Interactive ARAP - Final Report

Ömer Köse
TUM
go56tof@mytum.de

Natalie Adam
TUM
natalie.adam@tum.de

Timur Krüger
TUM
timur.krueger@tum.de

Kilian Peis
TUM
kilian.peis@tum.de

February 12, 2024

Abstract

In the university project as-rigid-as-possible (ARAP) deformation [13], we implement an interactive ARAP framework following the guidelines by Sorkine & Alexa [11]. Users are able to load a mesh into a visualizer and interactively manipulate the mesh by setting deformation constraints and moving vertices. The ARAP algorithm performs the deformations with respect to shearing, scaling, energy preservation and presents the updated mesh in real-time for the users.

1 Motivation

The ARAP framework is a tool for interactive shape modelling. By setting deformation constraints, the user can interact with meshes. During the deformation, the mesh needs to keep its shape. That means it should only be rotated or translated. The ARAP implementation takes shearing and scaling effects into account to avoid destroying the shape of meshes. For optimal deformation results, the ARAP algorithm tries to keep the surface transformation in each cell as rigid as possible to avoid stretching or shearing of the mesh [6, 7, 11].

2 Related Work

Surface deformation is the process of altering the shape or structure of a surface or an object. This can be done by stretching, bending, or twisting the surface. In computer graphics, surface deformation is a technique used to animate characters, objects, or environments by adjusting their shapes to simulate movement and interactions.

Prior to ARAP, Laplacian Surface Editing was an alternative that was used for surface editing. In this method, researchers suggested using the Laplacian operator to calculate a smooth version of the surface. By manipulating the Laplacian coordinates of the vertices, the deformations can be performed on the surface while minimizing distortion [4].

In our examination of the existing literature on ARAP surface modeling, we also analyzed the work of Igarashi et al. [7]. They implemented an ARAP framework that handles the deformation of 2D meshes. Their ARAP model allows for user-driven deformations that maintain local rigidity.

3 Method

For our implementation, we primarily followed the ARAP framework introduced by Sorkine & Alexa [11] and divided our project into four main steps:

- Preprocessing: Given a mesh, identify its vertices, edges and faces. Then, determine the neighbourhood for each vertex.
- Local Step: Compute the optimal local rotation for each vertex using SVD to find the rotation that minimizes the local rigidity energy.
- Global Step: Update the position of all vertices,

minimizing the global energy function by solving a linear system.

- Iterative Process: Repeat the last two steps iteratively until the algorithm reaches convergence or the rigidity energy falls below a threshold.

Our visualizer uses the libigl library [8] for geometry processing which provides functions for loading mesh files into Eigen types and viewing meshes with OpenGL [10] and GLSL [9]. Further, the ImGui library [3] is responsible for user interaction. The ImGuiFileDialog library [1] enables the user to load and replace meshes in the visualizer.

We take a mesh as input. The used meshes stem from the repository of Alec Jacobson [2] and the Stanford 3D Scanning Repository [12], which contain common 3D test models.

Eigen library [5] is used to calculate the ARAP deformations. First, we compute its neighbor matrix $N(i)$, which consists of the vertices connected to the vertex v_i.

$$w_{ij} = \frac{1}{2} * (\cot \alpha_{ij} + \cot \beta_{ij}) \tag{1}$$

The weight matrix ensures that our implementation is mesh-independent and can handle non-uniformly shaped cells [11]. We use the cotangent weight formula w_{ij} to define the weights per edge where α_{ij} and β_{ij} are the angles opposite of the edge (i, j). Then, we set the diagonal of the weight matrix to $w_i = 1$.

$$S_i = P_i D_i P_i'^T \tag{2}$$

We utilize the covariance matrix S_i to compute the rotation matrix R_i. P_i is a $3 \times |N(v_i)|$ matrix containing the edges $e_{ij} = p_i - p_j$ with $p_i \in \mathbb{R}^3$ as the position of the vertex v_i. Similarly, P_i' defines a $3 \times |N(v_i)|$ matrix containing the edges $e'_{ij} = p'_i - p'_j$ of the deformed vertex v'_i. D_i is the diagonal matrix containing the weights w_{ij}.

$$R_i = V_i I U_i^T \tag{3}$$

To obtain V_i and U_i, we perform SVD on the covariance matrix: $S_i = U_i \sum V_i^T$. Multiplying with the identity matrix I ensures the correct computation of the rotations.

$$E(S') = \sum_{i=1}^{n} w_i \sum_{j \in N(i)} w_{ij} \|(p'_i - p'_j) - R_i(p_i - p_j)\|^2 \tag{4}$$

The rigidity energy sums up the deviations from the rigidity per cell where S' is a deformed triangle mesh.

$$Lp' = b \tag{5}$$

The core of the ARAP implementation is the computation of the deformation by iteratively solving the system of equations.

The Laplacian matrix L represents the system matrix, while p' is the position of the deformed vertices. b is a n-vector containing the expression:

$$b_i = \sum_{j \in N(i)} \frac{w_{ij}}{2} (R_i + R_j)(p_i + p_j) \tag{6}$$

4 Results

In this section, we present the outcomes of our project, focusing on the deformation capabilities of our ARAP framework and the visual fidelity of our mesh deformations.

We rigorously tested our ARAP framework by applying it to several meshes, including complex geometries. Through user-driven constraints, we achieved various deformations. Figure 1 and 2 visualize the ARAP algorithm performed on the cactus mesh.

We decided to use a maximum of 10 iterations considering the efficiency tradeoff between performance and stability. The convergence of the rigidity energy can be seen in Figure 4.

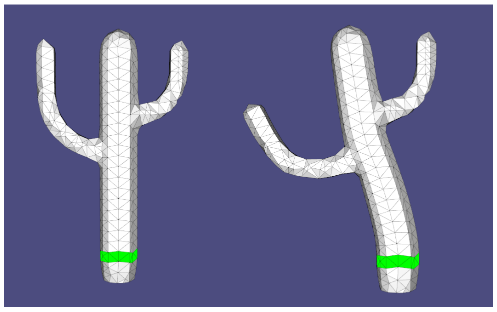


Figure 1: Initial cactus with one anchor point on the left side and cactus after 10 iteration steps on the right side.

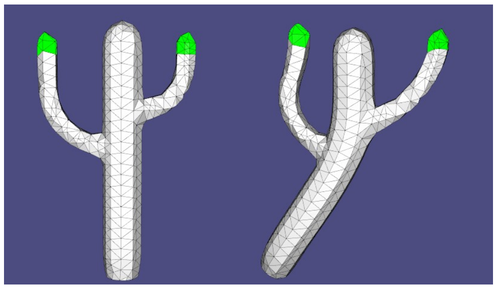
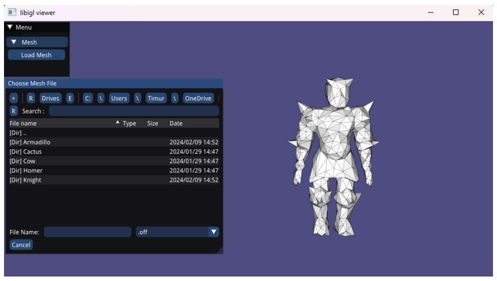


Figure 2: Initial cactus with two anchor points on the left side and cactus after 10 iteration steps on the right side.



```
File name                Type  Size  Date
[Dir] ..
[Dir] Armadillo                      2024/02/09 14:52
[Dir] Cactus                         2024/01/29 14:47
[Dir] Cow                            2024/01/29 14:47
[Dir] Homer                          2024/01/29 14:47
[Dir] Knight                         2024/02/09 14:52
```

Figure 3: Libigl visualizer with the ImGui menu and the ImGuiFileDialog on the left side and a loaded Armadillo mesh on the right side.

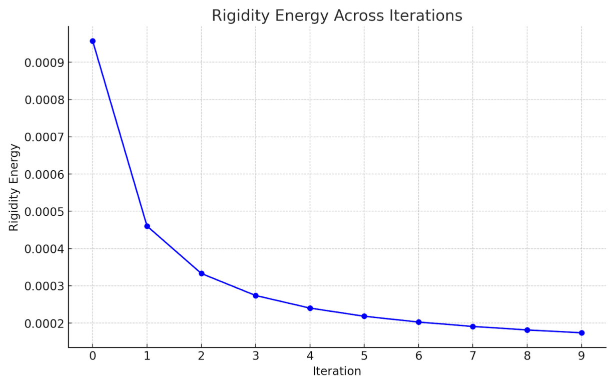


Figure 4: Example graph of convergence in rigidity energy for various meshes across iteration.

Mesh	Number of Vertices	Time (ms)
Armadillo_250	202	4.854
Armadillo_500	252	6.371
Armadillo_1k	864	23.122
Armadillo_2k	1002	26.408
Armadillo_5k	2502	73.776
Armadillo_10k	5002	155.843
Cactus	620	15.456
Cow	2903	79.053

Table 1: Performance analysis of multiple meshes for 100 iterations.

5 Analysis

In our implementation, we used sparse matrices as our matrix representations. This was achievable because the weight and system matrices have values only on the diagonals and the neighbors of each vertex. By taking advantage of this structural information, we constructed and manipulated these matrices, which allowed us to avoid unnecessary row operations. Additionally, since only the non-zero values are stored, these matrices are highly memory efficient.

According to the paper by Sorkine & Alexa [11], the method aims to minimize the local rigidity energy which serves as the loss function for the optimization process. The convergence of this function determines whether the method reaches a solution or not. In our experiments, we observed that the rigidity energy always converges, even when there are large deformations. It behaves as intended, starting low for small deformations and converging quickly, but starting high for larger deformations and requiring more iterations to converge. As stated in the paper by Sorkine & Alexa [11], the local rigidity energy is the function that the method tries to minimize. It can be seen as the loss function of the optimization. Therefore, it is the identifier that whether the method converges to a solution or not. In our experiments, we have seen that the rigidity energy always converges even under large deformations. Its behaviour also works as intended. For small deformations, it starts low and converges much faster, for the larger deformations it starts high and needs more iterations to converge.

As our solver, we use SparseLU from Eigen Library [5]. We were planning to use Sparse Cholesky solver as the system matrix is the Cotangent Laplacian Matrix which is positive semi-definite and as a common practice it can be made positive-definite with Tikhonov regularization. Due to its structure, Cholesky factorization is suitable for it and it is very efficient. However, in our implementation even though our Cotangent Laplacian is correct, we tested it by comparing against libigl's Cotangent Laplacian [8] in terms of Frobenius norm, the deformations we get with Cholesky was incorrect.

The ARAP Method is based on iterative optimization. It has two parameters that determine the number of iterations. The first parameter is the maximum number of iterations while the second parameter is a threshold value used to compare the rigidity energies between two iterations. If the difference between the rigidity energies is less than the threshold we have set, the optimization will stop early, giving us a performance boost, especially for small deformations. We have chosen 10^{-6} as the threshold in our implementation.

The number of vertices impacts the size of the system and weight matrix, affecting system solving performance, as depicted in Table 1. Aside from the number of vertices, the number of faces in a mesh is also a significant factor in affecting performance. This is because constructing the neighborhoods and weight matrix involves iterating over the faces. Due to the local structure of the weight matrix, denser neighborhoods in a mesh tend to take more time to set up the matrix.

6 Conclusion

Overall, our results demonstrate the efficacy of our ARAP framework in providing high-quality, interactive mesh deformations. The framework is capable of maintaining mesh integrity while allowing stable deformations for a large number of meshes. For some meshes with over 10.000 vertices the framework still creates artifacts.

6.1 Future Work

- Achieving more stable deformation for large meshes
- Parallelizing the ARAP Solver
- Trying out different energy functions

References

[1] AieKick and contributors. Imgui file dialog: A file dialog for dear imgui. https://github.com/aiekick/ImGuiFileDialog, 2022. 2

[2] Alec Jacobson. Common 3D Test Models, 2023. https://github.com/alecjacobson/common-3d-test-models/. 2

[3] Omar Cornut and contributors. Dear imgui: Bloat-free immediate mode graphical user interface for c++ with minimal dependencies. https://github.com/ocornut/imgui, 2022. 2

[4] Olga Diamanti, Alec Jacobson, Maks Ovsjanikov, Leonidas Guibas, and Olga Sorkine-Hornung. Laplacian mesh editing. *ACM Transactions on Graphics (TOG)*, 32(4):91:1–91:11, 2013. 1

[5] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010. 2, 4

[6] Takeo Igarashi and Yuki Igarashi. Implementing as-rigid-as-possible shape manipulation and surface

flattening. *journal of graphics, gpu, and game tools*, 14(1):17–30, 2009. 1

[7] Takeo Igarashi, Tomer Moscovich, and John F Hughes. As-rigid-as-possible shape manipulation. *ACM transactions on Graphics (TOG)*, 24(3):1134–1141, 2005. 1

[8] Alec Jacobson, Daniele Panozzo, et al. libigl: A simple C++ geometry processing library, 2018. https://libigl.github.io/. 2, 4

[9] Khronos Group. Opengl shading language specification, 2017. 2

[10] Khronos Group. Opengl specification, 2017. https://www.opengl.org/documentation/. 2

[11] Olga Sorkine and Marc Alexa. As-rigid-as-possible surface modeling. In *Symposium on Geometry processing*, volume 4, pages 109–116. Citeseer, 2007. 1, 2, 3, 4

[12] Stanford Computer Graphics Laboratory. Stanford 3D Scanning Repository, 2023. https://graphics.stanford.edu/data/3Dscanrep/. 2

[13] Ömer Köse, Natalie Adam, Timur Krüger, and Kilian Peis. 3ds-mc: Interactive arap. https://github.com/TimurKrueger/3DS-MC, 2023. 1