CIS660: Advanced Topics in Computer Graphics and Animation

AA Shape-Up

Mesh Modeling Tool with Constraint Projection

A Maya Plug-in Development Tool

By: Xuntong Liang, Yiyang Chen

PROJECT SUMMARY

The goal of this project is to design an interactive mesh modeling tool with some operations applied on the geometry based on effective constraint projection. These geometry processing operations help us to produce certain geometric shapes with geometric constraints, given input of meshes. These geometric constraints are often created for specific goals, such as deformation or mesh quality improvement. This tool helps artists and game designers create high quality 3D models more effectively. For example, when modeling a mesh, some vertices in a polygon may not be coplanar, which often produces unpredictable results. Planarization operation makes the polygons in meshes more planar, which reduces that condition and makes it easier for further editing. It also helps some researchers get 3D geometric data that meet their requirements for further research, such as physical simulation. Although many of the geometry processing tasks are acceptable if they are processed offline, the efficiency still affects the user experience and proficiency of adjusting the parameters.

We will develop this authoring tool “AA Shape-Up” as a Maya plug-in using the C++ API and the MEL script. First, we will implement a geometry optimization framework based on the local-global solver in Bouazzi et. al’s paper [1], as well as some shape constraints. Then, we will implement some core operations of our tools such as planarization, wire mesh design, and as-rigid-as-possible deformation, as well as create some UI for adjusting parameters of the operations using the MEL script. After that, we will implement the Anderson Acceleration method on these operations to make them more efficient. The remainder of the time will be allotted for further optimization on interface and performance, debugging, and even providing CUDA version of these algorithms for those who have NVIDIA’s GPU.

Our authoring tool is based on the SigGraph Paper "Anderson Acceleration for Geometry Optimization and Physics Simulation" [3], which proposed an accelerated method for geometry optimization problems with the framework using local-global solver [1].

1. AUTHORING TOOL DESIGN

1.1. Significance of Problem or Production/Development Need

Many computer graphics problems require computing geometric shapes whose elements are subject to certain constraints. In geometric modeling, such constraints can be related to aesthetics, performance, or fabrication requirements of the shape. Within computer graphics, first-order methods have been developed for these optimization problems in geometry processing, and Bouaziz et. al's seminal paper [1] proposed a unified local-global optimization framework for geometric constraints, which is applicable in many different application domains like deformation [8] and parametrization [9]. These applications have typically been solved by specialized optimization algorithms before.

Bouaziz et. al [1] use a local-global solver for geometry optimization, but the efficiency of local-global solvers comes at the cost of accuracy. Specifically, although they are efficient to produce an approximate solution, it can take a very long time to compute an accurate result, because their convergence rate is sublinear in general. So, we use Anderson acceleration (short for AA) to speed up their convergence. The Anderson acceleration method was originally proposed for solving nonlinear integral equations by Anderson et. al's seminal paper [2] and has been successful for a variety of problems in different domains such as computational chemistry and computational physics. Yue et. al [3] use the Anderson acceleration method [4] to speed up the convergence of a local-global solver and address the stability issue. They apply the acceleration technique on geometry optimization problems with the Shape-Up solver [1]. This method makes this kind of geometry optimization much more efficient, which gives a mind for us to design an interactive mesh modeling tool with efficient operations.

1.2. Technology

The most essential SigGraph paper for our tool is "Anderson Acceleration for Geometry Optimization and Physics Simulation", which provides the core algorithms of the geometry processing and acceleration. In addition, the paper "Shape-Up: Shaping Discrete Geometry with Projections" also provides the idea of how to complete the framework for the operations of our tool. Along the logic of the development of motivation, we introduce the latter paper first, and then the former.

- **Shape-Up: Shaping Discrete Geometry with Projections**
  This paper introduces a unified optimization framework for geometry processing based on shape constraints. This framework is based on a shape proximity function and shape projection operators. The shape proximity function plays a role in an objective function which contains the information of the distance between the current shape and the target shape with some constraints. The shape projection operators tend to minimize the proximity function by introducing projection functions for auxiliary variables. With more variables, apply block coordinate descent to form a local-global solver. The local step consists of projecting every element onto the constraint manifold, i.e., solving a small nonlinear problem per element. The global step combines the results of individual projections, finding a compromise between all the individual constraints by solving a

sparse positive definite linear system with a fixed matrix. The efficiency and robustness make the solver very powerful to get approximate solutions of geometry optimization problems, while the sublinear convergence rate makes it hard to improve the accuracy.

- **Anderson Acceleration for Geometry Optimization and Physics Simulation**
  This paper treats each local-global step as a fixed-point iteration, applying Anderson acceleration, a well-established technique for fixed-point solvers, to speed up the convergence of a local-global solver. The paper also addresses the stability issue of classical Anderson acceleration and proposes a simple strategy to improve the stability of the optimization process. This technique significantly reduces the number of iterations required to compute an accurate result of geometry optimization and physics simulation problems, with only a slight increase of computational cost per iterations. Its simplicity and effectiveness make many existing algorithms effective, as well as enables us to design efficient new algorithms.

1.3. Design Goals

Based on the goal of our tool, we can specify the design goals from different aspects, such as the target users, their objectives, the features that help them to reach the goals, the input, and the output.

1.3.1 Target Audience

Our main audience are modeling artists and game designers who are capable to create some simple models for games and CG animations. Moreover, we also expect some researchers can use this tool for geometric data which helps their research.

1.3.2 User Goals and Objectives

When modeling artists, game designers and researchers want to model a 3D object with high quality, they are very likely to make the shape of the mesh fulfill some requirements which can be described as constraints. Given some operations, they can process the shape of the mesh to satisfy the constraints. We hope these operations can run efficiently, and user friendly.

1.3.3 Tool Features and Functionality

1. **Planarization**
   Planar and circular meshes are important in the field of architectural design. Users can get a mesh in which the vertices of each polygon lie on a plane by applying a planarization operation. It is also possible to design a circularization that puts the vertices of each polygon lie on a circle to get a circular mesh.

2. **Wire mesh design**
   Wire meshes represent an innovate medium for industrial applications including composite materials and architectural facades. Wire meshes often

contain constraints such as edge length constraints and corner angle constraints. These constraints are related to some properties of a specific wire material. Users can get a wire mesh model by applying a wire mesh optimization operation.

3. **As-rigid-as-possible deformation**
   Users apply a deformation node on a mesh and move user-driven handle vertices to achieve deformation with local feature preservation. A series of as-rigid-as-possible constraints help maintain local features of a mesh after deformation. We mainly focus on ARAP deformation on polygonal meshes in 2D cases.

1.3.4 Tool Input and Output

The input will be a mesh provided by modeling or 3D reconstruction. The mesh may be problematic for some usages so that we should optimize some properties of the mesh.

After the processing of our tool, which is applied by a Maya node contains operation, the mesh with shape that fulfill the constraints will be the output mesh. Users can export the output mesh to use it in other applications or apply further modeling on it.

1.4. User Interface

Since our tool contains several operations which apply nodes on meshes, there should be user interface of the operations and the nodes. Users can select which operation will be applied to the mesh in the operation GUI and adjust the parameters in the node GUI. We can also provide interface for adjusting the parameters in the operation GUI before creating nodes if necessary.

1.4.1 GUI Components and Layout

- **Menu**
  The menu bar of our tool is very simple. We need to create deformer nodes for different operations, as well as buttons for creating windows that help us adjust parameters of these operations. For ARAP deformation, we need to create a weight painting tool for defining handle vertices. Figure 1 shows how the menu items look like.

Figure 1: A mockup of the GUI of menu items.

- **Option window**
  In an option window, users can adjust the parameters of the corresponding operation. For example, the weight of each term in the proximity function. Figure 2 shows how the option windows look like. The parameter adjustment process is similar to adjusting parameters in the node attribute panel.

Figure 2: A mockup of the GUI of option windows.

- **Weight painting tool**
  The window of the weight painting tool is nearly identical to the built-in tool setting window. Figure 3 shows how the window looks like.

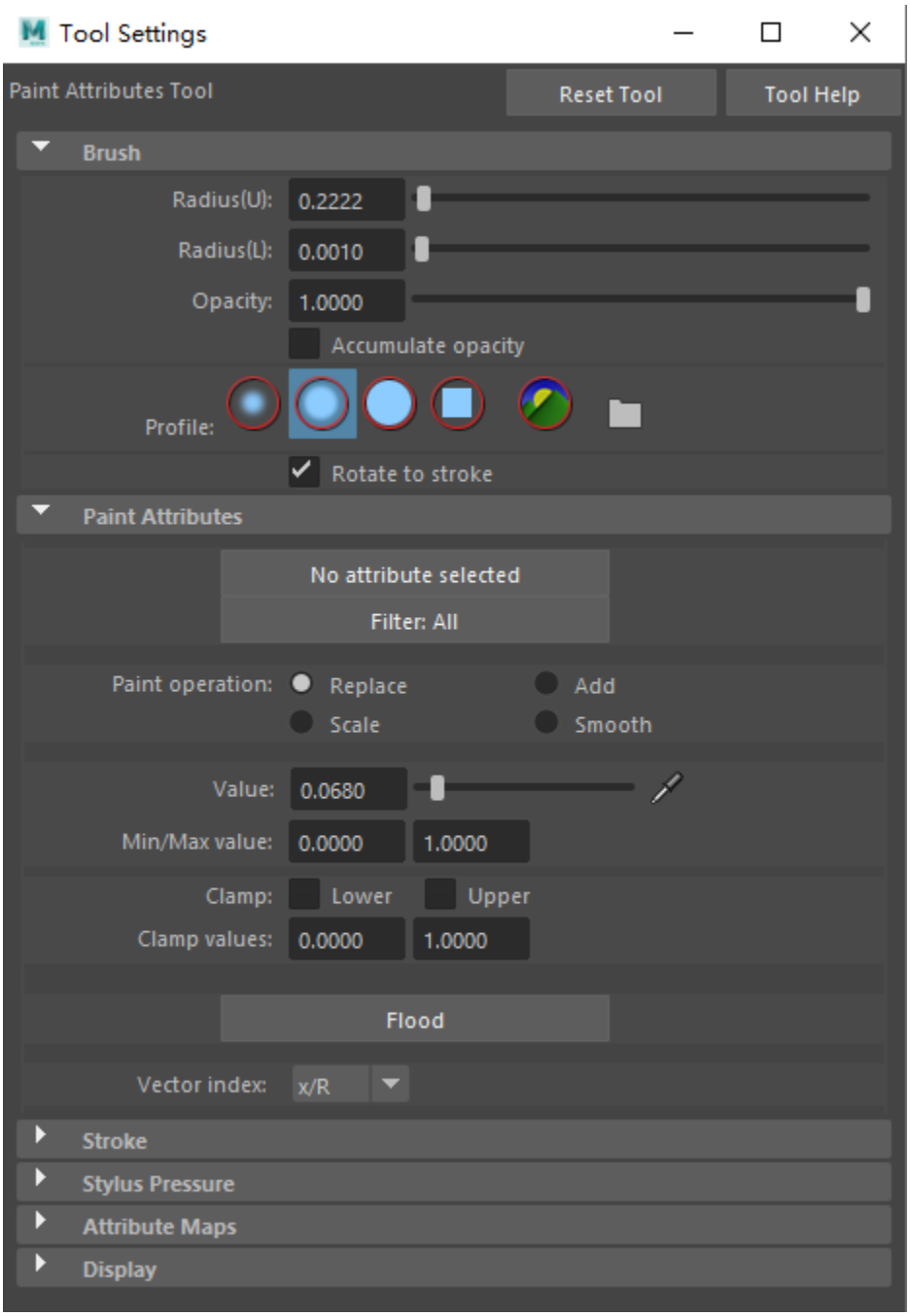


Figure 3: The GUI of the window of the weight painting tool.

1.4.2 User Tasks

Users apply a node to a mesh to process the specific operation. Before creating nodes, users can also open the option window to adjust the parameters. They can also adjust parameters in the node attribute panel. For deformation operations, users can paint weights on the vertices of the mesh, defining which vertices are handles.

Since the GUI is relatively simple, users can only know what the effect of these operations are. In addition, users should know how to use the weight painting tool if they apply deformation. The usage of the weight painting tool is identical to the built-in weight painting tool of Maya.

1.4.3 Workflow

The workflow of our authoring tool is very simple. Figure 4 shows the workflow. The input mesh can be obtained by manually modeling in various applications or 3D reconstruction.

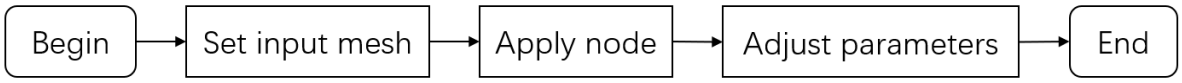


Figure 4: The workflow of our authoring tool.

2. AUTHORING TOOL DEVELOPMENT

2.1. Technical Approach

2.1.1 Algorithm Details

In this subsection, we introduce the general framework and the algorithm of the geometry optimization first, and then we introduce the details of each task we will solve.

- **Local-global solver**

  Our implementation algorithm is based on a local-global solver of geometry optimization with constraint projection. Every local-global step forms an iteration of the solver. In the local step, we compute the projections using the current estimated state. In the global step, we update the state by minimizing the proximity function keeping the projections fixed. At each step, the proximity function will decrease even if some values of the elements increase. Figure 5 shows an example of minimizing the summed square distance between the state x and all the constraints as the proximity function $\phi(x)$.

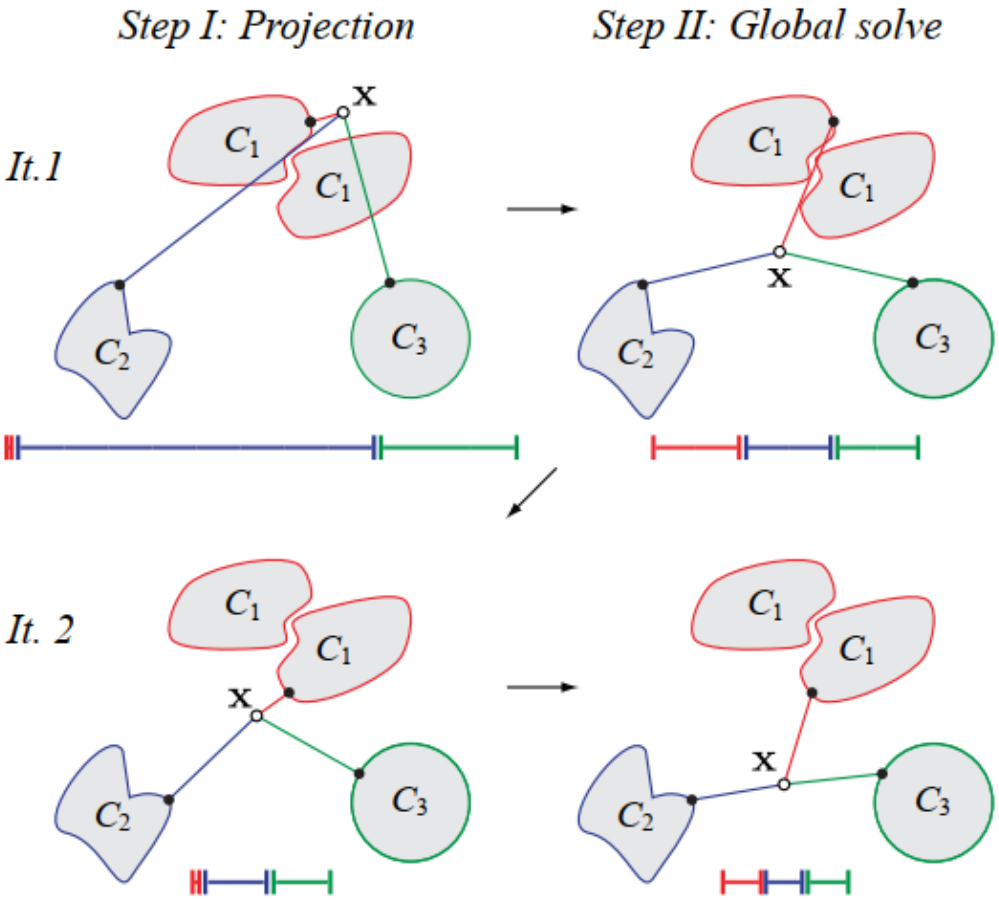


Figure 5: Two iterations of the local-global solver.

We often design proximity function that makes the local step to solve analytic solutions parallel, while the global step is to solve a sparse positive definite linear system with a fixed matrix.

For geometry optimization problems with point constraints, the proximity function is often modeled by:

$$\phi(\boldsymbol{Q}, \{P_i\}) = \sum_i \frac{w_i}{2} \|A_i \boldsymbol{Q} - P_i\|_F^2 + \sigma_i(P_i)$$

with matrix of vertices $Q = [q_1 \ \dots \ q_n]^T \in R^{n \times d}$ and the auxiliary variables $P_i \in R^{k_i \times d}$ subject to point constraint functions $\sigma_i(P_i) \in R$ that equal to zero if P_i satisfies the constraint i. The formula of the indicator functions allows us to write regularization terms in a unified way. Here matrix $A_i \in R^{k_i \times n}$ selects the relevant points for constraint i and applies linear transformation to derive an approximate representation, which means that P_i is the closest projection of $A_i Q$ onto the feasible set C_i of constraint i. The local-global solver finds both the projections and the vertices that minimize the proximity function, that is:

$$\underset{Q, \{P_i\}}{\operatorname{argmin}} \phi(Q, \{P_i\})$$

In this case, we fix Q to solve $\{P_i\}$ in the local step with analytic solutions, while we fix $\{P_i\}$ to solve Q in the global step with linear equation:

$$\left(\sum_i w_i A_i^T A_i\right) Q = \sum_i w_i A_i^T P_i$$

Since the system matrix is fixed for all iterations, we can precompute its Cholesky factorization and efficiently solve for each right-hand-sides in parallel.

- **Anderson acceleration**

  Since we want to get the solution more accurately and keep the efficiency, we introduce the Anderson acceleration technique to accelerate the convergence of the solver. Regarding the iteration of the local-global solver as a fixed-point iteration:

  $$Q_{LG}^k = G(Q^{k-1})$$

  With the fixed-point iteration form, we can apply Anderson acceleration. The key idea of Anderson acceleration is to utilize the current state Q^k as well as the previous m states $Q^{k-1}, \dots, Q^{k-m}$ to derive a new state Q^{k+1} that decreases the residual norm $F(Q) = G(Q) - Q$ as much as possible:

$$\boldsymbol{Q}(\boldsymbol{\alpha}) = \boldsymbol{Q}^k + \sum_{j=1}^{m} \alpha_j \left(\boldsymbol{Q}^{k-j} - \boldsymbol{Q}^k\right)$$

where $\boldsymbol{\alpha} = (\boldsymbol{\alpha_1}, \dots, \boldsymbol{\alpha_m})$ is the affine coordinates. With an approximation of the mapping G via barycentric interpolation, we can find the $\boldsymbol{\alpha}$ with the smallest residual norm:

$$\boldsymbol{\alpha}^* = \underset{\boldsymbol{\alpha}}{\operatorname{argmin}} \left\| F^k + \sum_{j=1}^{m} \alpha_j \left(F^{k-j} - F^k\right) \right\|^2$$

Then the new state is a combination of $Q(\boldsymbol{\alpha}^*)$ and the approximation of mapping G, or just use $Q(\boldsymbol{\alpha}^*)$ for a special case. For better efficiency instead of recomputing the differences between F^k and all m previous residuals, we solve an equivalent problem:

$$(\theta_1^*, \dots, \theta_m^*) = \underset{\theta_1, \dots, \theta_m}{\operatorname{argmin}} \left\| F^k - \sum_{j=1}^{m} \theta_j \Delta F^{k-j} \right\|^2$$

where $\Delta F^i = F^{i+1} - F^i$. This minimization problem can be solved by constructing the normal equation:

$$(D^T D)\boldsymbol{\theta} = D^T F^k$$

where $D = \left[\Delta F^{k-1} \ \dots \ \Delta F^{k-m}\right]$. The new state becomes:

$$Q_{AA}^{k+1} = G(Q^k) - \sum_{j=1}^{m} \theta_j^* \Delta G^{k-j}$$

where $\Delta G^{k-j} = G\left(Q^{k-j+1}\right) - G\left(Q^{k-j}\right)$. In this way, we can determine Q_{AA}^{k+1} by computing the difference vectors, updating the small linear system of the normal equation, and solving the affine coordinates $\boldsymbol{\theta}$.

To improve stability, we can compare the target proximity function of Q_{AA}^{k+1} with that of the previous state Q^k. If the function value decreases, then it can be chosen as the new state. Otherwise, we choose Q_{LG}^{k+1} which is produced by the original local-global solver.

The pseudocode of the local-global solver with Anderson acceleration is shown below.

Algorithm 1: Anderson acceleration for the local-global solver.

Data: $\mathbf{Q}^0$: initial node positions;
Local-Step $(\cdot)$: local step of projection onto feasible sets;
Global-Step $(\cdot)$: global step of updating node positions;
G: the mapping combining the local and global steps;
E: the target energy function;
m: number of previous iterates used for acceleration.

Result: A sequence $\{\mathbf{Q}^k\}$ converging to a local minimum of E.

$\mathbf{Q}^1 = G(\mathbf{Q}^0)$; $F^0 = \mathbf{Q}^1 - \mathbf{Q}^0$; $E_{\text{prev}} = +\infty$;
for $k = 1, 2, \ldots$ **do**
  // Make sure $\mathbf{Q}^k$ decreases the energy
  $\mathbf{P} =$ *Local-Step* $(\mathbf{Q}^k)$;
  if $E(\mathbf{Q}^k, \mathbf{P}) \geq E_{\text{prev}}$ **then**
    $\mathbf{Q}^k = \mathbf{Q}_{\text{LG}}$; $\mathbf{P} =$ *Local-Step* $(\mathbf{Q}_{\text{LG}})$;
  end
  $E_{\text{prev}} = E(\mathbf{Q}^k, \mathbf{P})$;
  // Anderson acceleration
  $\mathbf{Q}_{\text{LG}} =$ *Global-Step* $(\mathbf{P})$;
  $G^k = \mathbf{Q}_{\text{LG}}$; $F^k = G^k - \mathbf{Q}^k$; $m_k = \min(m, k)$;
  $(\theta_1^*, \ldots, \theta_{m_k}^*) = \arg\min \|F^k - \sum_{j=1}^{m_k} \theta_j \Delta F^{k-j}\|^2$;
  $\mathbf{Q}^{k+1} = G^k - \sum_{j=1}^{m_k} \theta_j^* \Delta G^{k-j}$;
end

- **Planarization**
  Starting from the initial quad mesh and a reference triangle mesh, we planarize the quad mesh by minimizing a target energy about is vertex positions:

$$E = w_{planar}E_{planar} + w_{ref}E_{ref} + w_{fair}E_{fair} + w_{2nd}E_{2nd}$$

  where w_{planar}, w_{ref}, w_{fair}, and w_{2nd} are positive weights. E_{planar} is a planarity term that measures the sum of squared distance between each face an its best fitting plane. E_{ref} is a reference term that sums the squared distance from each vertex to the reference mesh. E_{fair} and E_{2nd} are Laplacian fairness and relative

Laplacian fairness terms:

$$E_{fair} = \sum_i (\|\delta_i\|^2)$$
$$E_{2nd} = \sum_i \left(\left\|\delta_i - \delta_i^0\right\|^2\right)$$

where δ_i is the Laplacian operation defined by its one ring neighborhood. The projection operators for the planarity terms are computed via SVD, while the projection operators of the reference term are evaluated by finding the closest point on the reference surface from each quad mesh vertex which is implemented using an AABB tree.

- **Wire mesh design**
  The wire mesh is represented as a quadrilateral mesh, subject to the following geometric constraints that model its material properties: (1) all edge lengths equal to a constant l; (2) within each face, all four corner angles are between 45 and 135 degrees. We compute the final wire mesh shape by minimizing the target function

$$E = w_{edge}E_{edge} + w_{angle}E_{angle} + w_{ref}E_{ref}$$

where E_{ref} is defined as that of planarization. E_{edge} and E_{angle} are shape proximity terms for the edge length constraints and the angle constraints respectively, as follows:

$$E_{edge} = \sum_{i\in\varepsilon} \left\|\mathbf{q}_{i_1} - \mathbf{q}_{i_2} - \boldsymbol{p}_i\right\|^2 + \sigma_{edge}(p_i)$$
$$E_{angle} = \sum_{(i,j,k)\in A} \left\|\begin{bmatrix}\mathbf{q}_i - \mathbf{q}_j \\ \mathbf{q}_k - \mathbf{q}_j\end{bmatrix} - \begin{bmatrix}\mathbf{e}_{ijk}^1 \\ \mathbf{e}_{ijk}^2\end{bmatrix}\right\|_F^2 + \sigma_{angle}\left(\begin{bmatrix}\mathbf{e}_{ijk}^1 \\ \mathbf{e}_{ijk}^2\end{bmatrix}\right)$$

where ε is the index set of mesh edges. $\mathbf{q}_{i_1}$ and $\mathbf{q}_{i_2}$ are the vertices of edge i. A is the index set of vertices that form a face corner.

- **As-rigid-as-possible deformation**
  We perform ARAP deformation of 2D triangle meshes according to user-driven handle vertices, by solving the problem subject to hard constraints of handle vertex positions:

$$E = \sum_{i\in F} \|\mathbf{J}_i\mathbf{Q}_i - \mathbf{R}_i\|_F^2 + \sigma_{rot}(\mathbf{R}_i)$$

where we optimize all the positions of non-handle vertices. F is the index set of triangles. $\mathbf{Q}_i \in R^{(d+1)\times d}$ collects the positions of the $d+1$ vertices in cell i. $\mathbf{J}_i \in$

$R^{d\times(d+1)}$ is a linear map that computes the deformation gradient of cell i between positions $\mathbf{Q}_i$ and initial positions $\mathbf{Q}_i^0$. The projection operator in the local step finds the closest rotation matrix to a give matrix, which can be computed using SVD.

2.1.2 Maya Interface and Integration

- **User interface**
  The user interface in Maya will be written using the MEL scripts. With the help of the Maya UI and the proxy nodes, we can process different operations by triggering different menu items and encapsulate different operations into different deformer nodes.

- **Commands and nodes**
  Some of the commands called by GUI will be written using the MEL scripts, while the other commands and nodes will be written in the C++.

- **Data structures for algorithms**
  We can encapsulate the algorithms into data structures that are independent of the Maya API, and we can encapsulate these data structures into the class inherited from MPxDeformerNode class.

  Each node contains a local-global solver, a proximity function, and a series of constraints with information of initial state of the input mesh. The details of these classes will be shown in the next subsection.

2.1.3 Software Design and Development

- **Class design**
  Users will apply nodes to the input meshes to get the goal output, so we design several nodes as Maya interface, which are inherited from the MPxDeformerNode. This class contains some information of geometry and weights, and it can apply deformation on the input geometry to get the output geometry. Figure 6 shows the class diagram of them.

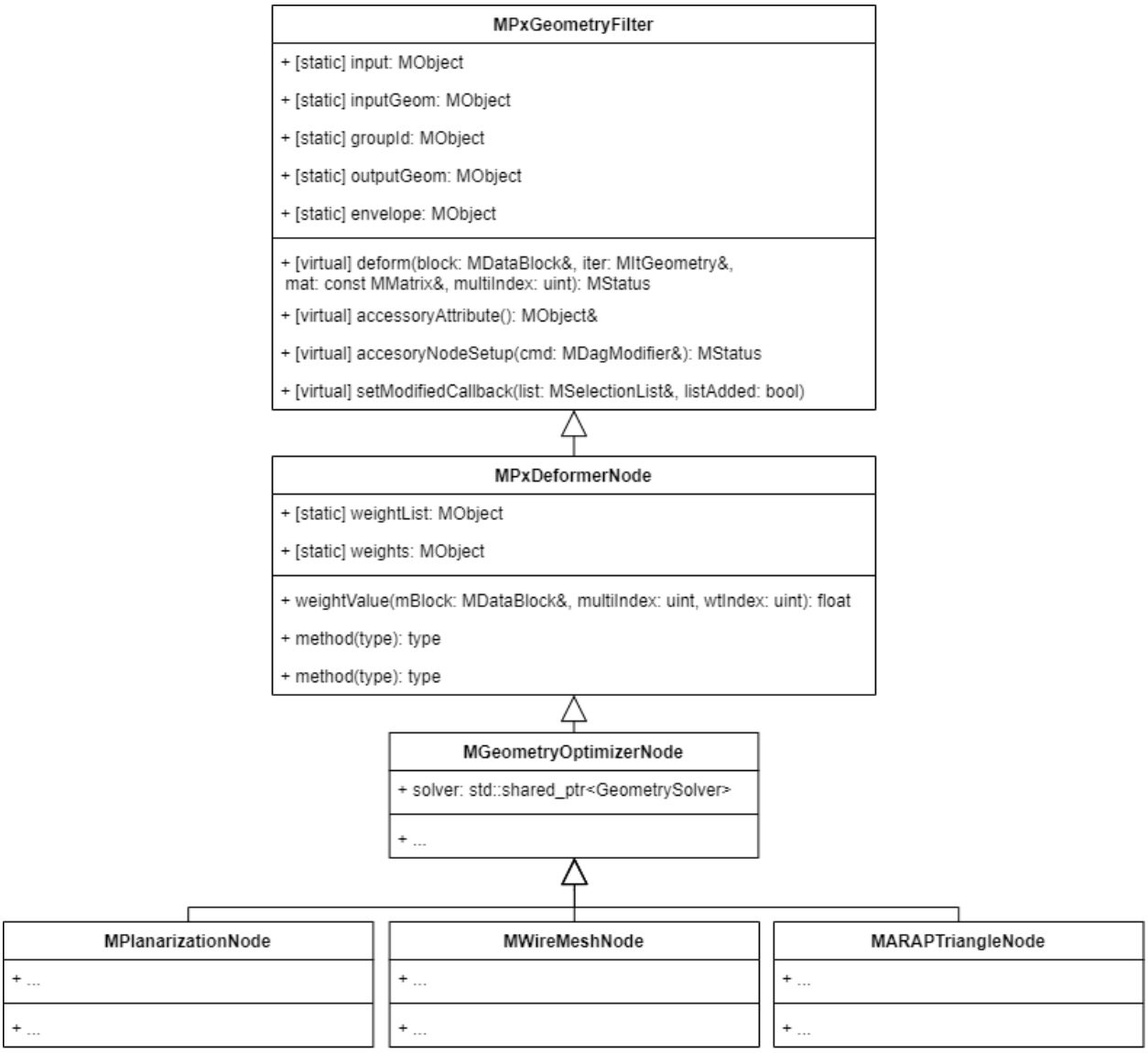


Figure 6: The class diagram of the related Maya interface.

The geometry optimizer node contains a solver for the computation of the core algorithms. Here we focus on the components required for computation first, and then show the class diagram of the solver.

To construct the form of the generalized proximity function, we need constraints and regularization terms. The constraints are essential to the local step. The constraints and the regularization terms form the system of the global solver. Based on the types of constraints and the types of regularization term, we design the classes like the diagrams in Figure 7 and Figure 8.

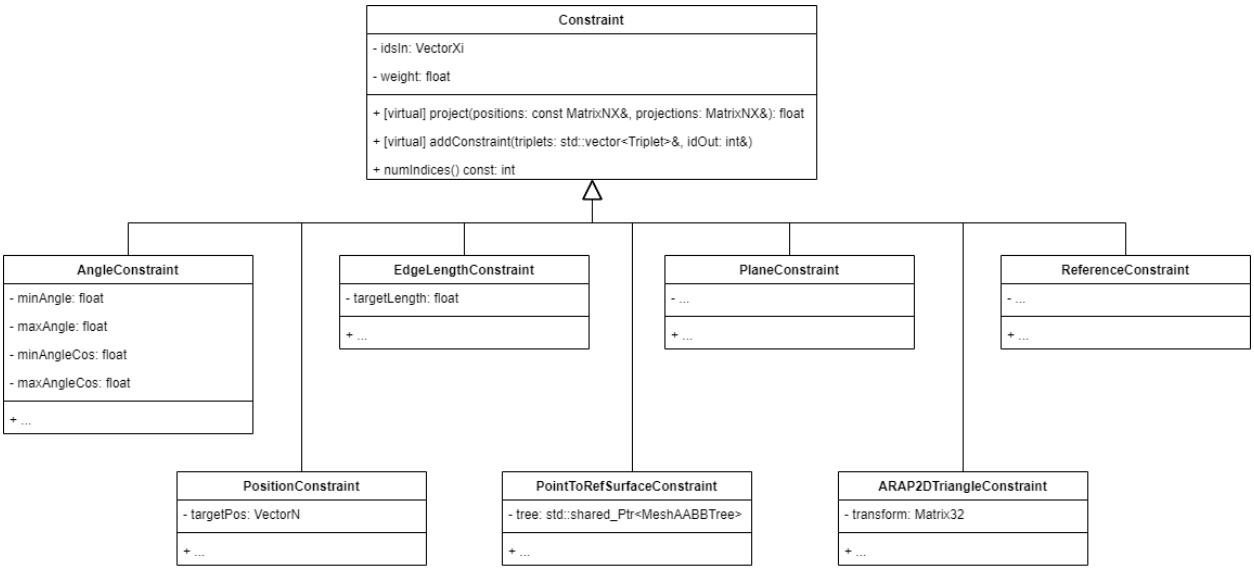


Figure 7: The class diagram of the constraints.

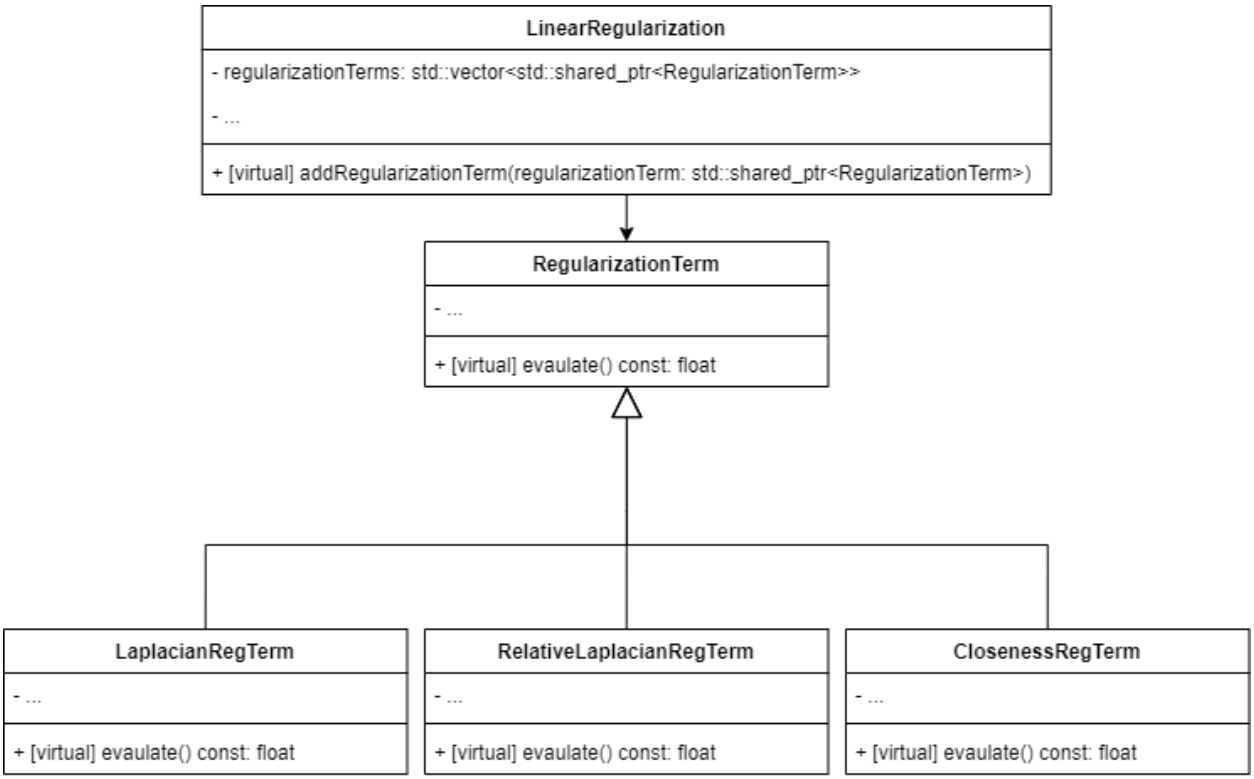


Figure 8: The class diagram of the regularization terms.

The high-level algorithm is implemented in the level of solver classes. Figure 9 shows the class diagram of the solver. Since we want to solve the system with symmetric positive definite matrix and apply Anderson acceleration, we need these solver as other classes.

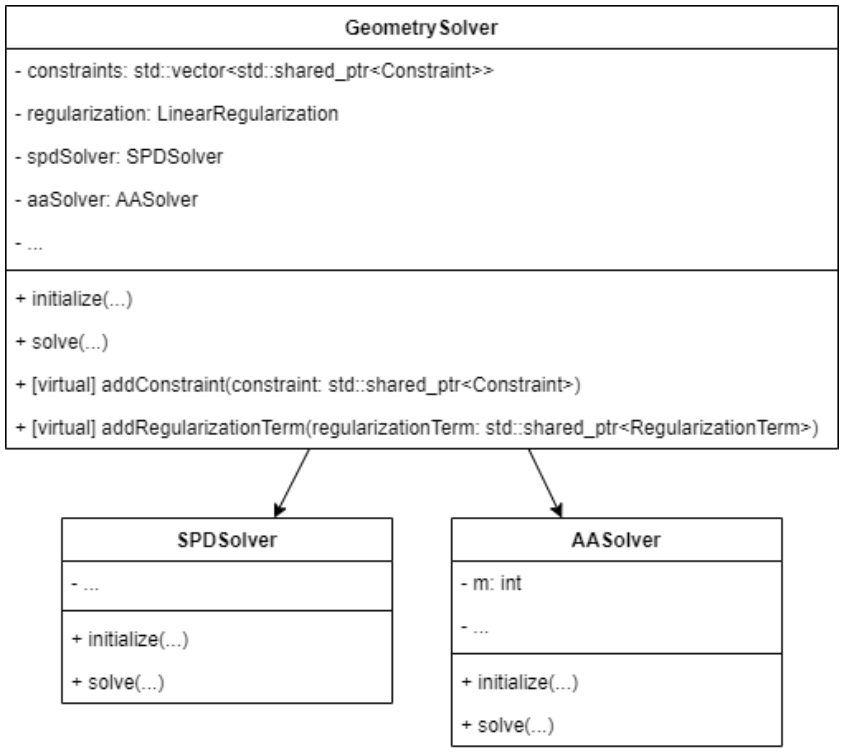


Figure 9: The class diagram of the solver.

- **Third-party software**
  We can solve sparse positive definite linear systems with fixed matrices with the help of the Eigen library. To make use of the AABB tree construction, we may also use the libigl library for convenience. When implementing the CUDA version of the algorithms, we will get the help of the Thrust library, the cuBLAS library, and the cuSOLVER library. We will also use the GLM library for convenience.

2.2. Target Platforms

2.2.1 Hardware

We are using Maya 2020 to develop and test this tool. Therefore, the hardware requirements are equivalent to running Maya 2020, as following:

- CPU: 64-bit Intel® or AMD® multi-core processor with SSE4.2 instruction set. Apple Mac models with M series chip are supported under Rosetta 2 mode
- RAM: 8 GB of RAM (16 GB or more recommended)
- Disk Space: 4 GB of free disk space for installation
- Pointing Device: Three-button mouse

To enable GPU operations, users should have NVIDIA graphics card.

2.2.2 Software

The software requirements are equivalent to running Maya 2020, as following:

- Maya 2020 or higher.
- Microsoft® Windows® 7 (SP1), Windows® 10 Professional, Windows 10® version 1607 or higher operating system

To enable GPU operations, users should install CUDA. We recommend users to install CUDA 11 or higher versions.

2.3. Software Versions

2.3.1 Alpha Version Features (first prototype)

For alpha version, we will implement the following features.

- Implement the framework of the local-global solver with Anderson acceleration.
- Implement the GUI framework.
- Implement the planarization operation.

We will provide some test meshes to show how these features work with the help of the visualization in Maya.

2.3.2 Beta Version Features

For beta version, we will implement the following features.

- Implement the wire mesh design operation.
- Implement the ARAP deformation for 2D triangle meshes.
- Try to implement more operations that are friendly to the local-global solver.
- Provide demo scenes and videos.

We will provide some test meshes to show how these operations work. Since we will implement more operations, we can show various samples.

2.3.3 Description of any Demos or Tutorials

To help new users to get used to our tool, we will provide a readme document with some instructions, a demo scene for examples of using the tool with some videos, and some tooltips shown in Maya GUI. In the demo scene, we will show all the operations on different input meshes. In the example videos, we will show how to import meshes, how to adjust the parameters, how to apply nodes, and even how to paint weights for deformation operations.

3. WORK PLAN

3.1. Tasks and Subtasks

Task 1 – Local-global solver framework

Build the framework of the plug-in. The solver will be used in all operations. Duration: 2 weeks

- ***Subtask 1.1. Local-global solver (Xuntong Liang)***

  Implement a unified local-global solver for those operations in our tool. The solver should be implemented as a class.

- ***Subtask 1.2. Anderson acceleration on the solver (Xuntong Liang)***

  Apply Anderson acceleration on the local-global solver. The Anderson acceleration method is introduced in 2.1.1.

- ***Subtask 1.3. Test the efficiency and generality (Xuntong Liang)***

  Test the acceleration effect with those operations and if it's efficient for real-time applications. Test its generality with various constraints cases.

Task 2 – Implement operations

Implement all operations listed below. An operation takes a mesh as input and use solver in task 1 to solve and then output the optimized mesh. Duration: 3 weeks

- ***Subtask 2.1. Planarization (Yiyang Chen)***

  Implement the planarization operation. Planarization is a process to make all the quadrangle faces of a quad mesh plane.

- ***Subtask 2.2. Wire mesh design (Yiyang Chen)***

  Implement the operation of wire mesh design. Wire mesh subject to some geometry constraints.

- ***Subtask 2.3. ARAP deformation (Yiyang Chen)***

  Implement the ARAP deformation for 2D triangle meshes.

Task 3 – Maya integration and GUI

Integrate the program to Maya and create GUI for user. Duration: 1 weeks

- ***Subtask 3.1. Create Maya node for operations (Yiyang Chen)***

  Implement operation nodes inherited from MPxDeformerNode and use the functions and classes created in task 1 and task 2.

- ***Subtask 3.2. Create Maya GUI (Xuntong Liang)***

  Use MEL script to create GUI, so that the user can do things like choosing operations, changing parameters.

Task 4 – CUDA version

Implement a CUDA version of the plug-in. Compute the local-global solver in GPU to further accelerate the process. Duration: 3 weeks

- ***Subtask 4.1. Develop a CUDA version (Xuntong Liang)***

  Implement the CUDA version of the local-global solver framework in task 1. Then the plug-in will use CUDA version if it's available.

- ***Subtask 4.2. Analyze the CUDA version (Xuntong Liang)***

  Test how much faster using CUDA with certain GPU. Test the compatibility with Maya. Come up with a performance analysis.

Task 5 – Optimization

Optimize the program, including debugging, optimizing interactions, improving solver efficiency and so on. Duration: 1 weeks

- ***Subtask 5.1. Debug (Yiyang Chen & Xuntong Liang)***

  Test with various cases to find corner cases and hidden bugs. Fix every bug and deal with each corner case.

- ***Subtask 5.2. Optimize interactions (Yiyang Chen)***

  After using the tool, identify interactions that don't make sense or need improvement. Optimize them to make the tool more user friendly.

- ***Subtask 5.3. Optimize code (Yiyang Chen)***

  Optimize the functions and classes to make them more reasonable and easier to extend. Try to optimize the solver to make it faster.

Task 6 – Document and demo (Yiyang Chen & Xuntong Liang)

Write document, make demo videos and add some tooltips in Maya GUI Duration: 2 days

3.2. Milestones

3.2.1 Alpha Version

For alpha version, we will implement the following features.

- Implement the framework of the local-global solver with Anderson acceleration.
- Implement the GUI framework.
- Implement the planarization operation.

Therefore, task 1 and task 2.1 and part of task 3 must be completed. Part of task 3 refer to the node and GUI related to planarization.

3.2.2 Beta Version

For beta version, we will implement the following features.

- Implement the wire mesh design operation.

- Implement the ARAP deformation for 2D triangle meshes.
- Provide demo scenes and videos.

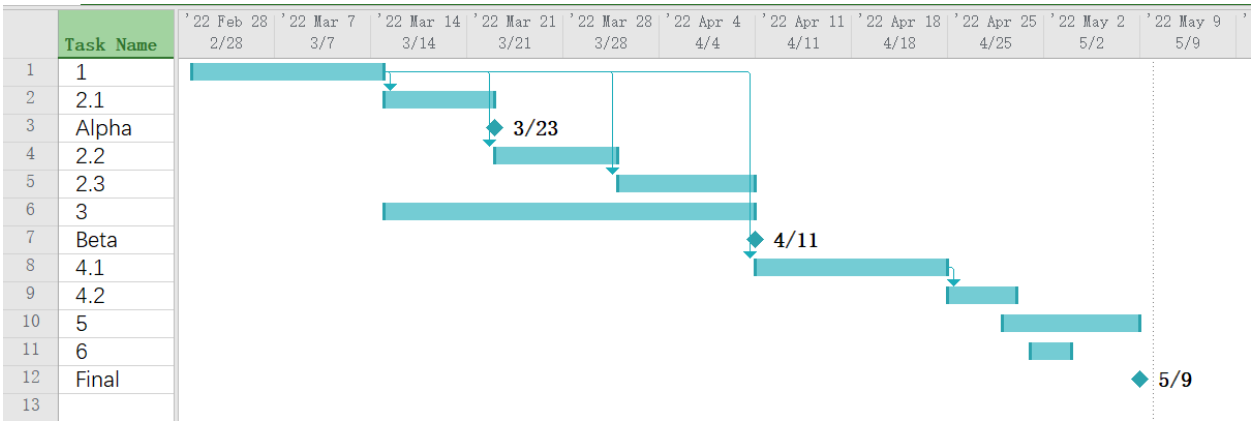
Therefore, task 2.2 and 2.3 and the remaining task 3 must be completed.

3.2.2 Final Version

For final version, task 4, 5, 6 must be completed. All part of the tool should be refined in the final version. In addition, implement some other geometry operations in our reference paper if possible.

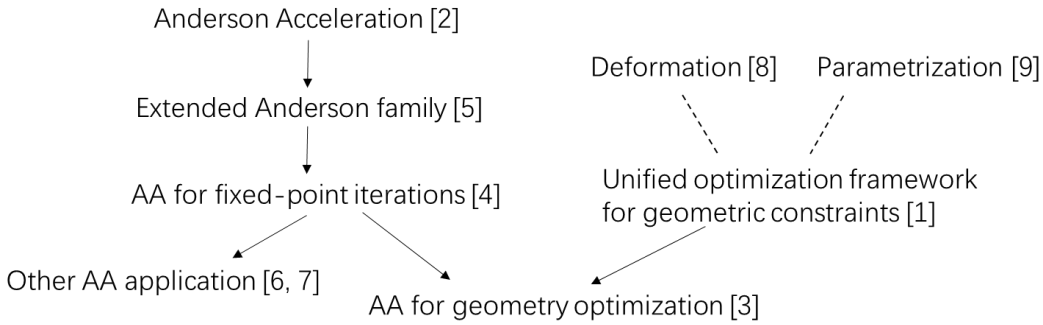
3.3. Schedule

The following is our proposed schedule of tasks. The timetable shows when each task should be completed. Milestones are indicated by the ◆ mark with dates.



4. RELATED RESEARCH

The research evolution is shown in the following graph (AA is Anderson acceleration for short):



[2] In this seminal paper, Anderson proposed a method for iterative solution of nonlinear integral equations. Specifically, the numerical solution of nonlinear integral equations involves the iterative solution of finite systems of nonlinear algebraic or transcendental equations. And the paper proposed some procedures that can accelerate the convergence of iterative vector sequences, which the author discussed that may prove useful in a wider context than the solution of a particular class of nonlinear integral equations.

Later, Pulay [10][11] introduced the same technique for stabilizing and accelerating the convergence of self-consistent field method in quantum chemistry computation, which proves to be effective. Because it's not the field of computer graphics, we won't go into detail here.

[5] The paper presents two classes of multisecant methods which allow to take into account a variable number of secant equations at each iteration. The first is the Broyden-like class, of which Broyden's family is a subclass, and Anderson acceleration is a particular member. The second class is that of the nonlinear Eirola–Nevanlinna-type methods. In the paper, the author clarified a remarkable relationship of Anderson acceleration to quasi-Newton (secant updating) methods and extended it to define a broader Anderson family of acceleration methods.

[4] Based on the paper by H. Fang and Y. Saad [5], the paper shed additional light on Anderson acceleration and to draw further attention to its usefulness as a general tool. They first prove that the Anderson acceleration method is "essentially equivalent" to some other methods under certain conditions to further illustrate it. Then they discuss practical considerations for implementing Anderson acceleration and illustrate its performance through numerical experiments involving a variety of applications. The application papers [3][6][7] all refer to the paper and use models and formulas from it.

[6] The paper explores the use of Anderson acceleration to improve the convergence of fixed-point iteration in Uzawa algorithm, which is an iterative method for the solution of saddle-point problems, which arise in many applications, including fluid dynamics.

[7] The paper applies Anderson acceleration on the numerical solving of advection-diffusion problems. Specifically, the paper considers the solution of systems of nonlinear algebraic equations that appear in a positivity preserving finite volume scheme for steady-state advection-diffusion equations and proposes and analyzes numerically an efficient strategy for accelerating the Picard method when it is applied to these systems. The strategy is based on the Anderson acceleration and the adaptive inexact solution of linear systems.

[1] In this seminal paper, they introduce a unified local-global optimization framework for geometry processing based on shape constraints. These constraints preserve or prescribe the shape of subsets of the points of a geometric data set, such as polygons, one-ring cells, volume elements, or feature curves. The framework is constructed by formulating a target function that is the weighted sum of squared distance from the constrained elements to their feasible shapes, with auxiliary variables representing the closest feasible configurations.

We think it's a seminal paper because it's a unified framework that can be applied to solve a large variety of different problems in geometry processing which have typically been solved by specialized optimization algorithms tailored to certain application domains. The approach is most closely related to the as-rigid-as-possible deformation method of Sorkine and Alexa [8] in that they also employ a two-step optimization strategy for shape constraint optimization. The paper [8] aims at deforming objects, but in this paper, they unify, formalize, and extend this concept.

The paper shows a wide range of applications. For example, deformation and parametrization are two prominent applications where preservation of certain geometric features is crucial. Specialized methods like near-isometric and quasi-conformal methods for deformations [8] or parametrization [9] have been proposed, while the unified framework in this paper can also solve these problems.

[3] This is the paper our authoring tool is based on. According to the paper, it's the first time to explore the use of Anderson acceleration within the computer graphics community. The paper uses the Anderson acceleration method [4] to speed up the convergence of the local-global solver and address the stability issue. And the local-global solver is used in geometry optimization [1] and physics simulation [12]. Moreover, their technique is effective beyond classical local-global solvers and can be applied to iterative methods with a common structure.

Because of the potential workload, we will mainly implement the Anderson acceleration for only geometry optimization in our authoring tool.

5. RESULTS

5.1. Documentation/Tutorial

For planarization:

1. Load a model (and a reference model) into Maya
2. Select the model (and the reference model in order) and click the “planarization” menu item.
3. (optional) Change parameters in attribute editor or the parameter window of the node. “Num Iteration” means the number of iteration in the local-global solver. The larger of the number, the better of the mesh after operation, but the speed will be slower. And those parameters of weight can affect the result. “Planarity Weight” is easy to understand. “Closeness Weight” means how close the result is with reference mesh. “Fairness Weight” and “Relative Fairness Weight” represent the fairness and smoothness of the result.
4. Apply operation. Then you can drag the “envelop” slider of the attribute editor of the node to see the difference between input and output.
5. Export the model

For ARAP deformation:

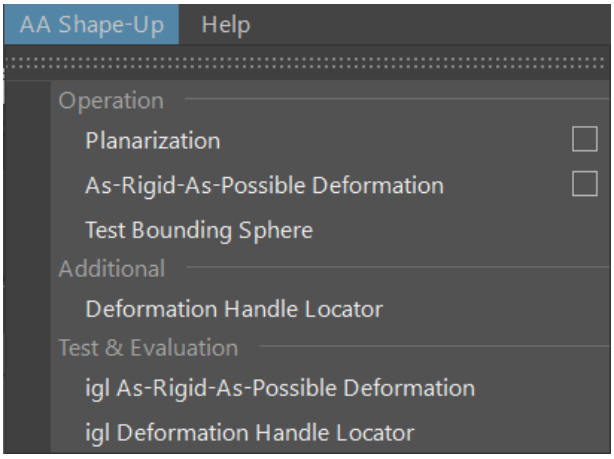
1. Load a model into Maya
2. Select the model and click the “ARAP deformation” menu item.
3. Change to vertex mode and select handle vertices, and then click the “Deformation handle locator” menu item.
4. (optional) Change parameters in attribute editor or the parameter window of the node. “Num Iteration” means the number of iteration in the local-global solver. The larger of the number, the better of the mesh after operation, but the speed will be slower.
5. Move some handles to deform
6. Export the model

P.S. See the UI images in 5.2.1 for better understanding.

5.2. Content Creation

5.2.1 User Interface

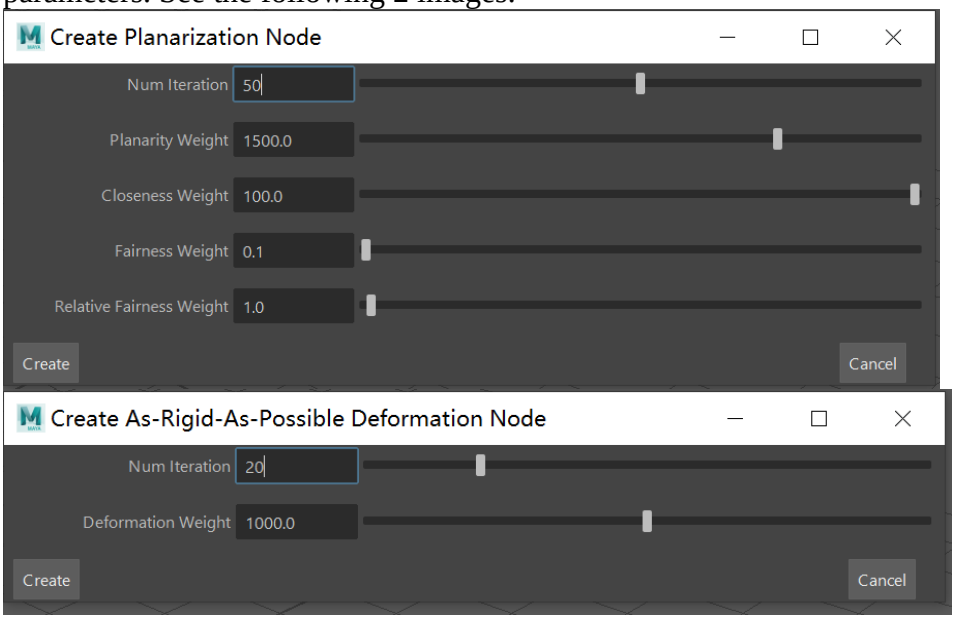
Menu



The “Test Bounding Sphere” is a simple operation that change the model to a sphere, which is just for test. The last 2 menu items are for test and

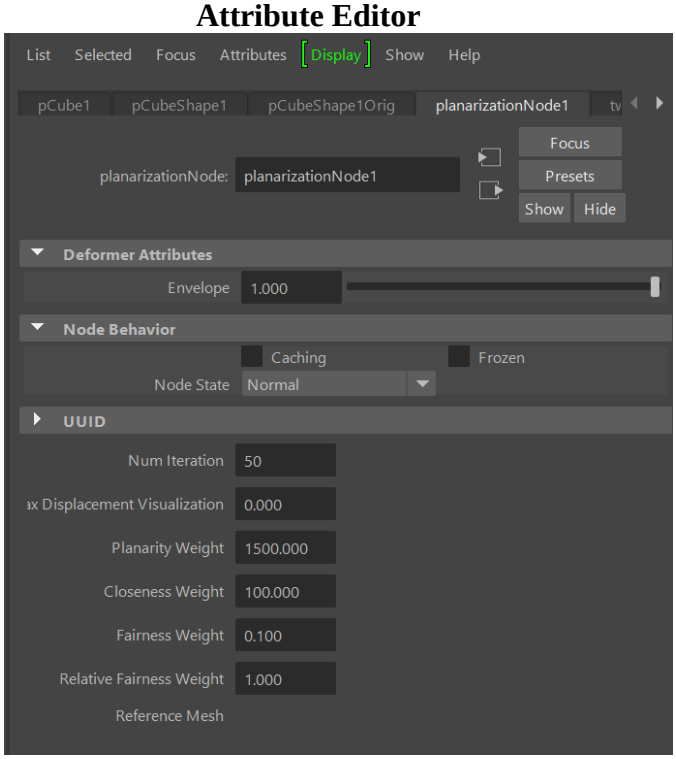
evaluation: we add an ARAP deformation using built-in function in libigl to compare the result of our implementation and that with libigl.

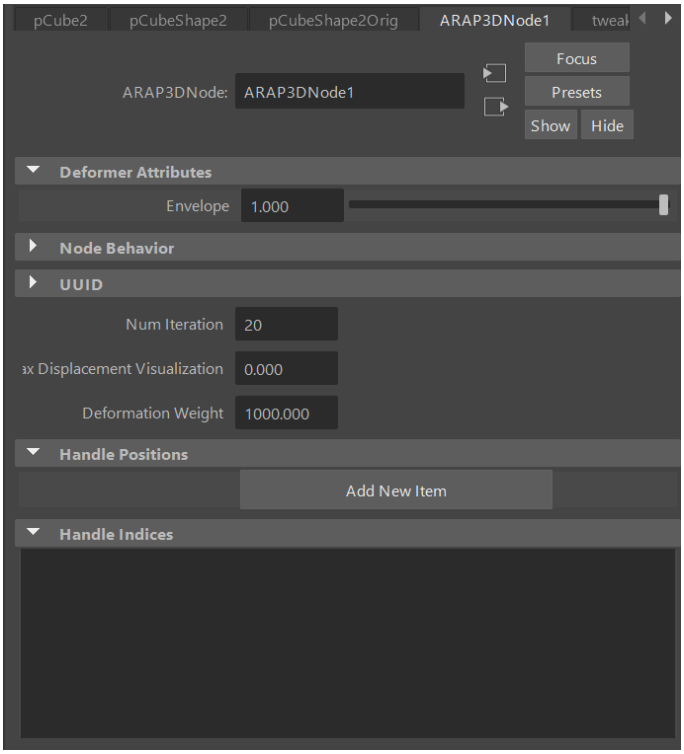
If you click the box after first two operations, you can change some parameters. See the following 2 images.



"Num Iteration" means the number of iteration in the local-global solver. The larger of the number, the better of the mesh after operation, but the speed will be slower. And those parameters of weight can affect the result.

Attribute Editor





Most parameters are the same. In the ARAP3DNode, the “Handle Positions” and “Handle Indices” show the handle information, and we recommend the user to add handles by selecting vertices on model rather than inputting here directly.

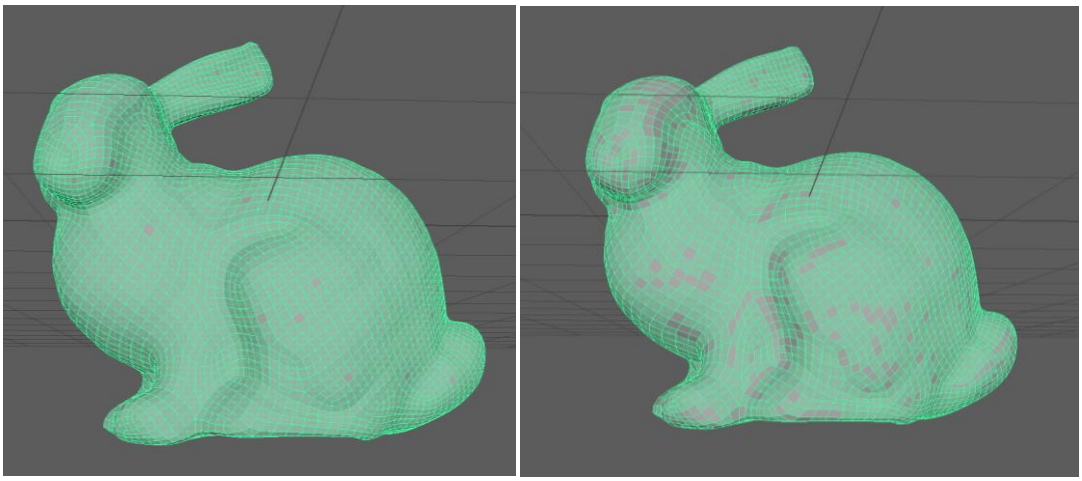
For the “Envelope” slider, if the number is 1, it shows the model after deforming, and if the number is 0, it shows the model before deforming.

5.2.2 Workflow

See 5.1

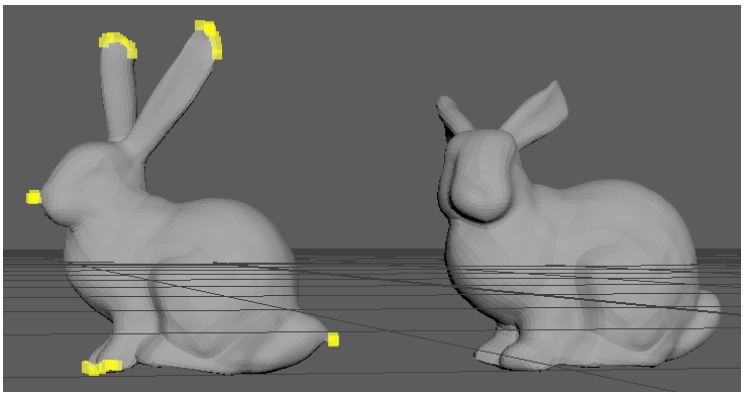
5.2.3 Typical output

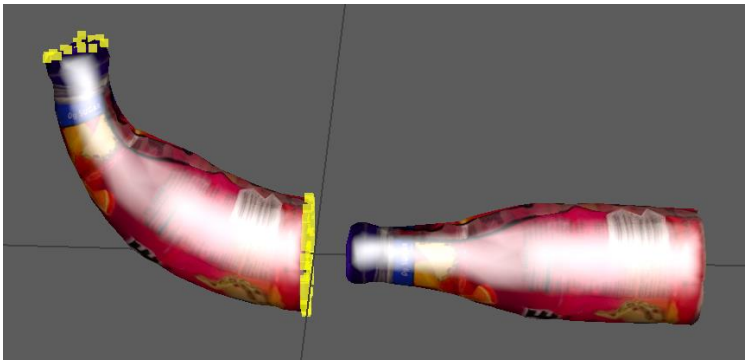
Planarization



Left is the origin model and right is the result. Green face means not plane, it’s obvious that there are more plane faces after planarization.

ARAP deformation

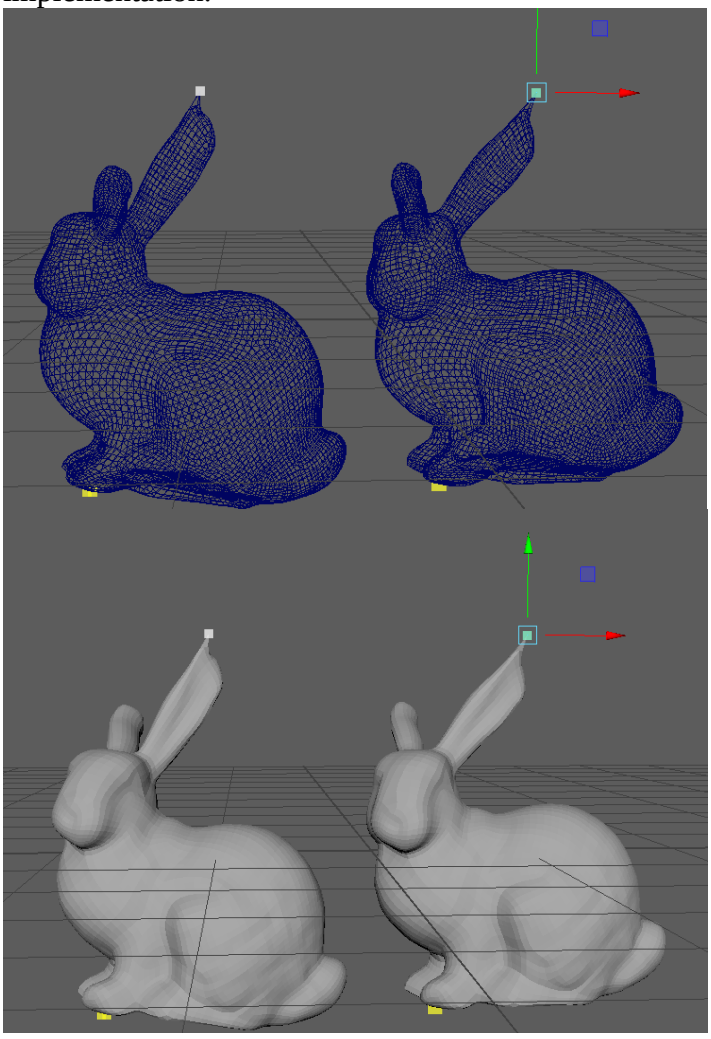


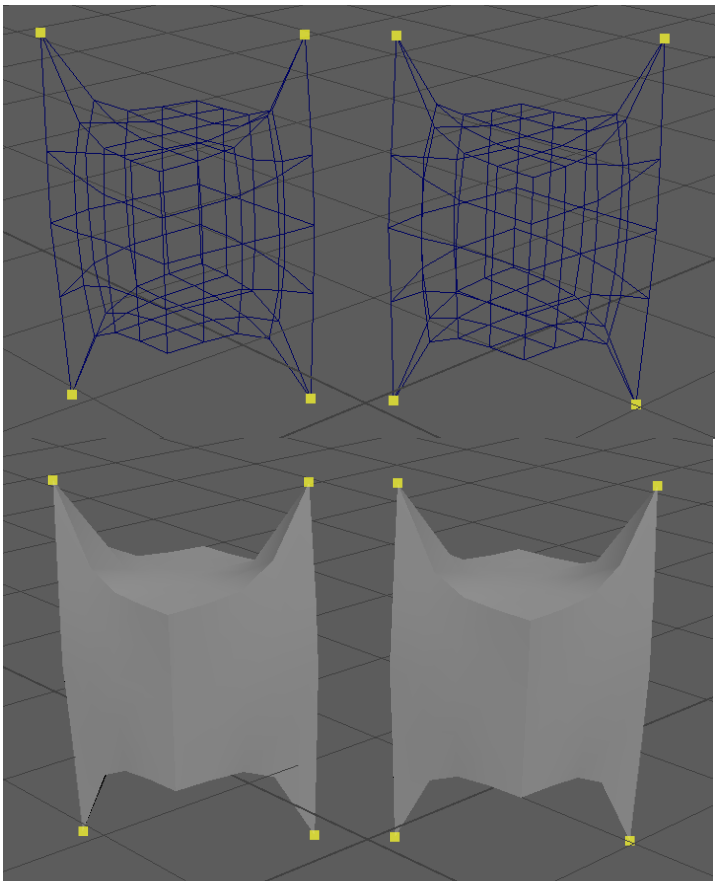


Right is the origin model and left is the result. The yellow points are handles.

Comparison

We add an ARAP deformation using built-in function in libigl to compare the result of our implementation and that with libigl. Images below show that our implementation is validate. Left is libigl and right is our implementation.





6. DISCUSSION

6.1. Accomplishments

6.1.1 Features that were implemented (all are worked)

1. Local-global solver framework. We then add different constraints to implement different operations.
2. Planarization operation. Make the face of the input quadrilateral mesh more “plane”.
3. ARAP deformation operation. Deform a mesh as rigid as possible.
4. Anderson acceleration. Successfully accelerate the local-global solver.
5. Maya plugin. Add a menu in Maya so that the user can select the model and then click the menu to apply operations and change parameters.
6. Standalone application (with only planarization operation). You can load an obj file, then change parameters and apply operations. You can choose render type which shows various info of the model.

6.1.2 Features that were not implemented

1. Wire mesh design operation

We think the other two operations are more meaningful in this authoring tool, so we focus on the other two operations, especially ARAP deformation.

2. CUDA version

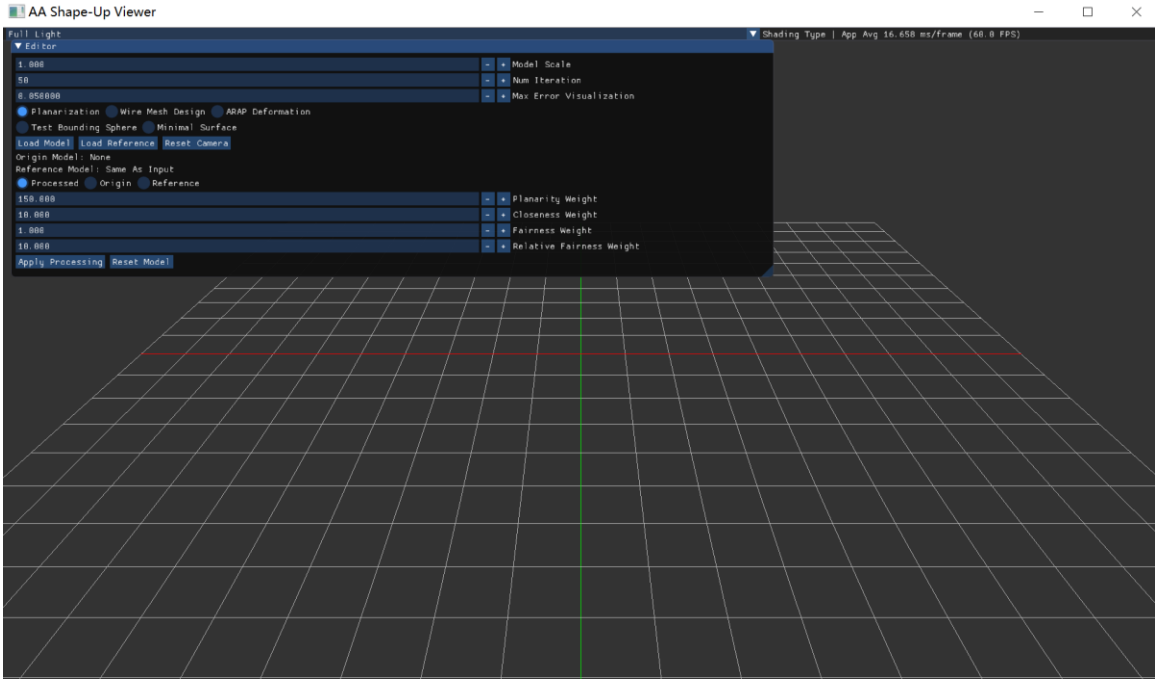
We find that it may cost too much time to implement a CUDA version, so we choose to focus on the functionality of the authoring tool.

6.2. Design Changes

There aren't any changes to the technical approach, but we change some details of implementation and add some features which are also design changes.

We changed some of the data structure to make them more extendable. We design template classes to support component pattern with static polymorphism. For example, we design separate interfaces for the function of constraints instead of just defining functions, such as generating triplets of sparse matrices and computing constraint projections. We found that even though the template classes are kind of difficult to debug, the new constraints are easier added.

We add a standalone application which can do the planarization operation (deformation is too complex to implement without the model modification in Maya) and show how plane each face is. We showed the application in our alpha version demo. We prompted the design change because the standalone application can help us debug the framework. See the image below.



6.3. Total Man-Hours worked

Yiyang Chen: 95 hours
Xuntong Liang: 99 hours

6.4. Future Work

1. **Further accelerate it with CUDA.** Currently if the model has a large number of faces and vertices, it's not real-time when moving the handle, which may affect usability.
2. **Optimize the handle selection.** Currently we use the vertex mode in Maya to select handle, which may be inconvenient when you need to select many handles in all parts of the model.
3. **Add more parameters in ARAP to customize results.** Artists may need to customize "rigid" to produce different results when deform. Although the method in our reference paper doesn't have parameters to control "rigid", we may try to modify it.

6.5. Third Party Software

We solve sparse positive definite linear systems with fixed matrices with the help of the Eigen library. To make use of the AABB tree construction, we use the libigl library for convenience. We use tetgen to generate tetrahedral mesh from surface mesh to deform it with ARAP method. In the standalone application, we use OpenGL to render models, use tinyobj to load obj files, and use imGUI to create GUI.

7. LESSONS LEARNED

1. **Tetgen.** Tetgen is a useful library to generate tetrahedral mesh from surface mesh. We learned to use it by reading documentation and source code.
2. **Maya API.** The Maya API is very complicated. We learned to find target nodes by listing connections and relatives. Also, we learned how to make custom locators to set handles of ARAP deformation. However, there are some features that are deprecated in some legacy Maya tutorials, such as the rendering of the custom locators.

References

[1] Bouaziz, Sofien, et al. "Shape-up: Shaping discrete geometry with projections." Computer Graphics Forum. Vol. 31. No. 5. Oxford, UK: Blackwell Publishing Ltd, 2012.
[2] Anderson, Donald G. "Iterative procedures for nonlinear integral equations." Journal of the ACM (JACM) 12.4 (1965): 547-560.
[3] Peng, Yue, et al. "Anderson acceleration for geometry optimization and physics simulation." ACM Transactions on Graphics (TOG) 37.4 (2018): 1-14.
[4] Walker, Homer F., and Peng Ni. "Anderson acceleration for fixed-point iterations." SIAM Journal on Numerical Analysis 49.4 (2011): 1715-1735.
[5] Fang, Haw-ren, and Yousef Saad. "Two classes of multisecant methods for nonlinear acceleration." Numerical linear algebra with applications 16.3 (2009): 197-221.
[6] Ho, Nguyenho, Sarah D. Olson, and Homer F. Walker. "Accelerating the Uzawa algorithm." SIAM Journal on Scientific Computing 39.5 (2017): S461-S476.
[7] Lipnikov, Konstantin, Daniil Svyatskiy, and Y. Vassilevski. "Anderson acceleration for nonlinear finite volume scheme for advection-diffusion problems." SIAM Journal on Scientific Computing 35.2 (2013): A1120-A1136.
[8] Sorkine, Olga, and Marc Alexa. "As-rigid-as-possible surface modeling." Symposium on Geometry processing. Vol. 4. 2007.
[9] Lévy, Bruno, et al. "Least squares conformal maps for automatic texture atlas generation." ACM

transactions on graphics (TOG) 21.3 (2002): 362-371.
[10] Pulay, Péter. "Convergence acceleration of iterative sequences. The case of SCF iteration." Chemical Physics Letters 73.2 (1980): 393-398.
[11] Pulay, Peter. "Improved SCF convergence acceleration." Journal of Computational Chemistry 3.4 (1982): 556-560.
[12] Bouaziz, Sofien, et al. "Projective dynamics: Fusing constraint projections for fast simulation." ACM transactions on graphics (TOG) 33.4 (2014): 1-11.