MS PROJECT

Virtual Surgery

Piyush Soni under the guidance of Dr. Jarek Rossignac, Brian Whited
Georgia Institute of Technology, Graphics, Visualization and Usability Center
Atlanta, GA
piyush_soni@gatech.edu, jarek@cc.gatech.edu, brian.whited@gatech.edu

1. Abstract

This tool of ours called Bender, which is in its first stage emerges out of a need for the surgeons to virtualize heart surgeries which involves besides visualizing a three dimensional view of the heart, its valves and arteries from different angles the fine grained surgical operations like cutting and bending of the patient's arteries and joining them in another appropriate place, all in real time. Bender uses triangle mesh and ray casting operations to achieve this, and inserts a cubic Bezier curve inside the cut artery as its 'spine' and projects the vertices of the cut part on it. When the user interactively and intuitively modifies any of the control points related to this spine, its length is maintained and corresponding vertices are 'unprojected' back from it after deforming the same.

2. Introduction

In our knowledge, surgeons who are at present involved in heart surgery operations don't really have a handy and intuitive graphical tool for simulating heart surgeries which maintains desirable properties and still manages to get a real time performance. This work is a step towards the same. It allows the surgeon to cut the arteries, bend and join them somewhere else in the cardio logical system. Professor Jarek Rossignac's Mesh Viewer program has been modified to achieve the same. When the user presses the Left Mouse Button (LMB from now on) on any triangle on the triangle mesh to select it, and then at some other location outside the triangle mesh, I draw a plane between this line drawn by the user, and the camera's eye. This plane is used to find a curve of intersection between itself and the triangle mesh. After tracing the curve of intersection and its triangles, I 'topologically' separate the triangles on both sides of the cutting plane. After the user inserts one or more cutting planes, he can initiate the bending of any of the parts separated by this act. The following section explain the techniques and algorithms involved.

3. Algorithms and Techniques

Here I present the techniques and algorithms implemented for the project.

3.1 Cutting Plane Triangle Traversal

The first and foremost important implementation of the project is the cutting plane traversal to find the triangles which actually intersect with the plane. To achieve this, a simple triangle walk algorithm is used, starting from the start triangle clicked by the user.

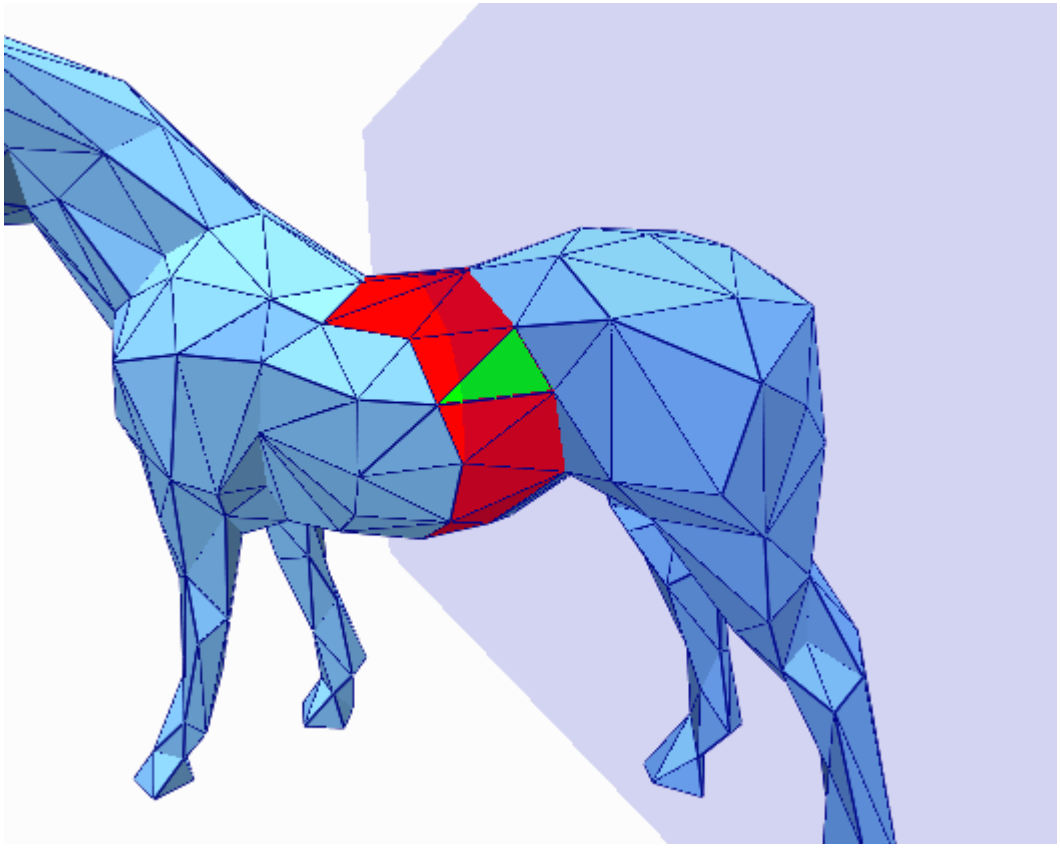


Figure 1: *Selected triangles by the cutting plane intersection with the mesh (shown in red)*

The above figure shows the triangles covered by the intersection algorithm, in which the initial triangle is selected by the click of the user, and then to find the complete loop, we walk along the plane by checking each time whether the next corner is on the left side or the right side of the plane.

This is the basic algorithm which does that:

```
while (mixed(PQ, V(P, g(n(c))), V(P, g(p(c)))) < 0)
{
      if (dot(V(P, g(c)), Np) < 0)
            c = l(c);
      else
            c = r(c);
}
```

Where Np is the normal of the plane, P is a point on the plane, (which is in this case the eye point), Q is the focus point and c is the current triangle corner.

3.2 Triangle refining along the cutting plane

Section 3.1 shows how to traverse the triangles which are intercepted by the cutting plane. Bender does more than that. It refines the triangles at the cutting plane intersection so that the cut is smooth and free of crests and troughs. It does this while finding the intercepting triangles only, to save time from looping through the triangles again.

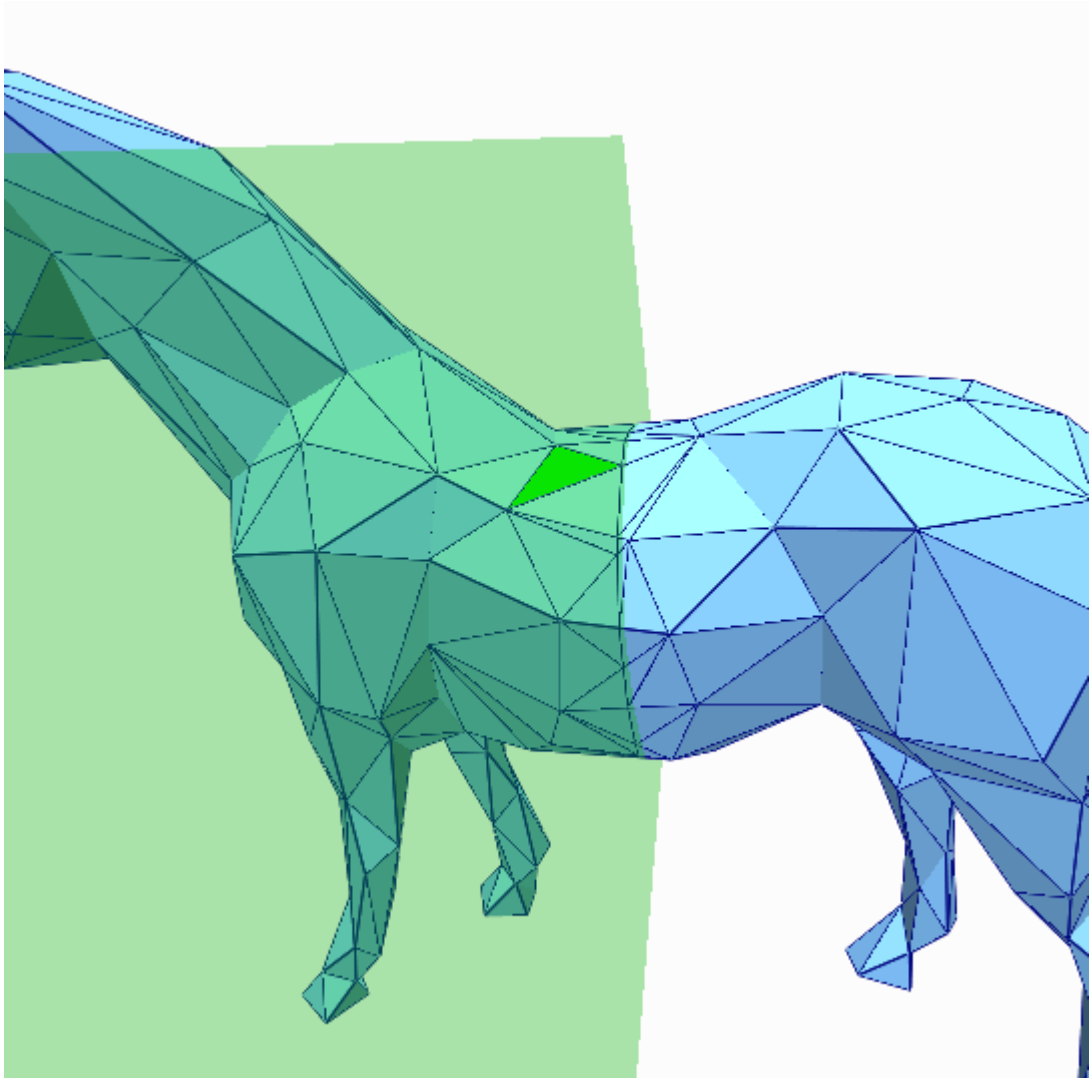


Figure 2: *Insertion of cutting plane refines the cut triangles along it*

In effect, this is achieved by just finding the intersection of the cutting plane with each triangle, which would give a line for each of the triangles. When the two end points of that line are found, three triangles are formed from each old triangle like this:

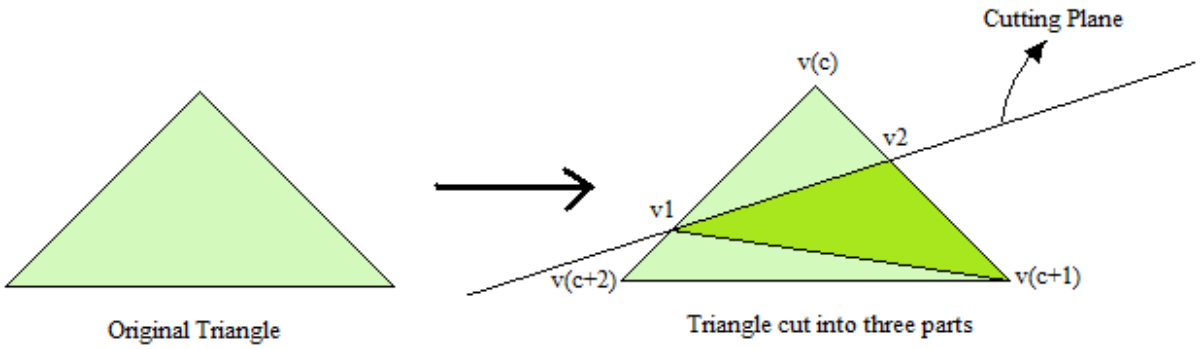


Figure 3: *Original triangle is broken into three triangles to refine the mesh at the cutting plane*

The two intersection points of the cutting plane with the triangle: v1 and v2 are added to the mesh, coordinates of one of the triangles changed, and two new triangles are added.
The following code snippet shows the above mentioned steps:

```
addVertex(G[v1]);
addVertex(G[v2]);
V[c] = V[c]; //no change
V[c+1] = v2; //change the vertices to new ones
V[c+2] = v1;

addTriangle(v1, v2, V[c+1]);
addTriangle(v1, V[c+1], V[c+2]);
```

The addVertex method increases the total count of geometric points in the system and addTriangle function actually adds a triangle from existing geometric points in the system increasing the total count.

3.3 Topological Mesh Separation

After the extra triangles are added, the two meshes are topologically separated without actually making two meshes out of one. This is done in three parts. First, the boundary curve is cloned (the curve of intersection of the cutting plane with the mesh). This is done by making a copy of each vertex on the curve and associating the vertices on either side of the cut with appropriate one or the other vertex. Second part is marking the opposite triangles on triangles of both the sides as -1(to erase the connectivity information) and the third part is fanning the hole created by this process. Professor Rossignac's fanHoles() method came handy here.
The fanHoles() method adds a new vertex at the center of the border loop to be fanned and then adds triangles equal in number to the border edges to the triangle mesh.
This way, in three steps the triangle mesh is topologically separated along the cut so that it behaves like two meshes but sharing the same corner table structure. User can insert multiple cuts in the triangle mesh before deciding for a part to be bent.

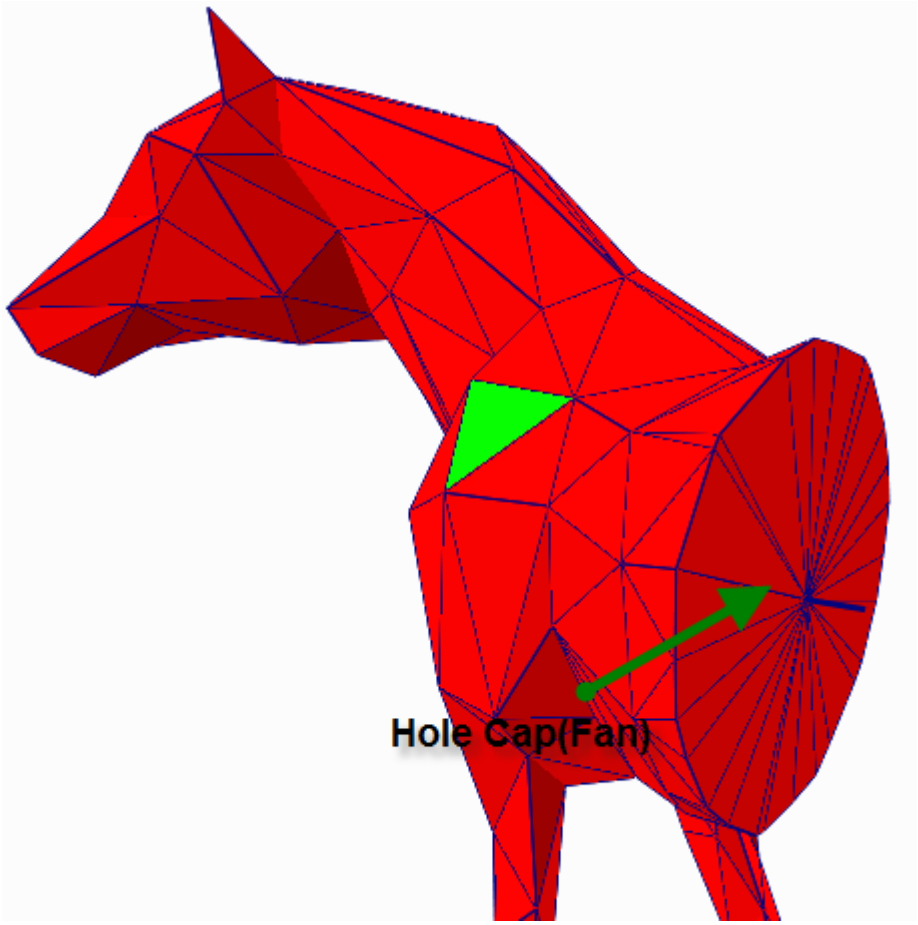


Figure 4: *After cloning the boundary, triangle fan is inserted along with the new center vertex*

3.4 Bending

After the previous three steps, the next step is the bending part. The basic idea employed for bending any part, (heart artery in this case) is to project all the vertices of the part to be bent on a spline, bend the spline with constant length, and then to 'unproject' the vertices at the proper place 'somehow' (discussed later) to give the effect of bending.
Here I discuss all the parts in sequence:

3.4.1 Projecting vertices on spline

The initial approach of projecting vertices just joined the center points of two fanned faces developed by each cutting plane's intersection with the mesh, took the straight line formed as the result and projected the vertices directly on to the line, and noting the value of the parameter 't' from 0 to 1 along the length of the line. The problem with projecting the points on a straight line is that it does not have a unique normal vector in 3D, which would make it impossible to unproject the vertices back to their original distance away, by the approach I earlier implemented (Which used frenet trihedron, discussed later in 3.4.2). Also, it was unnatural as most of the times the arteries or tubes to be bent cannot be assumed to be in straight lines and thus a more fitting curve would be better suiting this. So, the initial

spline taken to project the vertices was a Bezier curve, which was anyway the curve thought to facilitate active bending in the tubes, as they are very realistic in bending apart from being really smooth.

So, the later approach worked like this. When the user initiates the bending by pressing the 'B' key, the center points of the cutting plane faces are now the two end control points of the Bezier to be inserted, and the normals of the cutting plane become its initial tangents at the two ends. These tangents are extended at both ends by a suitably chosen value to get the two inner control points (which is currently taken as d/3, d being the distance between two end control points). Hence, the initiation step inserts a new Bezier on which the vertices are to be inserted.

The biggest hurdle in finding the projection on a Bezier curve (basically the value of parameter 't' where the point is projected) is that for this a fifth degree polynomial equation has to be solved, which does not have a direct and non iterative solution. Therefore, 'Jenkins-Traub' method of polynomial solving was used in finding the roots of the equation which enabled us to find the closest point on the Bezier curve from a vertex, and hence the parametric value 't' from 0 to 1.

3.4.2 Unprojecting vertices

Once the points are projected on the Bezier curve and the spline is bent in constant length according to the user input (details discussed in section 3.4.3), the final step is to unproject the vertices in the same way as they were projected. Conceptually, this technique is to find out a vector of every vertex in the local coordinate system on the curve at the projected point with parameter t, find the changed coordinate system, and then transform the initial position vector to the new frame. Hence, Frenet trihedron was a natural way to work with which can find tangent, normal and binormal vectors of a space curve at any given place. But, the problem with frenet vectors is that they can't be relied upon when dealing with bending a tube with fixed vertices. The problem is, the normal (or binormal) vectors take quick turns which would produce more twisting than what would really occur in the tube. The figure below illustrates this problem.

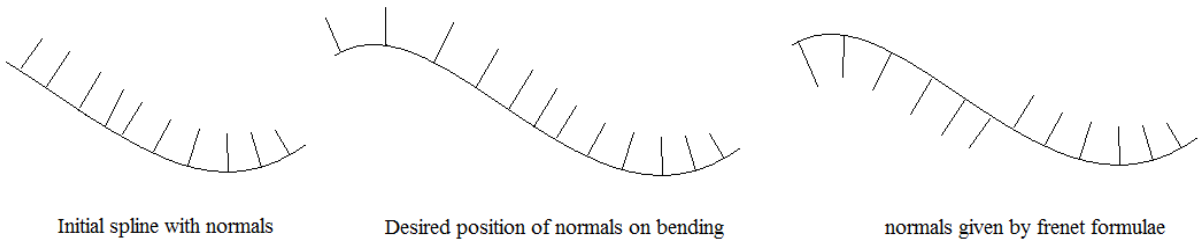


Figure 5: *Problem of working with the frenet formulae of finding tangent, normal and the binormal vectors at a given location of the spline. Note that while the normals in the third drawing should have been all on one side, they are given on the other by the frenet formulae.*

As seen in figure 5, the frenet-serret formulas cannot really be relied upon, because of their uncertainty in the resulting normal or binormal vectors, which act as a reference frames for the vertices. They do not communicate any orientation information from one frame of the animation to the other, so the animation after unprojecting the vertices is really weird.

To solve this problem, the frenet trihedron approach was discarded and a new technique (suggested by Brian Whited, GVU Lab, Georgia Tech) was used. It is evident that the tangent at any point on the Bezier can be easily found out. The only problem which is in finding unique normals, can be solved by taking the first normal vector (at parameter t = 0) in any arbitrary direction in the plane normal to the tangent, and then projecting this normal to each plane normal to the tangent at points at an interval of Δt on the curve. This way, the normals information will be propagated along the curve and thus the problem of random direction inversion of the normals won't occur. (The problem shown in the figure above is prevented). Note that for this method to work correctly, the planes on which the normals are projected (which act as the normal for that particular value of 't') must be separated by a very small value of Δt.

3.4.3 Bending of the artery

The arteries (or any part of the 3D model in theory) can be bent after 'initiating' the bend by pressing 'B'(which essentially projects the vertices on the 3D spline and sets the normals of the Bezier curve by the above described method). The Bezier curve is bent in constant length by matching the length of the curve at each step of the animation. As explained above in section 3.4.1, the two inner control points of the Bezier curve are found out by translating the two end points by the distance d/3 along the end tangents. This can be written as:

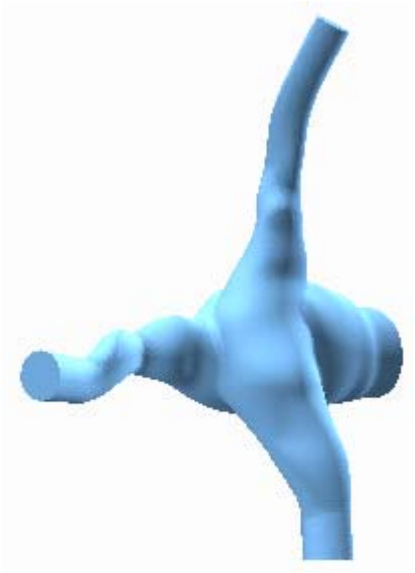
$$\mathbf{B} = \mathbf{A} + k_1(\mathbf{T_1})$$
$$\mathbf{C} = \mathbf{D} + k_2(\mathbf{T_2})$$

Where A, B, C, D are the bezier control points and T1 and T2 are the curve tangents at the two end points.

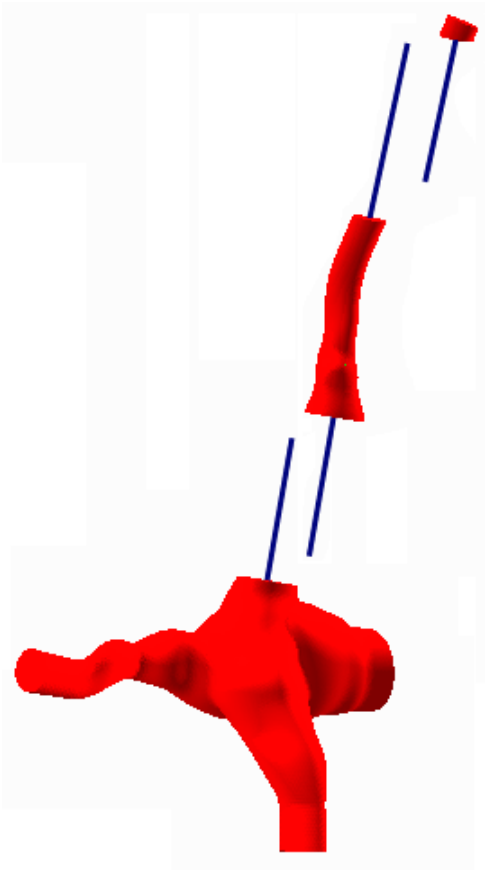
While bending, user tries to control the curve by changing the locations of the end points or the end tangents(one at a time). To maintain its original length, the values of constants k1 and k2 are changed in the above equations. If the user tries to drag the points A and D farther apart, the length tends to increase, which could be matched again to the original length by decreasing the value of k1 and k2 recursively, and vice versa. For now, k1 is kept equal to k2 which perhaps gives rise to a short 'kink' sometimes in the middle of the bezier curve, which looks a bit unrealistic. These should be separated to behave as two different constants, and the matchLength() function should not only check for maintaining constant length, but also chose the solution out of many possible solutions which minimizes the maximum curvature of the curve. This is a two dimensional optimization problem which is still worked upon for this particular case of bezier length matching.

4. Results

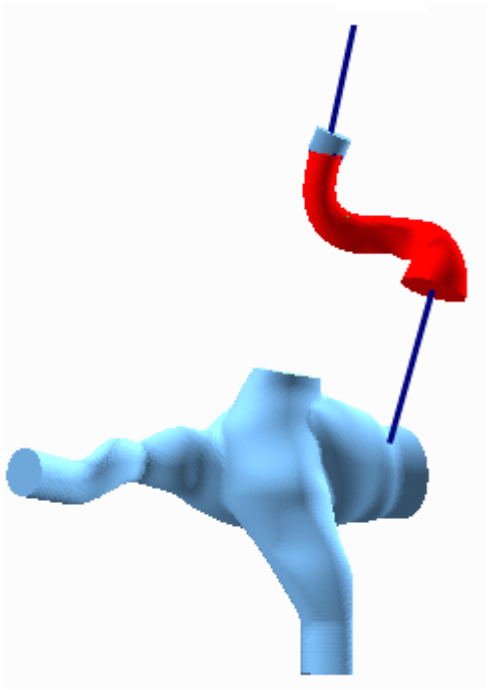
i). 3D model of a part of Heart:



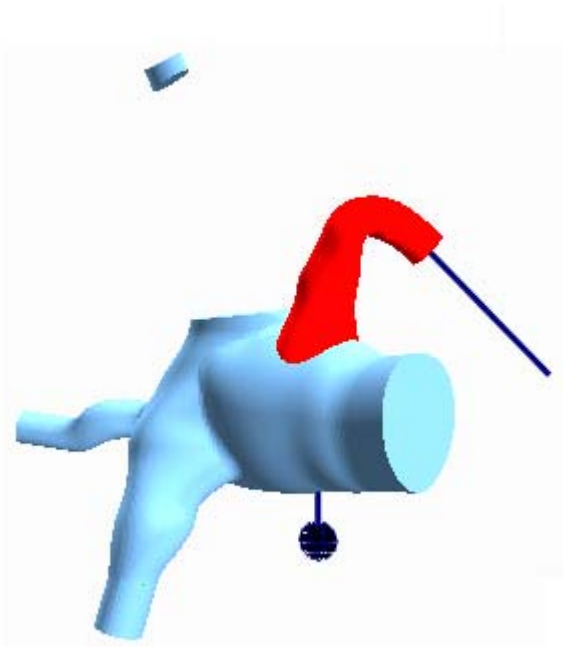
ii). Cut at two places (parts shown separated apart)



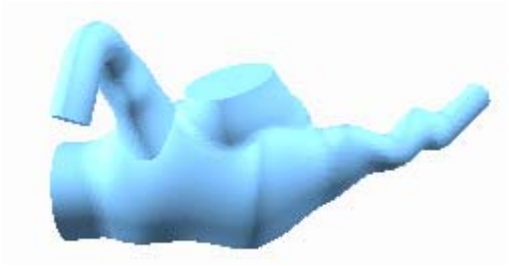
iii). Bending of one of the parts



iv) Final position of the part to be bent



v). Part after saving and reloading



5. Conclusions and Future work

Though the bender implements basic operations of cutting and bending of a 3D triangle mesh model and works in real time, a few things like better and 'kink free' bend could be implemented by minimizing the maximum curvature of the bezier curve while matching its original length. Also, the standard operation like union of two triangle meshes needs to be changed to operate on a single mesh for intra-joining. Also, the code to be added to make this work with standard 'phantom tools' (Phantom Haptic devices) has been already started, which also should be complete in near future. This will make the cutting, bending and twisting etc. very intuitive for the surgeons who work with it.

6. References

[1]. Jarek Rossignac, Kerem Pekkan, Brian Whited, Kirk Kanter, Shiva Sharma, Ajit Yoganathan, "Surgem: Interactive patient-specific anatomy-editor for hemodynamic analysis and surgery planning"

[2]. Frenet Trihedron, http://en.wikipedia.org/wiki/Frenet_trihedron

[3]. Length of a Bezier Curve by James FitzSimons, June 24, 2004

[4]. Solid and Physical Modeling, Jarek Rossignac

[5]. Adaptive subdivision and the length and energy of Bézier curves - Jens Gravesen(Department of
Mathematics, Technical University of Denmark)

[6]. Jarek Rossignac's class lectures

[7]. Cubic Curves - Steve Rotenberg, UCSD, Fall 2006