Spectral Based Mesh Segmentation

A Simple Algorithm for Mesh Segmentation

Bertay Eren†
Computer Science Departmen
Hacettepe University
Ankara Turkey
bertayeren@gmail.com

ABSTRACT

Mesh segmentation is an important task for various operations including CAD, reverse engineering, animation technologies. Mesh segmentation by its nature is often dictated by the topology of the mesh but in this paper we will try to discover usage of spectral graph theory in mesh segmentation applications. First, we will extract the data from Mesh by constructing a Laplacian Matrix to code information on connectivity of the mesh, then we will inspect its first non-trivial eigenvector (i.e. Fiedler Vector) to obtain necessary information for mesh segmentation. Here, we will present the results and will observe that we still need some human interaction to prevent over-segmentation, but results are encouraging for future work.

KEYWORDS

Fiedler Vector, Laplacian Matrix, Mesh Segmentation, Spectral Graph Theory

1 Introduction

Mesh segmentation is the process of decomposing a mesh into simpler and meaningful parts, and with the developing 3D Scanning Technology we can easily transfer almost every object to digital world hence extracting information from 3D objects can be crucial for the sake of easier integration of certain applications such as Simulation, Medical Imaging, Animation Rigging, Computer Aided Design (CAD), VR and AR apps (to create more interactable objects).

The problem itself faces some challenges to overcome, The term “meaningful parts” is ambiguous and may seem as a requirement for a Neural Network, also the automatization of identifying the sub-meshes of a mesh might be computationally expensive since these objects can have thousands of vertices. Also, we know that the 3D scanning has noise problems etc.

Despite these challenges, one must aware that in 3D graphics, meshes are essentially graphs with (mostly) undirected and unweighted edges. Hence, use of spectral graph theory can be beneficial. In this article, we will focus on the Fiedler Vector (the second smallest eigenvector of the Laplacian Matrix of a mesh object) because it was already shown to have powerful properties in terms of carrying useful information on mesh. Simplest terms, we will sort the fiedler vector’s entries for each vertex and then we will compute the second order differences to capture the second derivative (i.e. curvature information), the local maximas of the second order derivative will be our segmentation points mostly.

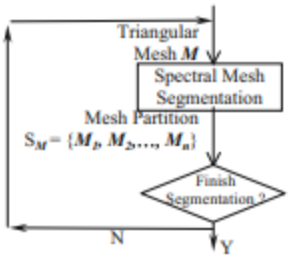


Figure 1. Flow of our implementation

Here above Figure 1 shows the flow of our algorithm.

1.1 Our Contribution

Our contribution to existing implementation to this paper was adding the ability of adapting the algorithm to calculate exact number of desired segmentations. The corresponding algorithm can be seen in Figure 2 below.

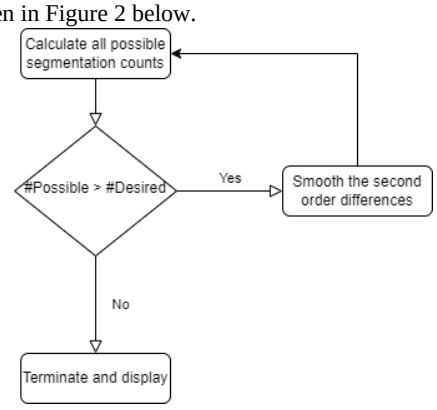


Figure 2. Adaptive Iterative Segmentation

The flowchart in Figure 2 will be explained in detail in following sections.

2 Literature Review

This approach has already been presented by Mejia et. al. at 2016[1].

3 Methodology

We will start our implementation with creation of Laplacian Matrix. For this purpose, we created a simple OpenGL framework for rendering OFF and OBJ files and made it able to paint vertices with any color that we want. And we stored the vertices of the object in a special format, while loading the object from file we are also storing it's neighboring vertices too. In fact, this is all we need to create Laplacian matrix.
The Laplacian Matrix is defined as $\boldsymbol{L = D - A}$ where **D** being Degree Matrix and **A** being adjacency matrix. Therefore, we have a sparse symmetric real-matrix in hand.

By calculating the Eigenvectors of this matrix, we can obtain the first non-trivial Eigenvector (i.e. the second smallest Eigenvalue's Eigenvector) also known as Fiedler Vector.
The main idea behind this approach is we have a N-dimensional space and Eigenvectors are storing the constant directions in that space where Eigenvalues are telling us the amount of scaling these Eigenvectors will experience. Hence, the first smallest Eigenvalue is 0, but the second Eigenvalue means that we have smallest change for that Eigenvector! Which means, it captures the low-frequency details of the object.

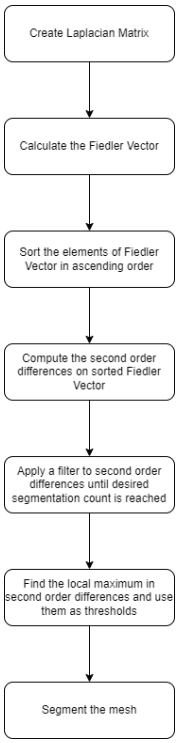


Figure 3. Improved Algorithm

One can see the all flowchart in Figure 3 which describes the improved flow of algorithm.

1. Create Laplacian Matrix: In this step, we will use Eigen Library of C++ because it has a representation for Sparse Matrices and it is crucial for us to have memory-efficient representation of a Laplacian Matrix since the most entries of the Laplacian Matrix are zero, it is trivial to make them memory costly.

2. Calculate the Fiedler Vector: In this step, we will use Spectra Library, which is a framework built on top of Eigen library because Spectra can calculate the desired number of Eigenvectors instead of calculating all N eigenvectors of an NxN matrix. Hence, it is way more efficient for as because all we need is the 2 smallest Eigenvectors.

3. Sorting the Fiedler Vector: This step will be used to calculate second derivative of Fiedler Vector. The sorting is performed to obtain a more derivative suitable function because the each entry of Fiedler Vector corresponds to a Vertex and these are just arbitrary elements, no information stored in them initially. Hence, sorting operation does not causes us to lose any but help us to have them in a more organized way. For this step, I've used STL's Quick Sort implementation (i.e. std::sort)

4. Compute the Second Order Differences: This is the step we are catching the curvature information on our mesh object essentially. The implementation is straightforward and shown in Equation 1 below.

$$|\, sFiedler[i-1] - 2 * sFiedler[i] + sFiedler[i+1] \,|$$

Equation 1. The Second Order Differences

In Equation 1, *sFiedler* indicates the sorted Fiedler Vector and one should observe that we are using the absolute value.

5. Apply a Filter to Second Order Differences: The mesh data can be noisy and these noise may decrease our segmentation performance. Also, this is the step where Figure 2 comes handy. In this step, first we are applying a simple Gaussian Filter with a size being 2.5% of the size of Second Order Differences array. This helps us obtaining a both smoother and valuable data and often this only 2.5% filtered version contains high number of local maximums. Our contribution improves the original paper by letting users to choose segmentation count at start and we keep applyingthe Gaussian Filter until we reach that desired number.

6. Find Local Maximums: We are finding the local maximums in this step.

7. Segment the Mesh: We are using our OpenGL framework to paint and display the segmented mesh.

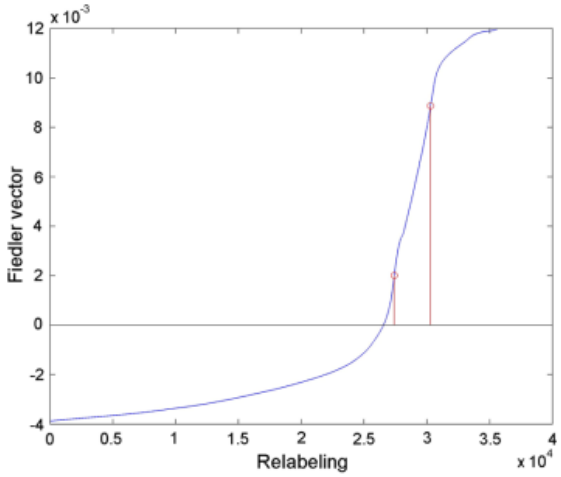


Figure 4. Sorted Fiedler Vector

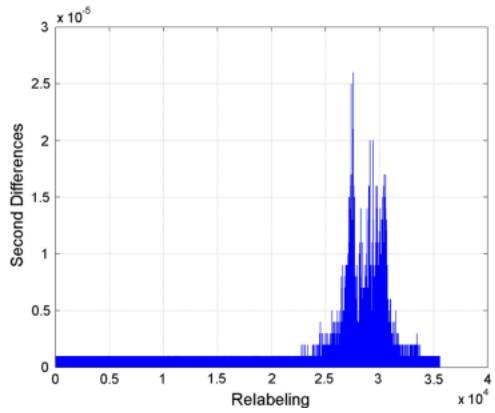


Figure 5. Second Order Differences

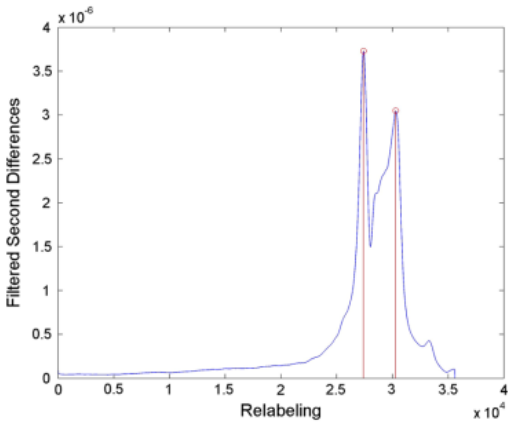


Figure 6. Filtered Second Order Differences

One can see the visual results of Steps 3, 4, 5, and 6 in the Figures 4, 5, 6.

4 Results

We now present the results of our algorithm and some comparisons of it with the original paper.

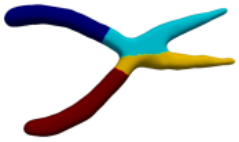


Figure 7. Original Paper's Segmentation of Pliers Object

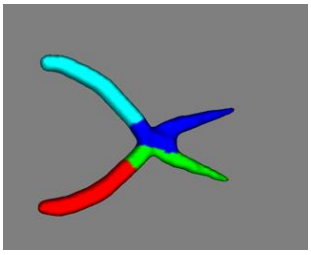


Figure 8. Our Implementation's Result for Pliers Object

The object shown in Figure 7. 4794 vertices and took our algorithm to compute it in 279899 ms. (in Debug Mode)

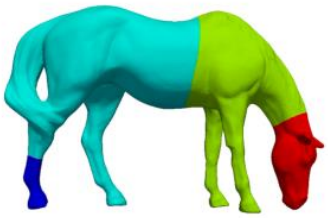


Figure 9. Original Paper's Segmentation of Horse

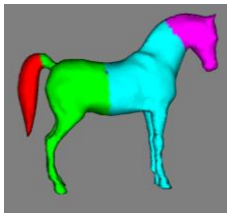


Figure 10. Our Implementations Segmentation of Horse with Desired Segmentation Count as 4

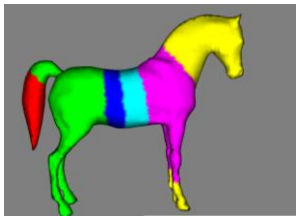
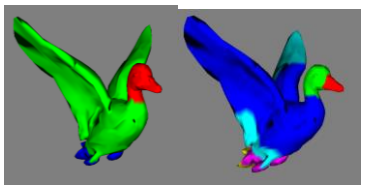


Figure 11. Our Implementations Segmentation of Horse with Desired Segmentation Count as 6

One can observe that our implementation also detects the tail whereas the original paper does not. This can be explained by considering the topology of the used mesh in original paper, the tail is connected to leg of horse and they are not separated during segmentation in the original paper.

Now we will present some other results without comparison because the intersection of our dataset and theirs end here.

Now let us show the difference between size of Gaussian filters used in raw version of second order differences, we should note that this difference was the main idea behind our contribution explained in Introduction 1.1.



(a) **(b)**

Figure 12. Segmentation Applied with Different Gaussian Filters

In Figure 12.a, we applied a Gaussian Filter with sigma is 5% of the size of second order differences, in Figure 12.b we applied a Gaussian Filter with sigma is 2.5% of the size of second order differences. Observe that smoothing the second order differences effects segmentation count in advance.

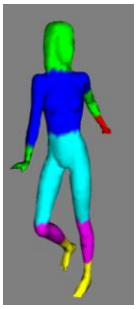


Figure 13. Our Implementation Applied to a Human Object

In Figure 13, we represent our implementations' results on a human object, as seen in the Figure we are able to determine major body parts meaningfully.

All the coding and related objects are reachable via https://github.com/bertaye/694-Segmentation

5 Conclusion

We started by the creation of Laplacian Matrix and then calculated Fiedler Vector from it to find curvature details of our mesh object, then we set some thresholds to segment objects. In conclusion, we observed that Fiedler Vector is a powerful tool for mesh segmentation, and it produces satisfactory results.

6 References

[1] Mejia, D., Ruiz-Salguero, O., & Cadavid, C. A. (2017). Spectral-based mesh segmentation. *International Journal on Interactive Design and Manufacturing (IJIDeM)*, *11*, 503-514