Mesh Cutting with Semi-automatic Rotation

Warunika Ranaweera
wranawee@sfu.ca

Zeinab Sadeghipour
zsadeghi@sfu.ca

Abstract

We present a mesh cutting tool to manually draw the boundary of segmentation while assisting the user with semi-automatic rotation. Based on the current endpoint of the sketch, we predict the potential subsequent position of the boundary point considering the minima rule. After obtaining the complete cutting contour, we segment the input mesh into two separate parts with a region growing algorithm. The experiments we conducted so far convey that our approach can facilitate manual mesh cutting whilst maintaining efficiency of the segmentation. Further improvements can be applied to our approach to produce more desirable and robust results.

1 Introduction

Mesh segmentation (or partitioning) is the task of decomposing a 3D mesh into a set of components which are visually simple and meaningful [1]. This problem stimulated increasing interest in the field of computer graphics by reason of its application in various geometry processing algorithms such as mesh parameterization, texture mapping, shape matching, morphing, mesh editing, skeleton extraction, collision detection and mesh compression [2, 3].

The precise definition for *meaningful* can vary widely considering both the special application problem and the complicated human perception [2]. Due to this deviation, establishing a fully automatic algorithm with desirable results has become a challenge. Thus, in some research works carried out on mesh segmentation, methods involving user interaction have been developed to bring the outputs closer to the human expectation.

In this project, we intend to implement a semi-automatic approach for mesh cutting which is a tool utilizing mesh segmentation as the principal component. Our method is mainly comprised of user drawing a closed cutting outline and a semi-automatic rotation to assist specifying the cutting contour. The main purpose of this hybrid algorithm is to obtain a more meaningful segmentation while reducing the effort of predicting human perception in a completely automatic procedure. The input to this problem is a general 3D mesh and after applying mesh cutting on this mesh, we expect a segmented set of parts as the output.

1.1 Related work

The most challenging task of designing a mesh cutting tool is the detection of a cut boundary which produces a semantically correct segmentation. Hence, the majority of the work in the literature is focussed on modeling human perception to increase the desirability of the result. Following is a breakdown of various such approaches to mesh cutting that attempt to produce meaningful cut boundaries, while maintaining the efficiency of the process.

Automatic partitioning: Methods that take a completely automatic approach to mesh decomposition can be considered under this category. Here, the boundary between regions is implicitly defined (for example, based on the minimum curvature points), instead of allowing the user to explicitly specify it. Subsequently, the regions are incrementally grown [4] or merged [5] to produce a seg-

mentation. The implicit definition of a mesh boundary, however, is a drawback in identifying user's preference of a perfect cut.

Semi-automatic segmentation: To avoid the pitfalls in implicit cut boundary detection, some semi-automatic approaches provide the user with a choice of potential boundaries [6]. Sketched-based interfaces have also emerged as a method to guide automatic segmentation algorithms with increased user interaction [1, 7]. Users can draw regions on the mesh which are most preferable to be placed inside the boundary. Although they are taking the user's choice into account, there is a restriction to the boundary produced by the region growing algorithms; hence, may not be mapped to the human perception of a semantically meaningful cut boundary according to the context.

A recent work by Chen, et al. [8] provides a rigorous evaluation of how current automatic mesh segmentation methods do not fit all types of objects. The authors also highlight the fact that the human perception is superior to such automatic segmentation approaches. Therefore, a manual mesh cutting tool which attempts to achieve the same level of efficiency as automatic or semi-automatic tools, is required.

1.2 Our approach

As previously mentioned, in our proposed approach, the user draws the cutting outline manually while the mesh is rotated automatically based on the approximation of the area the user would proceed to cut. Our tool is created according to the framework presented in Ji et al. [1], where the boundary to cut is acquired with the help of the *minima rule* and *part salience theory*. These rules state that humans take into notice the parts of a 3D shape along the concave discontinuity of the tangent plane of the mesh [9]. We also employ these rules to estimate the best angle for rotating the mesh in a way that does not obstruct the view of the user, in order to facilitate sketching the cutting contour. Afterwards, we separate the source mesh into two disjoint parts along a closed contour lying on the mesh [6].

2 Our mesh cutting platform

Given an input mesh, we apply the steps illustrated in Figure 1 to produce two parts of the mesh separated by the user-defined cut boundary.

As the first step, we capture user strokes and map them onto the mesh by traversing all the triangles which may surround (or overlap) the points in the stroke. Section 2.1 introduces the data structure we have used to store the mesh for ease of traversal. Then we attempt to identify potential points in subsequent user strokes based on the relationship between the minima rule and the human perception, as discussed in section 2.2. Potential points are then used to automatically rotate the mesh (section 2.3) to guide the user through the cutting step. After the boundary is completed, we set the seed points, and start growing the regions to perform the mesh segmentation. This step is discussed more in detail in section 2.4.

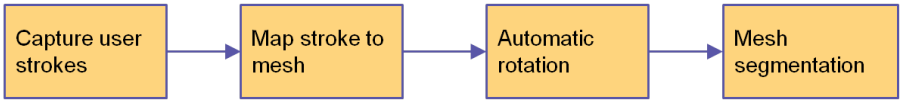

Figure 1: Algorithmic pipeline for our mesh cutting platform.

2.1 Data structure

In any mesh-related algorithm, efficient processing of adjacency queries is essential. Hence, with the aim to be able to traverse the mesh fast, we store the information about the geometry and the topology of the mesh, in a *data structure*.

This data structure keeps track of connectivity and local orientation around vertices of a triangular 3D mesh by saving arrays of vertices, faces and edges in the following ways:

- **CVertex:** For each vertex of the mesh, we incorporate the original (x, y, z) position, the normal vector, adjacent vertices, faces sharing that vertex and the curvature. In addition, there is a cluster number associated with every vertex which is identified during mesh segmentation.
- **CTriangle:** Although we customized our implementation to triangular meshes, it can easily be expanded for other types too. Connected with every face, the normal vector, the constructing vertices and the edges are included in the data structure.
- **CEdge:** Finally, we assign each edge the start and end vertices and the faces sharing that particular edge.

Therefore, all the components of a mesh are linked to each other leading to smaller complexity for operations required in the following steps. For instance, we can easily traverse all vertices in the mesh starting from one vertex, since we know the adjacent vertices for each one.

2.2 Curvature

In human cognitive vision theory, the minima rule has been proposed as an explanation for shape recognition of the human perceptual system [10]. The minima rule states that human perception tends to divide a surface into parts along *negative minima* of the principal curvature.

Principal curvature: For a curve that defines a surface, curvature relates to its second derivative, which represents how rapidly the tangent decreases or increases. Hence, curvature conveys how bent the surface is at a given point. Near the points where the curve has a rapid change in the slope, the magnitude of the principal curvature (either minimum or maximum) is high. In concave parts of the mesh, the principal curvature converges to a negative minima.

2.3 Automatic rotation

For each vertex in the mesh, we store the principal curvature in our data structure. Per user stroke, we traverse along the neighboring vertices to detect the one which has the highest potential to be included in the cut (i.e. the vertex having the minimum principal curvature value). This is based on the common observation that humans tend to cut along the negative minima of the curvature. To avoid a noisy rotation, we compare the direction of the vertex normals, as shown in Figure 2 (b); then we rotate the mesh if the normal directions have a drastic difference, which conveys a sudden bend in the mesh. The steps for detecting the rotation angle is as follows:

1. Calculate the minimum curvature for all the vertices
2. At each stroke p_0, calculate the nearest neighbor p_1 with the minimum curvature value c_{p1}
3. Find the angle $\theta_{p0,p1}$ using the inverse cosine of the vector dot product (Figure 2 (a))
4. Identify the axis of rotation using the change in x and y coordinates:
   if $|x_{p_0} - x_{p_1}| > |y_{p_0} - y_{p_1}|$ - rotate around the y axis
   else - rotate around the x axis
5. Calculate the difference between normal vector directions D (Figure 2 (b))
   if $D > theshold\_angle$, rotate the mesh

2.4 Mesh segmentation

Among all possible methods for segmentation, we utilize the local-greedy one referred to as the *region growing* algorithm. We begin with a set of seed vertices from the original mesh and grow corresponding sub-meshes incrementally [3].

In every region growing algorithm, two issues should be resolved; first, the criterion to add a vertex to an existing cluster, and secondly, the mechanism to select seeds. In our situation, we entail two clusters related to two disjoint parts of the mesh after cutting.

To choose seeds and ascertain their classes, we perform the following steps. Initially, associated with each point on the cut boundary specified by the user, we try to find the nearest vertex on

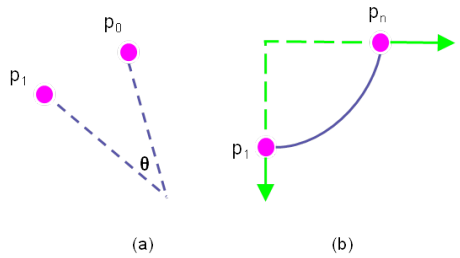

Figure 2: Assume p_0 is the current point on the sketch, p_1 is the neighboring point with the minimum curvature value and p_n is the last point with a rapid change in the normal direction. (a) Finding the angle of rotation using two points on the mesh; (b) Rotating only when a rapid change in the directions of normals occur.

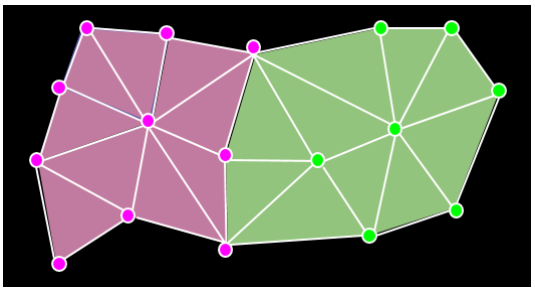

Figure 3: The result of region growing algorithm

the mesh. Next, iterating over the set of aforementioned vertices, for every pair of consecutive elements, we estimate the explicit representation of a *local plane* perpendicular to the mesh surface. Subsequently, we classify the neighbors of the vertices linked to the cut contour with regards to the sign of $f(v) = ax_0 + by_0 + c_0 + d$, where (a, b, c, d) are the coefficients defining the local plane and (x_0, y_0, z_0) are the coordinates of the adjacent vertex v; the vertex is set to *class 0* if the resulting $f(v)$ is positive, and is clustered as *class 1* otherwise. After we repeat the same procedure for all the vertices related to the cut outline, we obtain our seeds which can initiate growing regions.

As mentioned earlier, the other crucial aspect of the algorithm is the rule to decide about the cluster of other vertices. With the intention to simplify the method, the only criterion we consider is *adjacency*. Starting from every seed, we add its adjacent vertices to the same cluster; then we apply the same rule on those neighbors and repeat this process until all the vertices are clustered. Eventually, faces are colored based on the class of their vertices to visualize the segmentation (Figure 3).

3 Results and Discussion

In this project, we carried out two main sets of experiments on some models from Ji et al [1]. At first, we tried our segmentation approach with manual rotation performed by the user. The results for this part are shown in Figure 4 for *bunny, hand* and *horse* models. As it is illustrated, the algorithm works quite well in this case. In another round of experiments, we investigated the semi-automatic rotation which we have implemented and let the user rotate the mesh where he needed. Figure 5 includes the result for one instance of this round which still has some artifacts in separated segments. Results of the realtime rotation of our mesh cutting tool are illustrated in the demo (`https://youtu.be/EsZVKgbnZs8`).

Figure 4: The result of applying our segmentation on manual rotation

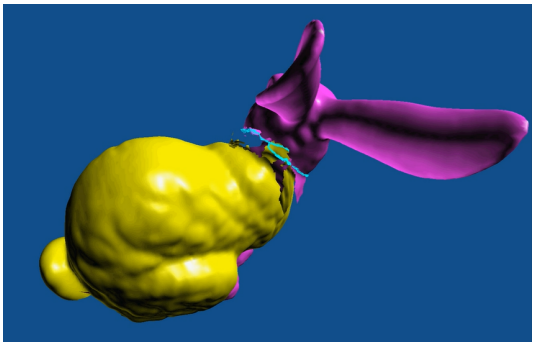

Figure 5: The result of our algorithm with semi-automatic rotation

4 Limitations and future work

The proposed method can be enhanced in several ways. For instance, in order to solve the problem of parts obscuring the user's field of view, we can eliminate those parts from the mesh once we are near those points. Another opinion could be to attempt to predict the direction of the sketch the user is drawing further away than just the next vertex. The second limitation that might be resolved in future works is the jaggy segments that occur after mapping the points on the cutting boundary to the vertices of the mesh. As suggested in previous works, the aforementioned segments could be refined with the assistance of *3D snakes*. The other artifact observed in some experiments is imperfect segmentation where we can improve the results by estimating the local planes with a higher accuracy. Furthermore, performing a user study to investigate the general human perception for cutting a mesh, may help implementing a fully-automatic rotation approach to users' expectations.

References

[1] Zhongping Ji, Ligang Liu, Zhonggui Chen, and Guojin Wang. Easy mesh cutting. In *Computer Graphics Forum*, volume 25, pages 283–291. Wiley Online Library, 2006.

[2] Marco Attene, Sagi Katz, Michela Mortara, Giuseppe Patané, Michela Spagnuolo, and Ayellet Tal. Mesh segmentation-a comparative study. In *Shape Modeling and Applications, 2006. SMI 2006. IEEE International Conference on*, pages 7–7. IEEE, 2006.

[3] Ariel Shamir. A survey on mesh segmentation techniques. In *Computer graphics forum*, volume 27, pages 1539–1556. Wiley Online Library, 2008.

[4] Alan P Mangan and Ross T Whitaker. Partitioning 3d surface meshes using watershed segmentation. *Visualization and Computer Graphics, IEEE Transactions on*, 5(4):308–321, 1999.

[5] Yu-Kun Lai, Shi-Min Hu, Ralph R Martin, and Paul L Rosin. Fast mesh segmentation using random walks. In *Proceedings of the 2008 ACM symposium on Solid and physical modeling*, pages 183–191. ACM, 2008.

[6] Yunjin Lee, Seungyong Lee, Ariel Shamir, Daniel Cohen-Or, and H-P Seidel. Intelligent mesh scissoring using 3d snakes. In *Computer Graphics and Applications, 2004. PG 2004. Proceedings. 12th Pacific Conference on*, pages 279–287. IEEE, 2004.

[7] Lubin Fan, Min Meng, and Ligang Liu. Sketch-based mesh cutting: A comparative study. *Graphical Models*, 74(6):292–301, 2012.

[8] Xiaobai Chen, Aleksey Golovinskiy, and Thomas Funkhouser. A benchmark for 3d mesh segmentation. In *ACM Transactions on Graphics (TOG)*, volume 28, page 73. ACM, 2009.

[9] Donald D Hoffman and Manish Singh. Salience of visual parts. *Cognition*, 63(1):29–78, 1997.

[10] Donald D Hoffman and Manish Singh. Salience of visual parts. *Cognition*, 63(1):29–78, 1997.