COL781 Assignment 2

Salil Gokhale
2021MT10237

Aniruddha Deb
2020CS10869

6 Feb, 2024

1 Half Edge Data Structure

We implemented the half edge data structure using arrays instead of pointers keeping in mind performance considerations. Half edges on the boundary were assigned dummy pairs. These dummy half edges do not belong to any triangle (their triangle pointer is -1), but are connected to other dummy half edges via the next pointer. Now we can implement mesh edit actions cleanly without worrying about half edge pairs not existing. Rendered image of both simple meshes is given below. The unit square mesh was placed at $z = -1$ to fit the entire mesh inside the window.

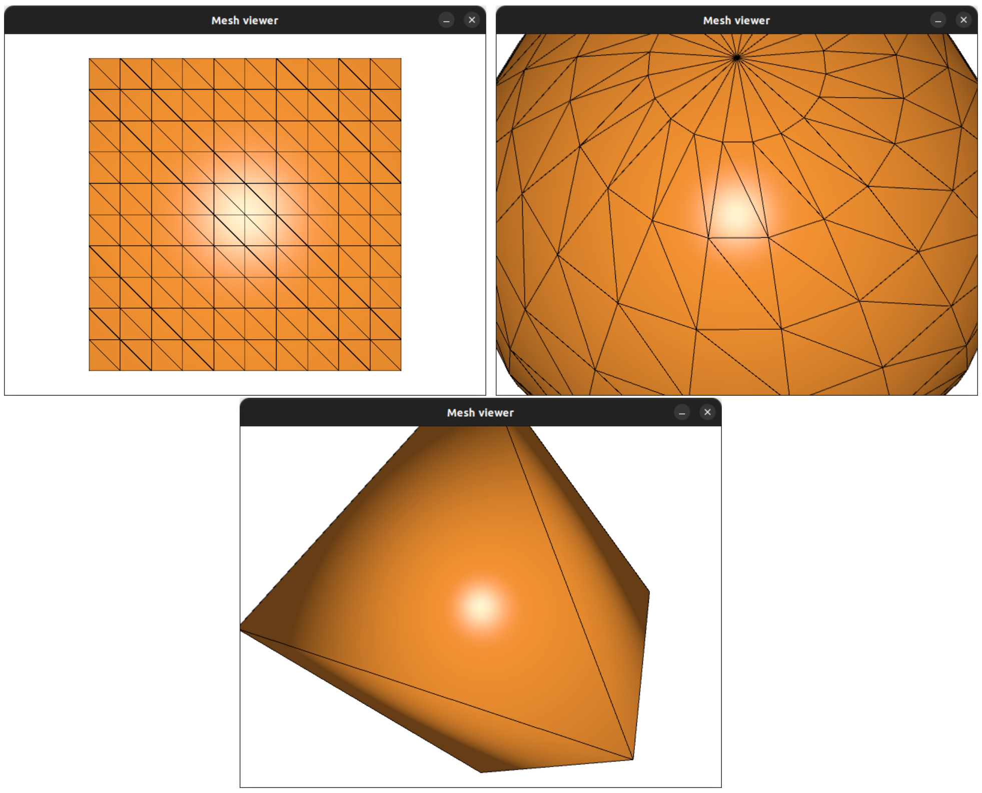


Figure 1: $10\times$ 10 square; $m = 20, n = 20$ sphere; $m = 4, n = 2$ sphere

2 Obj Meshes

We first pass over the obj file once to load all the vertices and half edges. In the first pass, boundary half edges do not have dummy pairs. After the first pass is over, we traverse over the half edges once again to set the boundary and dummy half edges.

We also implemented normal recomputation according to the scheme derived by [1]. Rendered images of all obj meshes after normal recomputation are shown below.

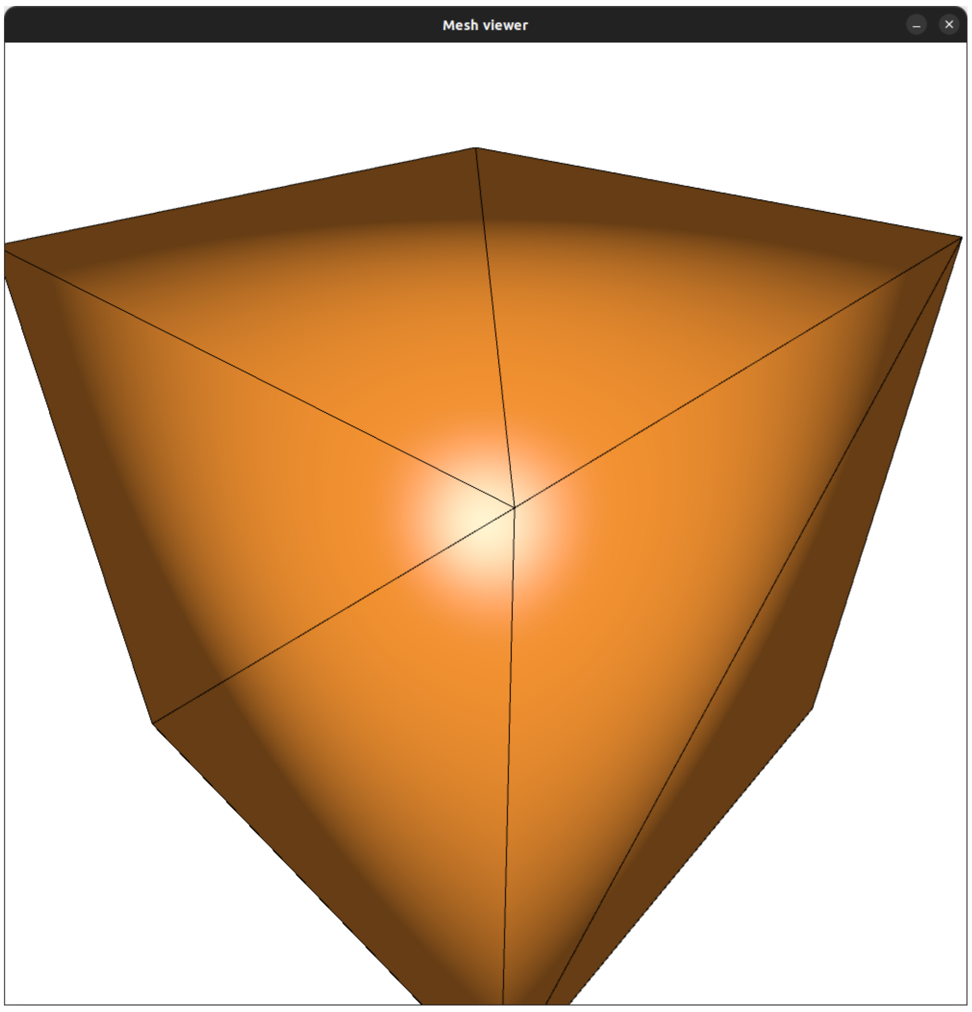


Figure 2: Cube

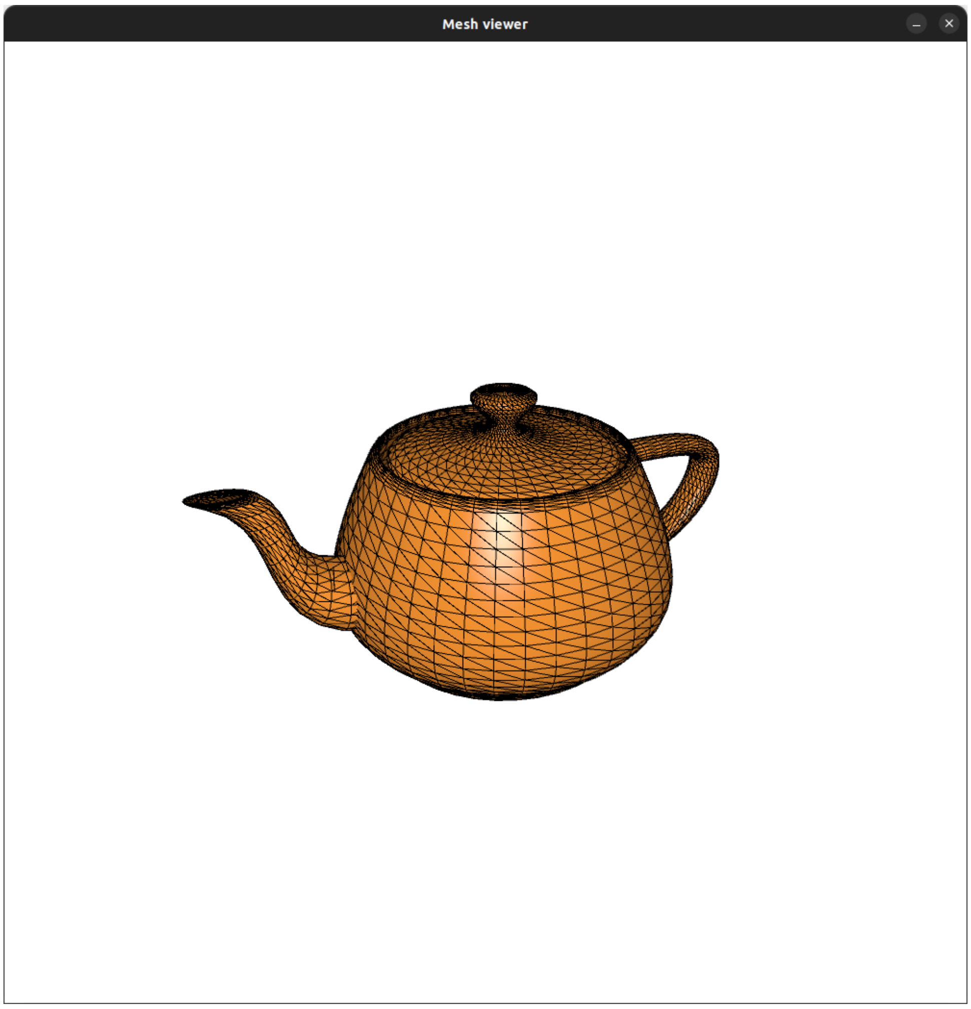


Figure 3: Teapot

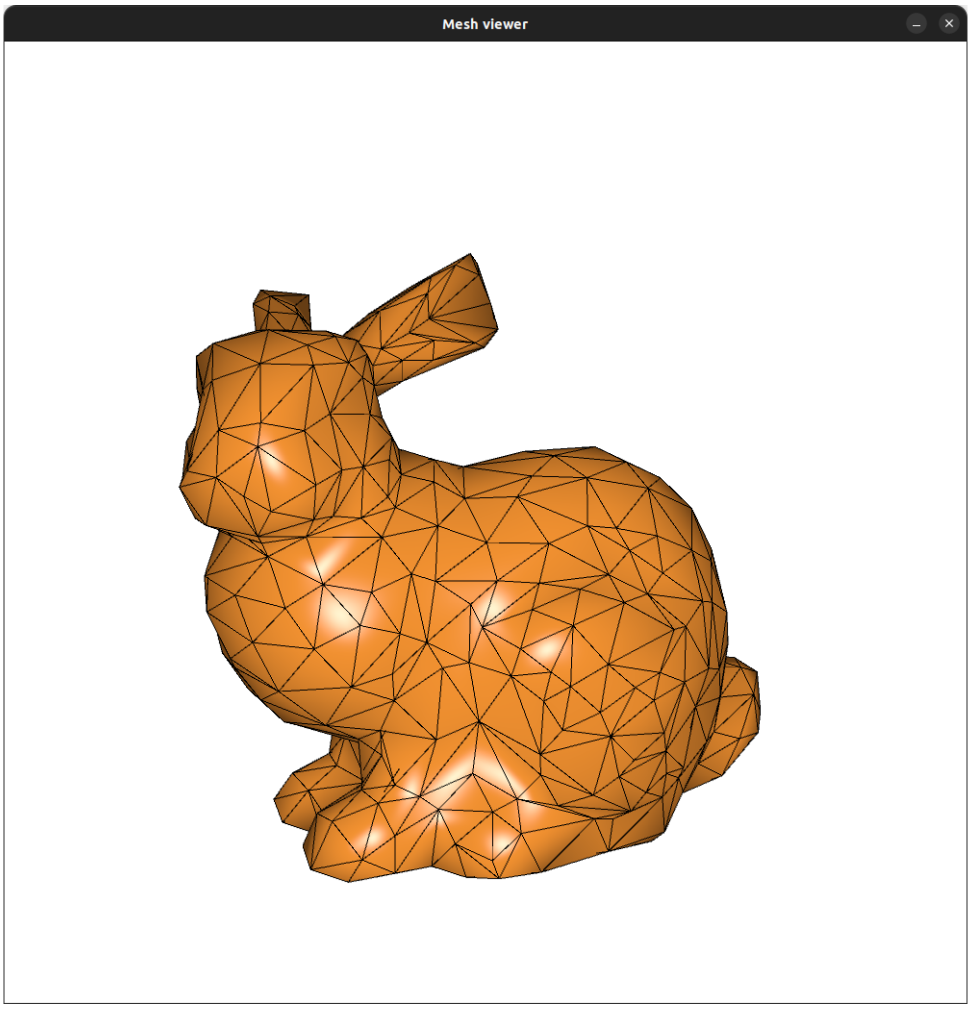


Figure 4: Bunny

3 Smoothing

Both naïve smoothing and Taubin smoothing are straightforward to implement. The results of these on the noisycube mesh are shown below.

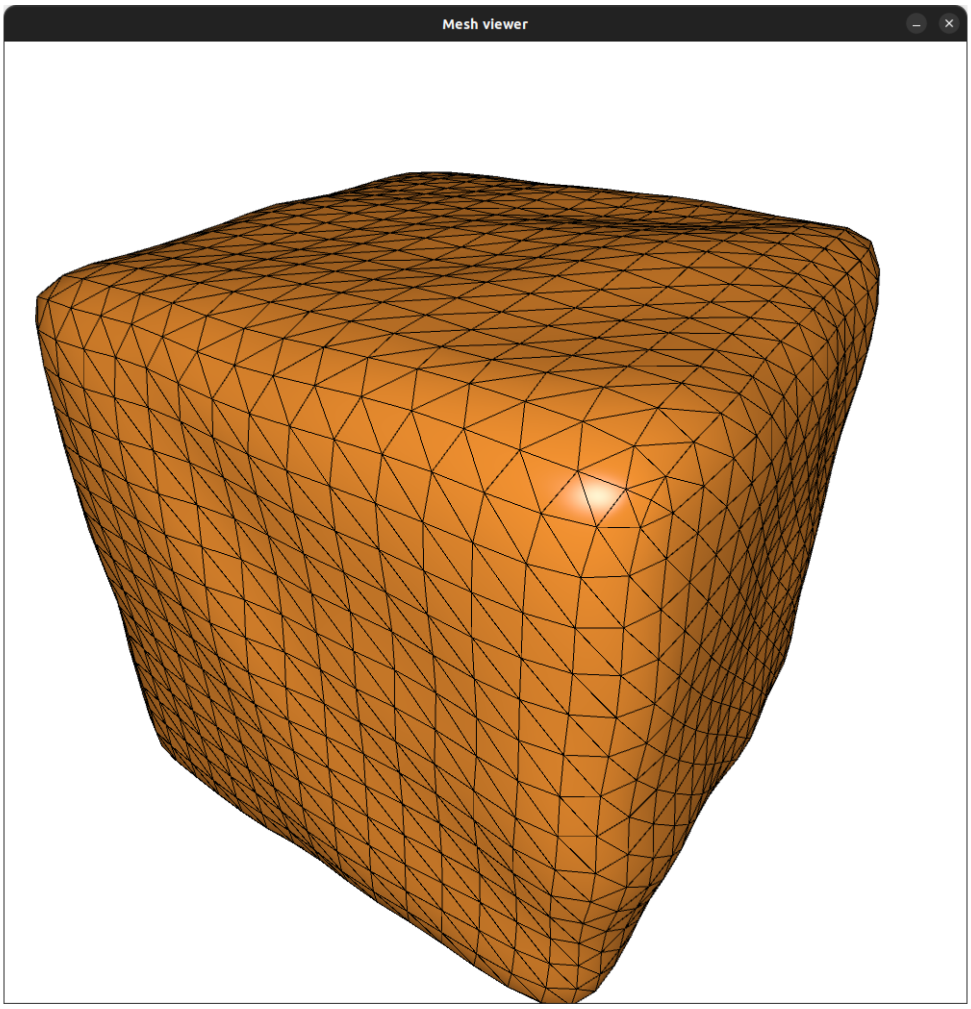


Figure 5: Naïve smoothing of noisycube with 10 iterations

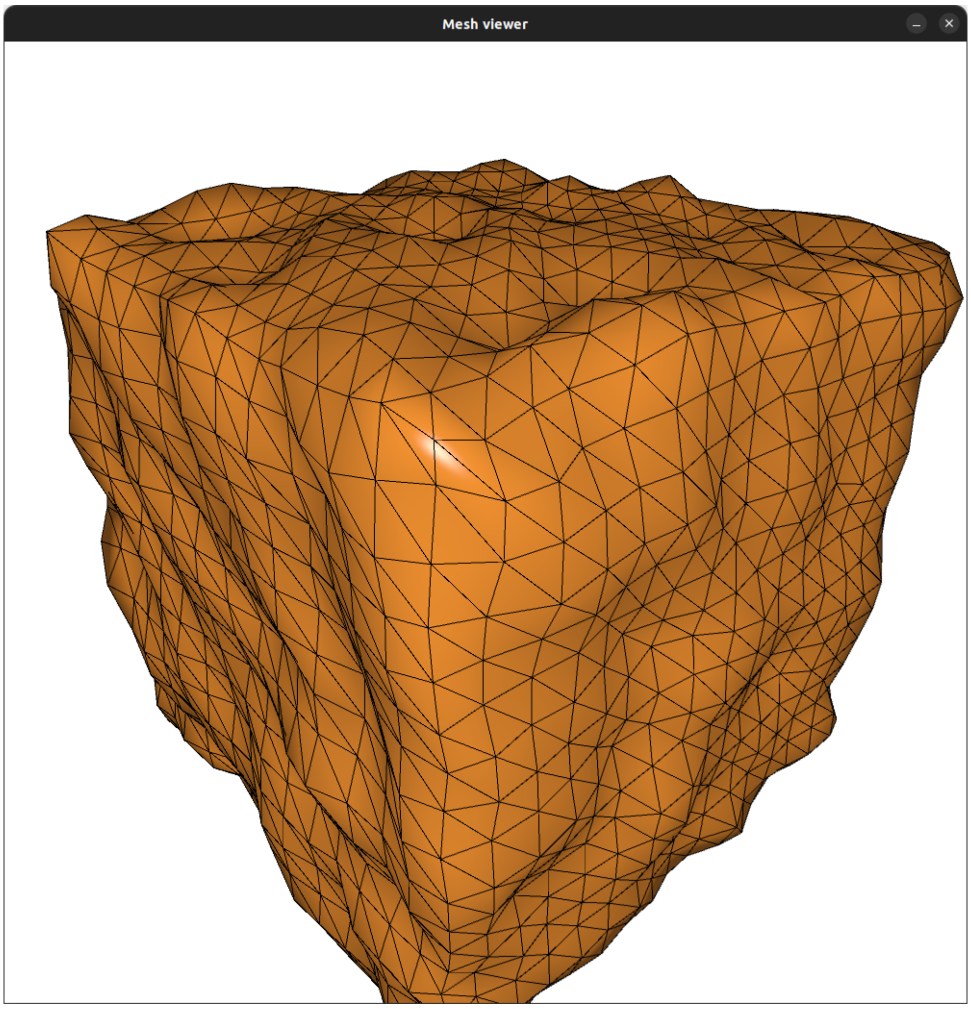


Figure 6: Taubin smoothing of noisycube

4 Mesh Editing Operations

4.1 Edge Flip

Since the number of half edges, vertices and triangles remains the same we don't need to increase or decrease the size of our arrays. We additionally implement a function flip_check() which checks if the given edge flip is a legal operation. In particular, we verify that the given edge is not on the boundary and that the quadrilateral involved in the edge flip is convex. The result of edge flip on the middle edge of the 3×3 square is show below.

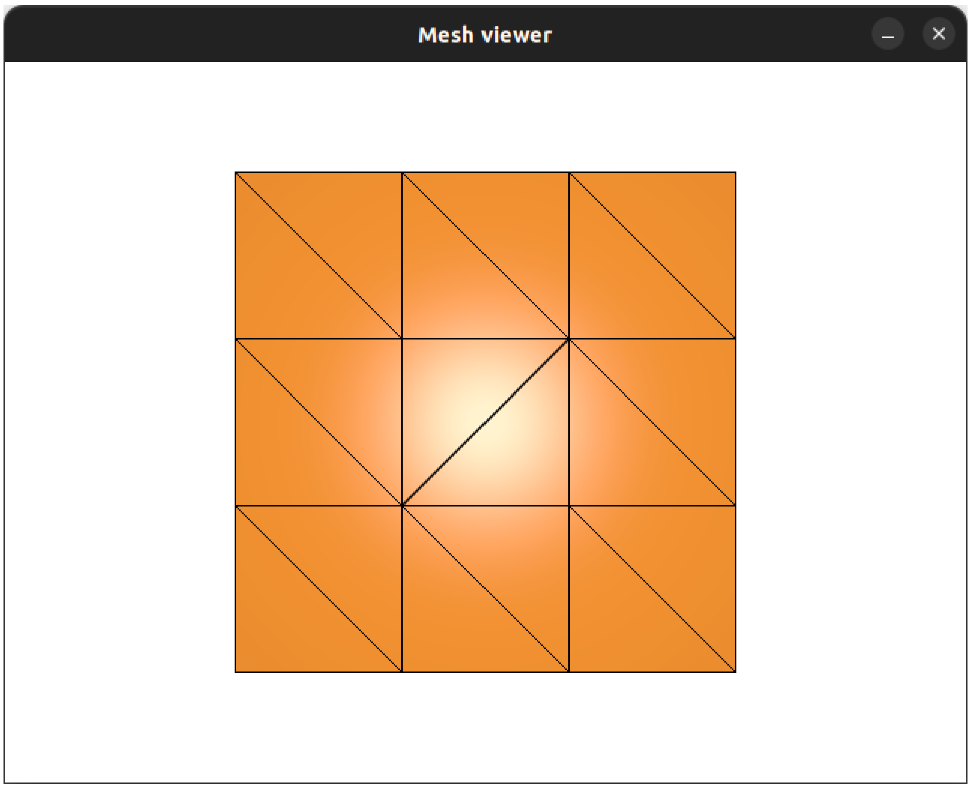


Figure 7: Edge Flip on the 3×3 square.

4.2 Edge Split

Since the number of half edges, vertices and triangles only increases we can easily append to our arrays. Also there is no need to check for the legality of an edge split operation, since it's always legal. We also implement edge splitting of boundary half edges. The result of edge split on the 3×3 square is show below.

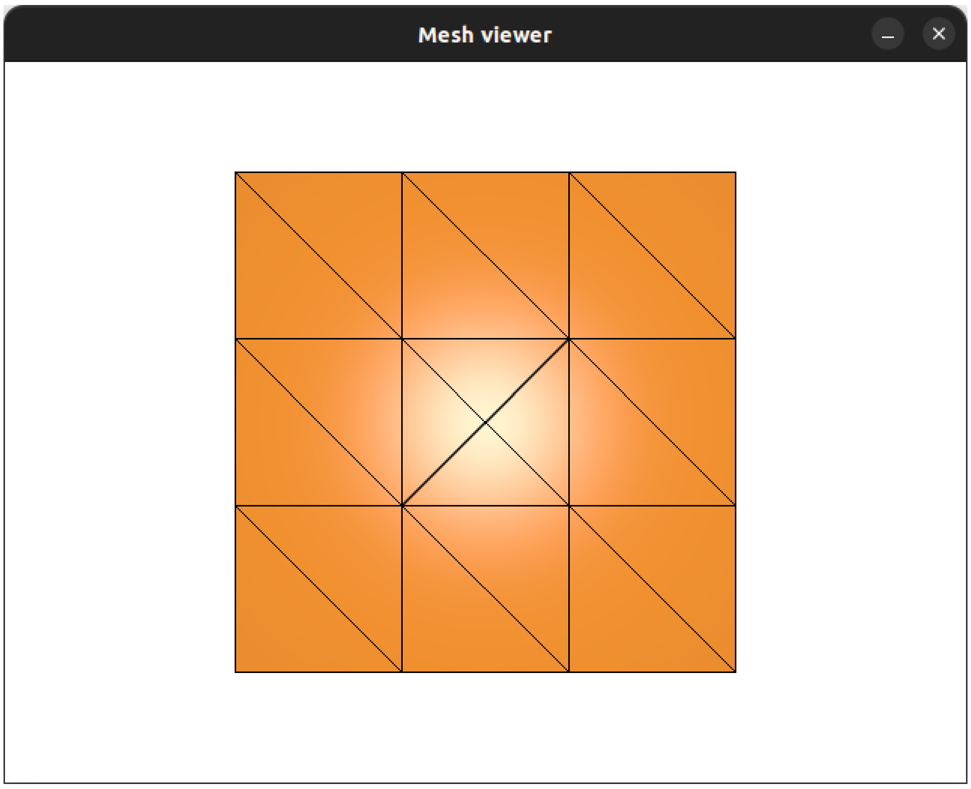


Figure 8: Edge Split on a interior edge.

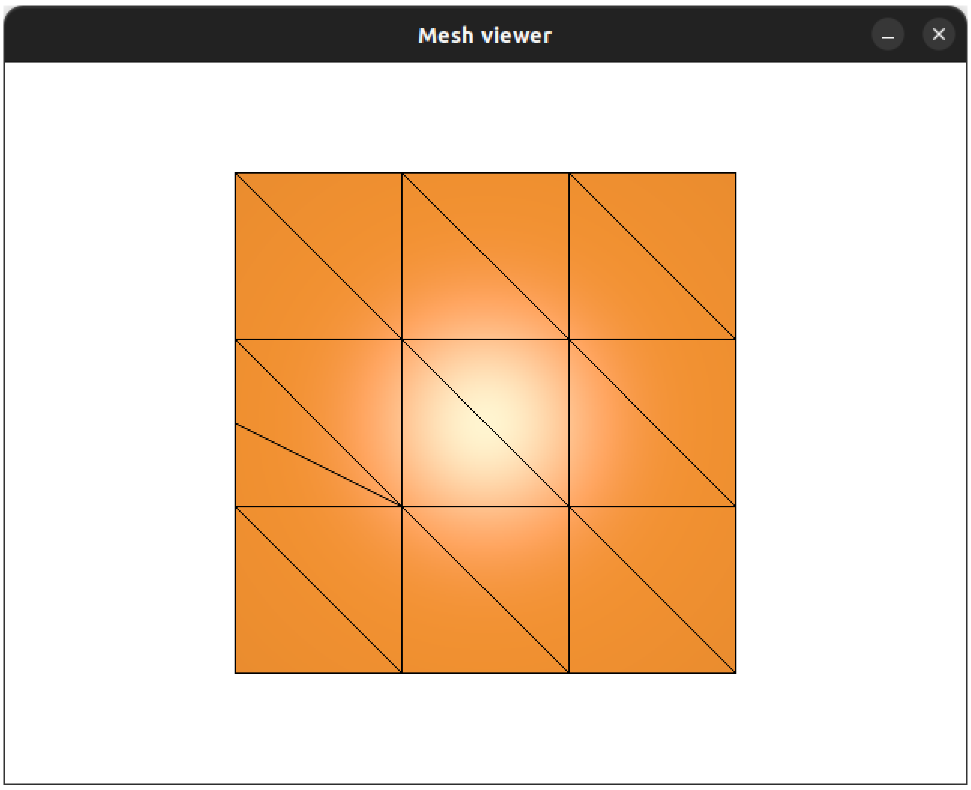


Figure 9: Edge Split on a boundary edge.

4.3 Edge Collapse

Implementing edge collapse was significantly harder compared to edge flip or edge split. First, we implemented a function `collapse_check()` which checks for the legality of the operation. Topological and geometric constraints were taken from [2]. In particular, this involved checking a few conditions like:

- If (v_1, v_2) are boundary vertices then the given edge is also a boundary edge.
- There are exactly 2 common vertices or exactly 1 common vertex when the given edge is a non-boundary or boundary edge respectively.
- The mesh has more than 4 vertices.
- The edge collapse doesn't flip any triangles, or create any sliver triangles. The methods given in [3] were used to check this.

The implementation of edge collapse was tricky when considering meshes with boundaries, and involved careful handling of many edge cases.

Another problem encountered was that we need to decrease the number of half edges, vertices and triangles during an edge collapse. Naively deleting these positions from our arrays will create holes with undefined references. We solved this issue by taking the elements with highest index and swapping them in these holes, thereby maintaining the continuity of our data structure. The order of deletion should be kept from highest index to lowest index whenever we delete multiple elements.

The result of edge collapse on the 3×3 square is show below.

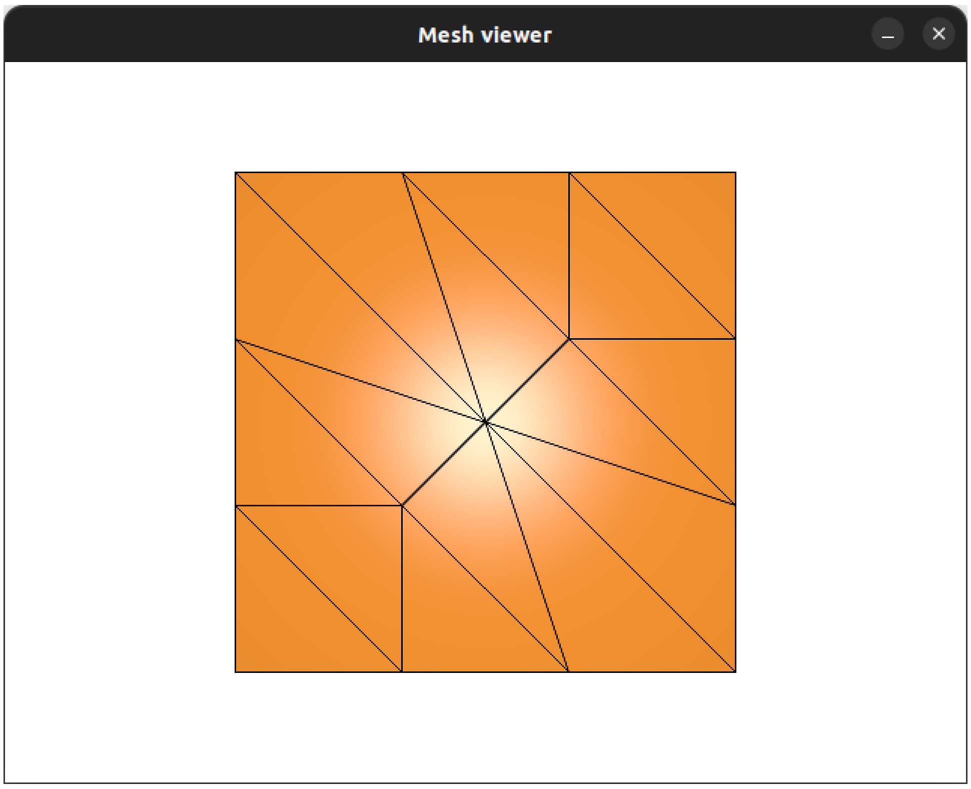


Figure 10: Edge Collapse on a interior edge.

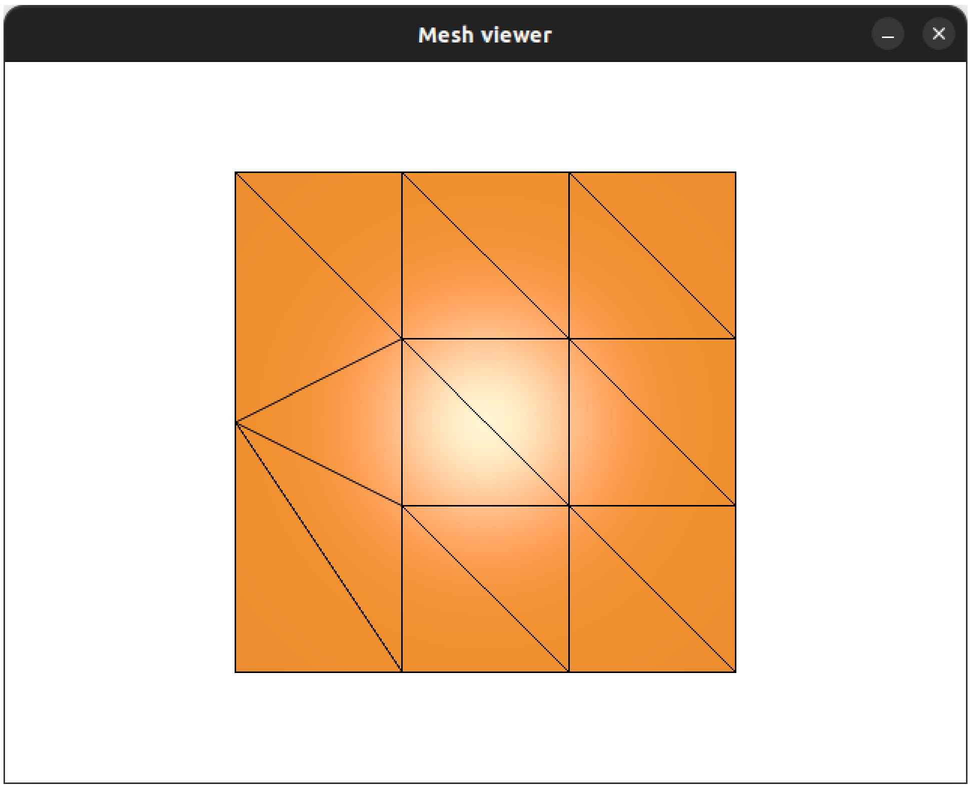


Figure 11: Edge Collapse on a boundary edge.

5 Invariants

We created a checking function check_invariants() which checked for various invariants. These were extremely useful while debugging and ensured that all our implementations were correct. Briefly, we list all the invariants below (vert is a vertex, he is a half edge and tri is a triangle):

1. Check that sizes of all arrays are correct.
2. Check that there are no out of bound indices in the arrays.
3. Check that vert.he.origin is vert
4. Check that tri.he.tri is tri
5. Check that tri has 3 unique vertices and all the edges between these vertices are contained in tri
6. Check that he and he.next have same tri
7. Check that he $\neq$ he.next
8. Check that he.target is same as he.next.origin
9. Check that he.pair.pair = he and he $\neq$ he.pair
10. Check correct winding order of triangles and he.target = he.pair.origin
11. Check that he.tri $\neq$ he.pair.tri

12. Check that each half edge is part of a closed loop and each interior half edge is a part of closed loop of size 3.
13. Check that neighbouring faces share 2 vertices.

6 Isometric Remeshing

The legality checking functions were now useful while determining if an edge can be flipped or collapsed. We implemented the tangential smoothening given at [4] using voronoi areas as the weights. The dampening parameter λ was set to be 0.1 and the desired length was set to be the average half edge length. We tested the isometric remeshing on bunny, teapot and sphere. The remeshing doesn't look perfect, but we can see that the triangles are approximately equally sized now. The results are given below.

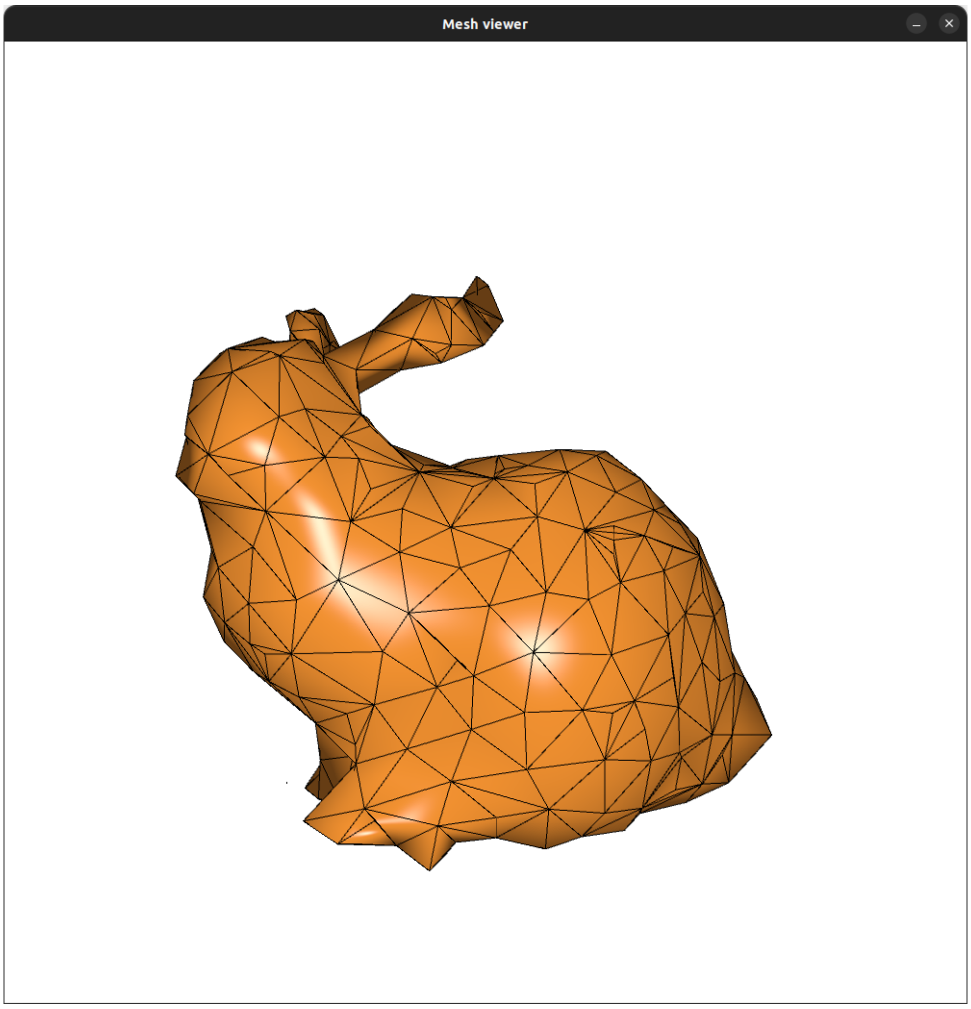


Figure 12: Isometric remeshing of bunny.

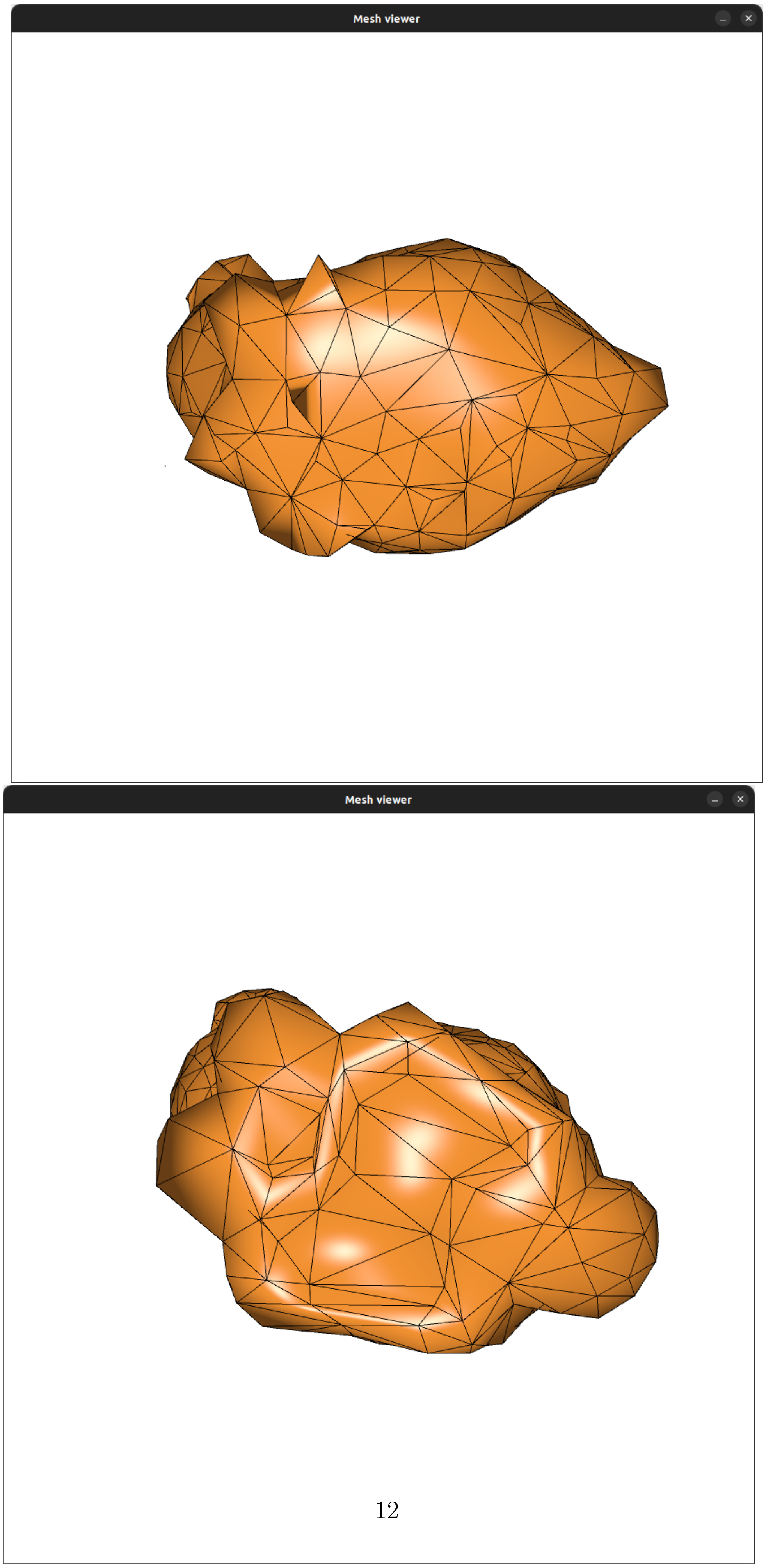


Figure 13: Isometric remeshing of bunny showing equal triangles as compared to original

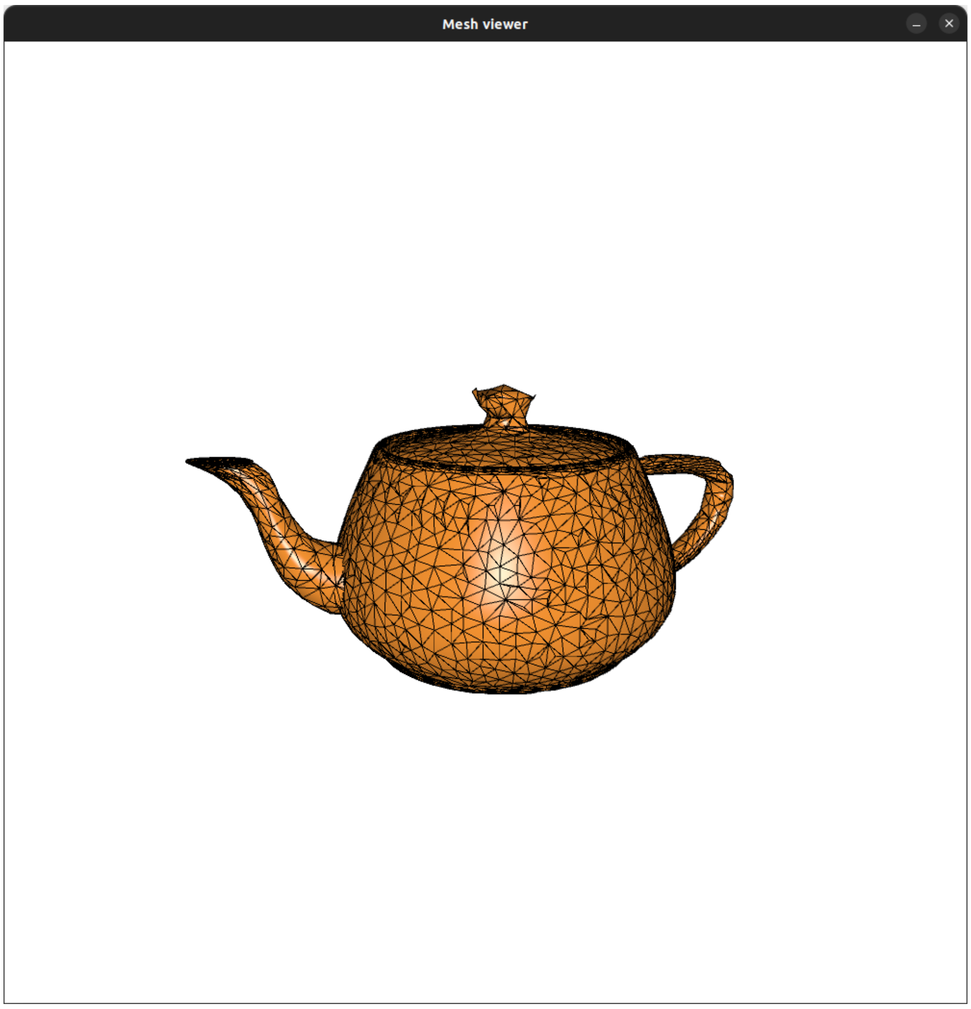
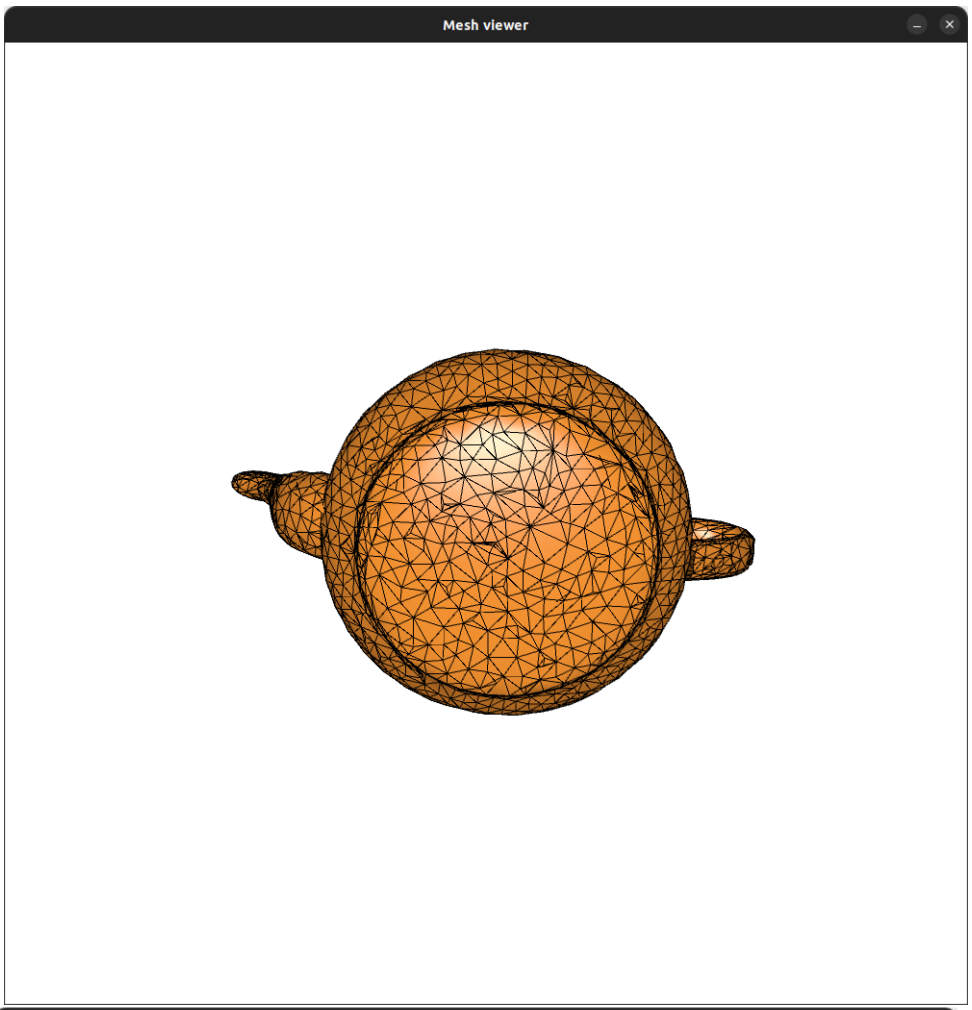


Figure 14: Isometric remeshing of teapot.



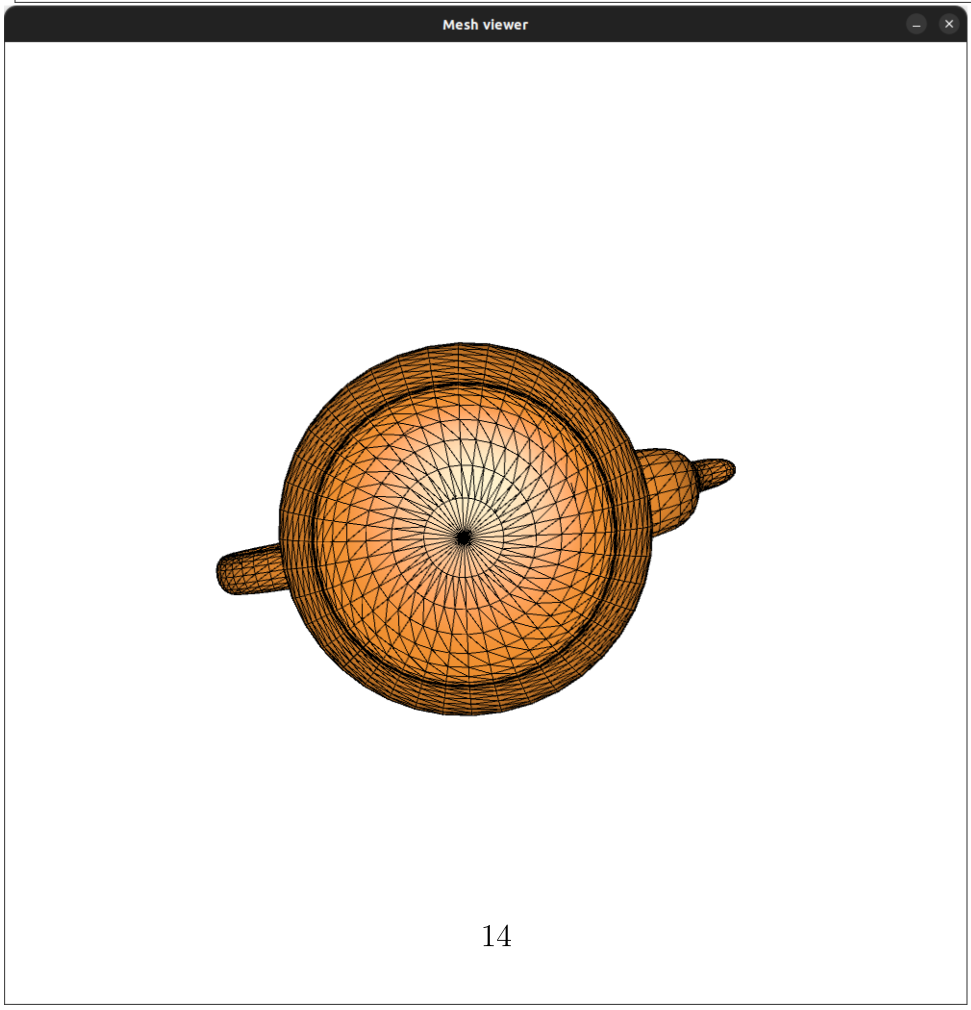


Figure 15: Isometric remeshing of teapot showing equal triangles as compared to original

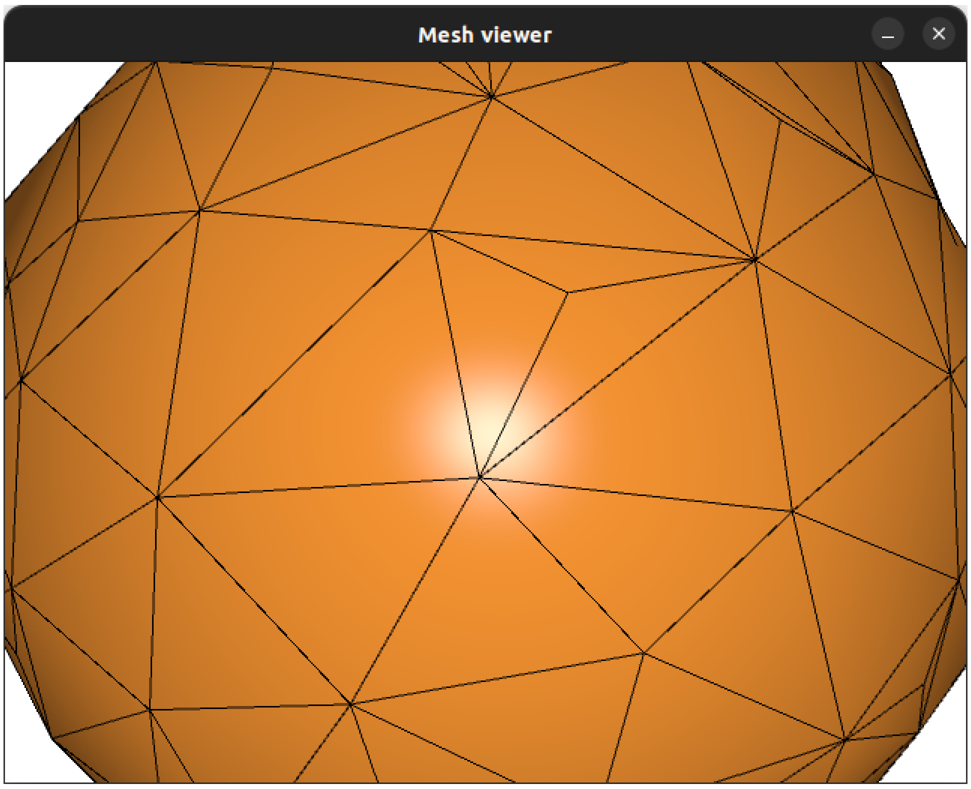


Figure 16: Isometric remeshing of sphere

We were thus able to implement **Every component of the assignment**.

References

[1] URL: https://escholarship.org/uc/item/7657d8h3.

[2] URL: //%20https://www.merlin.uzh.ch/contributionDocument/download/14550.

[3] URL: https://mgarland.org/files/papers/thesis-onscreen.pdf.

[4] URL: https://www.graphics.rwth-aachen.de/publication/03101/.