Final Report of Artistic Rendering Line Drawing

Yuefeng Wu Jason Narine

I. Introduction

I.I Project Overview

In its simplest form, an image solely consists of lines acting as contours, static lines that separate the front-facing from the back-facing regions, and suggestive contours, view dependent contours. As the complexity of an image increases, these contours and suggestive contours seem to move to the background while texture and shading usurp its position in the foreground. One would assume then that writing an algorithm void of the aforementioned complexities should be effortless for programmer, however this is not the case.

Years of painstaking research and mathematical formulas led Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella to figure out the perfect algorithm for calculating contours and suggestive contours in their paper titled, "Suggestive Contours For Conveying Shape."

The complexity of the algorithm matched with the simplicity of the result, contours and suggestive contours to convey shape, piqued our group's interest and led us down a similar path to DeCarlo and his associates. Using the "Suggestive Contours For Conveying Shape" paper as a guideline, our project is to replicate the results from the paper as similarly as possible.

I.II Project Team

Yuefeng Wu
Email: wilfredwu2012@gmail.com Yuefeng was the group member in charge of the coding and some of the presentations and reports.
Jason Narine
Email: jnnarine@dons.usfca.edu Jason was the group member in charge of some of the presentations, reports.

I.III Project Goal

The goal of our project is to replicate the result from the "Suggestive Contours For Conveying Shape" paper as closely as possible using the algorithm described in the paper to calculate the contours and suggestive contours.

II. Approach

As previously mentioned in Project Goal (I.III), our group will be utilizing the algorithm described in the "Suggestive Contours For Conveying Shape" paper to calculate the contours and suggestive contours. Our group decided to follow this line drawing algorithm because we believed it to be currently the best implementation of calculating contours and suggestive contours. Modeling our project after theirs will also give us a control with which to compare our results.

II.I Approach Overview

Contours are the set of points on the surface whose normal vector is perpendicular to the viewing direction. As shown in Figure 1a, when projected into the image, its visible portions are called the contour. In Figure 1b, a topographic map of the surface in Figure 1a with the contour generator shown in green. The portion that projects to the contour is drawn solid.

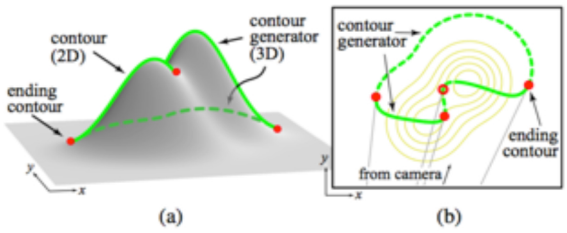


Figure 1

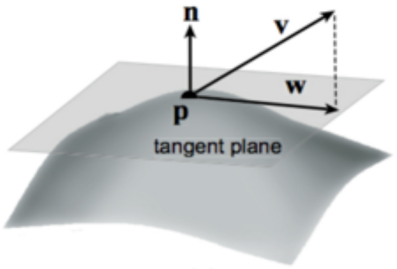


Figure 2

In our project, contours are defined as follows:

- The set of points that lie on this surface and satisfy the equation:
  - $n(p) \cdot v(p) = 0$
  - Where:
  - p is a point on the surface
  - n(p) is the vertex normal at p
  - v is the view vector: $v(p) = c - p$
  - c is the position of the camera

  See vectors in Figure 2

II.II Contours

Given the vertex normal and our knowledge of the camera view change and vector calculation, this approach worked well with our project.

II.III Suggestive Contours

In actuality, there are three different definitions of suggestive contours: zeros of radial curvature, minima of $n \cdot v$ definitions, and nearby viewpoints definition.

Based on these three definitions, there are two algorithms to calculate the suggestive contours: object-space algorithm and image-space algorithm. Our group decided to use the object-space algorithm thus we use the zeros of radial curvature definition to get the suggestive contours.

Zeros of radial curvature is defined as follows:

- The suggestive contours are the set of points on the surface Where:
- Its radial curvature Kr is 0
- The directional derivative of Kr in the direction of w is positive

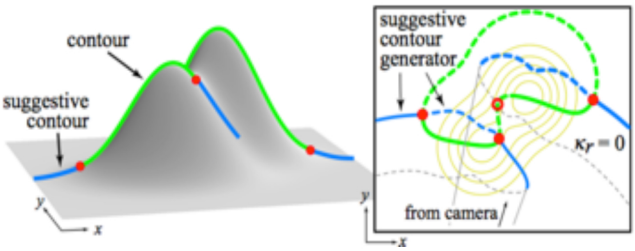


Figure 3

Suggestive contours (shown in blue) extend the actual contours of the surface (shown in green). A topographic view displays how the suggestive contour generators cross contours at the ending contours. The portion of the suggestive contour generator that projects to the suggestive contour is drawn solid, as shown in Figure 3.

In order to calculate the radial curvature, our group attempted to utilize the Euler Formula [do Carmo 1976]:

$$\kappa_r(\mathbf{p}) = \kappa_1(\mathbf{p})\cos^2\phi + \kappa_2(\mathbf{p})\sin^2\phi$$

Calculating K1, K1 and cos proved more difficult than our group anticipated thus we consulted the paper, "Estimating Curvatures And Their Derivatives On Triangle Meshes" by Szymon Rusinkiewicz. In this paper we came across a peculiar formula to calculate the normal curvatures for each vertex:

$$\kappa_{ij} = \frac{2\boldsymbol{n}_i \cdot (\boldsymbol{p}_i - \boldsymbol{p}_j)}{|\boldsymbol{p}_i - \boldsymbol{p}_j|^2}$$

We will explain this formula in the following section.

III. Methodology

III.I Methodology Overview

Our group's methodology consists of three steps: detection, estimation, and threshold. The aforementioned steps embody the critical aspects of the algorithm and are utilized to calculate contours as well as suggestive contours. These steps will be explained in-depth in the subsequent subsections.

III.II Detection

Detection is the first step in our methodology. Before further exploring detection, it is critical to understand the 1-ring neighborhood vertices.

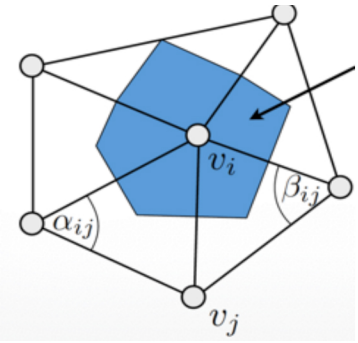


Figure 4

In Figure 4, assuming that Vi is our center vertex, the other 5 vertices in Figure 4 are the 1-ring neighborhood vertices since they are the closest neighborhood vertices in our mesh. Our algorithm is calculating normal curvatures of each vertex based on its 1-ring neighborhood vertices. We create a 2D index array to store every neighborhood vertices for each vertex in our mesh. Since we have already had the face-index array, we keep using it as our first step. After iterating the face-index array, we need save all vertices in each neighborhood face (except the center vertex) into our 2D neighborhood vertices array. It is crucial to be careful because there will be duplicate neighborhood vertices if conditional statements are not used to remove them from our 2D array. After building the neighborhood vertices array successfully, we can calculate the normal curvatures.

Because detection is the first step, our detection method is quite efficient and we have not come across a similar method that acts in the same manner.

III.III Estimation

Graphic scientists invented the methods of calculating contours and suggestive contours and thusly they created the n · v and Euler formulas to estimate the main parameters of said contours.

III.III.I Contours Estimation

For estimating contours, we continue to use the vertex normal array from detection. The view vector is now needed, which we can be obtained using:

- v(i) = c(camera) – p(i)

We need put our camera in a fixed position, but if the user desires to change the camera angle that can be achieved through our user interface. After calculating both the vertex normal vectors and the view vectors, we can obtain the n · v value for each vertex by utilizing the dot product formula for these two vectors. Currently, the method our group used is the only uniform method to define contours in the line drawing field.

III.III.II Suggestive Contours Estimation

While contours are important, the most difficult contours – suggestive contours – are critical to the success of our group's final project. In Suggestive Contours (II.III), we opted to use the Euler Formula.

But obtaining Kr proved to be difficult because we need obtain K1, K2 and cos beforehand. Unfortunately, the "Suggestive Contours For Conveying Shape" paper does not offer a solution to obtaining the aforementioned values. Luckily, through research of our own, "Estimating Curvatures And Their Derivatives On Triangle Meshes" by Szymon Rusinkiewicz explains how to obtain said values through the formulas to get the normal curvatures:

- $\kappa_{ij} = \frac{2n_i \cdot (p_i - p_j)}{|p_i - p_j|^2}$
- W vectors: w = v – n(n · v)

As mentioned in Detection (III.II), the neighborhood vertices array is extremely helpful. Through repetitive iterations (loops), we can obtain every normal curvature for each vertex in our mesh using the normal curvature formula. After obtaining the normal curvature for each vertex, calculate the two kings in our project, K1 and K2.

- K1 and K2 are defined as the principal curvature,Where:
- K1 is the maximum of the normal curvature at a given point on a surface
- K2 is the minimum of the normal curvature at a given point on a surface

Using this definition, after obtaining the normal curvatures, we iterate over every normal curvature and find out the maximum and minimum value, which are K1 and K2. To save the vertex index whose normal curvature is K1, we need to create a K1-index array. The purpose of this step will be more defined in the subsequent step.

To calculate $\cos\phi$

ϕ is the angle between w(p) and the principal curvature direction corresponding to K1.

This is where the significance of the K1-index array is apparent. The K1 direction vector is equal to:

- p(i) – p(j)
- Where:
- i is the center vertex
- j is the K1 index vertex

Finally, we understand:

- cos = w(p) · v(ij) / —w(p)— · —v(ij)—

Thus, we can achieve the Kr easily through the Euler formula:

However, it is noted in other research groups that they calculate the radial curvature using a different method. Other research groups calculate the value of K1 and K2 using:

$$\kappa_1 = H + \sqrt{H^2 - K}$$
$$\kappa_2 = H - \sqrt{H^2 - K}\,.$$

- Where:
- H is Mean curvature
- K is Gaussian curvature

Unfortunately, these formulas were far too complex for our group's understanding and are noted to produce a higher accuracy than the formulas we decided to utilize.

Our group did reach a problem in calculating the derivative of radial curvature DwKr using our formulas. We were unable to find a solution to this problem even through research of our own. We came across the formula:

$$D_{\mathbf{w}}\kappa_r = \mathbf{C}(\mathbf{w},\mathbf{w},\mathbf{w}) + 2K\cot\theta$$

While we could obtain the values for K and cot, C(w,w,w) remained unobtainable.

III.IV Threshold

After obtaining the n·v value for each vertex, the next step is to display the vertices whose n·v = 0. Unfortunately, our group faced another quandary because n·v was not returning values equal to 0. A solution to this problem is to increase the n·v (contours) and Kr (suggestive contours) threshold. Since our data type is GLdouble, its accuracy is relatively high thus the threshold had to be adjusted several times to achieve a perfect threshold for n·v (contours) and Kr (suggestive contours).

Figure 5 depicts the best threshold value for different meshes as tested by our group. Our user interface allows users adjust the thresholds accordingly by pressing certain keys.

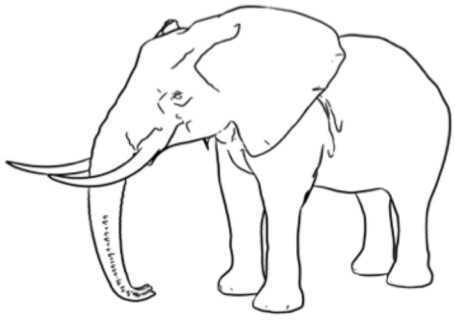
Threshold	Elephant	Cow	Dragon	Sphere
Contours	0.05	0.00025	0.00001	0.01
S_contours	0.00005	0.0008	0.00001	0.025

Figure 5

III.V Connecting Points

Connecting points created many issues, mainly regarding the order in which the vertices were being created. When applying line strips to the vertices, the image was very distorted with lines extending from the top left edge of the mesh to the bottom right edge of the mesh and lines extending from the top of the middle of the mesh to the bottom of the left. Correctly connecting the points via a line would require an iteration over the entire mesh, finding the closest vertices, and then drawing a line strip connecting those two points. After following advice from our professor, and mentor, Alark Joshi, our group decided to plot points using the function “glEnable(GL POINT SMOOTH)” rather than lines to show contours and suggestive contours because the mesh still takes the form of its intended image with little-to-no distortion. However, there still exists a problem with having too many or too few points and often points in the wrong areas or overlapping between contour and suggestive contour regions.

IV. Results



'igure 6 – "Suggestive Contours For Conveying Shape"

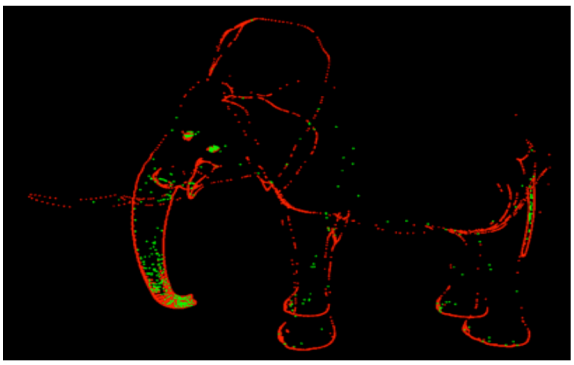
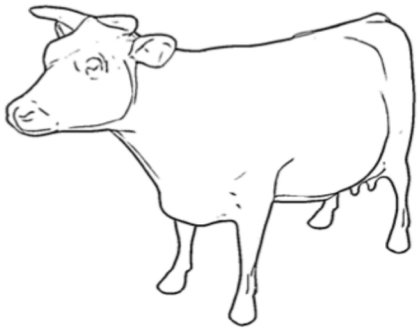


Figure 7 – Our result, Contours and Suggestive Contc

Figure 6 from the "Suggestive Contours For Conveying Shape" paper displays a masterful use of contours and suggestive contours on a mesh of an elephant. Our result (shown in Figure 7) is a promising attempt at duplicating the results from Figure 6. The results achieved by our group utilize points instead lines, as mentioned earlier, but still conveys strong contours (red points) and less potent suggestive contours (green points).



'igure 8 – "Suggestive Contours For Conveying Shape"

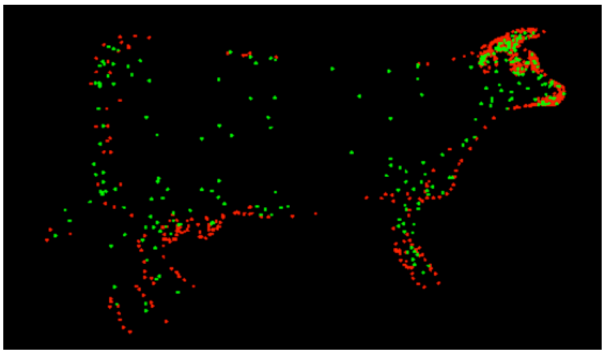


Figure 9 – Our result, Contours and Suggestive Contc

In regards to the mesh of a cow, our group experienced stronger suggestive contours (green points) than contours (red points), which was expected, but still has several outlier points that can be seen near the top of the back closest to the neck. Also, there are overlapping points near the back of the cow. Given a better understanding of mathematical concepts such as derivatives, these minor issues should be remedied.

V. Discussion

V.I Issues

Our group had difficulties with the mathematical concepts behind implementing contours and suggestive contours using the algorithms described in "Suggestive Contours For Conveying Shape." If there were a better understanding of derivatives, our results would look much closer to that of DeCarlo and his associates.

V.II Unimplemented Methods

Fortunately, our group only had two unimplemented methods, calculating the derivative of Kr and drawing using line strips. These methods did not hinder our results as much as anticipated. Our results still display contours and suggestive contours in their respective areas, but could use some improvements. We did not experience any distortion drawing using points rather than line strips and the mesh was still recognizable.

V.III Promising

Our project is very promising given our lack of knowledge in mathematics and the unforeseen issues that arose. The contours and suggestive contours were prevalent in all of the meshes tested without having several months to a year to complete the project, without having the assistance of the TriMesh2 library, and, as mentioned earlier, without a strong background in mathematics. Given any of the three setbacks, the project would have been closer to the results experienced in the "Suggestive Contours For Conveying Shape" paper.

VI. Conclusion

As stated several times prior, given our group's lack of knowledge in mathematics, the TriMesh2 library, and lack of time, the project was still successful. Our group was able to display strong contours and strong suggestive contours where necessary. Though we were not able to implement line strips, the meshes were still recognizable using points and not distorted at all. The two unimplemented methods, derivative of Kr and line strip, were minor issues and not as prevalent as we anticipated thus our results were not significantly worse than those in the "Suggestive Contours For Conveying Shape" paper. Overall, our group is pleased with the results achieved given our circumstances and we believe that the project was successful.

VII. References

1. Suggestive Contours For Conveying Shape – http://www.cs.rutgers.edu/ decarlo/pubs/sg03.pdf

2. Estimating Curvatures and Their Derivatives on Triangle Meshes http://gfx.cs.princeton.edu/pubs/2004 ECA/curvpaper.pdf

3. Interactive Rendering of Suggestive Contours with Temporal Coherence https://www.cs.rutgers.edu/ decarlo/pubs/npar04sug.pdf

4. (Discrete) Differential Geometry http://graphics.stanford.edu/courses/cs468 10 fall/LectureSlides/05 Diff Geo.pdf

5. Digital Geometry Processing http://www.hao li.com/cs599/slides/Lecture03.2.pdf