CG_5_ARAP Parameterzation Report

PB21010479 Caoliwen Wang

April 2, 2024

1 Problem Statement

In this experiment, we aim to achieve parameterization with no fixed boundaries. In the previous experiment, we have already implemented the mapping of a grid to a given boundary, where the generation of the entire grid involves solving a sparse system of equations. Now, we need to consider how to minimize the deformation of three-dimensional triangles, i.e., minimizing an energy function that measures distortion. The work being replicated in this paper is Ligang Liu's work from 2008. Additionally, we need to continue learning about the usage of the Eigen library, establishing and solving sparse matrices, SVD decomposition, and node programming.

2 Propose Algorithms

Firstly, we need to consider the general principles of the algorithm for non-fixed boundary parameterization, which involves minimizing the following energy function:

$$E(u, L) = \frac{1}{2}\sum_{t=1}^{T}\sum_{i=0}^{2}\cot(\theta_t^i)||u_t^i - u_t^{i+1} - L_t(x_t^i - x_t^{i+1})||^2. \tag{2.1}$$

In this expression, T represents the set of all triangles in the mesh, i iterates over each vertex, $\cot(\theta_t^i)$ measures the weight of the area, u_t^i represents the 2D coordinates of the i-th vertex of the t-th triangle (chosen as a plane in the initial parameterization), L_t represents the distortion of the t-th triangle (computed by transforming the 3D triangle into an equivalent 2D triangle and calculating the Jacobian matrix), and x_t^i represents the 2D coordinates obtained after mapping the 3D triangle in a distortion-minimizing manner.

Therefore, the essence of non-fixed boundary parameterization is to find suitable (u, L) that satisfy equation 2.2 as much as possible.

$$(u, L) = argmin_{(u,L)}E(u, L). \quad L_t \in M. \tag{2.2}$$

Where M is the desired linear transformation.

In this paper, we have reproduced the ARAP (As-Rigid-As-Possible) algorithm, which requires the matrix M to satisfy the following requirements:

$$M = \left\{ \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} : \theta \in [0, 2\pi) \right\}$$

In fact, this is a non-linear optimization problem, so it cannot be directly solved by taking partial derivatives. Therefore, it requires the introduction of local/global methods. The general steps are as follows.

1. Implement an initial parameterization: This step can utilize any parameterization result obtained from your work in Assignment 4.
2. Implement the local iteration step of ARAP (Local Phase): Independently execute on each triangle t, relatively straightforward. Implement a second-order matrix SVD decomposition. Fix parameter coordinates, compute the local orthogonal approximation of the current parameterization Jacobian.
3. Implement the global iteration step of ARAP (Global Phase): Fix on each triangle t, update parameter coordinates, solve a global sparse linear system of equations. The coefficients of the equation system remain fixed and only need to be set once and pre-decomposed.
4. Iterate several times and observe the results.

To implement Step 1, I added a new interface in the interactive interface, allowing the input of the initialized geometry, which can be connected to the boundary mapping results obtained in Assignment 4. It is shown in Figure 1.

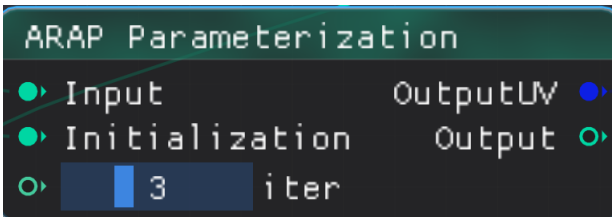


Figure 1: Node ARAP

"Input" refers to the initial geometry passed in, while "Initialization" refers to the initialized geometry, which can include the boundary mapping result from the fourth assignment. "iter" can be used to set the number of iterations. "Output" refers to the geometry after flattening.

Afterwards, it is necessary to emphasize the process of flattening, and we assume that the result of the flattening is as follows:

$$x_1(0,0,0),\ x_2(|v_1v_2|,0,0),\ x_3(|v_1v_3|\cos\theta,|v_1v_3|\sin\theta,0).$$

Here, θ represents the angle at vertex 1, and the part related to v represents the triangular mesh faces of the 3D triangle.

For the "Local" stage, keeping the initialization fixed (i.e., halfedge_mesh_ini), for each triangular mesh t, we solve for the L_t that satisfies the conditions. Here, we use the following SVD decomposition:

$$S_t(u) = \sum_{i=0}^{2} \cot\theta_t^i (u_t^i - u_t^{i+1})(x_t^i - x_t^{i+1})^T = USV^T. \tag{2.3}$$

$$L_t = UV^T. \tag{2.4}$$

L_t represents the optimal approximation rotation matrix that implies deformation.

For the "Global" stage, we fix L_t and solve for the optimized mesh coordinates $\boldsymbol{u}$, which essentially involves solving a sparse matrix.

$$\sum_{j \in N(i)} (\cot \theta_{ij} + \cot \theta_{ji})(u_i - u_j) = \sum_{j \in N(i)} (\cot \theta_{ij} L_{t(i,j)} + \cot \theta_{ji} L_{t(j,i)})(x_i - x_j) \ \forall i = 1, \cdots, n. \tag{2.5}$$

Where θ_{ij} represents the dihedral angle between half-edge ij in the mesh, which can be obtained through some operations on the half-edge data structure. This way, we can obtain the new optimized mesh. Then, we utilize the custom function "Update_" to perform updates, and proceed to the "Local" stage for iteration until reaching the specified number of steps.

3 Programming

3.1 Halfedge Structure

In this task, the representation of the half-edge structure mainly involves traversing each face. We adopt the following approach:

```
for (const auto& face_handle : halfedge_mesh->faces())
```

Other processing remains unchanged compared to before.

3.2 Node programming

Regarding the nodes needed for this task, they are depicted in Figure 1. The connections for "OutputUV" are shown in Figure 2, while the connections for "Output" are illustrated in Figure 3.

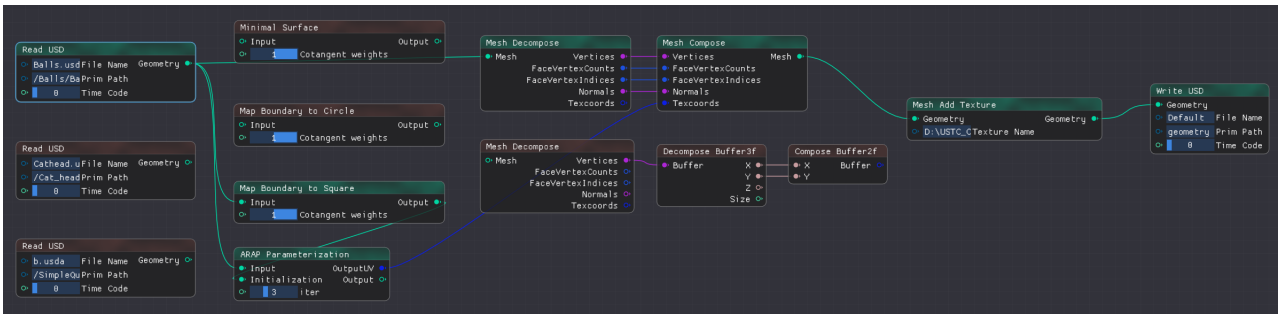


Figure 2: connection of "OutputUV"

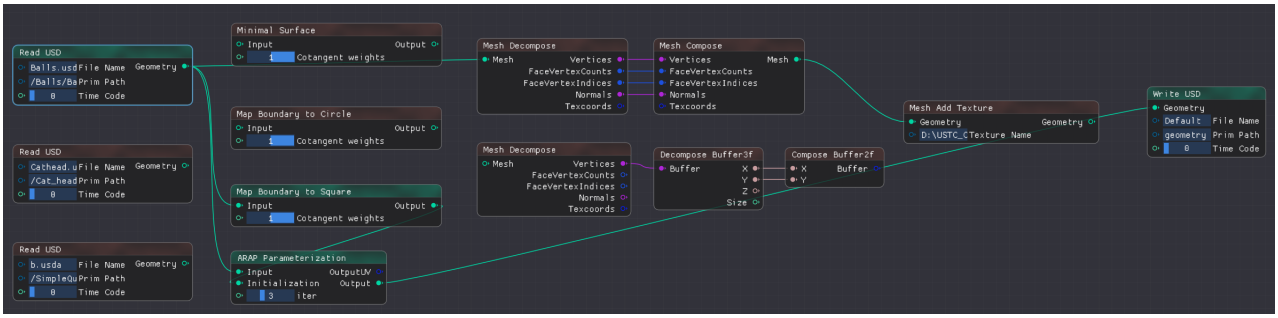


Figure 3: connection of "Output"

3.3 Code Optimization

In this project, we have performed the following code optimizations.

Firstly, for the iteration process, we only utilized the following code. Doing so involves calling functions, which makes the code relatively concise.

```
for (int i = 0; i < iter; i++)
{
    auto L = Local(X, Theta, halfedge_mesh_ini);
    Update_(L, X, halfedge_mesh, A, halfedge_mesh_ini);
}
```

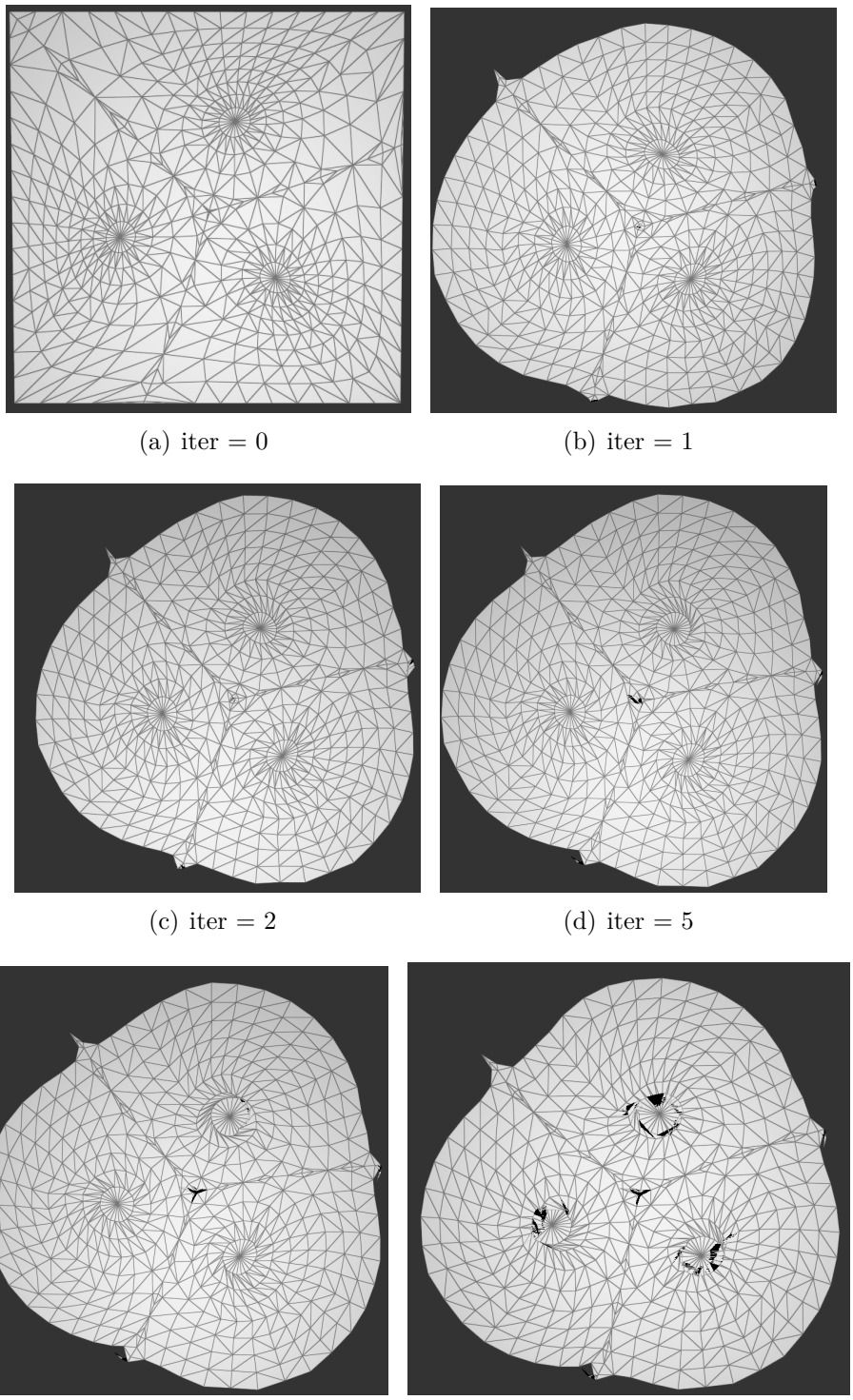
We noticed that the coefficient matrix of the sparse matrix only depends on the properties of the mesh itself. Therefore, we can calculate it beforehand and then pass it as a parameter.

4 Experimental Results

I think the results of this algorithm are fantastic.

First, we use the balls to test. The 2D geometry results are in Figure 4. The u-v coordinates results are in Figure 5.

We can observe that increasing the number of iterations does not necessarily lead to better results.



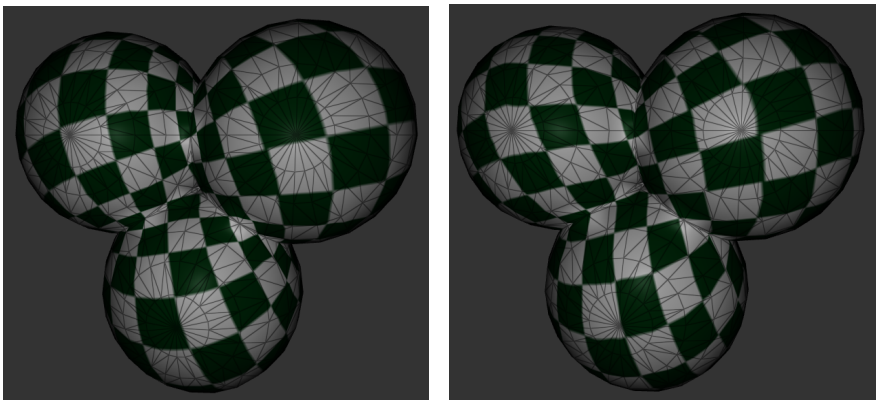
(a) iter = 0 (b) iter = 1

(c) iter = 2 (d) iter = 5

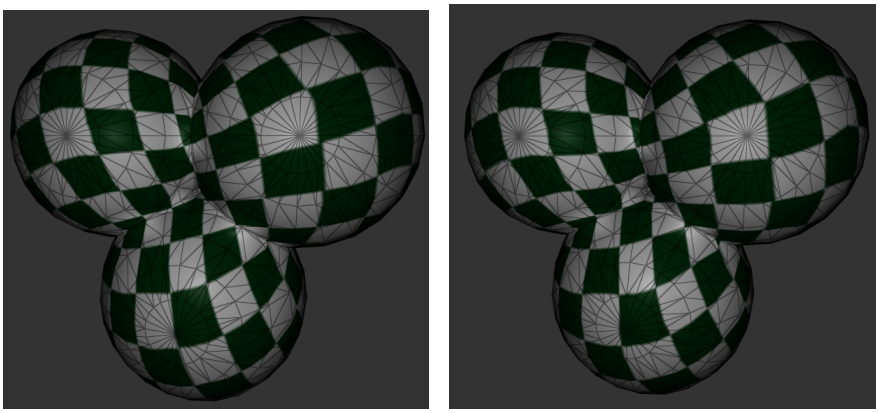
(e) iter = 10 (f) iter = 50

Figure 4: Balls: 2D Geometry

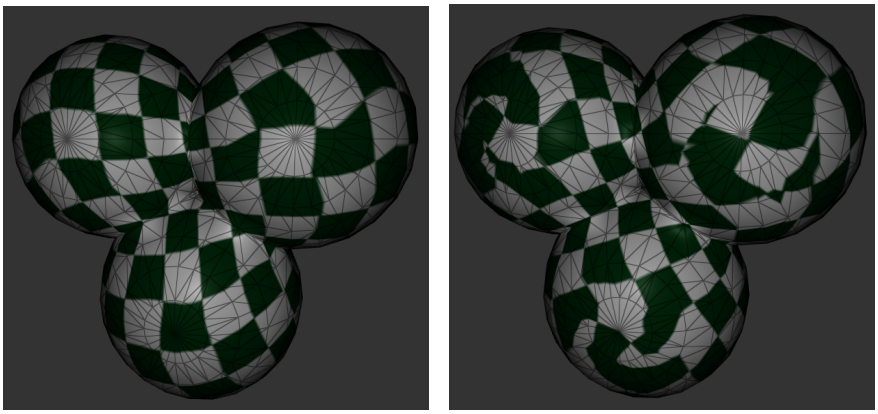
We also use cow to test, a very classical figure in this paper. The 2D geometry results are in Figure 6. The u-v coordinates results are in Figure 7.



(a) iter = 0 (b) iter = 1



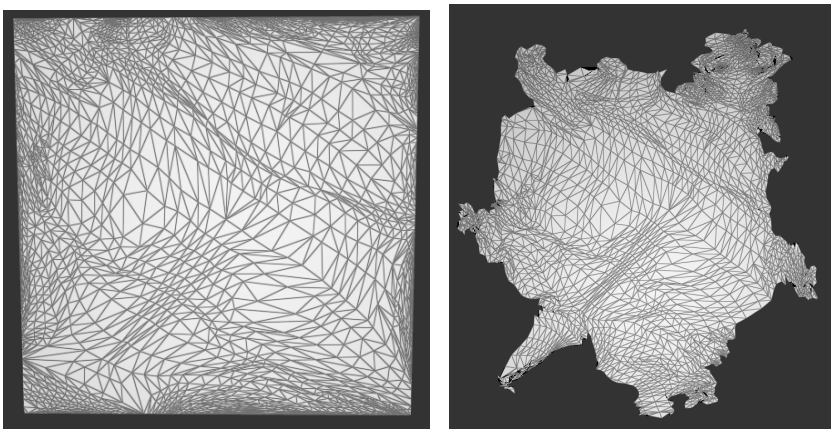
(c) iter = 2 (d) iter = 5



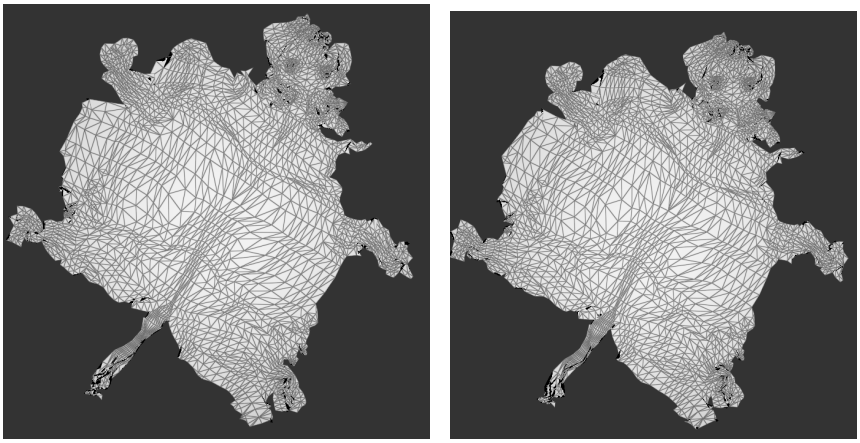
(e) iter = 10 (f) iter = 50

Figure 5: Balls: with texture

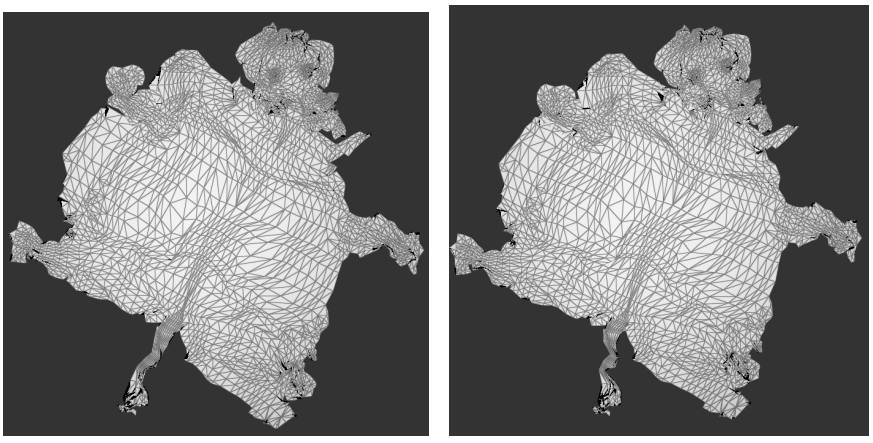
We can observe that convergence is relatively fast, and conducting additional iterations may not significantly alter the outcome.



(a) iter = 0 (b) iter = 1



(c) iter = 10 (d) iter = 30



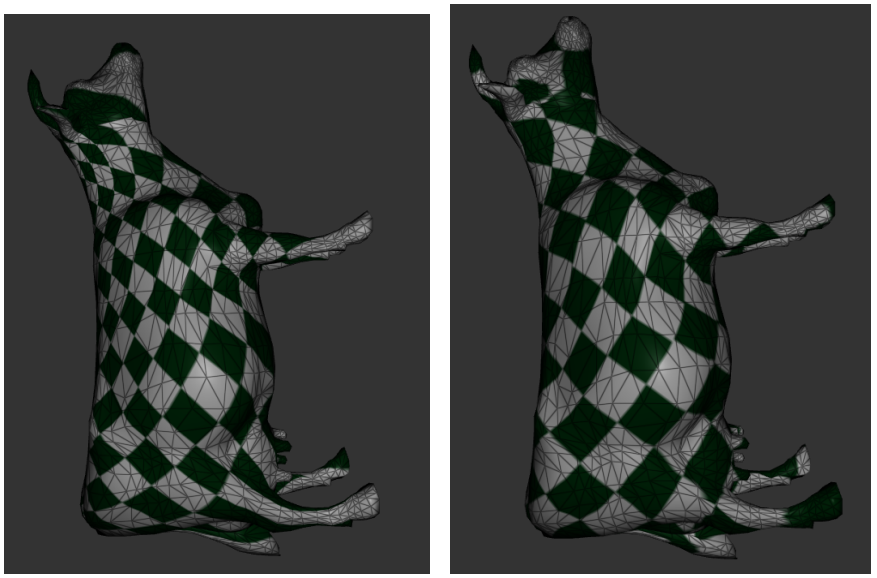
(e) iter = 60 (f) iter = 100

Figure 6: Cow: 2D Geometry

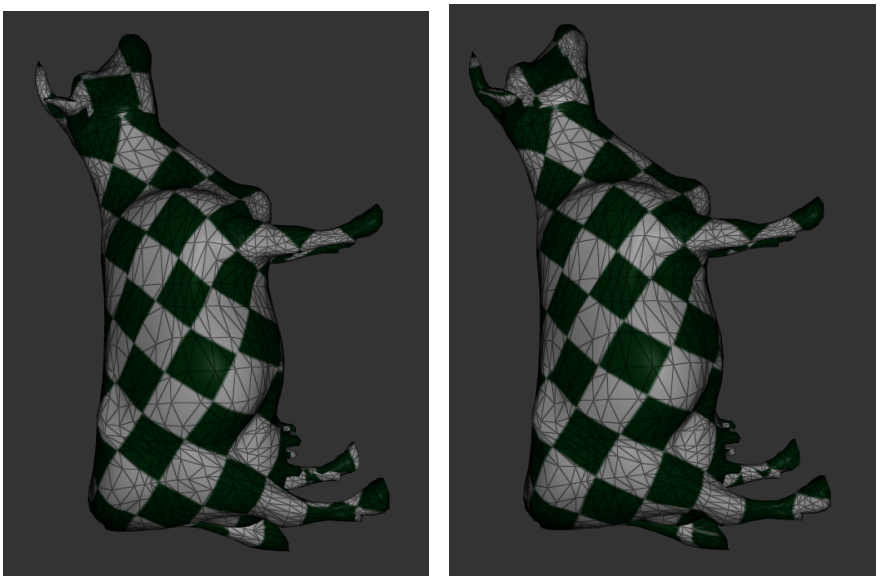
5 Summary

In this assignment, we successfully implemented 2D mesh deformation with non-fixed boundaries using the As-Rigid-As-Possible (ARAP) algorithm. This algorithm is indeed exquisite

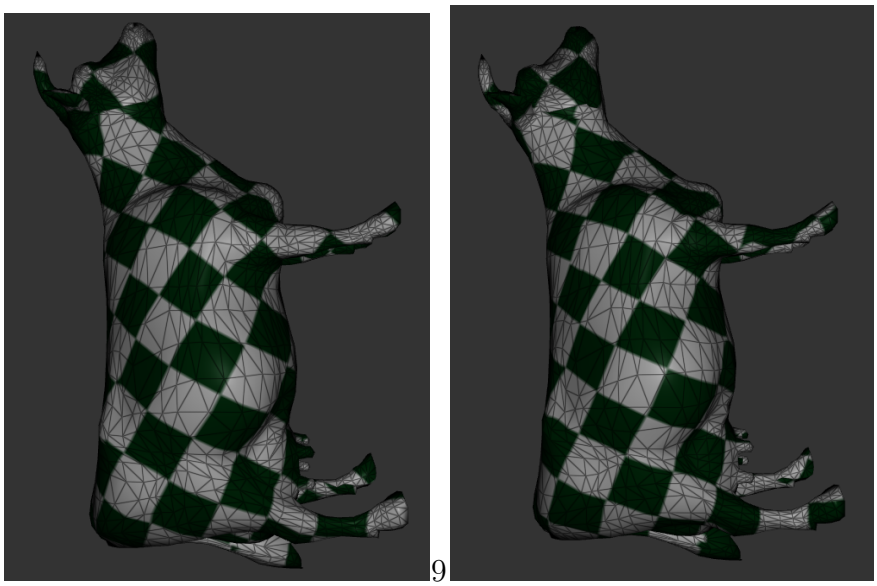
as it strives to preserve the geometric shape of the object as much as possible. I also spent a considerable amount of time refining the code and managed to achieve relatively good results. However, I encountered some flipping and overlapping issues with a few small triangles at the boundary. Unfortunately, I ran out of time to explore and implement alternative algorithms.



(a) iter = 0 (b) iter = 1



(c) iter = 10 (d) iter = 30



(e) iter = 60 (f) iter = 100

Figure 7: Cow: with texture