3D Scanning and Motion Capture Project Report
As rigid as possible (ARAP)

Anas Shahzad - 03737750, Batuhan Erden - 03738750, Cansu Yıldırım - 03740404, Alexander Epple - 03659403

1. Introduction

As-rigid-as-possible surface deformation (ARAP) is an iterative mesh processing scheme in which the shape is stretched or sheared while the small parts of the object are preserved locally, i.e., small parts are modified to be as rigid as possible. In our work, we implement this ARAP deformation algorithm by referring to the work of Sorkine et al. Our motivation for choosing this topic was that it is interactive and independent of additional hardware. In the remainder of the paper, we review related work, the methods used for the algorithm, our results, and the conclusion, which includes the challenges we encountered and our future work.

2. Related work

Interactive shape modeling is a task where the shape must retain its local structure when sheared or stretched to conform to specific conditions. If we simply select a desired point in a 3D object and then drag it to the desired position, we get a spiky surface with little or no local surface information. Therefore, 3D deformation is much more difficult due to the nonlinearity of similarity transformations.

As-rigid-as-possible implementation is the pioneering work in the field of non-rigid deformation [3]. Prior to the ARAP implementation, **Mesh Editing with Poisson-Based Gradient Field Manipulation** by Yu et al. [4] was an informative standard in the field of mesh deformation. In this method, the authors used a poison equation and applied it to the mesh by estimating each vertex as a set of boundary conditions. The boundary conditions consist of F, a set of local frames defining the local orientations of the vertices, S, a set of scaling factors associated with the vertices, and R, a force field. However, this method introduces detailed distortion and a lack of smoothness near the boundary region.

Cage based deformation transfer by Chen et al. [2] uses a cage proxy where the vertices are a linear combination of shapes to calculate the deformation of the model. Once the weights and cages are known, we can move the object simply by manipulating the cage vertices. Due to the much smaller number of vertices, the armadillo model can be deformed in about 56 milliseconds using this method. However, a major limitation of the model is that it depends largely on the shape and tessellation of the cage.

Another method that uses a proxy to estimate deformation much faster is **Skeleton Based As-Rigid-As-Possible Volume Modeling**. In this method, the skeleton is embedded in the mesh and each vertex is calculated as a linear combination of bone vertices. The method results in a much faster implementation than ARAP, but at the cost of a higher error rate (0.126). The method is also very unstable for configurations that intersect each other.

A much more recent method is that of **Embedded Deformation for Shape Manipulation** [3]. In this method, the optimization of the transformation is performed on a deformation graph where each vertex is defined by a set of affine transformations. The method uses Guass-Newton iterations to compute the deformed vertices in a time as short as 41 milliseconds.

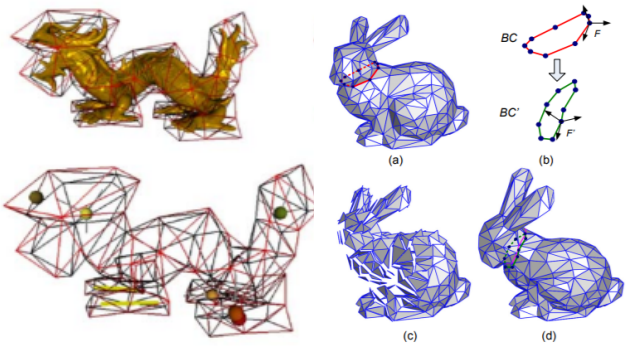

Figure 1. Cage-ARAP and Poison method

3. Method

3.1. Mathematical Overview

A transformation is said to be rigid if there is a rotation R_i that rotates the vertex to its new location such that

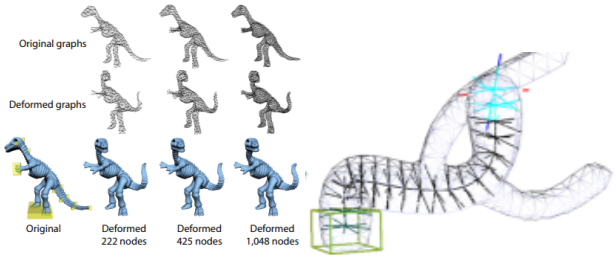

Figure 2. skeleton Arap and Embedded Graph

$$p'_i - p'_j = R_i(p_i - p_j) \quad (1)$$

If there is no such rotation, we try to find the one that minimizes the energy function:

$$E(C_i, C'_i) = \sum_{j \in N(i)} w_{ij} \left\| (p'_i - p'_j) - R_i(p_i - p_j) \right\|^2 \quad (2)$$

where p and p' are the original and deformed vertex positions, R is the rotation, and w_i & w_{ij} are the weights. We use this energy function to measure the stiffness of deformations. After calculating the gradients for both sides of the equation and assuming that $w_{ij} = w_{ji}$, we get this simple formula:

$$\sum_{j \in N(i)} w_{ij}(p'_i - p'_j) = \sum_{j \in N(i)} \frac{w_{ij}}{2}(p_i - p_j)(R_i + Rj) \quad (3)$$

The above equation can then be simplified to $Lp' = b$, where p' is the position of the deformed vertices and b is the expression of the right-hand side of the equation. The matrix B is the discrete Laplace-Beltrami operator with $L \in \mathbb{R}^{V*V}$. It is defined as follows:

$$L_ij = \begin{cases} -w_{ij}, & \text{if is an edge} \\ \sum_{k \in nei(i)} w_{ik}, & \text{if } i == j \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

The choice of these weights is also crucial for the energy function. The weight matrix must be symmetric so that it can be solved efficiently with a simple Cholesky solver. In our experiments, we consider two different weights: the constant weights and the cotangent weights. For the constant initialization, we set each w_{ij} value to 1. For cotangent weights, we compute only the weights for the neighbors of each vertex, while the other positions are set to zero. The formula for calculating the cotangent weights is shown below:

$$w_{ij} = \frac{1}{2}(\cot \alpha_{ij} + \cot \beta_{ij}) \quad (5)$$

Note that instead of $R = V * U.Transpose()$, we have changed the equation to $R = U * V.transpose()$. This ensures that the rotations are calculated correctly.

3.2. Implementation Details

3.2.1 Libraries

First, we used simple OpenGL to develop our graphical user interface from scratch. We used the sphere tracing algorithm to find the closest point when clicking on the screen. We then implemented the shaders and lighting options for the GUI. However, for performance reasons, we moved our ARAP class to the libigl framework, which performs the same tasks with little overhead.

To ensure that we can compute the rotation matrices more efficiently, we modified the equations a bit to ensure that the determinant remains positive.

In the Libigl version of our ARAP implementation, we provide several configurations to run our ARAP project. The user can choose between constant weights and cotangent weights. He can also choose to start the optimization with naive linear minimization. We use the Eigen library for the linear operations in the ARAP algorithm. In addition, OpenMP is used for parallelization. We run our model on an Intel(R) Core(TM) i5-8279U CPU @ 2.40GHz (4c / 8t) 16 GB machine.

Since there are no mutual dependencies in the computation of the rotation, weight, and Laplace-Beltrami operator matrices, we parallelized our implementation using OpenMP. We compute the rotation matrices in parallel by creating a thread for each loop. We also use BLAS routines to increase the speed of the linear solver. More information can be found in our Github repository. [1].

4. Results

Model	Runtime in sec. without OpenMP	Runtime in sec. with OpenMP
Armadillo 250	0.25	0.16
Armadillo 500	0.33	0.19
Armadillo 1k	1.38	0.93
Armadillo 2k	1.89	1.08
Armadillo 5k	5.53	4.63
Armadillo 10k	13.47	11.86
Cactus	0.91	0.54
Cactus (High Res)	3.59	2.54
Dino	64.22	51.19

Table 1. Runtime with and without openmp (200 iterations)

-	SparseLU	SparseQR	BIGGSTAB
runtime	0.93	21.23	1.69

Table 2. Comparison between the different solvers (200 iterations)

-	Small Displacements	Medium Displacements	Larger Displacements
-	Hand is raised	Bended half way to the left	Bended all the way to the left
250	2.36	16.78	34.39
500	2.94	18.78	34.33
1000	2.34	14.64	31.62
2000	1.93	11.24	31.44
5000	2.56	12.35	34.99
10000	1.82	13.24	25.76

Table 3. Comparison of the energy terms for different displacements

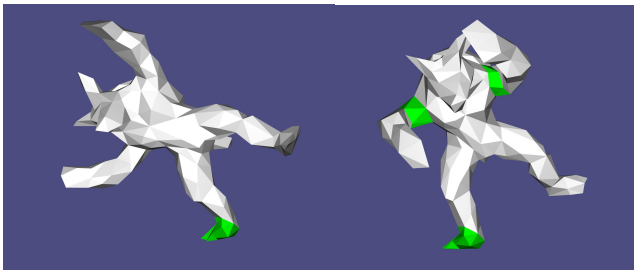

Figure 3. Different Positions for the armadillo Arap

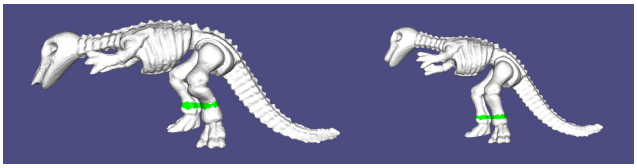

Figure 4. Different Positions for Dino Arap

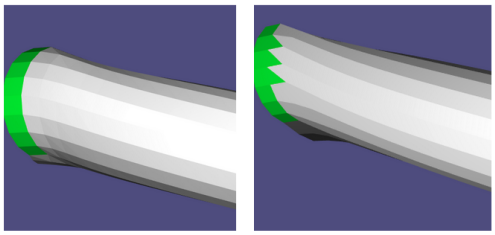

Figure 5. Cotangent vs Constant weights

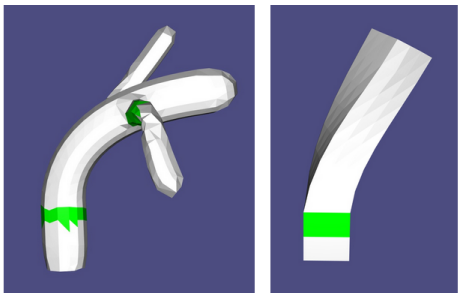

Figure 6. Cactus and Bar transformations

5. Analysis

Above we have the results for the ARAP implementation. We used a naive initializer for the optimization. As described in the paper, we obtained larger energy values for large sudden transformations with the initializer. The value is 31.78 without initialization and 31.55 with initialization. However, the ARAP method manages to recover afterwards.

Also, you can see in our results that the farther away from the original mesh we are, the higher our energy values are. In the optimization part, we try to minimize the energy difference between iterations, not the energy value. Only for the 10k mesh did we visualize worse deformation at larger displacements.

Our OpenMP implementation resulted in a large speedup, especially for large models like the Dino model, which has a difference of more than 13 seconds when we run our ARAP for 200 iterations. Moreover, the error decreases as the size of the mesh increases. This is because large meshes have smaller edges for the same size, so they are robust to position changes of a single vertex. We also found that the time required for convergence increases linearly with the number of vertices.

In our implementation, the SparseLU solver resulted in a lower convergence time than all other solvers. In addition, using cotangent weights results in a better curve and less artifacts on the surface than using constant weights. In the figure below, you can see that the beam model is much smoother and more cylindrical than its counterpart. Also, the edges point in the same direction as the surface of the object.

6. Conclusion

6.1. Problems

We spent most of our time implementing the graphical user interface from scratch using OpenGL. Implementing the sphere tracing algorithm and determining the space coordinates from the screen coordinates is a computationally intensive task that adds to the bottleneck of the ARAP implementation. In the algorithm part, we spent most of our time trying to obtain the local stiffness during bending. Rounding errors and the instability of the cotangent function led to undesirable results at the beginning of our implementation.

6.2. Future Works

At the beginning of this project, we planned to apply the ARAP scheme to meshes, to cage geometry, to skeletal geometry, and to tetrahedral geometry. Due to the challenges we did not anticipate and the time constraints we had, we only managed to apply ARAP directly to meshes. In our future work, we can implement the rest of the rigid transformations by directly using this ARAP class. The parallelization of the rest of the code can also be further improved by

determining the optimal number of threads. A holistic comparison between all the major models would be an exciting project for future research.

References

[1] https://github.com/erdenbatuhan/interactive-arap. 2

[2] Lu Chen, Jin Huang, Hanqiu Sun, and Hujun Bao. Cage-based deformation transfer. *Comput. Graph.*, 34:107–118, 2010. 1

[3] Robert W. Sumner, Johannes Schmid, and Mark Pauly. Embedded deformation for shape manipulation. *ACM Trans. Graph.*, 26(3):80–es, jul 2007. 1

[4] Yizhou Yu, Kun Zhou, Dong Xu, Xiaohan Shi, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Mesh editing with poisson-based gradient field manipulation. *ACM Trans. Graph.*, 23(3):644–651, aug 2004. 1