Spectral Mesh Flattening

Gabrielle Littlefair

March 17, 2025

1 Summary of Method

Spectral Mesh Flattening is the process of finding a 2-dimensional parameterisation of a 3-dimensional mesh using the eigenvalues of its Laplace-Beltrami.

For a parameterisation, we want it to be one-to-one or bijective so that each vertex has a unique mapping onto the 2D plane. This is useful for texture mapping, and also means that mesh reconstruction is possible.

We looked into this paper [1] and due to the following corollary (corollary 6.2), we developed our methods:

> Suppose T is any triangulation and that ϕ: $D_T \rightarrow R^2$ is a convex combination mapping which maps the the cyclically ordered boundary vertices $v_1, ..., v_n$ of T to the cyclically ordered vertices $\phi(v_0)..., \phi(v_n)$ of an n-sided convex polygon. Then ϕ is one-to-one.

This is essentially saying that given a triangular mesh, if we create a convex combination function ϕ that takes in the 3-dimensional mesh and outputs a 2-dimensional parameterisation, if ϕ maps the boundary vertices of the mesh in a cyclical order to an n-sided convex polygon, then ϕ is one-to-one.

This is exactly what we want to achieve, and so we begin our algorithm by mapping the boundary of meshes to either a circle or a square. The circle is essentially an n-sided polygon where n is the number of boundary vertices that we are mapping. It is also convex. The square is a 4-sided convex polygon. This is implemented in our functions `circle_boundary` and `square_boundary`. See the sections below for more detail on these two methods.

Once we implemented that, we now have a parameterisation for our boundary points and we need to find for the interior vertices. So we use Tutte's embedding method [2] with these boundary constraints in order to find our parameterisation. The equation is as follows:

$$\Delta_s \mathbf{u} \quad = \quad 0$$

Where Δ_s is the Laplace-Beltrami (for the surface) and $\mathbf{u}$ is the 2D parameterisation (it will be $\mathbf{u} = (u, v)$). This is equivalent to finding the smallest eigenvectors of the Laplace Beltrami. We do however have the constraints to the equation above, we implemented this by changing the rows of Δ_s for the boundary values to be equivalent to the corresponding row of an identity matrix, and then we added the values that we found for the paramaterisation of the boundary vertices to the 0 vector on the right hand side of the equation. This is implemented in the function `tutte_embedding`.

1.1 Circle Boundary

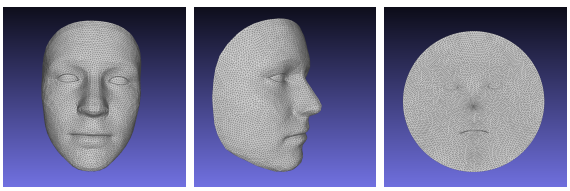


Figure 1: Face mesh flattened using the `circle_boundary` function.

The algorithm to map the boundary vertices to a circle is as follows:

1. Assign each boundary edge a weight determined by the length of the edge, divided by the total boundary length (this means all weights add up to 1).
2. Assign the first vertex to any point on the circle (in the 2D u, v plane).
3. Then follow the first vertex to the next, and place the next vertex at position

$$u = r\cos(2\pi\Sigma_{j=0}^{i} w_j)$$

$$v = r\sin(2\pi\Sigma_{j=0}^{i} w_j)$$

where r is the radius of the circle and w_j is the weight of edge j and edge i is the the edge between the current vertex and the previous vertex. Repeat this step until every vertex has been added (this will cover the full circle because the weights add to 1).

The output of this function is a list of (u, v) coordinates for the boundary vertices that lie on a circle and are spaced based on the lengths of the edges.

1.2 Square Boundary

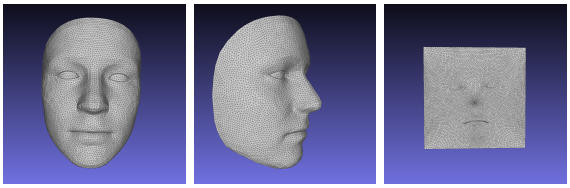


Figure 2: Face mesh flattened using the `square_boundary` function.

The algorithm to map the boundary vertices to a square is as follows:

1. Find the length of every edge.
2. Pick a starting vertex, and calculate one quarter of the total distance around the boundary (using `get_quartile_length`).
3. Set the starting vertex to be at one of the corners of the square, and follow the edges from this vertex round (until the quarter length has been reached at which time we move down a new edge) - setting them at a weighted distance of

$$(u, v) = A + \frac{d_i}{\Sigma_{i=0}^{n}}(B - A)$$

Where A and B are the two (u, v) coordinates of of an edge of the square, i is the index of the list of edges to follow, d_i is the length of edge i, and n is the edge that ends at the vertex that is being mapped to (u, v).

The output of this function is a list of (u, v) coordinates for the boundary vertices that lie on a square and are spaced based on the lengths of the edges.

1.3 Free Boundary

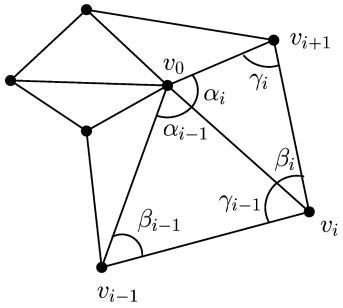
We implemented a function `free_boundary` that flattens the mesh using Tutte's embedding without any boundary conditions. We implemented this in order to compare with the square and circle boundary conditions. This is done by not setting any boundary constraints when solving the Poisson equation, and finding the smallest 3 eigenvectors of the Laplace-Beltrami, and setting the second and third smallest eigenvectors to be (u, v). This function does not work with meshes that are not very uniform.

2 Extension Work

2.1 Other Laplace-Beltrami Discretizations

We implemented two other discretizations of the Laplace-Beltrami matrix in order to see if a better parameterisation could come out of using a different discretization.

We implemented a discretization based on mean value coordinate weights [3], which we refer to as the Mean Value Discretization. This code can be found in `mvLaplaceBeltrami`. The weights for vertex v_0 are defined as follows:



$$\lambda_i = \frac{w_i}{\sum_{j=1}^{k} w_j}, \quad w_i = \frac{\tan(\alpha_{i-1}/2) + \tan(\alpha_i/2)}{\|v_i - v_0\|},$$

We also implemented a discretization that we refer to as the Improved Laplace-Beltrami from this paper [4]. Where the weights are as follows:

$$W_{ij} = \begin{cases} \frac{1}{\kappa_j} \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{4t}\right) & \text{if } \|\mathbf{x}_i - \mathbf{x}_j\| < \varepsilon \\ 0 & \text{otherwise,} \end{cases}$$

Where we took the ϵ constraint to be that the vertices i and j are neighbours, and x_i is the position of vertex i, x_j is the position of vertex j, and κ_j is the number of neighbours of vertex j. We then created the diagonal matrix by summing the weights along a row, and then as per the paper the Laplace-Beltrami is

$$L = W - D$$

.

2.2 Finding a Seam for Closed Meshes

We decided that we wanted our seam to go through high curvature points because these points are where more significant changes occur on the mesh, so we thought that if we cut along these it would result in less distortion across the mesh.

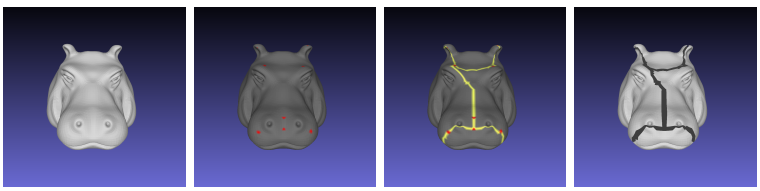


Figure 3: Seam Algorithm: Find Waypoints, Join Waypoints, Remove Seam.

We decided to use Gaussian curvature because we wanted areas where both minimum and maximum curvatures are high. We created the function `waypoints` that finds these waypoints. Once the highest curvature points have been found, we then decided to keep only one in a certain radius (i.e. remove any that were too close to each other) - our `threshold` input determines this radius.

Once we have the points, the function then creates a full connected graph (where every node is connected to every other node), and then creates a minimal spanning tree of this graph in order to find the order to traverse the waypoints. This list of edges of the minimal spanning tree is then the output of the waypoints function.

Our seam function then uses these waypoints combined with a function (`djikstra_path` [5]) that finds the shortest path between two points based on weights to find paths between connected waypoints. We decided to use weights that depend on both length of the path and the gaussian curvature so that the algorithm would prioritise short lengths and high curvatures (see `curvature_weights` function).

Once we found the shortest path between all of the waypoints, this then created our seam. We then used our function `delete_faces_along_seam` to remove the faces along the seam so that our mesh becomes open. We originally intended to remove the faces and then add them back so that we had duplicates of our seam vertices. We would then flatten the mesh, then use one half of the circle or square's boundary vertices as the parameterisation (since the vertices would be duplicated). Unfortunately due to time constraints we did not manage to complete this. However the code that we do have can be found in the "Work in Progress" section of our notebook. The functions are `reconstruct_faces` and `split_seam`. Because of this, all of our meshes that have the seam applied have a hole in them along the seam. I have visualised these with the seam underneath them in yellow in order to make it clear what is going on. If we had managed to fill in the seam, we would still have discontinuities along the seam because of how the parameterisation works.

For results of using the seam please see the Evaluation section.

3 Meshes

3.1 Mesh and Texture Download Links

Below are linked the meshes used in this report, however there are more meshes in our folders that we used for testing and the links for those can be found in the `README.md`.

- Face https://sketchfab.com/3d-models/face-mesh-wael-tsar-2a344a540fc2426c95ac6176b91164d7
- Horse Head https://free3d.com/3d-model/horsehead-v3--43341.html
- Hippo https://free3d.com/3d-model/noveltyhead-partial-hippo-v1--526682.html
- Ear https://free3d.com/3d-model/ear-v1--113169.html
- Hand https://free3d.com/3d-model/hand-v3--902450.html
- Check Texture https://www.vecteezy.com/free-vector/black-white-checkered

For all of these meshes, we applied some prepocessing techniques to clean them up and make sure that they were triangular. We did this using tools in MeshLab [6]. We also used MeshLab to create smaller versions of our meshes, as well as to cut open some of our meshes.

3.2 Visualisation of Meshes

To visualise the paramaterisations we used a check texture that is applied based on the (u, v) mapping. The code for this can be found in the two functions `export_textured` and `export_textured_free`.

When I show the split mesh with texture, so that it is clear I have a mesh underneath with the seam shown in yellow (since there is a hole in our mesh).

4 Evaluation

4.1 Circle vs Square vs Free Boundary Flattening

For this investigation I used the "open_meshes/Face.obj" mesh.

For my first investigation, I decided to investigate the differences in results when using the circle boundary, square boundary, or not setting any boundary conditions. The results can be seen in the image below.

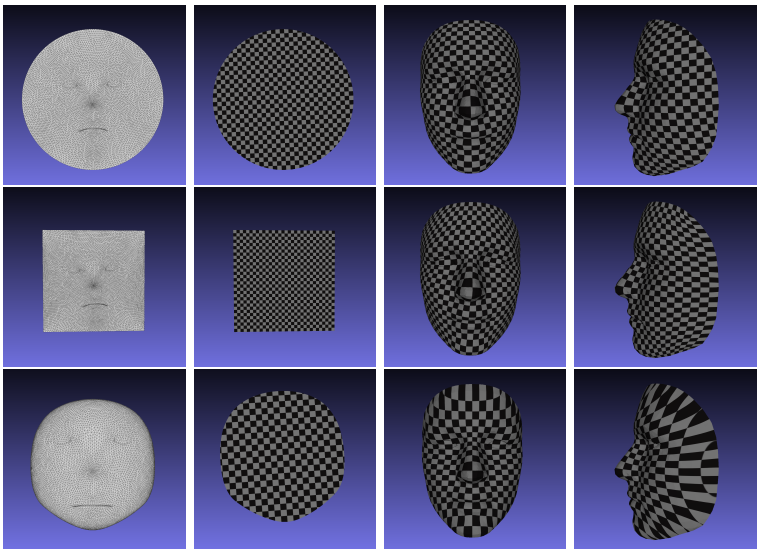


Figure 4: Top Row: mesh flattened with circle boundary values (`circle_boundary`), Middle Row: mesh flattened with square boundary values (`square_boundary`), Bottom Row: mesh flattened with no boundary constraints (`free_boundary`).

From the images above we can see first that the free boundary solution has a lot more distortion along the boundary of the mesh. This is especially obvious when looking at the side view of the face, and when looking at the forehead of the front view. For the circle and square boundaries the front view looks very similar, however from the side view, we can see along the forehead and the chin that the square boundary introduces some distortion. This appears to be the places where the corners of the square are.

This makes sense because the boundary of the face mesh is very round and doesn't have any sharp corners or turns. The square boundary would potentially be better for a boundary that is already less rounded / naturally shaped.

To conclude this investigation, since most open meshes seem to have a rounded boundary, I will be continuing to use the circle boundary for future experiments.

4.2 Different Laplace-Beltrami Discretizations

For this investigation I used the following meshes: "open_meshes/Face.obj" and "open_meshes/Horse2_smaller.obj".

For my next investigation I decided to look into different discretizations of the Laplace-Beltrami matrix used for calculating Tutte's Embedding. We used four different discretizations - 1. Uniform Laplace-Beltrami, 2. Cotan Discretized Laplace-Beltrami, 3. Mean Value Laplace Beltrami, and 4. The Improved Laplace-Beltrami [4]. The results are shown below.

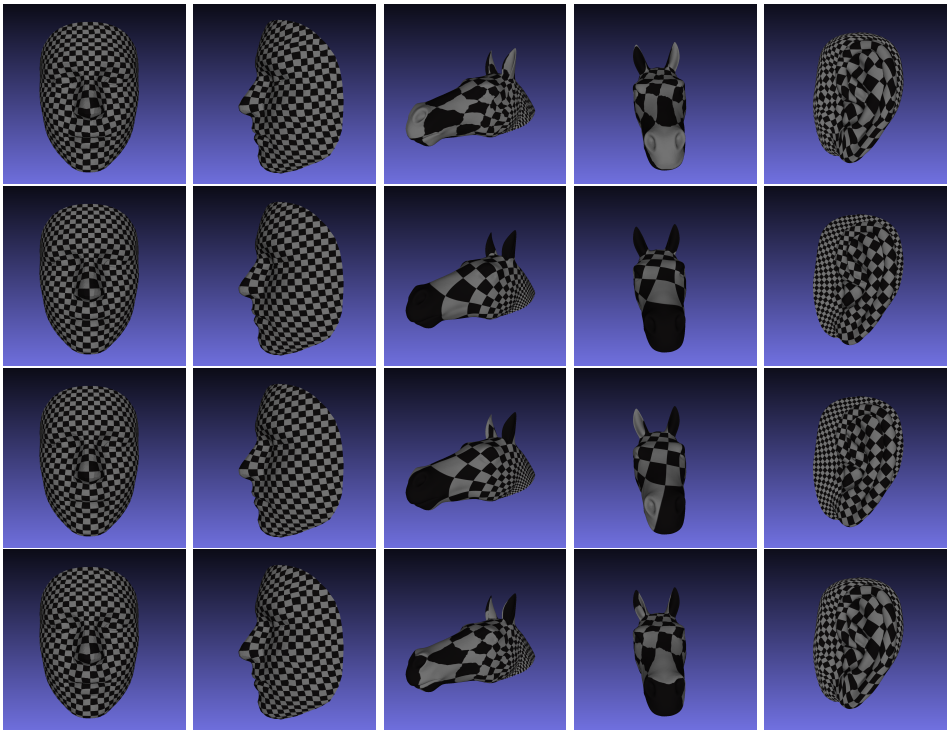


Figure 5: Top Row: Uniform Discretization, Second Row: Cotan Discretization, Third Row: Mean Value Discretization, Fourth Row: "Improved Laplace-Beltrami" [4].

The Uniform and Improved Laplace-Beltrami results show a lot more distortion than the Cotan and Mean Value. They add a sort of *wobble* along some of the check edges that isn't present at all for the other two. This is especially clear on the horse mesh. This may be due to the lack of angle incorporation into the mesh discretization. The straight lines on the Cotan and Mean Value results suggests that vertices and faces along that line have their *relationship*, or that it behaves more linearly, however for the Uniform and Improved Laplace-Beltrami this does not appear to be the case.

For the face, the Mean Value and Cotan results look nearly identical, but for the horse and the ear very slight variation can be seen. On the horse from the front view, we can see that for the Cotan discretization, the entire horse nose is black, and there is one white square above it which for the Mean Value results is two squares, and the nose is also two squares. From the side view, there is no obvious difference between the two.

To conclude, the Mean Value and Cotan discretizations result in much less distortion than the Uniform and Improved Laplace-Beltrami. There is only very little difference between the Mean Value and the Cotan discretization, and it is only visible on one out of three meshes, therefore I will continue with the Cotan discretization for the next experiments.

4.3 Mesh Complexity

Next, I decided to look into whether the complexity (number of vertices) affects the parameterization. The three meshes I used had 38019 vertices, 9557 vertices, and 4805 vertices.

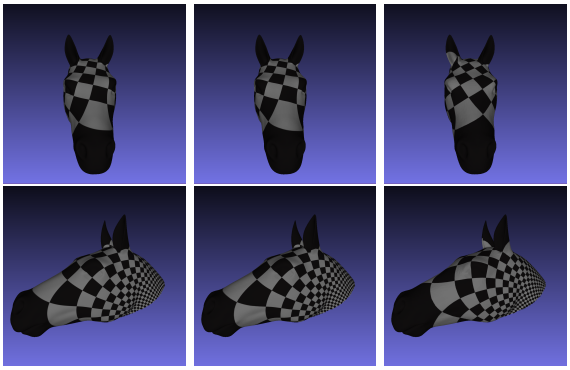


Figure 6: Resulting 2D parameterised visualised with a check texture on three different sized meshes. Left: 38019 vertices, Middle: 9557, Right: 4805.

We can see from the results above that the complexity of the mesh had very little effect on the parameterisation. From the front and view of the horse, the left and middle horse look identical (38019 and 9557 vertices), and there is very slight change to the right horse (4805). There is no obvious difference in distortion, only slight differences in the actual parameterisation.

To conclude, the method is robust for varying mesh sizes. However it is worth noting that these meshes all had *good* triangulations - i.e. the triangles were very uniform. So it could be the case that different numbers of vertices would matter if the triangulations were worse.

For this investigation I used the following meshes: "open_meshes/Horse2.obj", "open_meshes/Horse2_small.obj" and "open_meshes/Horse2_smaller.obj".

4.4 Varying Maximum Distance From Boundary

We found that when we run this algorithm for meshes that had a boundary, and then had elements of the mesh that were quite far from the boundary, that we would end up with large distortion along the faces that were further away. For this investigation, I cut a mesh at different points, so that the further points would be at different distances from the boundary. The results can be seen below.

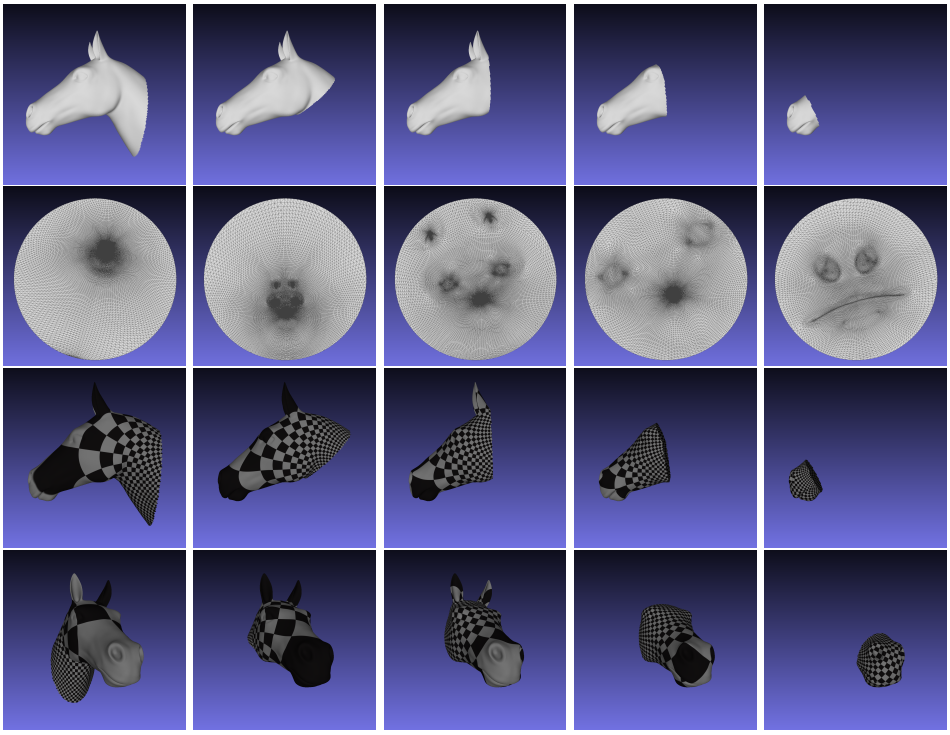


Figure 7: Top row: Meshes with varying maximum distances from the boundary, Second row: 2D parameterisation of top row meshes with boundary set to a circle, Third and Fourth Row: Visualisation of parameterisation with check texture.

The results of this investigation are what we would expect. The left column shows us that when the furthest point is at a larger distance from the boundary, the faces that are far away have very small area in the circular parameterisation (as can be seen in the second row of the images above where the nose is the whole circle on the furthest right column, and the entire face becomes indistinguishable in the furthest left) than the faces that are closer to the boundary. The difference in the horse's noses across all the different meshes effectively shows that as the distance gets smaller, the parameterisation along the nose becomes much better (with much less distortion). This could potentially be improved upon by finding a better seam, which is what led us onto our research into finding a good seam.

For this investigation I used the following meshes: "open_meshes/Horse1.obj", "open_meshes/Horse2.obj", "open_meshes/Horse3.obj", "open_meshes/Horse4.obj", and "open_meshes/Horse4.obj".

4.5 Good Triangulation vs Bad Triangulation

For my final investigation with open meshes, I decided to look into whether the parameterisation is affected by how *good* (how uniform the triangles are) the triangulation is.

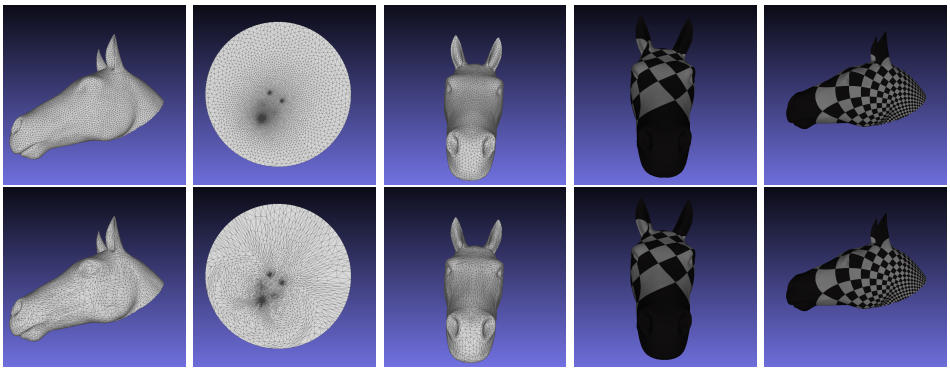


Figure 8: Mesh with very uniform triangulation (top row) compared with a mesh with a less uniform triangulation (bottom row).

From the figure above we can see that even though one of the meshes is visibly worse (the triangles are much less uniform), that there is no obvious difference in distortion. There are fewer vertices in the worse mesh, however after our experiment above we know that this is not going to cause any distortion. We can now also conclude that the algorithm is robust to different triangulations! However it is worth noting that the triangulation could be much worse and it might be worth investigating that as well.

For this investigation I used the following meshes: "open_meshes/Horse2_smaller.obj" and "open_meshes/Horse2_smaller_remeshed.obj".

4.6 Length of Seam vs Gaussian Curvature

For our seam, we use Dijkstra's Algorithm to find minimal paths between waypoints. For this, it minimizes a weight that we created that is composed of length and gaussian curvatures, so that the seam will prioritise short paths that go through high curvature points. For this investigation, I decided to vary how important the length is in the weight - I ran it for 4 different weightings with it having high importance (100 times as much as curvature), medium high importance (2 times as much as curvature), medium low importance (the same as for curvature) and low importance ($\frac{2}{3}$ times as much as curvature). The results are shown in Figure 9.

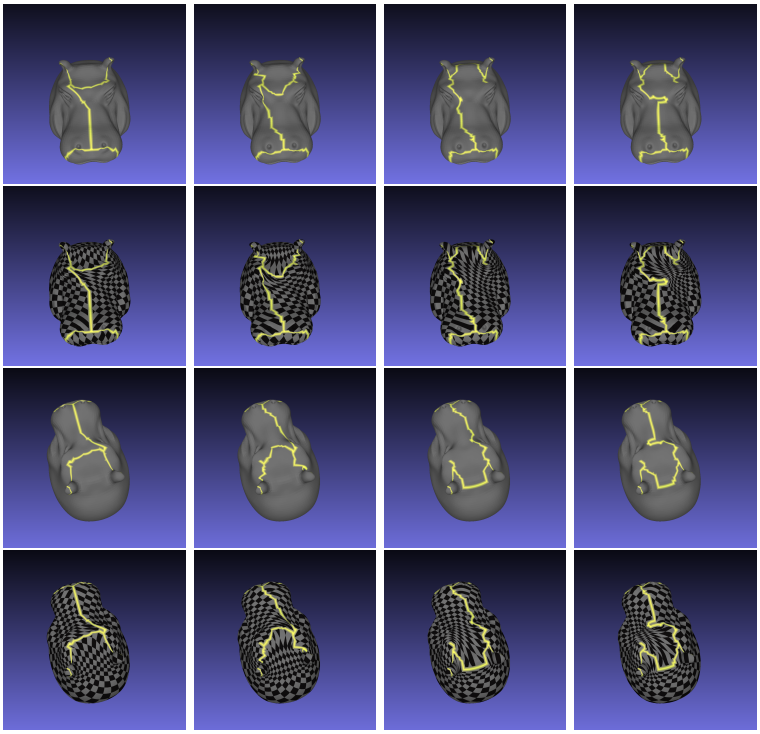


Figure 9: Left: High length importance, Middle Left: Medium high length importance, Middle Right: Medium low length importance, Right: Low length importance.

We can see from rows 1 and 2 that as the importance of the length decreases, the seam because much more *wiggly*, and also longer (as expected). We can see from rows 2 and 4 of the figure (the textured meshes) that as the importance of the length decreases, the distortion increases. This is especially obvious from the top down view of the hippo.

We can conclude from this that the length should contribute more to the cost of each edge than the gaussian curvature. It also might be the case that the gaussian curvature should not be factored in at all (beyond for finding the waypoints) and that minimum length should be found. In the next experiments I will use high importance of length (100 times as for the curvature).

4.7 Varying the Number of Waypoints

A waypoint, in terms of our seam, is the number of high curvature vertices to use as points that the seam has to go through. This experiment looks into the effects of varying this number.

I found the seam for the hippo using 6 waypoints, 17 waypoints, 26 waypoints, and 53 waypoints.

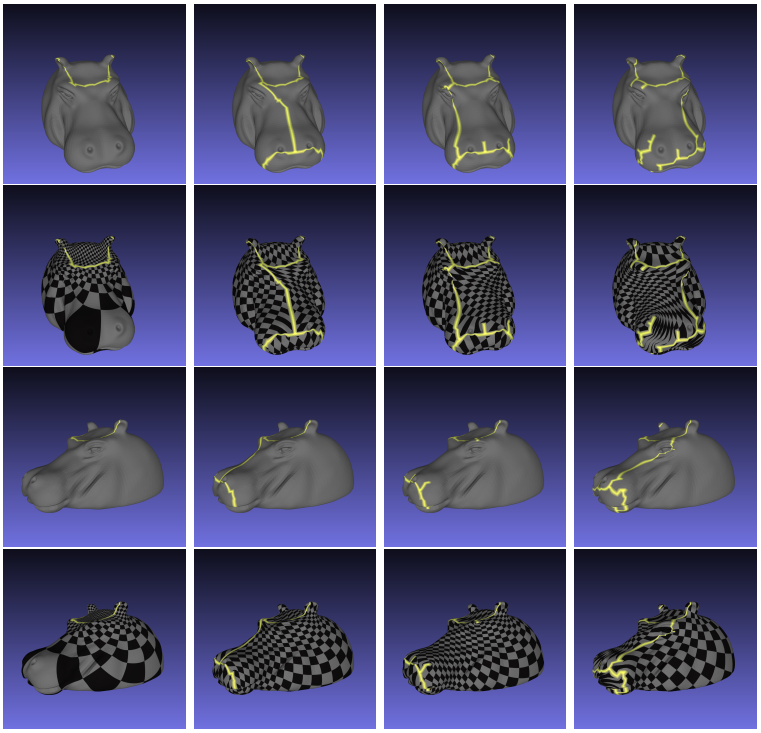


Figure 10: Left: 6 waypoints, Middle Left: 17 waypoints, Middle Right: 26 waypoints, Right: 53 waypoints.

In Figure 10, the results of this experiment can be seen. Because there are so few points for the first hippo, the seam is very short, which creates the problem of some of the vertices being too far from the seam and so this does not produce very good results. When using 53 waypoints, we can see that this causes too much of the *twisting* distortion. The seam becomes very bumpy (and it becomes longer because it has to take more detours along the route) and all along the seam the distortion is very bad (which in turn means more distortion). However for 17 and 26 waypoints, we can see that distortion is much less - and there is very little difference in distortion between the two of these. This implies that we need to find a good balance for the number of waypoints because we want enough that the seam is not too short, but not too many that it causes windy routs and longer paths than necessary. It might be worth in future investigating if placing more waypoints in more different locations (so increasing the threshold in our `seam` function) would change the results at all.

4.8 Comparison of Seam Cut with Opening the Mesh

For this experiment, I wanted to investigate how our seam method compares to use removing a section of the mesh (that makes sense) for the parameterisation. In Figure 11, I have show images of the input meshes for the two methods. The top row shows the meshes that I opened in MeshLab [6] that are input straight into the boundry and embedding functions, and the bottom row shows the closed meshes, that are then opened using our `seam` and `delete_faces_along_seam` functions before being flattened.

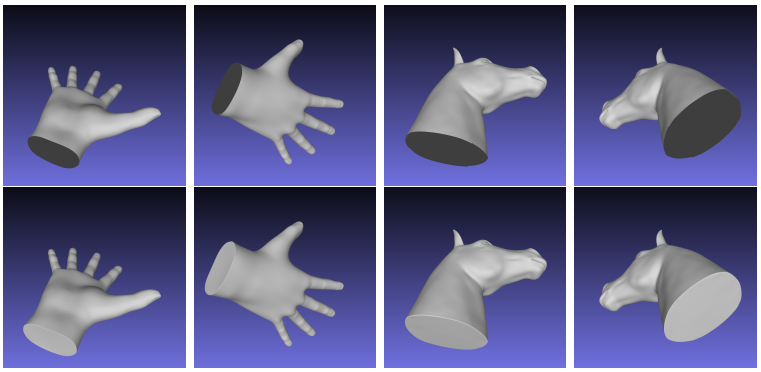


Figure 11: Top row: Open version of the meshes, Bottom: closed version of the meshes.

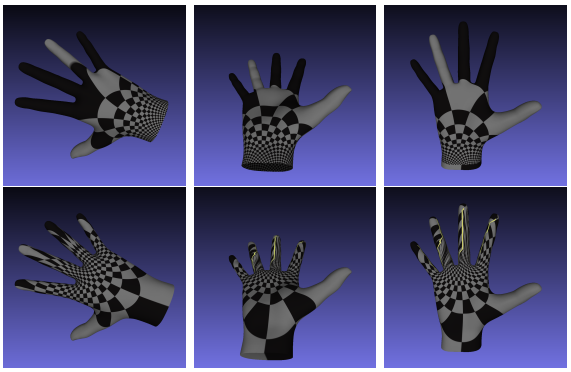


Figure 12: Top Row: Results for open hand mesh, Bottom Row: Results for closed hand mesh.

In Figure 12, we can see that for both the open mesh and the closed mesh there is a lot of distortion. However, for the closed mesh, there is much less distortion at the higher curvature areas (the finger tips) and there is more distortion at the end of the hand. This makes sense, because we know that the distortion is worse at a larger distance from the boundary/seam.

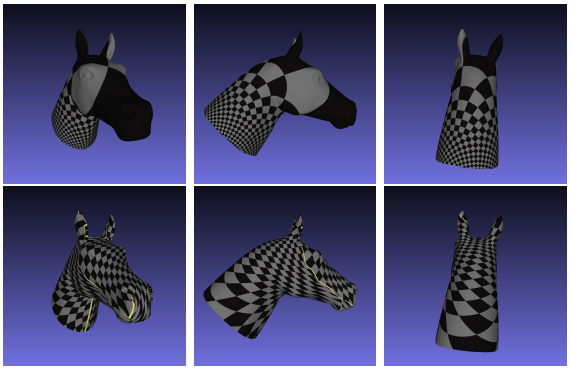


Figure 13: Top Row: Results for open horse mesh, Bottom Row: Results for closed horse mesh.

In Figure 13, we can see the results for the open and closed horse mesh. The closed mesh has more *twisting* distortion, and the open mesh has much more scale distortion. As always, there would also be discontinuities along the seam. Which of the two of these is better would be down to preference and for what it was actually being used for. It is also hard to fully compare without our seam being closed.

For this experiment I used the following meshes: "open_meshes/hand.obj", "closed_meshes/closed_hand.obj", "open_meshes/open_horse_head.obj", and "closed_meshes/closed_horse_head.obj".

4.9 Seam Results

Meshes used: "closed_meshes/horse_60000.obj", "closed_meshes/camel.obj", and "closed_meshes/bumpy-cube.obj".

We can clearly see from all of these that the seam is not perfect. There is a lot of distortion, especially around the seam. This is due to use forcing an organic shape into a circle and so the distortion is expected.

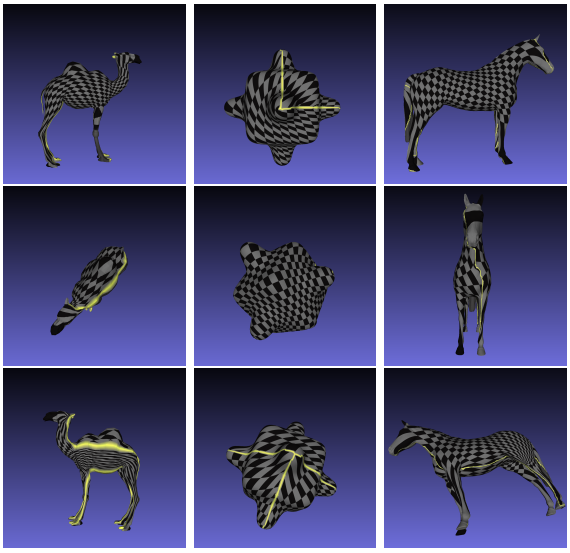


Figure 14: Results of closed meshes with seam.

References

[1] M. Floater, "One-to-one piecewise linear mappings over triangulations," *Mathematics of Computation*, vol. 72, no. 242, pp. 685–696, 2003.

[2] N. J. Mitra, "Mesh parameterization," lecture slides, UCL, March 2024. [Online]. Available: https://ucl-eu-west-2-moodle-sitedata.s3.eu-west-2.amazonaws.com/26/df/26df446d5dcdd82c98b5fca8b15dd207b87ce906?response-content-disposition=inline%3B%20filename%3D%2207_Parameterization_23-24.pdf%22&response-content-type=application%2Fpdf&X-Amz-Content-Sha256=UNSIGNED-PAYLOAD&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIA47YHZF637GKGWUJC%2F20240417%2Feu-west-2%2Fs3%2Faws4_request&X-Amz-Date=20240417T072739Z&X-Amz-SignedHeaders=host&X-Amz-Expires=21561&X-Amz-Signature=2805f0e26ae46c24e03319589c0e4659978aafc9cb6acdcab4b97bec888e8629

[3] M. S. Floater, "Mean value coordinates," *Computer Aided Geometric Design*, vol. 20, no. 1, p. 19–27, Mar. 2003. [Online]. Available: http://dx.doi.org/10.1016/S0167-8396(03)00002-5

[4] A. Baghani and B. N. Araabi, "Improved laplacian eigenmaps."

[5] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.

[6] MeshLab Development Team, "Meshlab," http://www.meshlab.net/, Accessed: 2024.