Simulation of the interactions between a surgical tool and elastic tissue

Student: Fabian Prada Nino
Advisor: Austin Reiter

September 30, 2017

1 Introduction

In this project we developed a toolbox to simulate the interactions between a surgical tool (rigid shape) and sinus tissue (elastic shape). Our toolbox provide intuitive user interfaces to perform the following tasks:

1. Specify a region of interest and a path within an anatomical model.
2. Design a surgical tool.
3. Simulate tool-tissue interactions when moving the tool along the path.

Our goal is to compute a trajectory of a surgical tool from its initial pose to a target position that minimizes collisions with the surrounding tissue. Ideally, we would like to have a trajectory that is collision-free, however, such trajectory might not exist given the shape of the tool and the anatomic model. Instead, we will tolerate moderate collisions which we will try to minimize.

Our simulation moves the tool according to an extrinsic vector field that pushes the tool towards the target position while keeping it away from the tissue. When collisions are detected our simulation deforms the elastic shape to return to a collision free state.

Instructions on the installation of our toolbox for Windows and Linux systems are provided on Section 2. Description of users interface is presented on Section 3. We provide a brief overview of the implementation on Section 4, and evaluation on a simple set of tools models on Section 5.

2 Installation

The code of this project can be accesed from:

`https://github.com/FabianRepository/SinusProject`

The following is the list of external libraries (open source!) that are used within our toolbox. We present a brief description of their usages:

- Ceres [1]: Non linear solver.
- Embree [8] : Ray tracer.
- PNG [6] : Image storage.
- LibIGL [3] : 3D Geometry.
- Eigen [2] : Sparse linear solver.
- Fast Geodesics [7]: Geodesics on triangle meshes.
- Misha's library [4] : Visualization, image and geometry processing.

The provided code was compiled and tested on Windows and Linux systems. For the Windows system, the code was compiled on Visual Studio 2017 and executed on Windows 10. For the Linux system, we used GCC 5.4 and Ubuntu 16.04.

LibIGL, Eigen, Fast Geodesics and Misha's library are headers libraries and do not require any compilation. The code of these libraries is already included within our distribution. These are the system specific instructions to install our toolbox:

- Windows: In the folder */ForWindows* we provide the header files and libraries for Ceres, Embree and PNG. Copy the folders */Ceres*, */embree2*, */glog*, */PNG*, and */ZLIB* into */Code/include* and the uncompressed wlibs.zip into */Code*. Now we are ready to compile the code: just open the provided solution in Visual Studio and build it. Finally, copy the dynamic libraries in the wdlls.zip to the */x64/Release* folder before you run the applications.

- Linux: We do not provide compiled versions of Ceres, Embree or PNG for Linux. On Ubuntu, PNG can be directly installed from command line as `sudo apt-get install libpng-dev` (make sure the version installed is libpng12). We compiled and installed the source code for Ceres and Embree as described in their project pages (this will install the headers in */usr/local/include* and the libraries in */usr/local/libs*). To compile the code of our toolbox we provide a Makefile for each application: just go to the folder of each aplication an run `make`. This will create the binay for each application in */Code/Bin*.

3 User Interface

Our toolbox is composed by four different applications: `SelectPathAndROI`, `SelectFixedVertices`, `ToolBuilder`, and `Simulator`. Each application provide a command line interface and graphic interface to the user. To see the options of the command line interface just call the program with no arguments. In this section, we present a brief description of the usage and options of each application.

3.1 Path and ROI selection

The `SelectPathAndROI` program allows you to specify the region of interest (ROI) on the elastic shape, and the (oriented) path that the rigid tool is intended to traverse. The input to this program is a triangle mesh in .ply format[1]. Run this program from command line as follows:

```
SelectPathAndROI.exe --input nose.ply
```

[1]Use some external tool like Meshlab to convert from your mesh format to .ply.

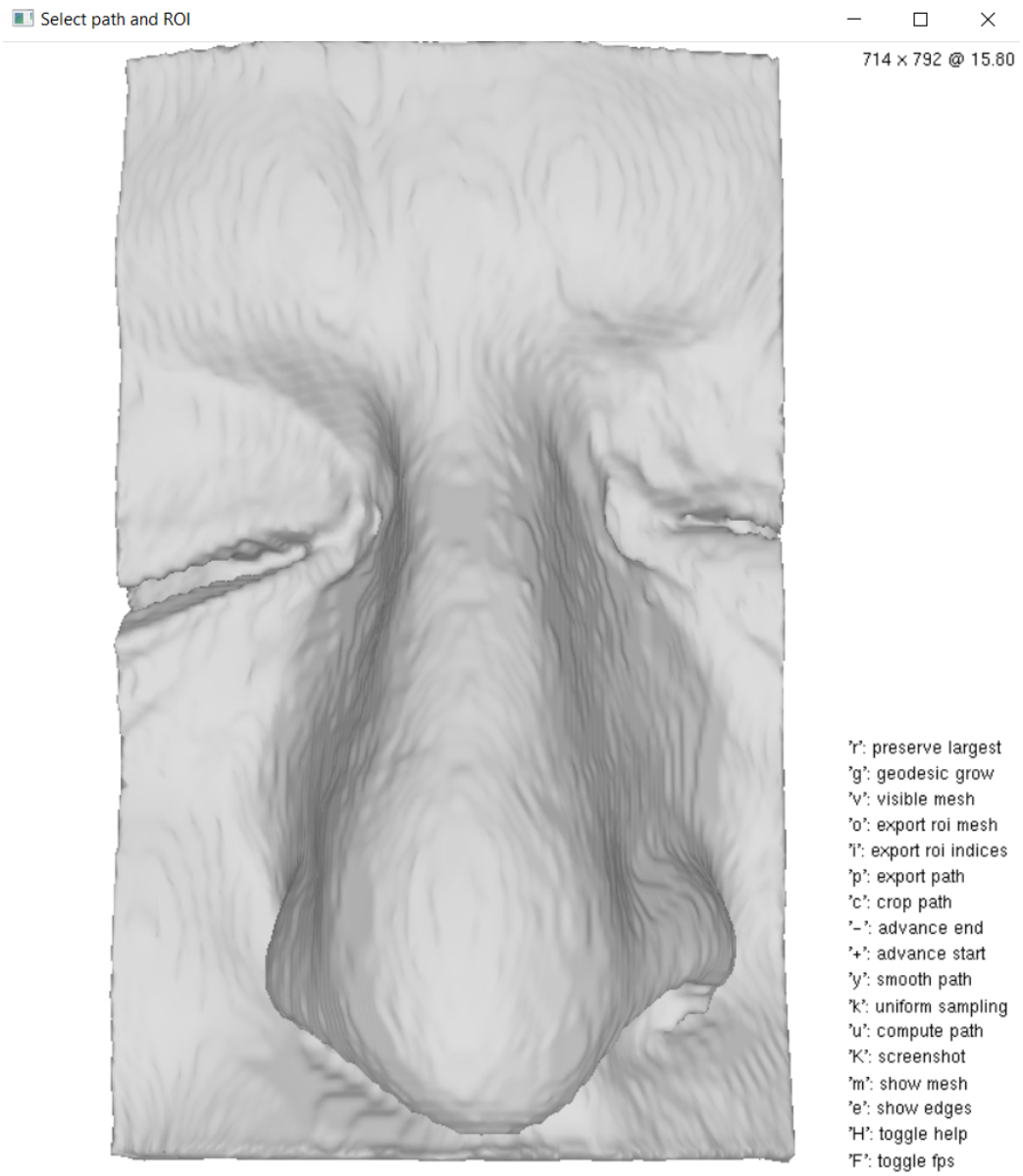

Figure 1: `SelectPathAndROI` user interface.

This command will initialize the user interface showed in Figure 1. The interface provides a rendering of the input mesh together with a list of commands displayed on the right side of the window. You can move the camera

by keeping pressed one of the mouse bottons. Use the left button for translation, right button for rotation and CTRL + left button for zoom in. To select the path and ROI from the user interface follow the next instructions (also illustrated in Figure 2):

a) Select a source node (blue) using SHIFT + left button, and a target node (red) using SHIFT + right button.

b) Press 'u' to get the shortest path between source and target nodes (use 'm' to enable/disable the mesh view).

c) Press 'y' to successively align the path with the medial axis of the shape.

d) Press '+' (resp. '-') to move the source (resp. target) node apart from its initial position along the computed path.

e) Press 'c' to crop the path, keeping only the portion within the adjusted source and target nodes.

f) Press 'v' to identify the portion of the mesh that is visible from the computed path. You will be prompted to specify the threshold distance for which points despite being visible should be discarded (default is 18mm).

g) Press 'g' to apply a geodesic dilation of the selected region. You will be prompted to specify the radius of dilation (default is 0.5mm).

h) Using ALT + left button you can manually select additional regions to be included in the ROI[2].

i) Using ALT + right button you can manually select regions to be deleted from the ROI.

You can keep the largest connected component of the ROI by pressing 'r'. Finally, you can export the ROI by pressing 'o' (we will denote the output ROI as `roi.ply` in the following examples).

[2]In Linux , the ALT key might be already mapped to other functionality so you would need to change the keyboard configuration

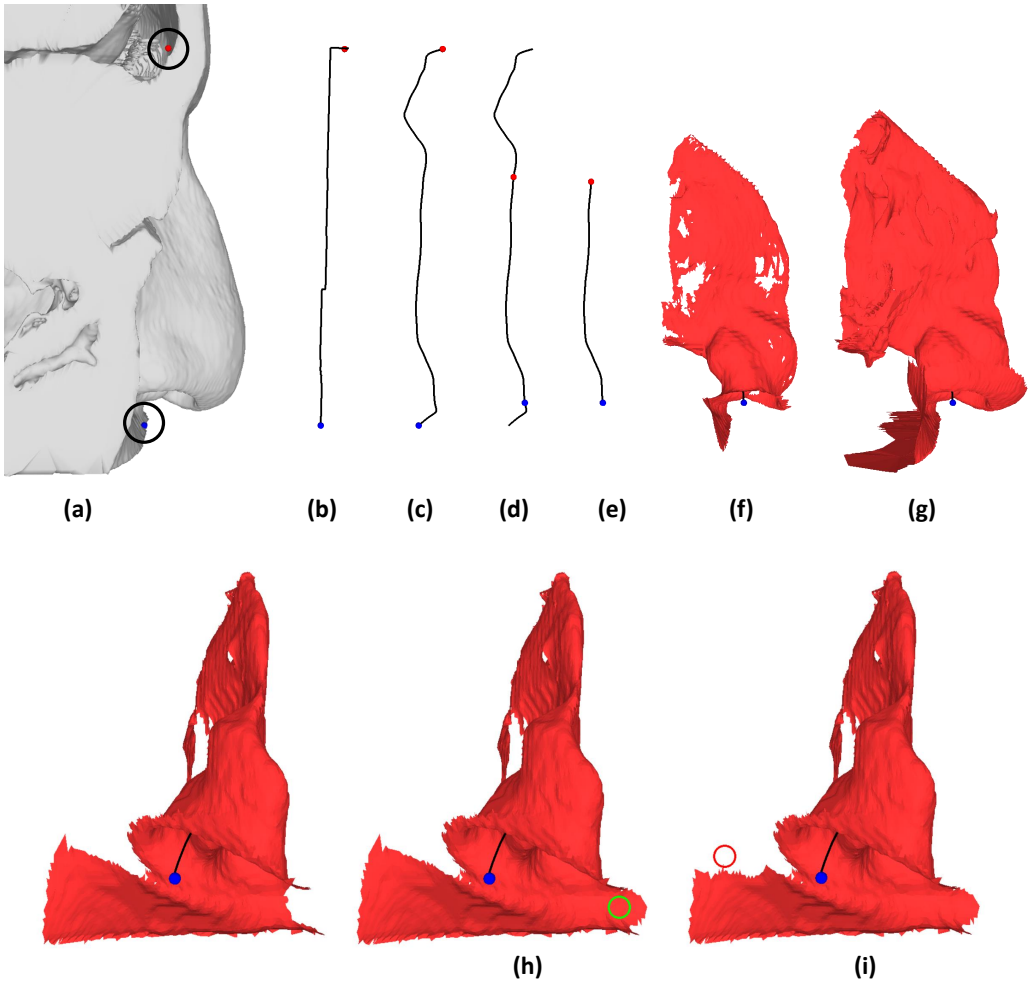

Figure 2: Steps to select path and ROI.

You can also resample the path by pressing 'k' (default is 20) and export it by pressing 'p' (we will denote the output path as `path.bin` in the following examples).

3.2 Fixed regions selection

The `SelectFixedVertices` program allows to specify vertices on the ROI that will be keep fixed during the simulation stage. The inputs to this program are the ROI (necessary) and a surrounding mesh associated to rigid

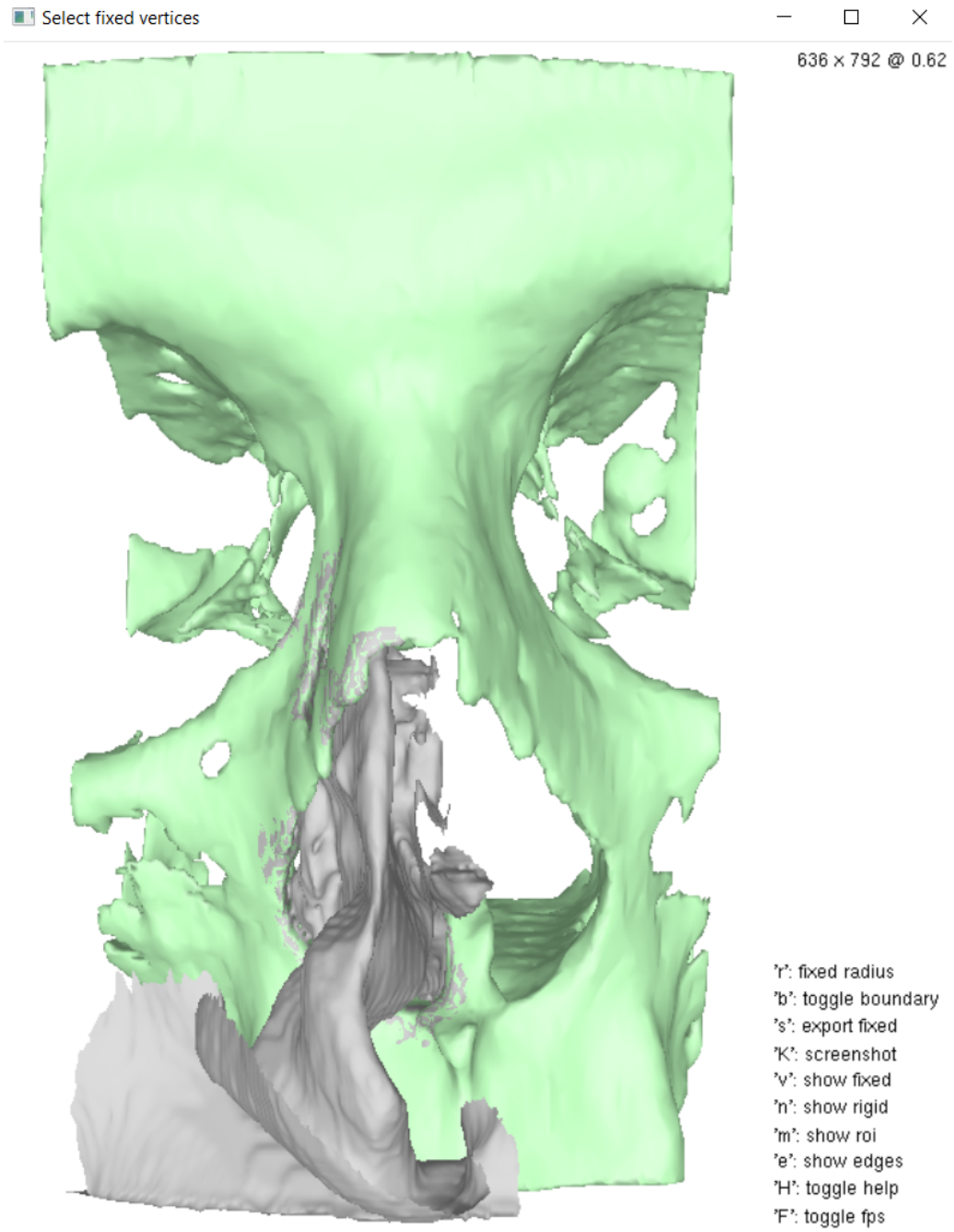

Figure 3: `SelectFixedVertices` user interface.

structures (optional, and referred as `bone.ply` in the next example). Running this program from command line,

```
SelectFixedVertices.exe --roi roi.ply --fixed bone.ply
```

provide you the user interface shown in Figure 3. To specify the set of vertices that can be fixed by this applications follow the next steps (see also Figure 4 for guidance):

a) Press 'r' to specify the radius of the region of influence of the rigid mesh. All the vertices in the ROI that are within this radius will be marked as fixed vertices (default value is 0.1 mm).

b) Press 'b' to mark/unmark the vertices in the boundary of the ROI as fixed vertices.

c) Using SHIFT + left button you can manually select vertices to be marked as fixed.

d) Using SHIFT + right button you can manually delete fixed vertices.

Finally, use 's' to export the set of selected fixed vertices (we will denote the output fixed region as `fixed.bin` in the following examples).

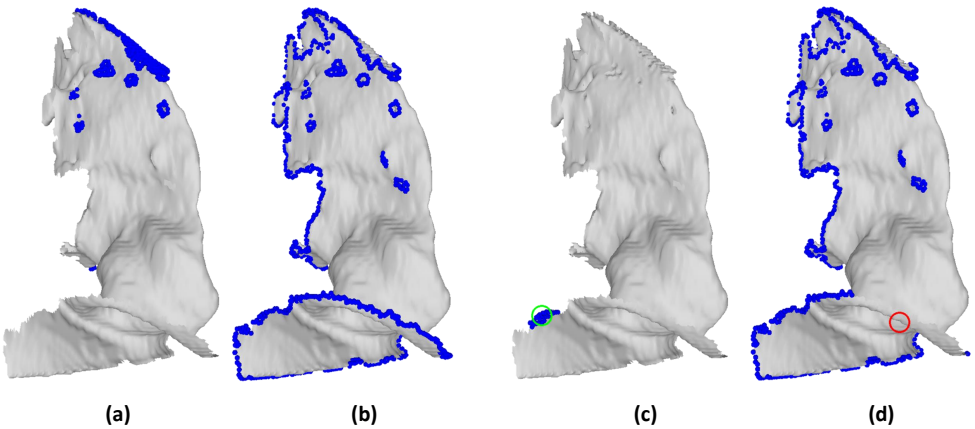

Figure 4: Steps to select fixed vertices.

3.3 Tool design

The `ToolBuilder` program provides you a simple interface to design a tool as observed in Figure 5. Run this program from command line as follows:

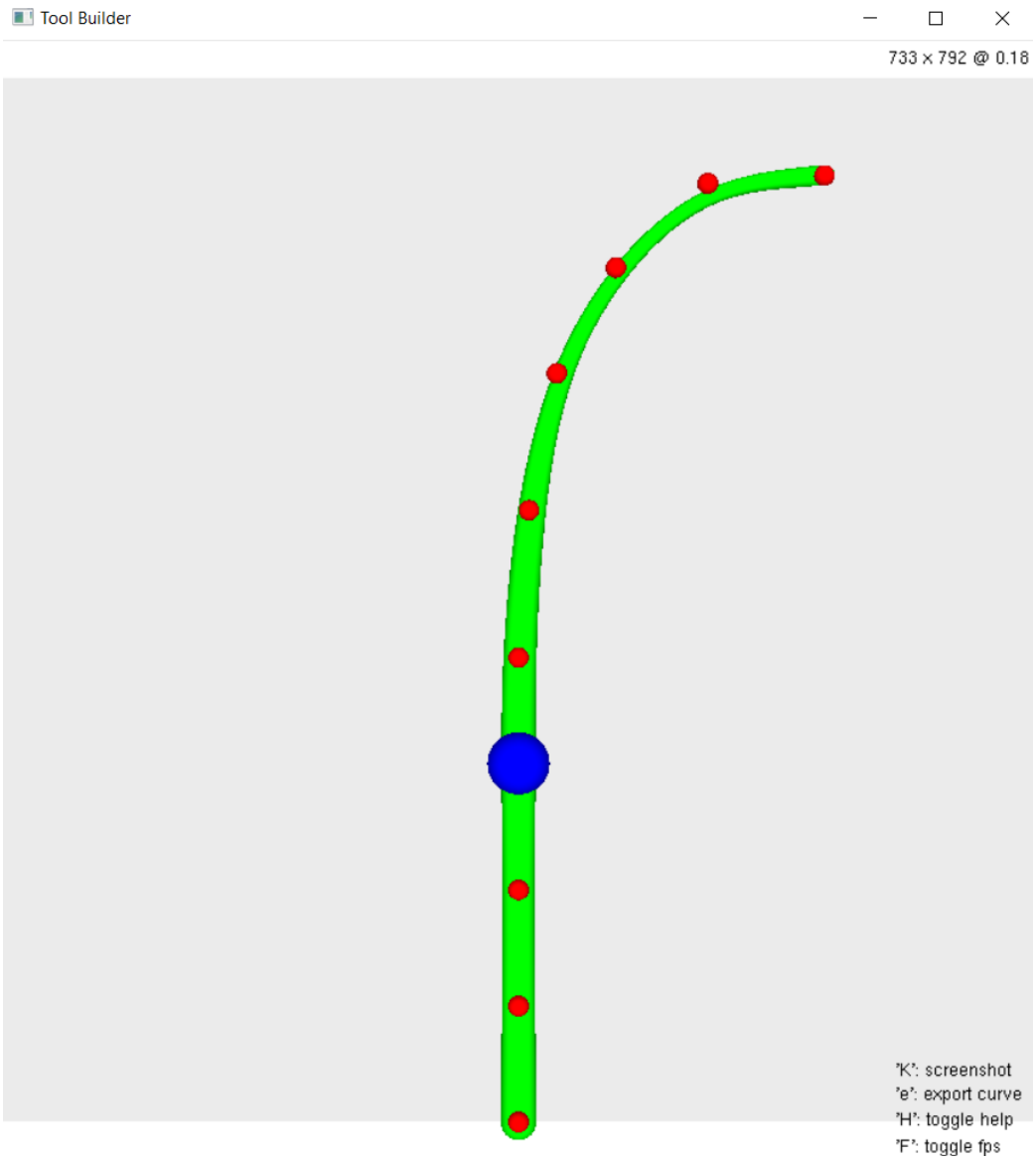

Figure 5: `ToolBuilder` user interface.

```
ToolBuilder.exe --nodes 10 --outSamples 20 --scale 50
```

The parameter `--nodes` specifies the number of control points used during the design process (default is 10),`--outSamples` specify the total number of samples that will be exported from the final design (default is 20), and `--scale` specifies the output size of the tool (default is 50mm).

Use SHIFT + left button to select a control point. Keeping SHIFT + left (resp. right) pressed and moving the cursor you can translate the control point in a perpendicular (resp. parallel) direction to the viewing direction. Pressing '+' (resp. '-') will increase (resp. decrease) the radius of the tool at the control position. Repeat this operation on other control points. To export the final result press 'e' (we will denote the output tool as `tool_positions.bin` and `tool_radius.bin` in the following examples).

3.4 Simulator

The `Simulator` program provides you an interface to visualize the interaction between an elastic shape (ROI) and a rigid tool that is traversing the target path. From command line you can specify multiple configuration parameters for this program. Run the program for the first time as follows:

```
Simulator.exe --input roi.ply --fixed fixed.bin
--toolPos tool_positions.bin --toolRad tool_radius.bin --path path.bin
```

From the provided interface you can visualize and execute each of the steps of a simulation that moves the rigid tool along the specified path. This is done as follows (see Figure 7 for guidance):

a) Use SHIFT + left button to translate the tool and SHIFT + right to rotate it. Once you have identified and inital position for the tool you can save it by pressing 'w' (we will denote the output tool pose as `pose.bin`). You can also use the mouse buttons to set the camera position. Export a desire camera view by pressing 'S' (we will denote the output camera view as `camera.bin`).

b) Press 'v' to advance the tool in the direction of a extrinsic vector field.

c) Press 'k' to move the tip of the tool towards the path center.

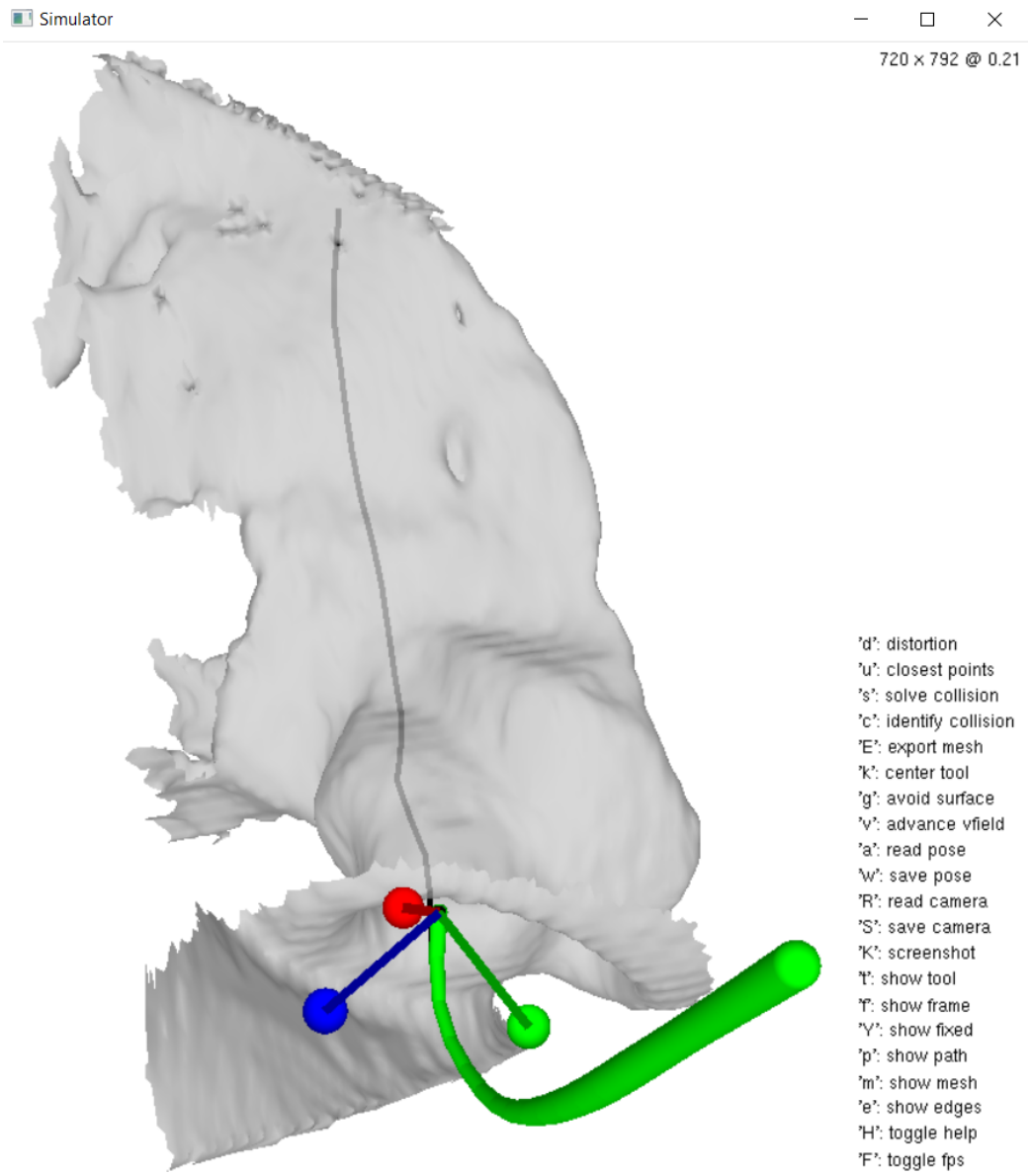

Figure 6: `Simulator` user interface.

d) Press 'u' to identify the closest points between the tool and the ROI.

e) Press 'g' to move the tool away from the surface.

f) Press 'c' to identify collisions between the tool and the surface.

g) Press 's' to deform the elastic shape away from the tool.

h) Press 'h' to visualize the distortion on the surface after the deformation. If you want to export the deformed surface to a .ply mesh press 'E'.

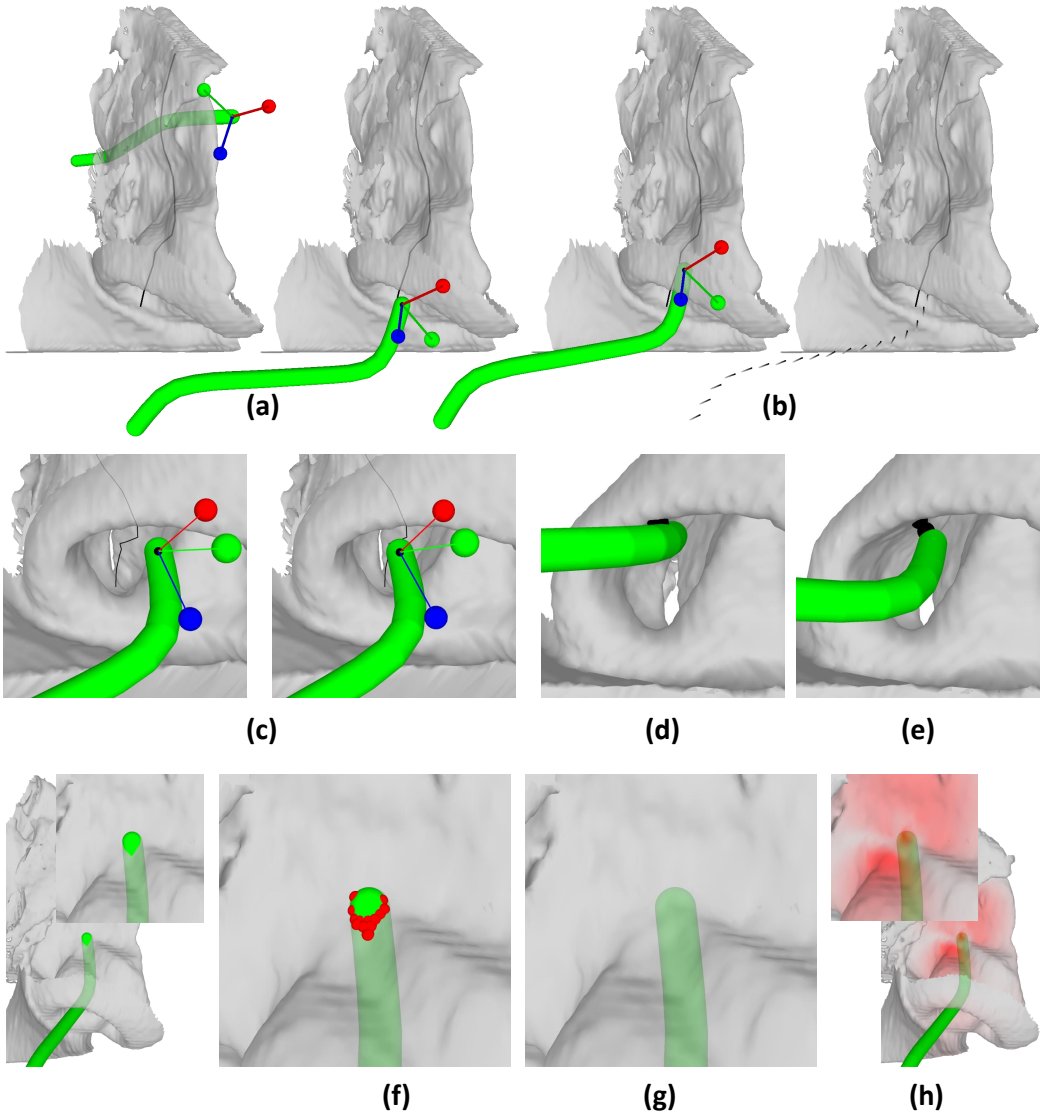

Figure 7: Steps to run a tool-tissue interaction simulation.

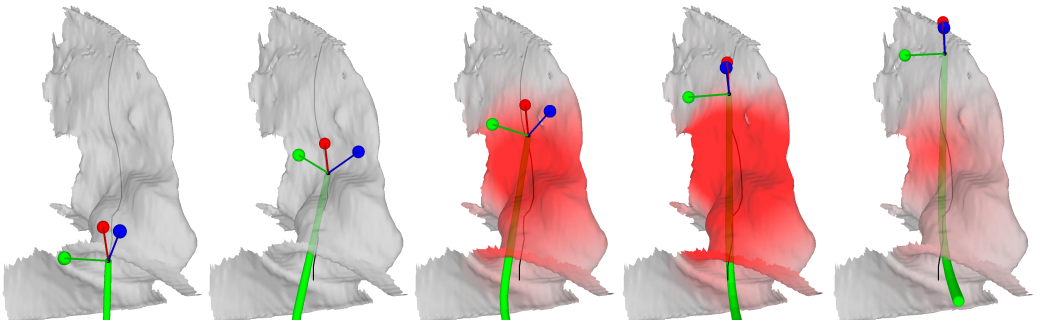

Figure 8: Result of simulation.

The user interface provide an intuitive visualization of all the steps involved in our simulation. However, running a complete simulation from manual interaction might require more than 200 steps. Instead, you can automatically run a complete simulation from command line as follows:

```
Simulator.exe --input roi.ply --fixed fixed.bin
--toolPos tool_positions.bin --toolRad tool_radius.bin --path path.bin
--pose pose.bin --camera camera.bin --output results --simulation
```

The application run a complete simulation by alternating between advancing the tool, centering the tool, avoiding the ROI surface, and solving for collision. This is done by including the `--simulation` parameter. For visualization, we output a frame rendered from the camera view point for each step of the simulation (see Figure 8).

Additionally, use the `--output` parameter to specify a prefix for the results of the simulation. This command generates the file `results_DeformedROI.ply` which stores the deformed ROI mesh at the final configuration. It also generates files `results_Rotations.bin` and `results_Translation.bin` which store the pose of the rigid tool at each of the simulation steps. The file `results_Statistics.txt` reports measures of surface distortion and complexity of tool manipulation used for evaluation.

4 Implementation

In this section we describe some of the algorithms implemented within our toolbox. The code for these algorithms is included in header libraries so can be reused in further applications.

4.1 Path and ROI Selection

In order to provide interactive computation of a path, we voxelize the space around the input mesh and mark each voxel as interior or exterior (see Figure 9(a)). Then, we define a graph structure among exterior voxels: we assign an edge to adjacent exterior voxels whenever their connecting segment does not intersect the surface (i.e., the exterior voxels are visible to each other).

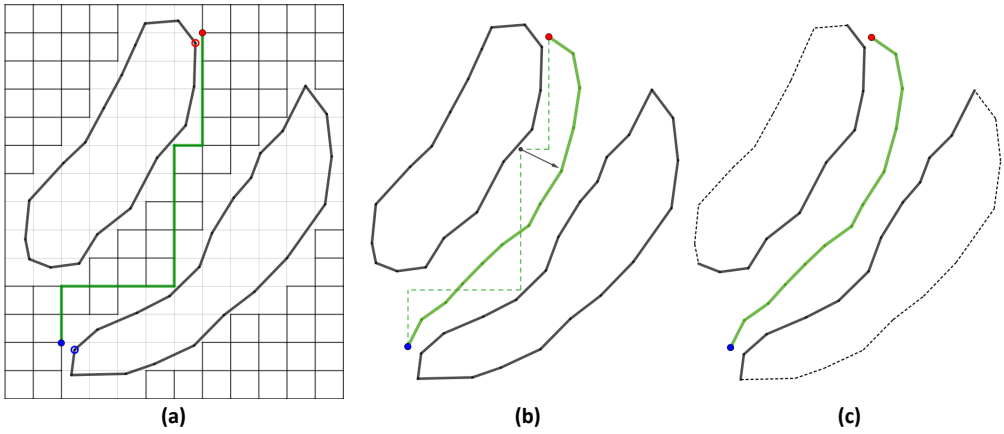

Figure 9: Selection of path and ROI implementation.

After the user specifies two points in the surface (shallow cirles in Figure 9(a)), we map them to the closest exterior voxels (filled circles) and compute the shortest path in the exterior voxel graph. This generates a path that is aligned with the voxel grid.

We smooth this path as follows: first, each path node is offset in the opposite direction to the closest surface point, $x_k \leftarrow x_k + \epsilon n_k$, second, the new position is computed by averaging among its neighbours $x_k \leftarrow$

$\alpha x_{k-1} + (1 - 2\alpha)x_k + \alpha x_{k+1}$. We repeat this process until the curve stabilize along the medial axis (Figure 9(b)).

Finally, we set the ROI to the portion of the input surface that is visible from the selected path. For each triangle in the mesh we trace a ray between the triangle baricenter and the closest point in the path. If this ray does not intersect the surface we add the triangle to the ROI (Figure 9(c)).

4.2 Directional field

The motion of the rigid tool is driven by a vector field that simultaneously pushes the tool towards the end point of the path (Figure 10(a)) and repels away from the surface (Figure 10(b)). This vector field can be evaluated at any point of the 3D space.

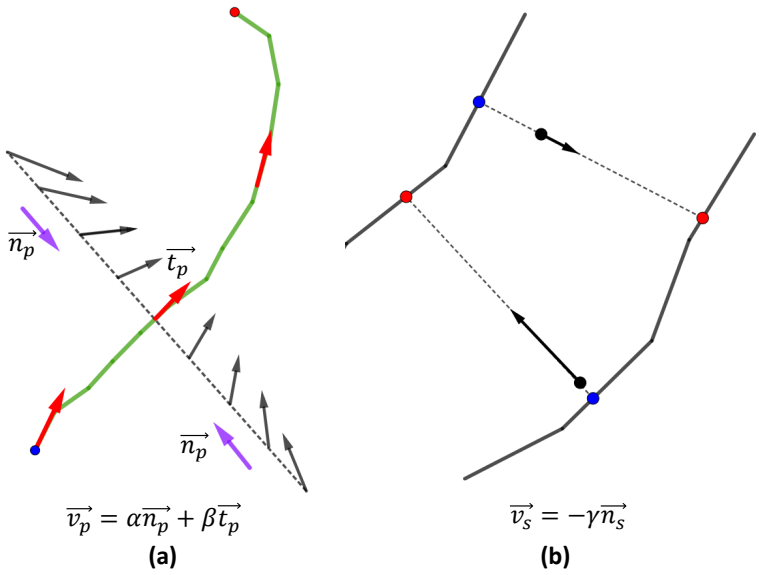

Figure 10: Extrinsic vector field.

The force that pushes the tool towards the path end is computed by first identifying the closest point path. Denoting $\vec{t}_p$ the tangent direction at the closest point and $\vec{n}_p$ the direction pointing towards the closest point, we com-

pute the pushing force vector as $\vec{v}_p = \alpha\vec{n}_p + \beta\vec{t}_p$. We set $\alpha^2 + \beta^2 = 1$ and $\alpha \to 1$ as the point separates from the path.

The force thas pushes the tool away from the surface is computed by first identifying the closest surface point (in blue) and a second surface point (in red) obtained by intersecting a ray in the opposite direction ($-\vec{n}_s$). We compute the repelling force vector as $\vec{v}_s = -\gamma\vec{n}_s$, where γ is a function of the distance from the sampling position to the closest and opposite surface points. The final vector is computed by adding the two forces: $\vec{v} = \vec{v}_p + \vec{v}_s$.

4.3 Rigid Motion

We parametrize the motion of the tool as a sequence of rigid transformations $\{(R_i, t_i)\}_{i=0}^{N}$. The pair (R_0, t_0) is specified by the user from the graphic interface, and we call it the initial configuration. Subsequent transformations are computed by moving the tool according to the vector field.

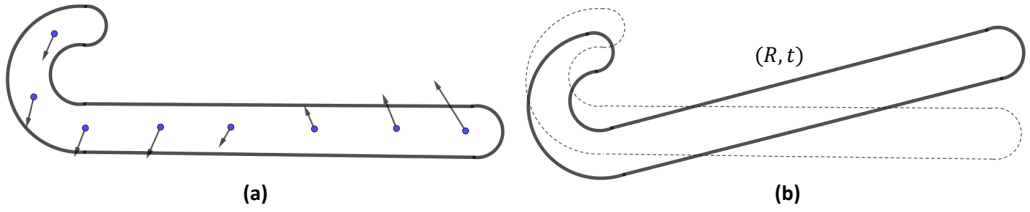

Figure 11: Rigid Motion.

We sample the extrinsic vector field along the medial axis of the tool (Figure 11(a)). This associate at each sample p_i a target position by moving along the vector direction, $q_i = p_i + \vec{v_i}$. We obtain the rigid deformation (Figure 11(b)) by solving the Procrustes problem:

$$\min_{R,t} \sum_i w_i ||q_i - (Rp_i + t)||^2 \tag{1}$$

A similar strategy is implemented when centering the tip of the tool towards the path. In this case the target position for the tool tip is set to the closest point in the path, and the target position for any other point is set to its current position.

4.4 Collision detection

The motion sequence that takes the rigid tool from its initial configuration to the target position should avoid collisions as much as possible. However, the geometry of the ROI and the tool might prevent the existence of a collision free trajectory. Thus, we allow elastic deformations on the ROI induced by contacts event with the rigid tool.

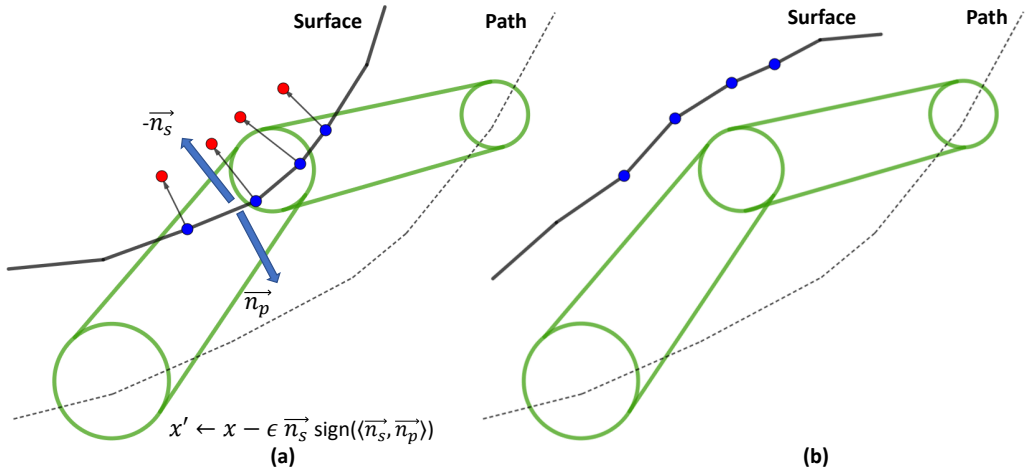

Figure 12: Collision.

We say that a vertex in the ROI surface is a collision vertex when it is in the interior of the rigid tool (see blue vertices in Figure 12). For a geometric object parameterized as a collection of simple primitives (e.g., spheres, cones, cylinders) the interior point test is done analytically. For a geometric object parametrized by a triangle mesh the interior point test is done by a ray casting test.

For each collision vertex we define a target position away from the path. Given a collision point x, let $\vec{n_s}$ be the surface normal at x, and $\vec{n_p}$ the direction to the closest point in the path. The target position is given by $x' = x - \epsilon \vec{n_s}\, \text{sign}(\langle \vec{n_s}, \vec{n_p} \rangle)$.

Once the target positions for the collisions vertex is specified we proceed to solve for an elastic deformation as described in the next section.

4.5 Elastic deformation

Our goal is to compute a deformation of the ROI surface that moves the collision vertices towards the target positions, while preserving the surface stiffness. This is done by optimizing a deformation energy which composed by a fitting error and a rigidity error.

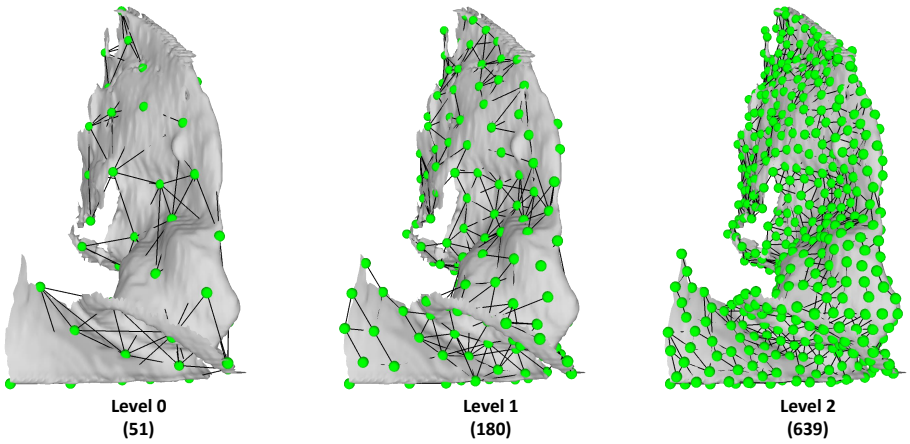

Figure 13: Hierarchical Deformation Graph

Rather than solving for deformation at the finest resolution, we use a hierarchical structure (see Figure 13) to solve for deformation from coarsest to finest as described in [5]. At each level, deformation is driven by a structure called *deformation graph*. The deformation graph $G = (N, E, W, T)$ is defined by a set of nodes $N = \{n_i\} \subset V$, and set of edges E joining spatially close nodes. Additionally, we have a set of weighting functions $W = \{w_i\}$ that define the influence of each node on each mesh vertex, and a set of rigid transformation $T = \{t_i\}$ that define how space is transformed around.

We compute the deformed position of a vertex by linearly blending the transformation induced by the graph nodes:

$$T(v) := \sum_{i \in N} w_i(v) t_i(v) \text{ where } \sum_{i \in N} w_i(v) = 1 \tag{2}$$

Given a subset of vertices $U \subset V$ that is intended to be deformed to match

positions $D(U)$ (collision vertices and their target positions), we define our fitting error as:

$$E_F(T) = \sum_{u \in U} |D(u) - T(u)|^2 \tag{3}$$

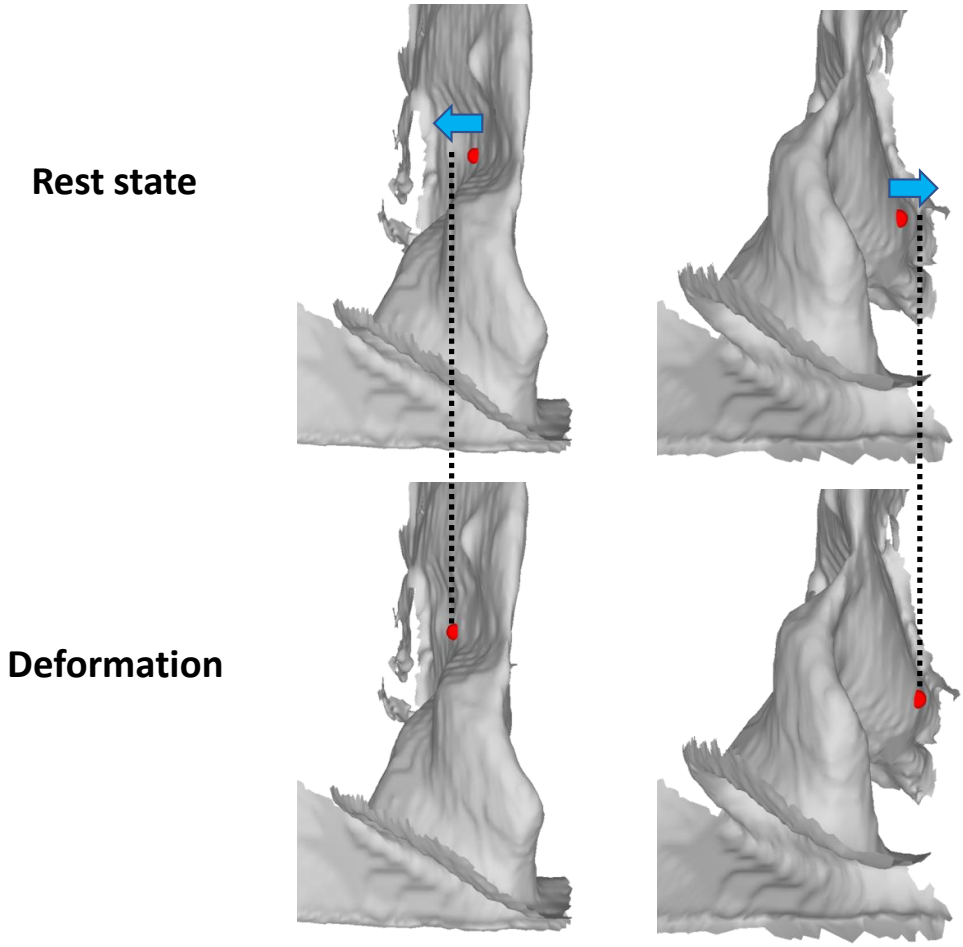

Figure 14: Elastic Deformation

On the other hand, rigidity is computed by comparing the action of transformation between adjacent nodes:

$$E_R(T) = \sum_{(i,j)\in E} |t_i(n_i) - t_j(n_i)|^2 + |t_i(n_j) - t_j(n_j)|^2 \tag{4}$$

Combining fitting and rigidity terms, the energy to minimize is,

$$\min_T E_F(T) + \lambda E_R(T) \tag{5}$$

In our current implementation this non-linear problem is solved at each hierarchical level using Ceres library[1]. Once we find the solution at each hierarchical level we use it for initialization to the next level.

In Figure 14 we present two examples of a simple deformation where a single vertex is pulled in the normal direction to the surface. As observed the deformation acts smoothly and preserve the local shape of the surface.

5 Evaluation

We run our simulation in a set of different tool shapes. For each tool we were interested on measuring two attributes: the amount of distortion induced by the tool on the ROI surface and the complexity of the tool manipulation.

At each step of our simulation, we measured surface distortion by integrating the norm of the displacement of each point in the surface respect to its rest configuration:

$$d_i := \int_\Omega ||x_i - x_0|| dx_i \tag{6}$$

We measured complexity of the tool manipulation as the magnitude of the tool rotation along the principal axis direction of the tool. Let $\vec{r}_i$ be the angle-axis representation of the rotation applied to the tool at the $i-$th iteration of the simulation, and assume $\vec{a}_i$ is the tool's principal direction. We define the complexity of the tool's motion as:

$$c_i := |\langle \vec{r}_i, \vec{a}_i \rangle| \tag{7}$$

In Figure 15 we compare our measures on four different tools. For each tool we reported the manipulation score ($\sum_i c_i$), the cumulative distortion ($\sum_i d_i$) and the max distortion ($\max_i d_i$).

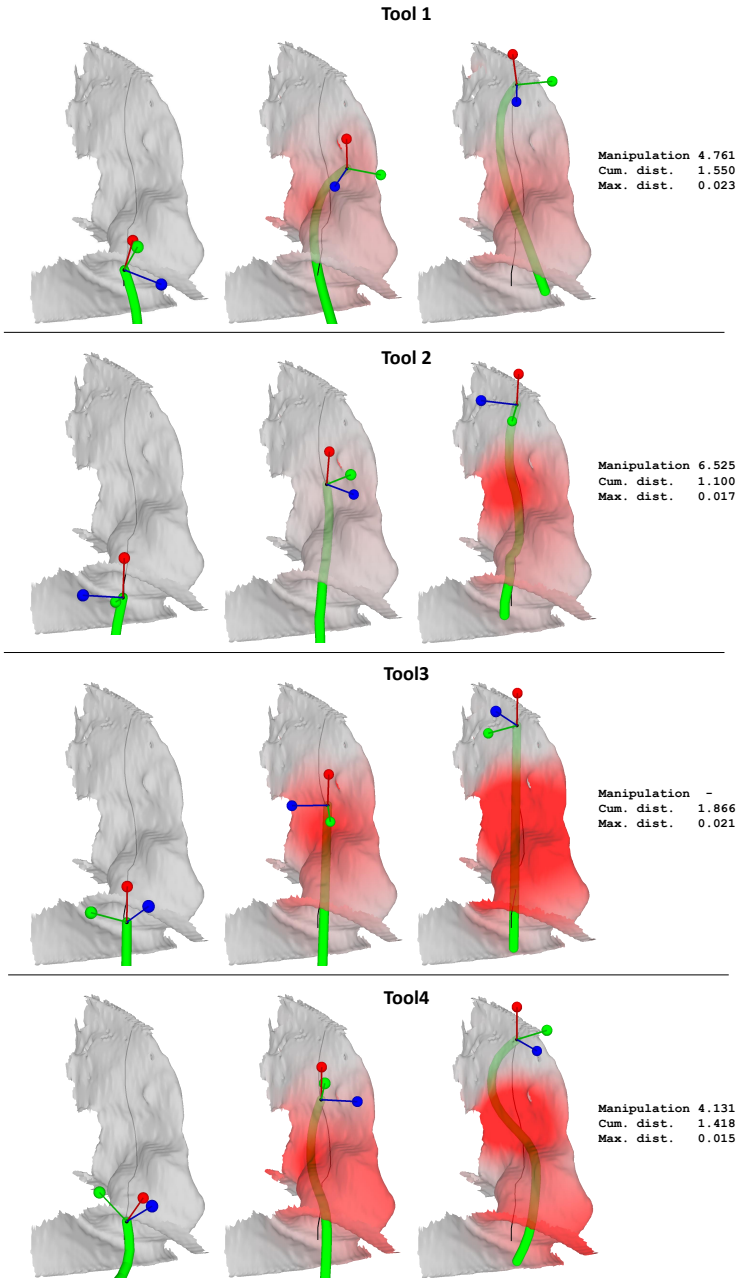

Figure 15: Evaluation

The first row shows a tool with a slight curvature on its tip, the second row shows a tool with the same shape than the medial axis, the third row shows a straight tool, and the bottom row a tool with a high curvature at its center.

The lowest cumulative distortion was achieved by the tool shaped similar to the ROI medial axis. As observed in the figure, the final pose of the tool closely matches the medial axis of the ROI, producing a small surface distortion for that configuration. Interestingly, the overall maximum distortion was not attained by Tool2 but rather by Tool4. This result motivates the exploration of other tool designs that reduces maximum distortion. As expected, the tool with the straight shape was the one that produced the largest distortion.

In terms of manipulation, Tool 2 (the only non planar tool) was the hardest to manipulate, while Tool 4 was the easiest. We must highlight that the measures on tool manipulation and surface distortion can broadly change depending on the initial pose of the tool.

6 Conclusions

This project provide a collection of tools to design and simulate the interactions between a rigid object and a elastic shape. The project was developed in the context of a surgical tool traversing a path along the nasal cavities, but should be adaptable to some other scenarios.

Our simulator computed feasible trajectories for several configurations of tool shape and initial pose. As observed in the results, this involved precise motion of the tool and moderate distortion of the elastic shape.

In this project we provide visually realistic results, however, evaluating physical realism is out of the scope. Introducing elastic properties of the materials (Young's modulus, Poisson ratio, etc.) to our deformation model would be a promising direction towards this goal.

Simulation of surgical interactions seems a first step in the direction of real impact applications like tool design and robotic surgery.

References

[1] Sameer Agarwal, Keir Mierle, and Others. Ceres solver, 2010. http://ceres-solver.org.

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

[3] Alec Jacobson, Daniele Panozzo, et al. libigl: A simple C++ geometry processing library, 2016. http://libigl.github.io/libigl/.

[4] Michael Kazhdan. Geometry and image processing tools. http://www.cs.jhu.edu/ misha/Code/.

[5] Hao Li, Bart Adams, Leonidas J. Guibas, and Mark Pauly. Robust single-view geometry and motion reconstruction. *ACM Trans. Graph.*, 28(5), December 2009.

[6] Guy Eric Schalnat, Andreas Dilger, John Bowler, Glenn Randers-Pehrson, et al. Png library. http://www.libpng.org/pub/png/libpng.html.

[7] Vitaly Surazhsky, Tatiana Surazhsky, Danil Kirsanov, Steven Gortler, and Hugues Hoppe. Fast exact and approximate geodesics on meshes, 2005. http://hhoppe.com/proj/geodesics/.

[8] Ingo Wald, Sven Woop, Carsten Benthin, Gregory S. Johnson, and Manfred Ernst. Embree: A kernel framework for efficient cpu ray tracing, July 2014. https://embree.github.io/.