, we 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:

- Step 1: Find the critical points (see an example in Figure 4(a)) on the mesh by Zhou and Huang [24].
- Step 2: Place each triangle T_i in the corresponding set S_i and find all adjacent sets to make them a pair.
- Step 3: Find the boundary vertices V_b in each pair of sets, thereby obtaining the fitting plane P_f by [5] and then calculate the cost C (i.e., by Equation (1)) of merging the pair.
- Step 4: Put all pairs of sets into a heap in order of small to large.
- Step 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 edges of the pair are removed from the mesh.
- Step 6: Update the cost of the set that is 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 the above merging processing is based on relative cost C on a heap, which is in increasing order.

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 Figure 5(a), S_1 and S_2 are the sets to be merged and their shared edges contain a user-specified edge. If they are merged, the user-specified edge is removed. In order to keep it, we do not merge this pair.

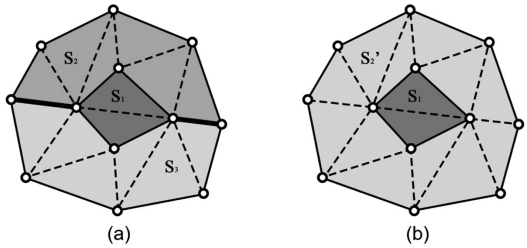


Fig. 6. The combination of S_2 and S_3 with discontinuous shared edges. This will separate the extracted edges. (a) The shared edges between S_2 and S_3 are not continuous. (b) S_2' is the combined set of S_2 and S_3.

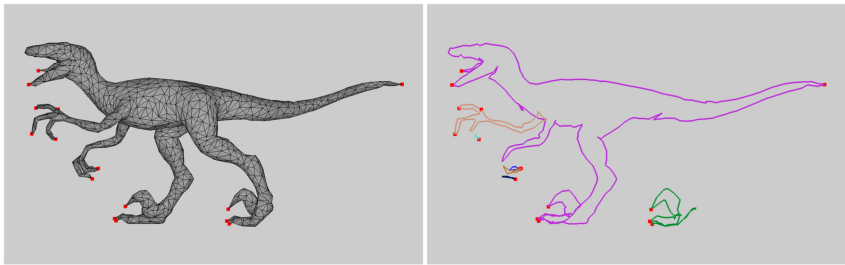


Fig. 7. The result without the constraint (3). The extracted edges are disconnected.

(2) **A critical point is on the shared edges.** As shown in Figure 5(b), 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).
(3) **The shared edges of the pair are discontinuous.** As shown in Figure 6(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 Figure 6(b). However, we want the resulting edge segments to be connected. Thus, we do not merge this pair in this case. Without this constraint, the result of extracted edge segments is shown in Figure 7(b).
(4) **Adjacent vertices of the sets are not on the shared edges.** As shown in Figure 8(a), S_3 and S_4 are the sets to be merged. Among the triangle vertices contained in the sets, there are three 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, this yields the result as shown in Figure 8(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 Figure 8(c), where S_2' is the merged set of S_2 and S_3'. To avoid this phenomenon, we do not recommend to merge S_3 with S_4 in advance. Without this constraint, the result of extracted edge segments is shown in Figure 9.

Using these two constraints (3) and (4), the main purpose is to make sure extracted edges are connected. However, for example, in both Figures 7(b) and 9(b), the extracted edges are disconnected if we do not have (3) and (4) constraints in these two examples.

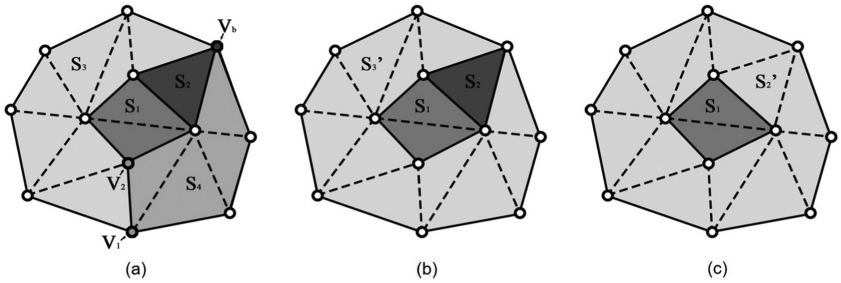


Fig. 8. The merging of S_3 and S_4 with shared vertex V_b not on the merged edge. This could also separate the extracted edges. (a) V_1, V_2, and V_b are all the merged vertices between S_3 and S_4. (b) The result of merging S_3 and S_4. S_3 is the merged set. (c) After merging of S_2 and S_3. S_2 is the merged set.

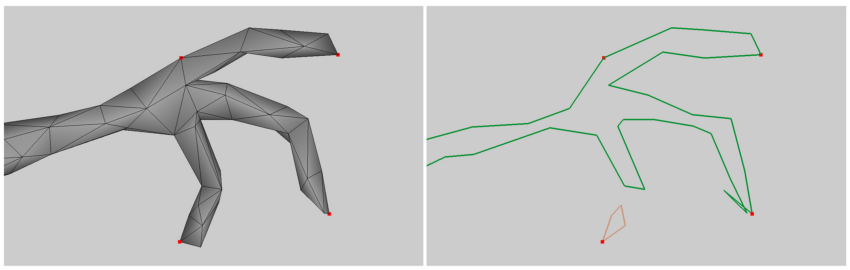


Fig. 9. Without this constraint (4), the extracted edges can be disconnected.

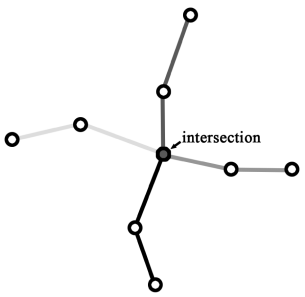


Fig. 10. Four edge segments (in different grayscale color) are separated at the intersection.

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 called *extracted edges*. We want to merge successive *extracted edges* into a single line segment. If there are intersections between the segments, then the edges that are adjacent to the intersections belong to different segments, as shown in Figure 10. We call them *edge segments*. Figure 11 is an example of the extraction of edge segments. There are 16 critical points, three retained sets, and three extracted edge segments. We will use these edge segments to create wire lines in later steps.

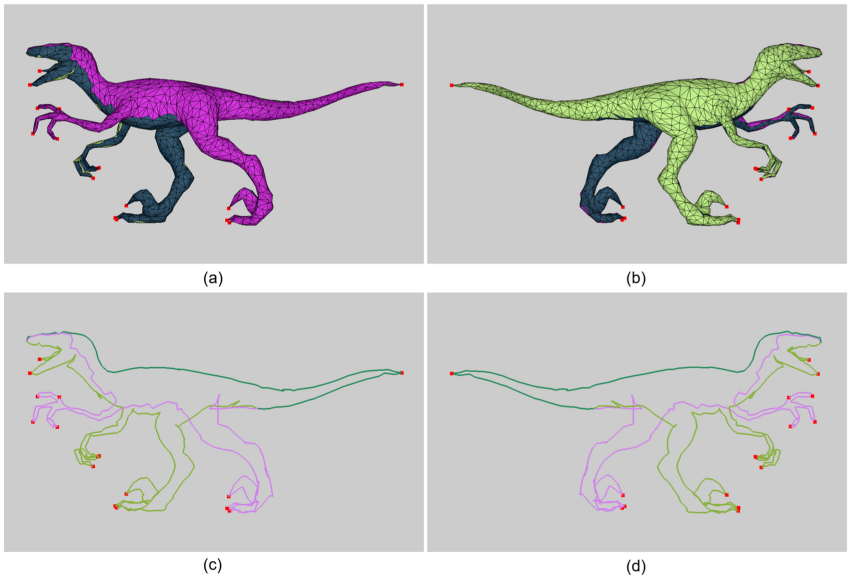


Fig. 11. An example of extracted edge segments. The result keeps the shape of the original model. Note that all critical points (in red) are located on extracted edges. (a) The result has three retained sets of triangles. (b) Another view of retained sets. (c) The result has three extracted edge segments. (d) Another view of extracted edge segments.

5 WIRE COMPOSITION

Liu et al. [13] proposed a method of connecting 3D 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 the probability of connection is. Since Liu et al. [13] treat the line segments as nodes to calculate the costs and then connect them, the resulting lines may not be continuous. In their work, the pair-wise cost between two line segments s_i, s_j is formulated as

$$e_{i,j} = d(s_i, s_j) + \mu * \Gamma, \tag{2}$$

where $d(.)$ denotes the distance in 3D between the closest endpoints of two line segments, and Γ is calculated as $(1 + cos\alpha)/2$; i.e., α is the angle between the tangent directions of these closest endpoints on two line segments. The larger the angle between the line segments is, the smaller the cost is.

We depict their method in Figure 12(a). In this example, the line segments are labeled 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, the distance from segment 1 to these four segments is calculated by Equation (2). The next segment is the one that has the smallest cost in such defined distances. In this case, the distance term of segments 2, 12, 14, and 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 ($\alpha = 180°$) 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 Figure 12(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 Figure 12(b),

needs five wires to connect a star. In contrast, our method (described later) just needs a wire to create a star in Figures 12(d) from 12(c). Note that in Figures 12(a) and 12(c), the number in each segment indicates the order of connection sequence after mTSP.

ALGORITHM 1: Algorithm of our mTSP

Input: Set of 3D line segments $C_{opt} = \{C_t\}$
Collect endpoints (P_i) of each C_t; /* Each C_t consists of two endpoints */
$P_{opt} = \{P_i\}$;
$\mathcal{V} \leftarrow P_{opt}, \mathcal{E} \leftarrow \{w_{i,j}\}$;
Construct graph $\mathbf{G} = (\mathcal{V}, \mathcal{E})$;
$V_0 \leftarrow$dummy node;
for each path p_k **do**
    $V^k = \{V_t^k\}$; /* Set of nodes that are visited in path p_k */
    $V_0^k \leftarrow V_0$; /* Starting a path with dummy node */
    $V_1^k \leftarrow V_i$; /* V_i is randomly selected in $\mathcal{V}$ */
    while $V_c \neq V_0$ **do** /* V_c is the last visited node */
        for each node $V_j \in \mathbf{G}(\mathcal{V} - V^k)$ **do**
            if V_j is on the same segment with V_c **then**
                Add V_j to V^k;
                $V_c \leftarrow V_j$;
            else
                Calculate cost from V_j to the nodes on $\mathbf{G}(\mathcal{V} - V^k)$ by Equation (2);
            end if
        end for
        Find the node (V_{min}) that has the smallest cost to V_j on $\mathbf{G}(\mathcal{V} - V^k)$;
        Add V_{min} to V^k;
    end while
end for
Find minimum number of paths by solving Equation (3) associated with the constraints in Equations (4) to (10);
Output: minimum number of paths.

Basic concept. Motivated by the aforementioned limitation in Liu et al. [13], our proposed method improves mTSP to connect line segments in space. In contrast to Liu's algorithm, 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. In this way, all the connections between the segments can be considered, rather than only the nearest ending points. In Figure 13, our varied mTSP only requires a single wire (Figure 13(c)), and Liu's method requires two wires (Figure 13(a)) in this example. As shown in Figure 13(d), there are three segments. Each of them composes of a pair of points $(P_{i,1}, P_{i,2})$, $i = 1 \ldots 3$. We consider these points as the nodes of the graph and constrain each of the 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 Figure 13(c)).

Later, in Section 7, we will show our comparison with Liu et al. [13] for wire composition in Figure 22. The results show that two methods create similar wire art, but our method requires a smaller number of wires.

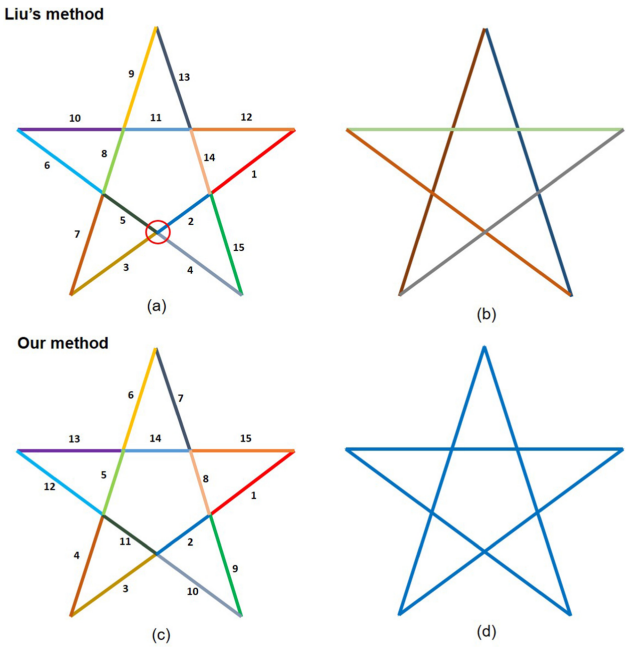


Fig. 12. (a), (c) The different colors of line segments denote the number of line segments to be connected and the numbers on the figure denote the connecting sequence. (b) The result of Liu's method using five different wires (i.e., with five different colors) to connect a star. (d) The result of our method using one wire to connect a star, i.e., needs a single blue color wire.

Method development and implementation. Given a set of 3D line segments $C_{opt} = \{C_t\}, t = 1, 2, \ldots, n$, where n is the number of segments, we can obtain a set including the endpoints of all segments $P_{opt} = \{P_i\}, i = 1, 2, \ldots, 2n$, where the two endpoints of C_t are P_{2t-1} and P_{2t}. We use P_{opt} to create a direct graph $\mathbf{G} = (\mathcal{V}, \mathcal{E})$, where each node $V_i \in \mathcal{V}$ represents an endpoint P_i. We also define a dummy starting node V_0 as the starting and ending node of each path. To find a path in the graph $\mathbf{G}$, we initially select a random node in $\mathcal{V}$. Since each node in the graph is an endpoint of a certain segment, we define the next visited node by two theories: (1) by the node that is on the same segment with the last visited node (V_c) or (2) by the node that has the smallest cost from V_c among those of the nodes on the graph. By chasing this procedure, a set of visited nodes in a path is defined. The mTSP is yielded by solving an optimization function. We summarized this procedure in Algorithm 1. Besides, to make our results better, we design the additional constraints that are explained later in this section.

In the completed graph $\mathbf{G}$, the value of each directed edge $e_{i,j} \in \mathcal{E}$ (from V_i to V_j) is assigned by a cost $w_{i,j}$. Such a cost $w_{i,j}$ is used to evaluate the pairwise relations between pairwise segments that V_i, V_j belong to. To calculate this cost, we adopt the pairwise cost by Liu et al. [13], which was mentioned earlier in Equation (2).

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 $\mathbf{G}$; each path starts and ends at V_0 and node $V_i (i > 0)$ is included in exactly one of the paths. Each path consists of the directed edges it passed through, and we assign the directed edge $e_{i,j} \in \mathcal{E}$ a binary variable $x_{i,j}$. $x_{i,j} = 1$ 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 $u_0 = 0$ corresponds to the dummy starting node V_0.

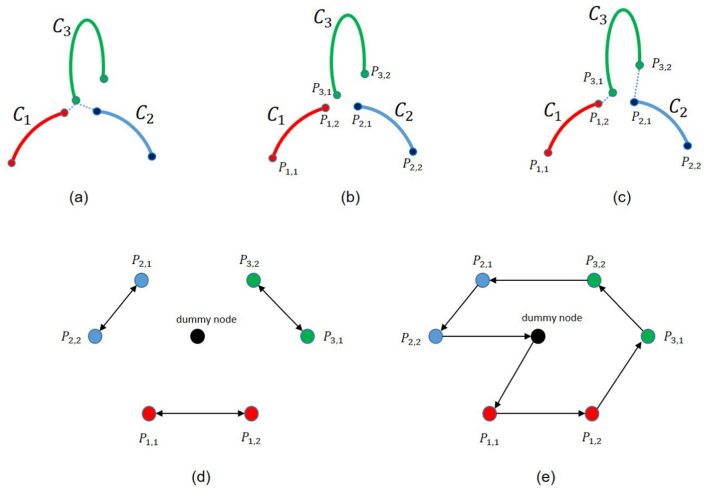


Fig. 13. (a) The nearest ending points between (C_1, C_3) and (C_2, C_3) are the same, resulting in non-continuous lines. (b) The two ending points $P_{i,1}, P_{i,2}$ on line segment C_i are the nodes of the directed graph. (c) The lines connected by our method are guaranteed to be continuous. (d) The 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; 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.

Our objective function is the same as that of Liu et al. [13] (see more detail in [13]). However, to ensure that the least number of lines is obtained, we set the *extra constraint* ξ as max $w_{i,j}$, leaving the variable k to minimum in our experiments. The following optimization function is to calculate mTSP:

$$\{k*, x*_{i,j}\} := argmin_{k, x_{i,j}} \sum_{i,j} x_{i,j} w_{i,j} + \xi k, \tag{3}$$

where k is the minimum number of paths, i.e., 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, $\xi = 1/10^{th}$ of the maximum edge cost and $x_{i,j}$ is a binary variable. $x_{x,j} = 1$ means that the node i to node j is included in one of the paths, and $x_{i,j} = 0$ otherwise. $w_{i,j}$ is the cost of node i to node j. In addition, the following constraints are used to associate with mTSP to solve Equation (3).

***Constraints* of mTSP.** We use the same constraints in Liu et al. [13] to solve Equation (3). To ensure that each node $V_i (i > 0)$ is visited exactly once, one of the incoming and outgoing edges of a node needs to be selected:

$$\forall i > 0 : \sum_{j, i \neq i} x_{j,i} = 1, \forall i > 0 : \sum_{j, j \neq i} x_{i,j} = 1. \tag{4}$$

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

$$\sum_{j, j>0} x_{j,0} = k, \sum_{j, j>0} x_{0,j} = k. \tag{5}$$

In order to avoid paths composed of disconnected cycles, we follow Liu et al. [13] and set up below some sub-tour elimination constraints (i.e., Equations (6) and (8)) as proposed by Kara and Bektas [10]):

$$u_i + (L-2)x_{0,i} - x_{i,0} \leq L - 1, \forall i > 0, \tag{6}$$

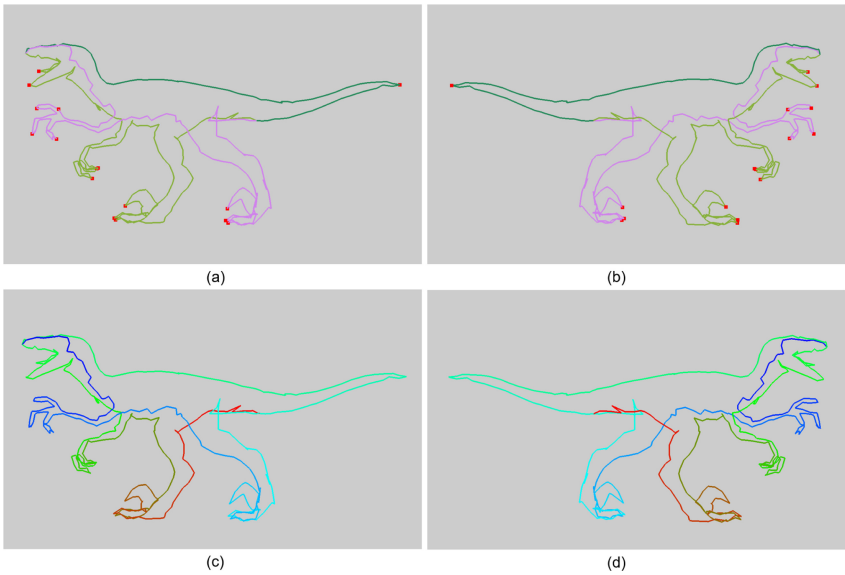


Fig. 14. The result of wire composition. (a) The three edge segments (in different colors) extracted in the previous step. (b) Another view of the edge segments. (c) After wire composition, only a single continuously rough wire is obtained. We use the red-green-blue gradient color to represent the detour of the wire. (d) Another viewpoint of the rough wire.

$$u_i + x_{0,i} \geq 2, \forall i > 0, \tag{7}$$

$$u_i - u_j + Lx_{i,j} + (L-2)x_{j,i} \leq L-1, \forall i,j > 0, i \neq j, \tag{8}$$

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 (Equations (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.

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

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:

$$x_{i,j} + x_{j,i} = 0, \text{ for disconnected } P_i, P_j. \tag{10}$$

We use the CPLEX Optimizer [3] to calculate the objective function (3), a binary integer optimization problem, and use the above constraints to solve the problem. Finally, we get k continuous lines $W = \{W_1, \ldots, W_k\}$; 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. Figure 14 is the result of wire composition.

6 WIRE ART SMOOTHING AND CREATION

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

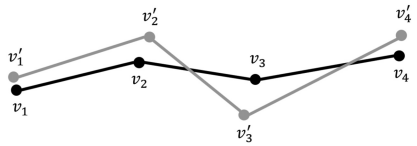


Fig. 15. The vertices v_i' of the rough wire (gray) and the ones v_i of the smoothed wire (black).

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 Figure 15, 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:

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

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 the 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 $\alpha = 1.0$. In addition, in order to preserve the features on the wire, we have the following constraints:

$$v_i = v_i' \text{ for each critical point } v_i. \tag{12}$$

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 Equation (11), constantly changing the position of the wire vertices until the difference ratio between the function value of the current iteration and that 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 the CPLEX Optimizer to solve this optimization problem. Figure 16 is an example of a smoothing wire. In Figure 17, we show the comparison of the results with different α values and experimentally find that $\alpha = 1.0$ can have smooth wires without creating overlaps or intersections in our examples.

In order to make the results look like real wire art, as shown in Figure 18, 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. Figure 19(a) shows an example. Our purpose is to use the torus to imitate the twisting effect by the real wire art. See an example in Figure 19(b). We follow the artists' 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 directly cross each other like the artist's design (see Figure 19(c)) without any twisting connection.

7 RESULTS AND DISCUSSION

Implementation Environment. In the extraction of edge segments, we implement the algorithm in C++. In the wire composition and smoothing part, we use the 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

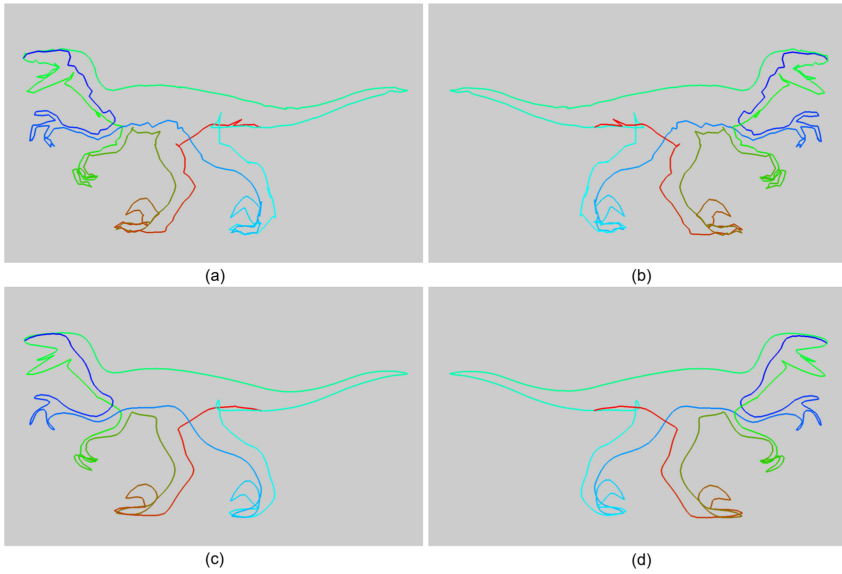


Fig. 16. An example of a smoothing wire. (a) Rough wire from previous step. Gradient color indicates the order of the detour. (b) Another view of the rough wire. (c) Smoothed wire after one iteration. (d) Another view of the smoothed wire.

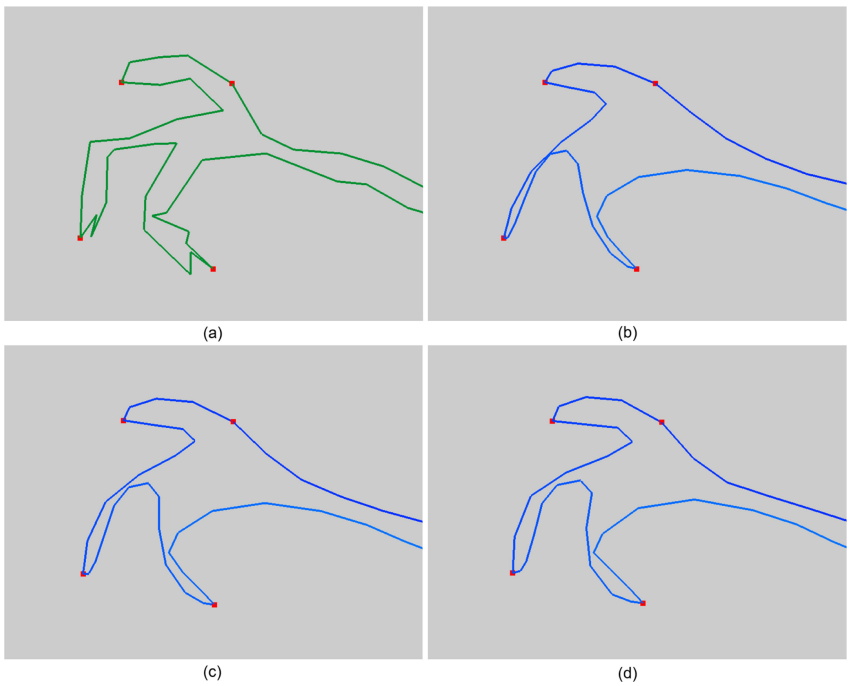


Fig. 17. The comparison of the results with different α values. (a) Rough wire. (b) Smoothed wire with $\alpha = 0.5$. (c) Smoothed wire with $\alpha = 1.0$. (d) Smoothed wire with $\alpha = 2.0$.

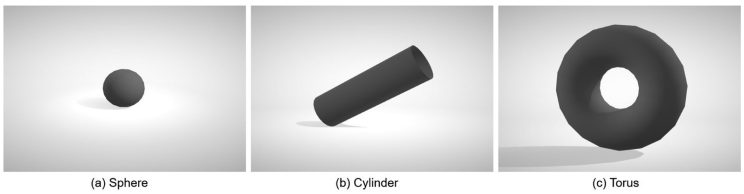


Fig. 18. The three kinds of surfaces used to construct the wire.

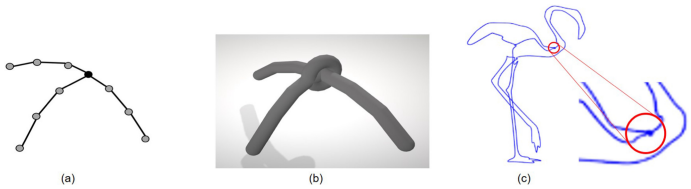


Fig. 19. Construction of the wire model; the final output of our pipeline (c).

section. All parameters are set the same for all our experiments. See the supplementary video for our results at http://graphics.csie.ncku.edu.tw/3D_Wire.

Automatically Generated Results. In this section, we present 10 auto-generated results. We compare two results with the results with user-specified edges to prove that our method allows the user to produce a variety of results. Tables 1 and 2 show statistics for extracting edge segments. The more triangles of the model, the more iterations of pair combinations and the more execution time are needed to check the constraints. Tables 3 and 4 show statistics for wire composition and how many wire lines are generated for each model. As shown in the table, the number of connected wires is extremely concise. The higher the 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 example, in 11-Fertility and 12-Elephant. 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 polygons are processed in our system. The timing cost of the wire smoothing (Tables 5 and 6) for our generated results is very fast, i.e., less than 0.104 second for all examples in our experimental results.

In this article, our goal is to achieve artist-style wire art with a single wire if possible. If not possible, we try to create it with the least number of wires as possible. As mentioned in Figure 1, similar to this artist work, our method cannot guarantee good perception at all views of created results. For example, for the results of model 1-Raptor in Figure 20(d), the 1-Raptor result cannot be easily viewed/recognized because it does not contain details in this view; i.e., details are not preserved from our extraction of edge segments. Therefore, similar to this artist work, our method creates view-dependent results; i.e., better viewing results are not at all viewing directions. Figure 20(c) also shows 1-Raptor's rendering result. This viewing also does not see its details well. In fact, our Figure 20(d) also looks similar to Figure 20(c). See our supplementary results, where we show more different views of created wire results. As a result, we can see most of the views are still model-recognized and only in some specific views is it not easy to recognize them. On the other hand, if we want our wire art visible at all views, i.e., view independent, we might do simplification for mesh view-independent and global silhouette preservation and so forth. But, this way may not create artist-style wire results, and it also cannot always create a single wire for an

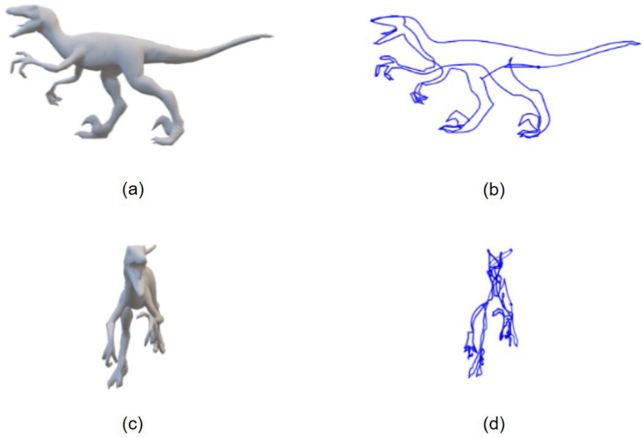


Fig. 20. Comparison between original model and our wire results. Distance error $D(a, b) = 0.000236924$ and $D(c, d) = 0.00131612$. Note we will define distance error in Equation (13) later.

Table 1. Statistics of Extraction of the Edge Segments for the Automatically Generated Results

Model	#Triangles	#Edge Segments	Duration (sec)
1-Raptor	2,500	3	1.666
2-Shark	2,500	8	1.374
3-Butterfly	2,500	3	1.360
4-Rabbit	4,000	6	3.197
5-Deer Head	10,000	3	19.707
6-Scorpion	5,000	3	5.196
7-Stork	10,000	2	19.79
8-Chair	10,000	17	19.94
9-Fertility	3,000	29	2.393
10-Elephant	3,000	23	2.047

Table 2. Statistics of Extraction of the Edge Segments for the Results with User-specified Edges

Model	#Triangles	#Edge Segments	Duration (sec)
11-Fertility	3,000	38	2.156
12-Elephant	3,000	44	1.979

object. Furthermore, it is impossible to easily recognize all views (see Figures 20(c) and 20(d)) even though we see original 3D models.

In the part of the wire composition, we also implement the method of Liu et al. [13] and compare the generated results by our method. The method of Liu et al. [13] 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. Figure 21 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

Table 3. Statistics of the Wire Composition for the Automatically Generated Results

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 4. Statistics of the Wire Composition for the Results with User-specified Edges

Model	#Triangles	#Edge Segments	Duration (sec)
11-Fertility	3,000	14	0.598
12-Elephant	3,000	16	0.555

Table 5. Statistics of the Smoothing Wires for the Automatically Generated Results

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 6. Statistics of the Smooth Wires for the Results with User-specified Edges

Model	Duration (sec)
11-Fertility	0.104
12-Elephant	0.072

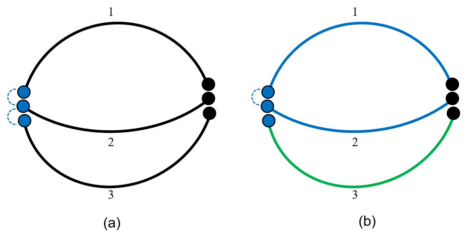


Fig. 21. (a) The three edge segments that were originally to be connected. (b) The result of disconnecting edge segment 2 and edge segment 3.

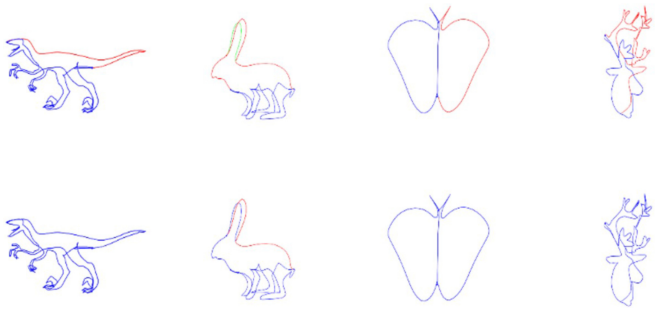


Fig. 22. Shows some comparisons with Liu et al. [13]. For each pair comparison, top side is created by [13] and bottom side is created by ours. In each example, different wires are drawn with different colors.

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 two continuous wires. Figure 22 shows a comparison with Liu et al. [13]. Our method can potentially generate a smaller number of wires than Liu's method. In this comparison, the results are similar, but we require a smaller number of wires; this indicates that our results are more matched with artists' desired goal, i.e., to achieve artist-style wire art with a single wire if possible and, if not possible, to try to create it with the least number of wires as possible. For more example comparisons, see our supplementary file.

To quantitatively evaluate our method with 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[2] 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 Figures 23 to 31. 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. Finally, about

[2]https://www.learnopencv.com/shape-matching-using-hu-moments-c-python.

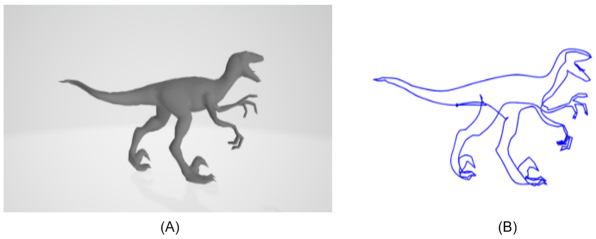


Fig. 23. Result of model 1-Raptor. $D(A, B) = 0.00304063$.

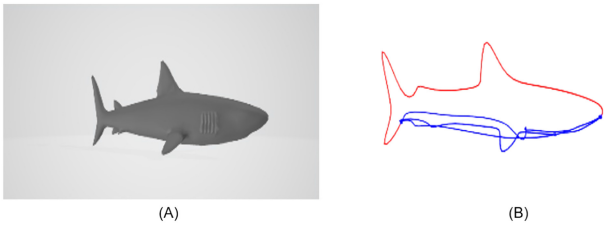


Fig. 24. Result of model 2-Shark. $D(A, B) = 0.00587722$.

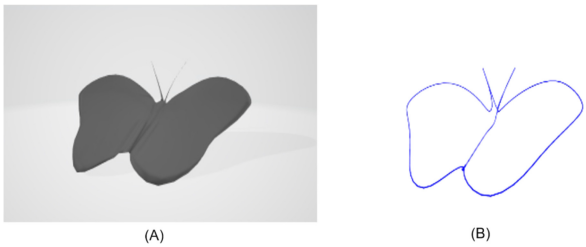


Fig. 25. Result of model 3-Butterfly. $D(A, B) = 0.000857082$.

Fig. 26. Result of model 4-Rabbit. $D(A, B) = 0.00364158$.

the perceptually consistent measure, the use of the Hu moments may be inadequate to provide a perceptually consistent measure. To give a better perceptual measurement, we include an extra user study in our supplementary file to evaluate user feelings about our method and created results.

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. 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

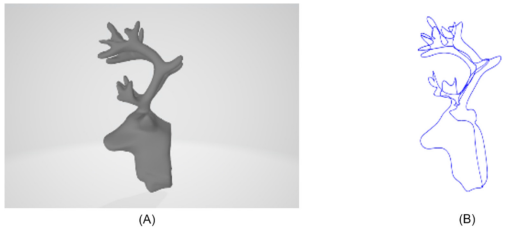


Fig. 27. Result of model 5-Deer Head. $D(A, B) = 0.000249258$.

Fig. 28. Result of model 6-Scorpion. $D(A, B) = 0.000614232$.

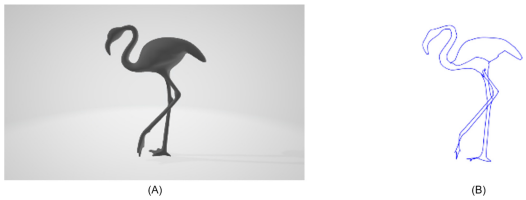


Fig. 29. Result of model 7-Stork. $D(A, B) = 0.00169231$.

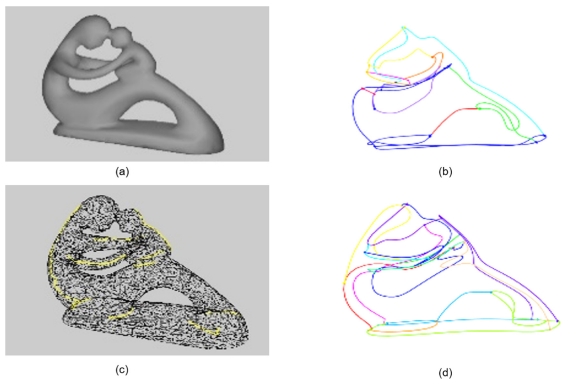


Fig. 30. (a) 9-Fertility and (b) our automatically generated result with $D(A, B) = 0.00239587$. (c) 11-Fertility with user-specified edges and (d) its generated result with $D(A, B) = 0.00197667$.

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 30 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 30(b). Similarly, the automatic

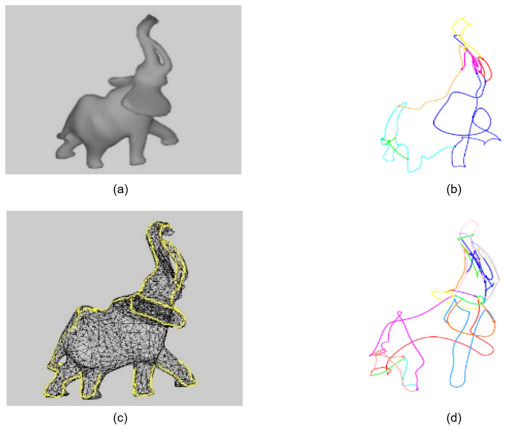


Fig. 31. (a) 10-Elephant and (b) our automatically generated result with $D(A, B) = 0.00365977$. (c) 12-Elephant with user-specified edges and (d) its generated results with $D(A, B) = 0.00188513$.

method may not create user-like results. To improve the result, we must add the user-specified edges (see Figure 30(c)) so that these edges are not removed during the extraction of the edge segments, as shown in Figure 30(d). Another example without/with requiring user-specified edges is shown in Figure 31.

8 CONCLUSION

In this article, 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 that can abstract 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 addition, our segmentation method may not satisfy the users' desire. Therefore, we also allow the users to interactively select some lines to yield their designed results. Currently, our computation timing is limited by mesh segmentation cost. In the future, we may find a better idea and fast way to automatically segment models. Finally, we also plan to design/generate multiple-view wire art from two or three models to generate a virtual wire art that can see the contours of different models from different perspectives by automatically finding out the best viewing contours.

ACKNOWLEDGMENTS

The authors would like to thank the reviewers for the many constructive comments that help improve the paper.

REFERENCES

[1] Mario Botsch and Leif Kobbelt. 2004. A remeshing approach to multiresolution modeling. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*. 185–192.

[2] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. 2004. Variational shape approximation. In *ACM SIGGRAPH 2004 Papers*. 905–914.

[3] IBM ILOG Cplex. 2009. V12. 1: User's manual for CPLEX. *International Business Machines Corporation* 46, 53 (2009), 157.

[4] Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. 2003. Suggestive contours for conveying shape. In *ACM SIGGRAPH 2003 Papers*. 848–855.

[5] Emil. 2017. *Fitting a Plane to Noisy Points in 3D*. https://www.ilikebigbits.com/2017_09_25_plane_from_points_2.html.

[6] Ran Gal, Olga Sorkine, Niloy J. Mitra, and Daniel Cohen-Or. 2009. iWIRES: An analyze-and-edit approach to shape manipulation. In *ACM SIGGRAPH 2009 Papers*. 1–10.
[7] Michael Garland and Paul S. Heckbert. 1997. Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. 209–216.
[8] Kai-Wen Hsiao, Jia-Bin Huang, and Hung-Kuo Chu. 2018. Multi-view wire art. *ACM Transactions on Graphics (TOG)* 37, 6 (2018), 242–1.
[9] Emmanuel Iarussi, Wilmot Li, and Adrien Bousseau. 2015. WrapIt: Computer-assisted crafting of wire wrapped jewelry. *ACM Transactions on Graphics (TOG)* 34, 6 (2015), 1–8.
[10] Imdat Kara and Tolga Bektas. 2006. Integer linear programming formulations of multiple salesman problems and its variations. *European Journal of Operational Research* 174, 3 (2006), 1449–1458.
[11] Shiwei Li, Yao Yao, Tian Fang, and Long Quan. 2018. Reconstructing thin structures of manifold surfaces by integrating spatial curves. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2887–2896.
[12] Wallace Lira, Chi-Wing Fu, and Hao Zhang. 2018. Fabricable Eulerian wires for 3D shape abstraction. *ACM Transactions on Graphics (TOG)* 37, 6 (2018), 1–13.
[13] Lingjie Liu, Duygu Ceylan, Cheng Lin, Wenping Wang, and Niloy J. Mitra. 2017. Image-based reconstruction of wire art. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1–11.
[14] Ravish Mehra, Qingnan Zhou, Jeremy Long, Alla Sheffer, Amy Gooch, and Niloy J. Mitra. 2009. Abstraction of man-made shapes. In *ACM SIGGRAPH Asia 2009 Papers*. 1–10.
[15] Eder Miguel, Mathias Lepoutre, and Bernd Bickel. 2016. Computational design of stable planar-rod structures. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 1–11.
[16] Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. 2007. Fibermesh: Designing freeform surfaces with 3d curves. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)* 26, 3 (2007), article no. 41.
[17] Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot, and David Salesin. 2008. Diffusion curves: A vector representation for smooth-shaded images. *ACM Transactions on Graphics (TOG)* 27, 3 (2008), 1–8.
[18] Rui S. V. Rodrigues, José F. M. Morgado, and Abel J. P. Gomes. 2018. Part-based mesh segmentation: A survey. In *Computer Graphics Forum*, Vol. 37. Wiley Online Library, 235–274.
[19] Ariel Shamir. 2008. A survey on mesh segmentation techniques. In *Computer Graphics Forum*, Vol. 27. Wiley Online Library, 1539–1556.
[20] R. Suzuki, M. Moriguchi, and K. Imai. 2017. Generation and optimization of multi-view wire art. In *Proceedings of the 25th Pacific Conference on Computer Graphics and Applications (Pacific Graphics'17) Posters*. 16–19.
[21] Fernando J. Wong and Shigeo Takahashi. 2011. A graph-based approach to continuous line illustrations with variable levels of detail. In *Computer Graphics Forum*, Vol. 30. Wiley Online Library, 1931–1939.
[22] Rundong Wu, Huaishu Peng, François Guimbretière, and Steve Marschner. 2016. Printing arbitrary meshes with a 5DOF wireframe printer. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 1–9.
[23] Jonas Zehnder, Stelian Coros, and Bernhard Thomaszewski. 2016. Designing structurally-sound ornamental curve networks. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 1–10.
[24] Yinan Zhou and Zhiyong Huang. 2004. Decomposing polygon meshes by means of critical points. In *10th International Multimedia Modelling Conference, 2004, Proceedings*. IEEE, 187–195.

Received August 2020; revised June 2021; accepted July 2021