CS348a Homework assignment 4

---------------------------------------

Team:

Nora Coler

Alexandra Tamplin

---------------------------------------

**Part 1: Mesh Decimation**

**Functions:**

*initDecimation:*

While iterating over the vertices of the mesh, a quadric for that vertex is calculated. Given the point, we iterate over the faces that contain our current point and calculate the face's contribution.

To calculate the contribution, we complete the formula ax + by + cz + d = 0, as defined in the paper "Surface Simplifications Using Quadric Error Metrics". <a, b, c> is defined as the normal at the point as it is orthogonal to the point. d is calculated as the negative dot product of the normal and the point.

With these values <a, b, c, d>, we create a Quadricd qi and add it to the quadric for the vertex. In the end, the vertex quadric is the sum of the quadrics of its faces.

*priority:*

To calculate the priority of a half edge, we first sum the quadrics from its two vertices (to and from vertites). We then return the calculation v.transpose(Q)v as our error. For simplicity, we are using only the calculation as a result from the to vertex.

*decimate:*

For the iteration, the number of iterations is \_n\_vertices. In a while loop that continues until the number of vertices is equal to \_n\_vertices, the vertices in the queue are examined. We find the half edge that originates at that vertex. If that half edge is legal to collapse, we break out and collapse the edge. The quadric from the to vertex is added to our current vertex (the from) before collapse.

Once the half edge has been collapsed, the vertices from and including the current vertex are requeued and the number of vertices is decreased by one.

**Images:**

Armadillo – 288.off

Human – 14.off

|  |  |  |
| --- | --- | --- |
| Original mesh | Mesh with 50% vertices | Mesh with 10% vertices |
| armadillotri100.png | armadillotri50.png | armadillotri10.png |
| bunnytri100.png | bunnytri50.png | bunnytri10.png |
| homertri100.png | homertri50.png | homertri10.png |
| horsetri100.png | horsetri50.png | horsetri10.png |
| humantri100.png | humantri50.png | humantri10.png |

**Part 2: Suggestive Contours**

**Functions:**

*isSilhouette:*

We obtain the two halfedges from the handle e. Then the view ray for the vertex from the first halfedge is calculated by subtracting the cameraPos from the vertex position.

With the view ray, we are able to calculate the dot product of it with the normal of each halfedge. If the products have different signs, the edge is a silhouette.

*isSharpEdge:*

We obtain the two halfedges from the handle e and their respective normals. If the dot product of those two normals is less than .5, the edge is a sharp edge.

*computeViewCurvature:*

To find the curvature at each vertex, we need to compute the equation . First we calculate the w vector which was the projective of the view vector onto the triangle face. Then we find cosθ as the principal 1 direction vector dotted with w. Similarly sinθ is the principal 2 direction vector dotted with w.

**Suggestive Contours (SC)** *renderSuggestive(true):*

To render the contours, we iterate over each face and check to see if there is a contour for that face. For each face, we see if two of the vertices have the same curvature sign while the other is different. This results in there being two spots on the edges of the face where the curvature is 0. Once we find the points on the edges where curvature is zero, we calculate the midpoint between them. From the midpoint, we calculate the w vector which is then normalized. Dotting the normalized -w vector with the normalized viewCurvatureDerivative vector gives us DwKr which must be greater than a threshold value. We also check to make sure it is greater than a threshold value. If DwKr passes these tests, then we draw the suggestive contour.

---------------------------------------

**Part 3:**

**Additional Feature: Highlights**

For our additional feature, we choose to implement *Highlight Lines for Conveying Shapes* (DeCarlo and Rusinkiewicz, 2007). We thought that the addition of highlights lines would nicely complement the suggestive contours that we had already drawn on the image.

We also implemented a toon shader to have our images appear closer to how they appeared in the paper.

**Suggestive Highlights (SH)** *renderSuggestive(false):*

Suggestive highlights are calculated the same way as suggestive contours. However you only draw the line if DwKr is less than a negative threshold value. This threshold value is greater than the one for suggestive contours.

**Principal Highlights (PH):**

To find the principal highlights, we iterate over the faces of the mesh. For all the vertices of the triangle, we find which one has the greatest k1 value. We then make sure all the principal 1 directions of the vertices are pointed in the same direction at the one corresponding to the largest k1 value. Then we check which edges have the highlight going across it by looking at the signs of the dot product of the view vector at each vertex dotted with the principal 1 direction at that vertex. We only draw the line segment for the face if the interpolation between the value of the vertices is greater than a threshold.

**Images:**

|  |  |  |
| --- | --- | --- |
| **SC** | **SC + SH** | **SC + SH + PH** |
| **armaSC.png** | **armaSH.png** | **armaPH.png** |
| **bunnySC.png** | **bunnySH.png** | **bunnyPH.png** |
| **homerSC.png** | **homerSH.png** | **homerPH.png** |
| **horseSC.png** | **horseSH.png** | **horsePH.png** |
| **humanSC.png** | **humanSH.png** | **humanPH.png** |