3D Scanning and Motion Capture Project Report

ARAP - As Rigid As Possible

Michael Dey
ge83tal@mytum.de

Bendegúz Timár
bendeguz.timar@mytum.de

Ankur Deria
ge83xoq@mytum.de

Andrea Solanas de Vicente
gexot83@mytum.de

February 11, 2023

1 Introduction

The field of computer graphics has been rapidly advancing in recent years, and new techniques for modeling and animating shapes are being developed constantly. One such technique is the As Rigid As Possible (ARAP) deformation method. ARAP is an approach to modeling the deformation of shapes that aims to preserve the rigidity of the object while allowing it to be reshaped and transformed. In this paper, we present our ARAP project, which explores the implementation and application of the ARAP algorithm as stated by Sorkine et al. [1]. The project provided us an insight into the ARAP technique and its applications in computer graphics. In this paper, we will discuss the design and development of the ARAP project, the results of its implementation, and the conclusion which includes all the problems and challenges we encountered during the project.

2 Related Work

The As Rigid As Possible (ARAP) deformation method has been widely studied and applied in the field of computer graphics. In recent years, a large number of research papers have been published on this topic, exploring various aspects of the ARAP method, including its mathematical foundations, algorithmic design, and applications in real-world scenarios.

Several research papers have explored the mathematical foundations of the ARAP algorithm. For example, Igarashi et al. [2] describe in their paper another approach to the algorithm. They use a two-step idea to deform 2D triangle meshes. The algorithm uses quadratic error metrics to turn each minimization problem into a system of linear equations. This allows for quick calculation of free vertex positions during interactive manipulation, resulting in a more natural and rigid deformation of the shape. Zollhöfer et al. present another approach to the mathematical basics of the algorithm by decoupling the runtime complexity from the mesh's geometric complexity [3]. They call their manipulation LARAP since it is based on the ARAP paradigm, and their proxy geometry automatically introduces volume awareness into the optimization problem, leading to more natural deformations.

Other research papers have decided to explore other ways to adapt the basic ARAP algorithm in order for it to be used with other types of meshes and forms. A good example of this builds upon a pseudo-skeleton with bones that are allowed to stretch and twist instead of the triangle meshes [4]. Solving for the unknown positions of the joints in an as-rigid-as-possible manner decouples the optimization problem from the geometric complexity of the input model allowing for real-time edits. As one can see it also uses the mathematical basis from the previous paper [3]. Another research paper by Koki Nagano and Hongyi

Xu, focuses on research and adaptation of the scheme for modeling heterogeneous deformable surfaces [5]. They are able to accommodate heterogeneous rigidities to the basic algorithm, and demonstrate an edge weight editing scheme using a texture as an input rigidity distribution.

In our project we used the mathematical approach presented in the paper by Sorkine et al. as most of the papers mentioned before [1]. We saw that it was one of the first papers to explain the ARAP algorithm and its basic approach. We will explain more in detail in the next section how the math of the algorithm works.

3 Method

3.1 Mathematical background

To determine the rigid transformation between a cell C at vertex i and its altered form C'_i, we examine the edges that originate from vertex i in both S and S'. If the change from C to C' is rigid, there must be a rotation matrix R_i that exists such that

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

In the previous equation p_i and p'_i are the original and deformed vertex positions. When the deformation is not rigid, we can still find the best approximating rotation R_i that fits the above equation in a weighted least squares sense [1], we try to minimize

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

where w_{ij} are weights from the weight matrix (this will be explained later (6)). This energy function is used to measure the stiffness of the deformations. For convenience, let us denote the edge $e_{ij} := p_i p_j$, and similarly for e'_{ij} for the deformed cell C'_i. With this we plug it in the previous energy function (2) and we obtain the following covariance matrix:

$$S_i = \sum_{j \in N(i)} w_{ij} e_{ij} e'^T_{ij} = P_i D_i P'^T_i \tag{3}$$

where D_i is a diagonal matrix containing the weights w_{ij}, P_i is the $3 \times |N(vi)|$ containing e_{ij}'s as its columns, and similarly for P'_i. One can derive R_i from the singular value decomposition of $S_i = U_i \Sigma_i V_i^T$:

$$R_i = V_i U_i^T \tag{4}$$

To ensure that we can compute the correct rotation matrices, we modified the equation (4) a bit to ensure that the determinant remains positive using $R_i = U_i * V_i^T$ instead. Making our way back to our energy function, after calculating the gradients of both sides we arrive at the following sparse linear system of equations:

$$\sum_{j \in N(i)} w_{ij}(p'_i - p'_j) = \sum_{j \in N(i)} \frac{w_{ij}}{2}(p_i - p_j)(R_i + R_j) \tag{5}$$

The linear combination on the left-hand side is the discrete Laplace-Beltrami operator applied to p'. The system can be rewritten as $Lp' = b$, where b is an n-vector whose ith row contains the right-hand side expression of the previous function. We also need to include the constraints, in this case the fixed vertices or the vertices chosen by the user, into this system. We can just set the specific constraints to 0 in the system matrix L so they are not deformed. As can be seen, the system matrix is not affected by the rotation matrices and only depends on the original mesh. Because of this we can use a direct solver such as the sparse Cholesky factorization with fill-reducing reordering. Finally as we mentioned before, we use a weight matrix with w_{ij} weights. The weights should compensate for non-uniformly shaped cells. We therefore use the cotangent weight formula for w_{ij}:

$$w_{ij} = \frac{1}{2}(\cot \alpha_{ij} + \cot \beta_{ij}) \tag{6}$$

where α_{ij}, β_{ij} are the angles opposite of the mesh edge (i,j).

Once we have explained the mathematical background, we can now explain the steps of the main algorithm. The ARAP algorithm follows minimizing the rigidity energy and it works as follows:

1. Firstly the weights in the weight matrix are precomputed using the cotangent formula. Also the system matrix L is precomputed along with the neighbors for all the vertices in the mesh.

2. Using an initial guess of the deformed vertices (in our case we chose to use the original vertices), the rotation matrix for each vertex R_i is estimated.

3. New vertex positions p'_1 are obtained by solving the system $Lp' = b$, plugging R_i into the right-hand side.

4. Then further minimization is performed by recomputing the rotations and using them to define a new right hand-side for the linear system.

With this at the end we calculate the rigidity energy using the original function, and obtain the new deformed vertices.

3.2 Implementation

To implement our project, several libraries and frameworks were required:

- A graphics API, in our case we chose OpenGL, for rendering the 3D mesh and displaying the deformation in real time. We also used a open-source C++ library for geometry processing and computer graphics called libigl which works in tandem with OpenGL.

- A library for linear algebra for performing the mathematical operations necessary for computing the deformation of the mesh. We chose to use Eigen.

- A parallel programming library or framework, such as OpenMP, for executing the deformation algorithm on multiple threads of a multicore processor simultaneously.

We used the Eigen library to perform the calculations necessary for computing the deformation of the mesh. Eigen is a popular C++ library for linear algebra, which provides tools for performing mathematical operations on matrices and vectors. Libigl provides a convenient high-level interface for processing geometric models, and OpenGL provides a low-level interface for rendering 3D graphics. After processing a model with libigl to compute topology, geometry, or other attributes, the result can be passed to OpenGL for rendering. We also used the library to read from a standard mesh file formats and view it in a 3D scene using its inbuilt GUI. OpenMP is a parallel programming model that allows for the easy development of programs that can run on multiple processor cores. This was very useful for improving the performance of our algorithm when it came to the deformation of the vertices. In the case of the ARAP deformation algorithm, using OpenMP can allow us to take advantage of multiple processor cores to compute the deformation of the mesh more efficiently.

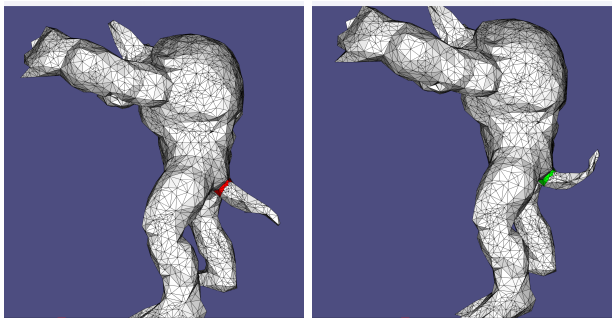
We used Cmake and Visual Studio 2022 to build and run the source code.

4 Results

In the following subsections we show some benchmarks we conducted to compare the different results of using the libraries and frameworks. 2 different machines were used for the measurements:

- Machine 1 - 32GB RAM with AMD RX 5800XT (8 core 16 thread) CPU.

- Machine 2 - 16GB RAM with 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz CPU



(a) Pre ARAP Deformation (b) Post ARAP Deformation

Figure 1: Images showing the selection of fixed faces and the deformation of other faces using ARAP.

4.1 Benchmarks

Our ARAP algorithm is programmed to run for 10 iterations or stop earlier in case an energy threshold is reached. The different measurements we obtained are displayed in Tables: 1, 2 and 3. We also show a few images of a mesh before we applied the ARAP deformation and after.

Model	No. Vertices	Without OpenMP(s)	With OpenMP(s)
Bar	338	0.016	0.009
Armadillo250	202	0.0137	0.008
Armadillo500	252	0.02	0.012
Armadillo1K	864	0.09	0.087
Armadillo2K	1002	0.118	0.089
Armadillo5K	2502	0.47	0.39
Armadillo10K	5002	1.1	0.56
Cactus-small	620	0.026	0.02
Cactus-highres	1856	0.086	0.076
dino	14070	2.45	2.22

Table 1: Runtime with and without OpenMP in seconds

The first experiment (Table 1) shows the difference in runtime (seconds) with and without using OpenMP for parallelization in the ARAP algorithm. We decided to test diferent number of vertices and faces to see if a similar movement would cost more time without OpenMP than with it. Next, the second experiment (Table 2) observes the increase in performance by increasing the number of threads in OpenMP from 2-16. This was done to see what the best number of threads we could use in our implementation of OpenMP. In the third experiment (Table 3) we wanted to get an insight into how much rigid energy is obtained depending on what area of the mesh is moved, how far it is moved (distance) and the corresponding runtime. For this we used the Armadillo 1k model which is also used as the example mesh in Figure 1. The moving vertex is moved by +0.05 units in every direction for the small displacement and +0.2 units in every direction for the large displacement. The displacement happens in 50 sub-steps to simulate mouse movement. The Time value is the average sub-step runtime.

No. of threads	Runtime Avg(s)
2	3.4183
4	3.3556
8	3.3386
16	3.2980

Table 2: Runtime per ARAP deformation loop for different number of threads for dino.off model with SparseLU solver.

Fixed Area	Moving Vertex	Displacement	Rigid Energy	Time
Knee	Foot	large	0.000041	0.020
Knee	Foot	small	0.000002	0.018
Hip	Foot	large	0.000026	0.019
Hip	Foot	small	0.000003	0.018
Hand	Foot	large	0.000028	0.019
Hand	Foot	small	0.000002	0.018

Table 3: The Rigid Energy of picked transformations

5 Analysis

Once the experiments have been briefly explained and we showed the results we can start to analyze them. Table 1 shows a decrease in runtime across several meshes using OpenMP. This proves that our ARAP algorithm supports multi-threading and can use it to enhance its performance in terms of runtime. It can be seen that the difference between using OpenMP and not using it, is larger for bigger models like the Dino model or the Armadillo 10K. This is because large meshes have smaller edges for the same size, so they are robust to position changes of a singe vertex, therefore the error decreases.

Table 2 shows a clear boost in performance due to the increase in the number of threads. It is evident that our ARAP algorithm scales well with increase in the core count of a processor. It does however depend on the specific characteristics of the computer it is run on.

The last experiment in Table 3 shows that the farther away from the original mesh we are, the higher our energy values are. Even if the fixed area and the moving vertex is the same, the energy largely depends how big the displacement is from the user's point. This means the bigger the distance that the vertex is moved, the higher the rigid energy is. How-

ever the runtime of the experiment shows that there is not a big difference in terms of time it takes to do each displacement, no matter how large or small the displacement is.

6 Conclusion

In this paper, we presented our As Rigid As Possible (ARAP) project. The project aimed to explore the implementation and application of the ARAP algorithm and technique in real-world scenarios. The results of the project showed that we were able to successfully implement the ARAP method and apply it to a variety of shapes and objects. The ARAP project provided us with a deeper understanding of the ARAP technique and its applications in computer graphics.

6.1 Challenges

Even though our project was successful and we were able to implement the algorithm correctly, it does not mean we did not encounter problems and challenges. One of our main problems was implementing the main algorithm. We dedicated most of our time to fixing it and finding all the mistakes in it, which was quite time consuming. A main problem is that it is not easy to test so we had to manually calculate the results to make sure it was working as it should.

6.2 Future Work

At the beginning of the project we planned to apply the algorithm to meshes but this is not the only way the algorithm could be applied as we saw on the different papers. For future work, there are several directions that could be explored. For example, new techniques could be developed to speed up the computation of the ARAP deformation, making it more suitable for real-time applications. Another direction could be to investigate new methods for handling topological changes in ARAP deformation, providing a way to model the addition or removal of vertices and faces in a shape. Finally, the ARAP method could be applied to new and innovative applications in the field of computer graphics and animation, such as virtual reality, robotics, and medical visualization.

References

[1] O. Sorkine and M. Alexa, "As-Rigid-As-Possible Surface Modeling," in *Geometry Processing* (A. Belyaev and M. Garland, eds.), The Eurographics Association, 2007. 1, 2

[2] T. Igarashi, T. Moscovich, and J. F. Hughes, "As-rigid-as-possible shape manipulation," *ACM Trans. Graph.*, vol. 24, p. 1134–1141, jul 2005. 1

[3] M. Zollhöfer, E. Sert, G. Greiner, and J. Süßmuth, "Gpu based arap deformation using volumetric lattices.," in *Eurographics (Short Papers)* (C. Andújar and E. Puppo, eds.), pp. 85–88, Eurographics Association, 2012. 1

[4] M. Zollhöfer, A. Vieweg, J. Süßmuth, and G. Greiner, "Pseudo-skeleton based arap mesh deformation," 2013. 1

[5] K. Nagano and H. Xu, "As-rigid-as-possible surface modeling for heterogeneous deformable surfaces," 2014. 2