# 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 changed to be as rigid as possible. In our work, we implement this ARAP deformation algorithm by referring to the paper of Sorkine et al. [3]. Our motivation for choosing this theme was that it is being interactive and independent from any additional hardware. In the remainder of the paper, we review the 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 that requires the shape to maintain its local structure when shearing or stretched to fit certain constraints. If we simply choose a desired point in a 3d object and then pull it to the position we end up with a spiky surface with little or no local surface information. Hence, 3D deformation is much more challenging because of the non-linearity of similarity transforms.

**As-rigid-as-possible** implementation is the seminal paper in non-rigid deformation [4]. Prior to the ARAP implementation, **Mesh Editing with Poisson-Based Gradient Field Manipulation** by Yu et al.[5] in Figure **??** was an informative standard in the area 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 contain, F is a set of local frames which define the local orientations of the vertices, S is the set of scaling factors associated with the vertices, and R is a strength field. However, this method results in detailed distortion and lack of smoothness near the boundary region.

**Cage based deformation Transfer** by Chen et al. [2] uses a cage proxy in which vertices are a linear combination of shapes, in order to compute the deformation of the model. Once the weights and cages are know, we can simply move the object around by manipulating the cage vertices. Due to a significantly lesser number of vertices, the method is able to deform the armadillo model in around 56 millisecs. However, a large limitation of the model is that it largely depends on the shape and tessellation of the cage.

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

A much more recent method is that of the **Embedded Deformation for Shape Manipulation** [4]. In this method the optimization for the transformation is carried out on a deformation graph where each vertex is defined by a set of affine transformations. The method uses Guass-Newton iterations in order to compute the deformed vertices in time as low as 41 millisecs.

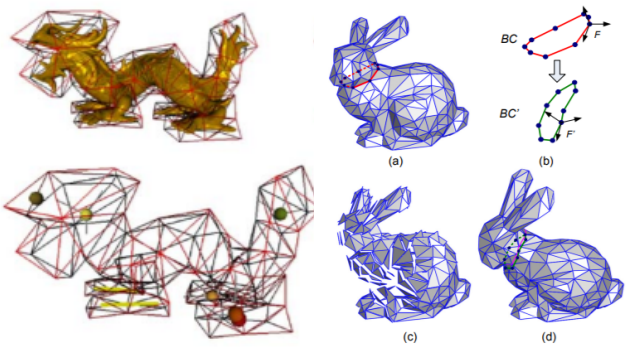


Figure 1. Cage-ARAP and Poison method

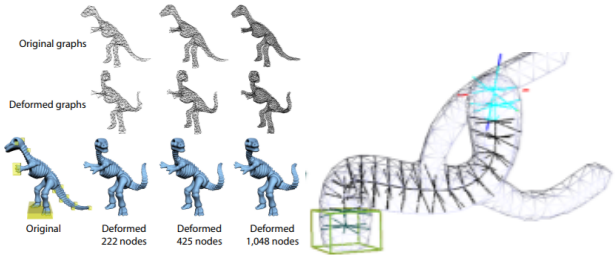


Figure 2. skeleton Arap and Embedded Graph

# 3. Method

## 3.1. Mathematical Overview

A transformation is considered as rigid, if there exists a rotation $R_i$, which rotates the vertex to its new location such that

$$p'_i - p'_j = R_i(p_i - p_j) \tag{1}$$

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

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

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 rigidity of deformations. After computing the gradients for both sides of the equation and assuming that $w_{ij} = w_{ji}$, we are left with this simple formula:

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

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 where $L \in \mathbb{R}^{V*V}$. It is defined like the following:

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

Also selecting these weights are crucial for the energy function. The weight matrix needs to be symmetric in order to be solved efficiently using a Simple Cholesky solver. We consider two different weights in our experiments; the constant weights and cotangent weights respectively. With constant initialization, we set each $w_{ij}$ value to 1. While in cotangent weights, we compute weights only for the neighbours of the respective vertex while the other positions are set to zero. Below is the formula to calculate the cotangent weights:

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

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

## 3.2. Implementation Details

### 3.2.1 Libraries

Initially we used simple OpenGL in order to build our GUI from scratch. We used the sphere tracing algorithm in order to find the nearest point on clicking the screen. We then implemented the shaders and lighting options for the GUI. However, due to performance reasons we shifted our arap class to the libigl framework which performs these same tasks with little overhead.

In order to ensure that we calculate 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 the use with different configurations to run our ARAP project. The user can choose from constant weights and cotangent weights. Furthmore, they can select whether to intialize the optimization using naive linear minimization. We use Eigen library for the linear operations in the ARAP algorithm. Additionally, OpenMP is used for the parallelization. We run our model on a Intel(R) Core(TM) i5-8279U CPU @ 2.40GHz (4c / 8t) 16 GB machine.

Due to no inter-dependencies during the computation of rotations,weights 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. Furthermore, we use BLAS routines to increase the speed for the linear solver. You can find more information 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

| - | SparseLU | SparseQR | BIGGSTAB |
|---|---|---|---|
| runtime | 0.93 | 21.23 | 1.69 |

Table 2. Comparison between the different solvers after 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 for the energy terms for different displacements

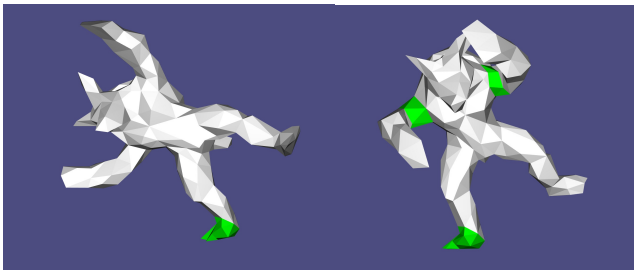

Figure 3. Different Positions for the armadillo Arap

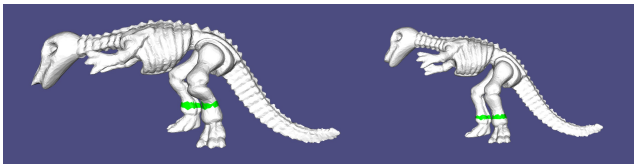

Figure 4. Different Positions for Dino Arap

## 5. Analysis

Above we have the results for the arap implementation. We used a naive initializer for the optimization. As expressed in the paper we obtained larger energy values with the initializer for large sudden transformations. It has a value of 31.78 without initial guess and 31.55 with initial guess. However, the arap method manages to recover afterwards.

Furthermore, in our results, one can see that the further away we have gotten away from the original mesh, the higher our energy values have become. In the optimization part, we try to minimize the energy difference between the iterations, not the energy value. Only with the 10k mesh, we have visualized worse deformation when we had larger displacements.

Our OpenMP implementation resulted in a large speedup especially for large models such as the Dino model which has a difference of more than 13 sec when we run our arap for 2000 iterations. Furthermore, as the size of the mesh increases the error decreases. This is because larges meshes of the same size have smaller edges, so they are robust against position changes of a single vertex. We also found that the time required to converge increases linearly against the number of vertices. In our implementation, the SparseLU solver resulted in lesser time to converge than all other solvers. Furthermore, the use of cotangent weights produces a better curve and little artifacts on the surface as compared to constant weights. In the figure below you can see that bar model is much more smoother and cylindrical than its counterpart. The edges are also pointing the same direction has the surface of the object.

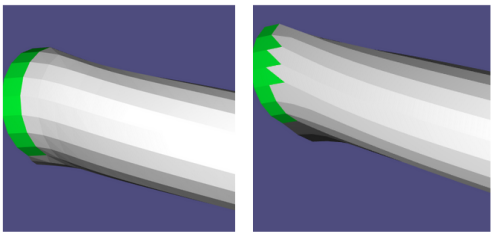

Figure 5. Cotangent vs Constant weights

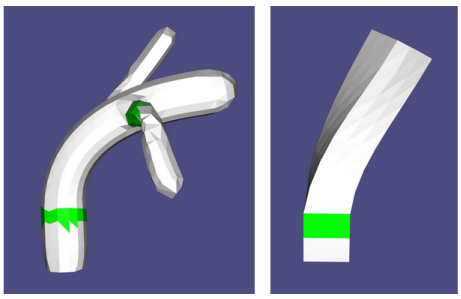

Figure 6. Cactus and Bar transformations

## 6. Conclusion

### 6.1. Problems

We have spent most of our time implementing the GUI from scratch using simple openGL. Implementing the sphere tracing algorithm and finding out world space coordinates from screen coordinates is a computationally expensive task which further adds to the bottleneck the ARAP implementation experiences. In the algorithm part, we have spent most of the time trying to maintain local rigidity during bending. Rounding errors and the instability of the cotangent function produced 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 on meshes, on cage geometry, on skeleton geometry and on tetrahedral geometry. Due to the challenges we have not expected and the time pressure we had,

we only managed to implement ARAP directly on mesh. In our future work, we can implement the rest of the rigid transformations by directly using this ARAP class. There is also room for further improvement in parallelizing the rest of the code by fixing the optimal number of threads. A holistic comparison between all 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] Olga Sorkine and Marc Alexa. As-rigid-as-possible surface modeling. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, SGP '07, page 109–116, Goslar, DEU, 2007. Eurographics Association. 1

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

[5] 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