

Progressive Re-Meshing for Cutting Thin-Membrane Surfaces for Surgical Simulators



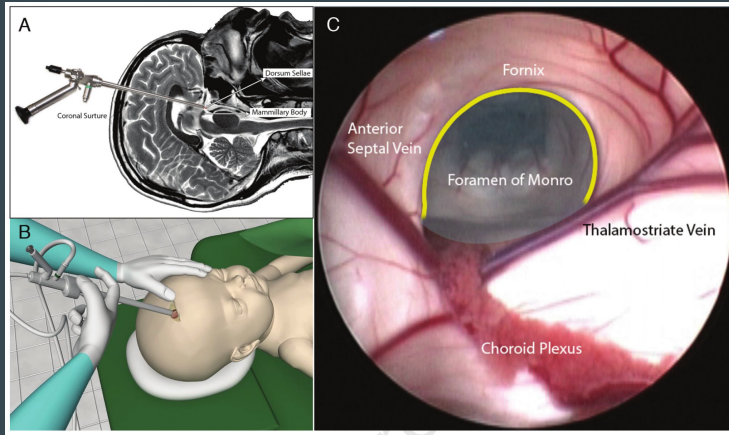
Presenter: Shrey J. Patel
Supervisor: Prof. Prem Kalra

Motivation

- Traditional surgical teaching and demonstration methods have many limitations:
 - a. Difficulty in obtaining subjects to experiment upon
 - b. Geographical barriers
 - c. Resource constraints
 - d. Restricted experimentation by students due to less margin of error
- Virtual Surgical Simulators based on Computer Graphics can overcome such limitations.
- Additional benefits:
 - a. Extreme versatility
 - b. Customizable based on the needs/abilities of the instructor/student
 - c. Easy and intuitive UI

Background

- A large variety of surgical procedures can be supported.
- **Our focus:** Procedures which involve working on membranes.
- Examples: Endoscopic Third Ventriculostomy(ETV), Thoracotomy, Lumbar Puncture/Spinal Tap



Endoscopic Third Ventriculostomy
(Source: [5])

Problem Statement

To **simulate tearing of a membrane** when subject to different kinds of forces using various surgical instruments along generalised trajectories.

Two aspects of the problem:

Geometrical/Topographical:

- a. Conversion of mesh from one configuration to another.
- b. Manipulation of data structures involves creation/removal of vertices and/or faces.

Physical:

- a. To simulate the behaviour of membrane under the effect of forces, given a configuration.
- b. Manipulation of data structures involves only changing the coordinates of the vertices.
- c. No change in topology of the mesh.

Current Objectives

To handle the **topographical aspect of the cutting/tearing of mesh** (we leave the rest of the physical modelling for later) i.e. to handle the re-meshing of the membrane when subject to a cut.

$\mathcal{M}(v, f)$: Mesh representation of the membrane (although the internal representation will be different)

v : Vertices of the mesh, f : Faces(indices) of the mesh

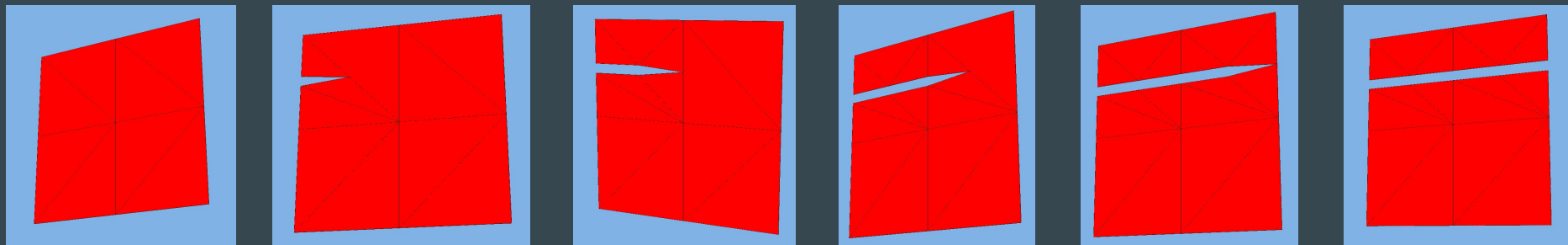


Progressive Simulation:

Given a cut and an initial configuration, we first break the cut into smaller segments (each spanning a face of the mesh) and generate intermediate configurations up till the final one.

Benefits:

- Easier to incorporate physical effects.
- Models the actual cut better, since it replicates the path of the instrument.
- Possibility of doing it in real-time, with minimal lag.



Implementation

1. Mesh Representation

Normally,

- vertices: list of coordinates
- faces: set of three indices(ordered) into the vertex list
- Easy to store
- Difficult to query and manipulate

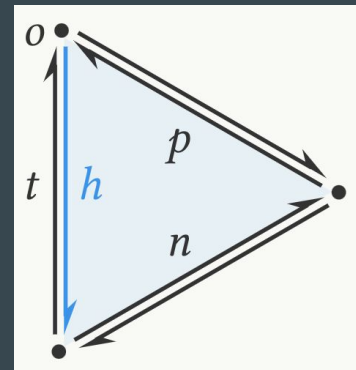
We will use the **Half-Edge Data Structure** instead

- Each edge of the mesh broken into two twin edges with opposite orientation
- Difficult to store and maintain
- Easy to query and traverse the entire mesh
- Easy to manipulate

Half Edge Data Structure

A half-edge(h) stores:

1. Starting Vertex (o)
2. Twin Half-Edge (t)
3. Next Half-Edge (n)
4. Previous Half-Edge (p)
5. Adjacent Face



Half-edge struct with all its attributes

(Source: [3])

Accordingly, we also need to change our vertices and faces:

A vertex now stores:

1. 3D coordinate
2. Any one half-edge originating at this vertex

A face now stores:

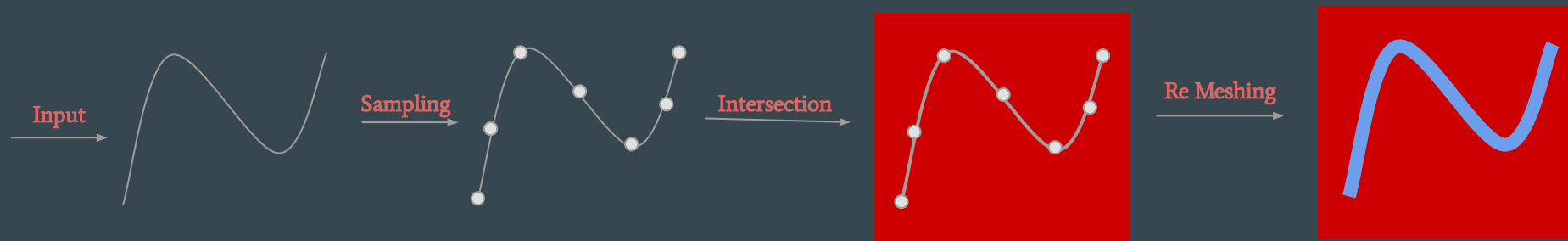
1. List of indices into the vertex list which form the face
2. Any one adjacent half-edge

The mesh is now represented as a list of vertices, list of half-edges and a list of indices.

2. Algorithm

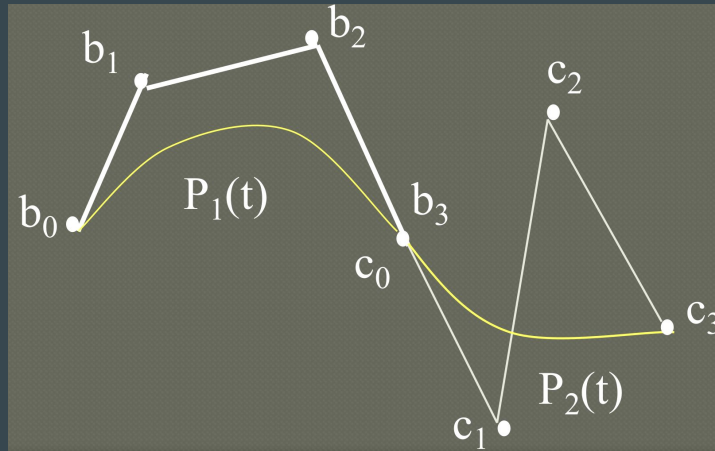
The algorithm consists broadly of the following steps:

1. Handling Input
2. Sampling points to define cut segments
3. Iterate through the sampled points and for each adjacent pair of points,
 - a. Define a plane of cut
 - b. Intersect the plane with the current mesh
 - c. Iterate through the intersection points and remesh.



Handling Input

- Need to be able to model both rectilinear and curvilinear paths.
- Since we have the info. of the complete cut, we can represent it using parametric curves.
- We will use **piecewise/composite Bezier Curves(of any degree)** for this purpose.
- We ensure at least C0 continuity while modelling.



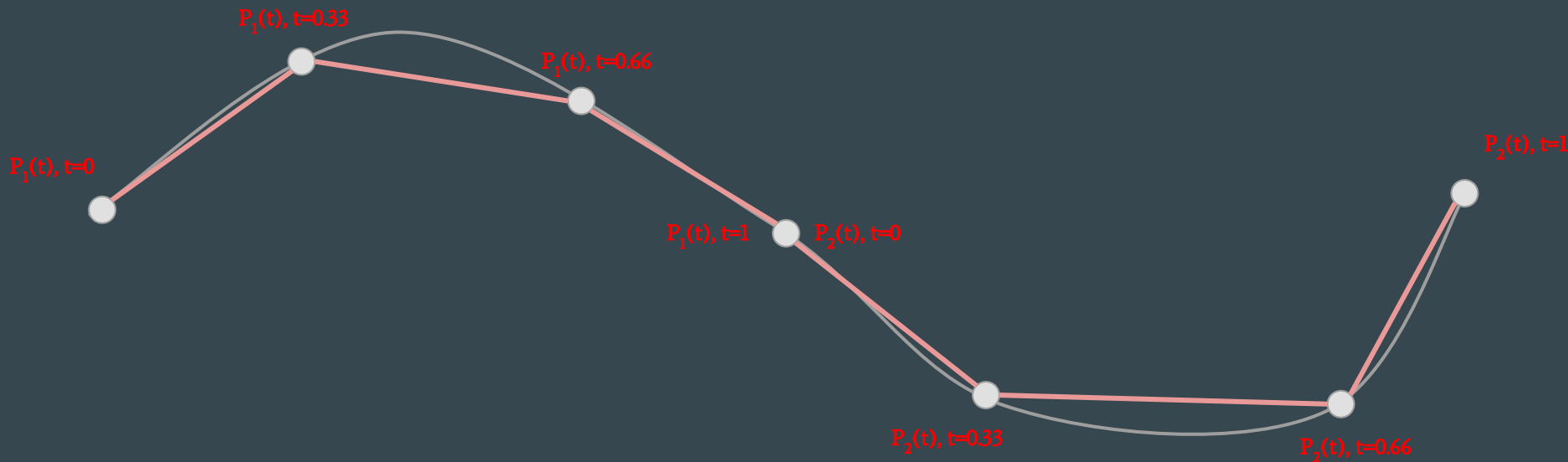
Composite Bezier Curve with pieces $P_1(t)$ and $P_2(t)$
(need not be of same degree)

(Source: [6])

- Input Specification: Control Points of the Bezier Curves which form the cut

Sampling Points

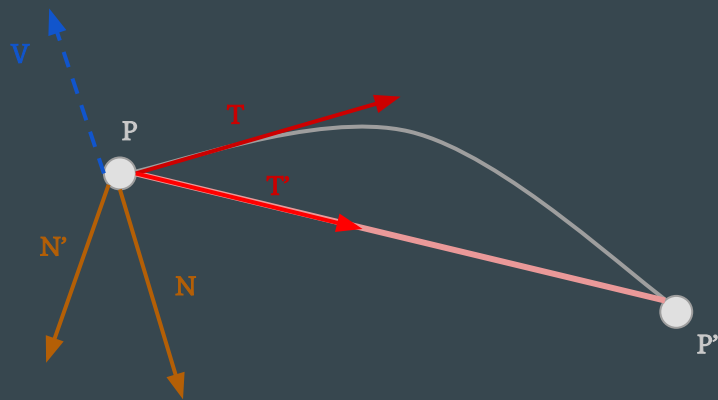
- Easier to re-mesh in case of straight lines.
- So, we approximate the cut using reasonably small straight cut segments, by sampling points on it.
- Since it is a parametric curve, we can obtain points at regular intervals of parameter t .
- We use **Uniform Parameterization** for varying t for ease of implementation.



Processing the Cut Segments

a. Defining a Cut Plane

The line formed by adjacent points may not lie on the mesh itself. But, the plane passing through that line will. So, we first obtain the plane of cut.



P: Current sample point

P': Next sample point

T: Tangent to the actual cut path

T': Tangent to the current cut segment(slope)

N: Normal to cut path, obtained from instrument

N': Normal to cut segment, thus to cut plane

V: Vertical vector = $\mathbf{N} \times \mathbf{T}$

The cut plane can be represented as the plane passing through P as origin with \mathbf{N}' as normal, where:

$$\mathbf{N}' = \mathbf{T}' \times \mathbf{V} = \mathbf{T}' \times (\mathbf{N} \times \mathbf{T})$$

b. Finding Intersection Points

Intersection Record: (3D Coordinate, Intersection Type, Index)

0: Vertex

1: Edge

2: Face (usually for endpoints)

If type = 0, index into vertex list

If type = 1, index into edge list

If type = 2, -1

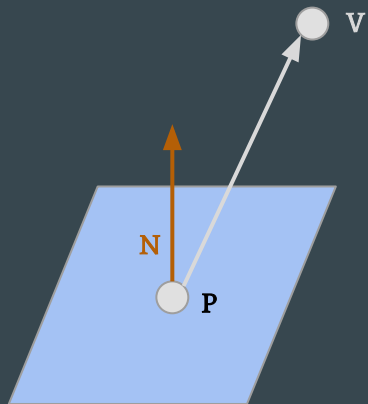
Algorithm for finding intersection points:

1. Obtaining the offsets of all vertices of the mesh from the cut plane
2. Identifying edges intersecting with the cut plane
3. Obtaining intersecting points from intersecting edges

Algorithm for finding intersection points:

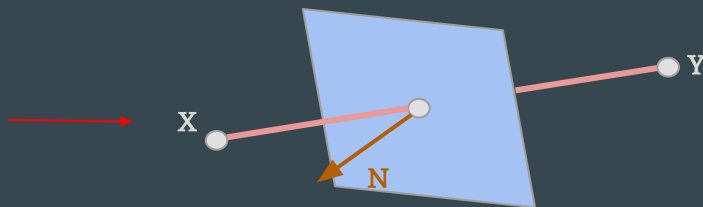
For each vertex V ,

$$\text{Offset}(V) = \mathbf{PV} \cdot \mathbf{N}$$



If $\text{Offset}(V) = 0$,
then V lies on the
cut plane, so add
 $(V, 0, \text{idx of } V)$

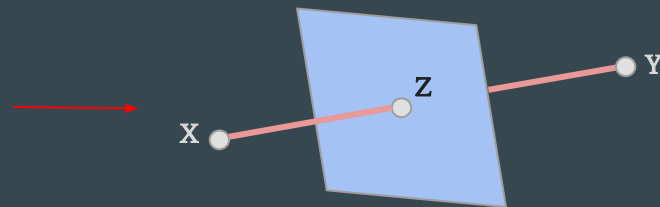
For each edge E
with endpoints X
and Y ,



If $\text{Offset}(X) * \text{Offset}(Y) < 0$, then
 E intersects the cut
plane.

For each edge E with endpoints X
and Y , which intersects the cut
plane, let

$$\alpha = \frac{\text{Offset}(X)}{\text{Offset}(Y) - \text{Offset}(X)} = \frac{\text{offset}(X)}{\text{offset}(Y)}$$



$Z = X * (1 - \alpha) + Y * \alpha$ is the
intersection point. So add
 $(Z, 1, \text{idx of } E)$

c. Filter And Sort



Define a quantity for each intersection point X, $\gamma = \mathbf{PX} \cdot \mathbf{PP}'$

- We don't require all intersection points, only those lying between P and P' when projected on the mesh. So, accept X if $0 \leq \gamma \leq |\mathbf{PP}'|^2$
- Also, since we require progressive re-meshing, we need to sort the filtered points in order of their γ values.

d. Re-Meshing

- Central part of our algorithm, which fulfills our current objective





- The *reMesh()* procedure takes the following input:
 - Last, Current and Next intersection records
 - Pointers from the last re-meshing operation

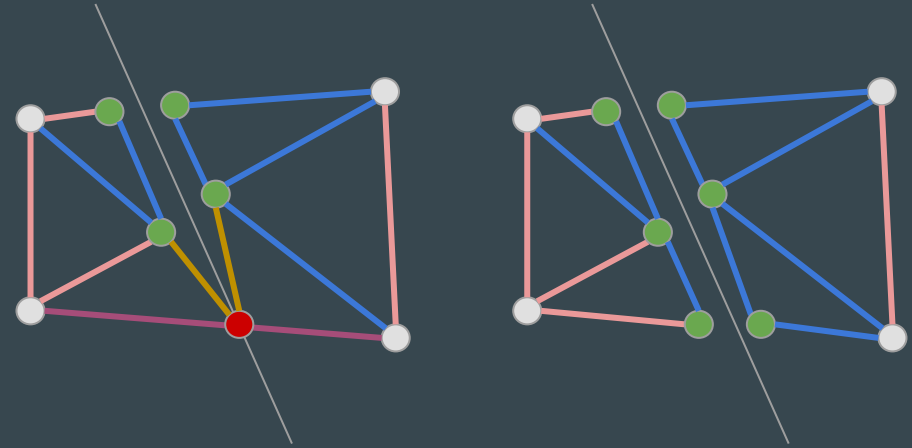
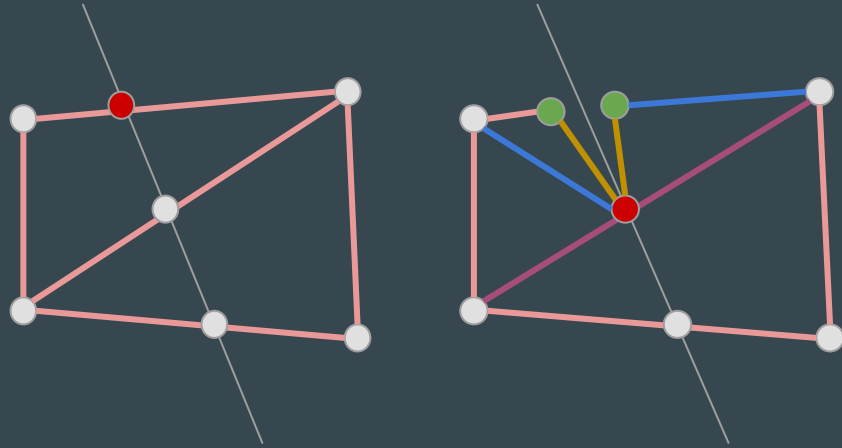
And it performs the following operations:

- Uses the information from the last re-mesh operation to maintain correctness
- Manipulates the current mesh state to split it at the current intersection point
- Passes the pointers of new vertices and half-edges created to next re-mesh operation.

These two types of edges need to be passed across reMesh() calls.

 **Cross Edges:** New edges joining the new vertices to the next intersection point

 **Side Edges:** Old/New edges on the side on which the next intersection point lies



-  Current Intersection Point
-  New Vertex
-  New Edge



Each edge is actually a pair of half-edges

Implementation of reMesh():

- a. Creating a new vertex by splitting the current intersection point:

If **position vector of current int. pt. = V** and **normal of cut plane = N** , then

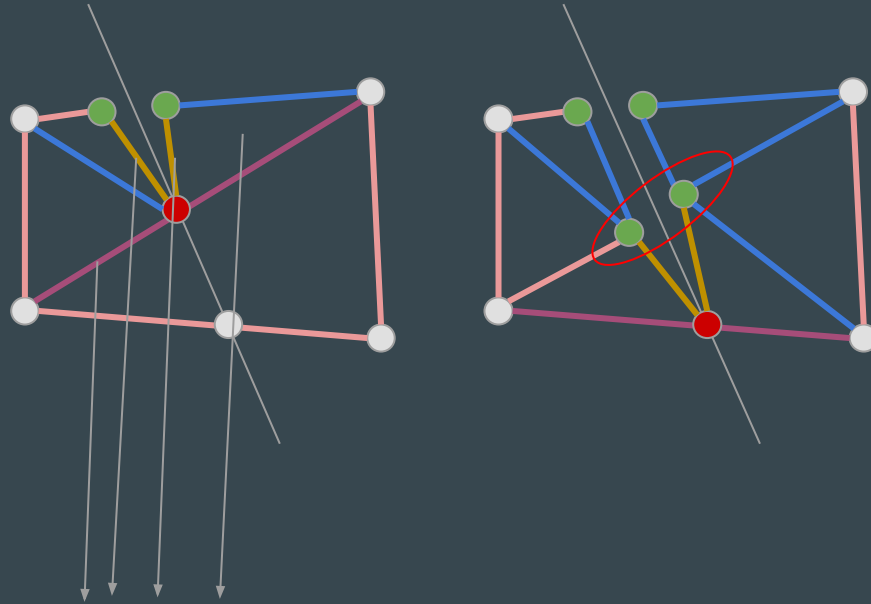


ϵ is a scalar which should ideally be smaller than the length of the smallest edge of the mesh.
(Current Value: 0.04)

- b. Deciding the type of the current re-meshing procedure:
 - Based on the last, current and next intersection point
 - Each with 3 possible codes (0, 1 and 2)
 - Cases with consecutive 2's not handled (i.e. 022, 122, 220, 221, 222)
 - Total number of cases = $27 - 5 = 22$

c. Making data structures consistent upto the last intersection point

Because of addition of a new vertex, some new edges created in the last reMesh() operation need to be updated again as illustrated below

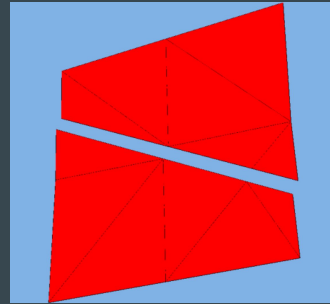
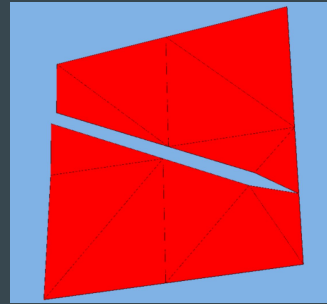
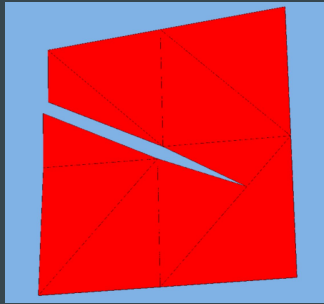
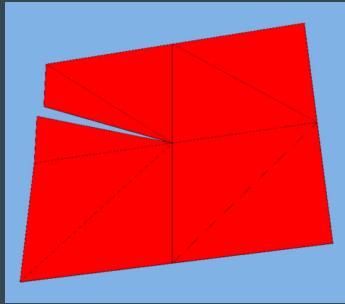
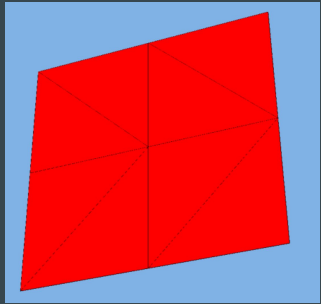


These cross edges, side edges and the new face are still inconsistent since the red dot will be split into two green dots

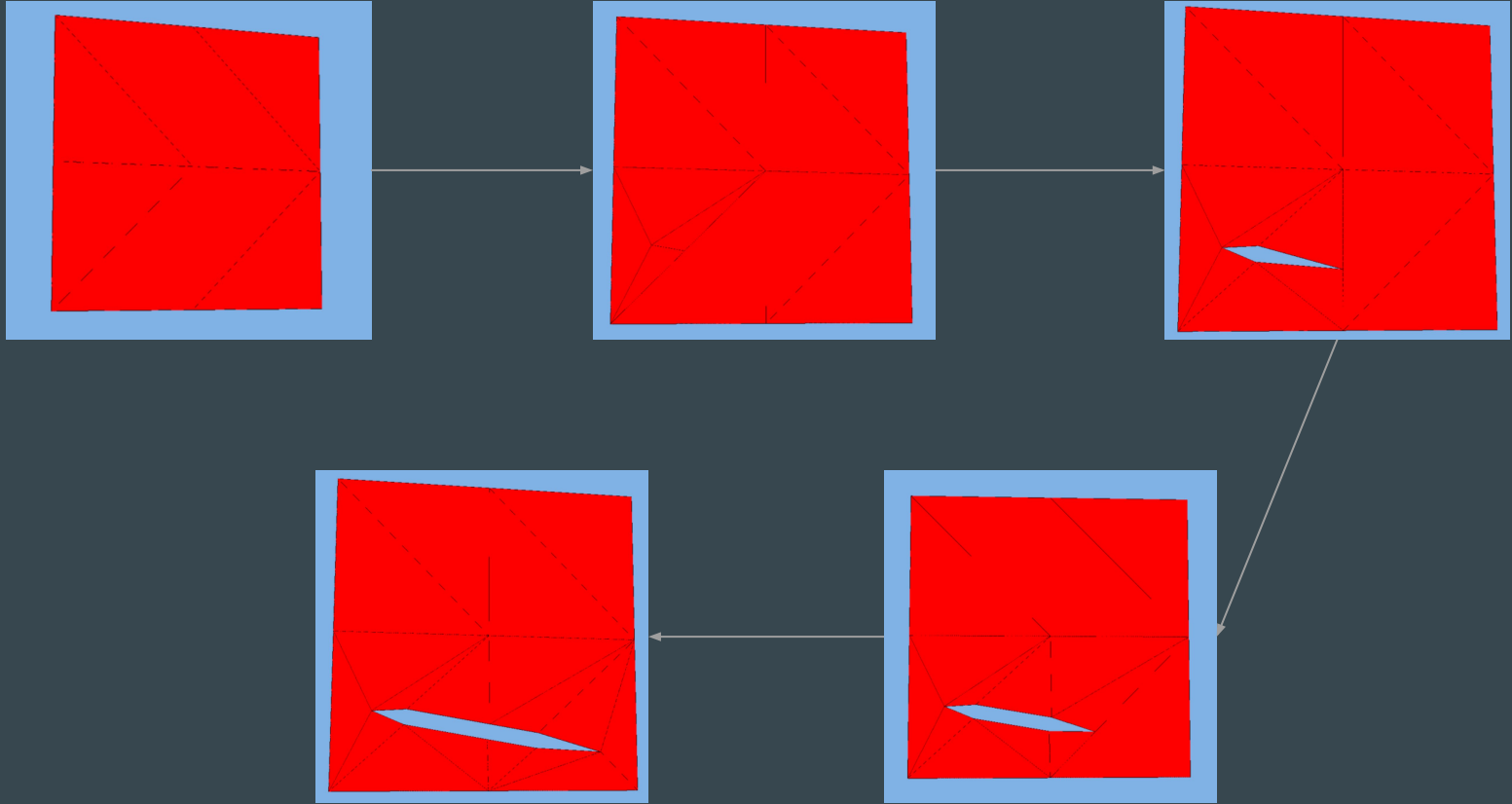
- d. Creation of new half-edges and faces due to creation of new vertex
- e. Adding/Updating attributes of new/old half-edges
 - Edge-Edge: Twin, Next, Prev.
 - Edge-Vertex: Start Vertex
 - Edge-Face: Adjacent Face
- f. Adding/Updating attributes of new/old faces
 - Face-Edge: Adjacent Half-Edge
 - Face-Vertex: Vertex Indices
- g. Adding/Updating attributes of new/old vertices
 - Coordinate: Already handled in 'a'
 - Vertex-Edge: Originating Half-Edge
- h. Updating the cross edges and side edges for the next intersection point
- i. Inserting the newly created vertices, edges and faces to the appropriate list

Results

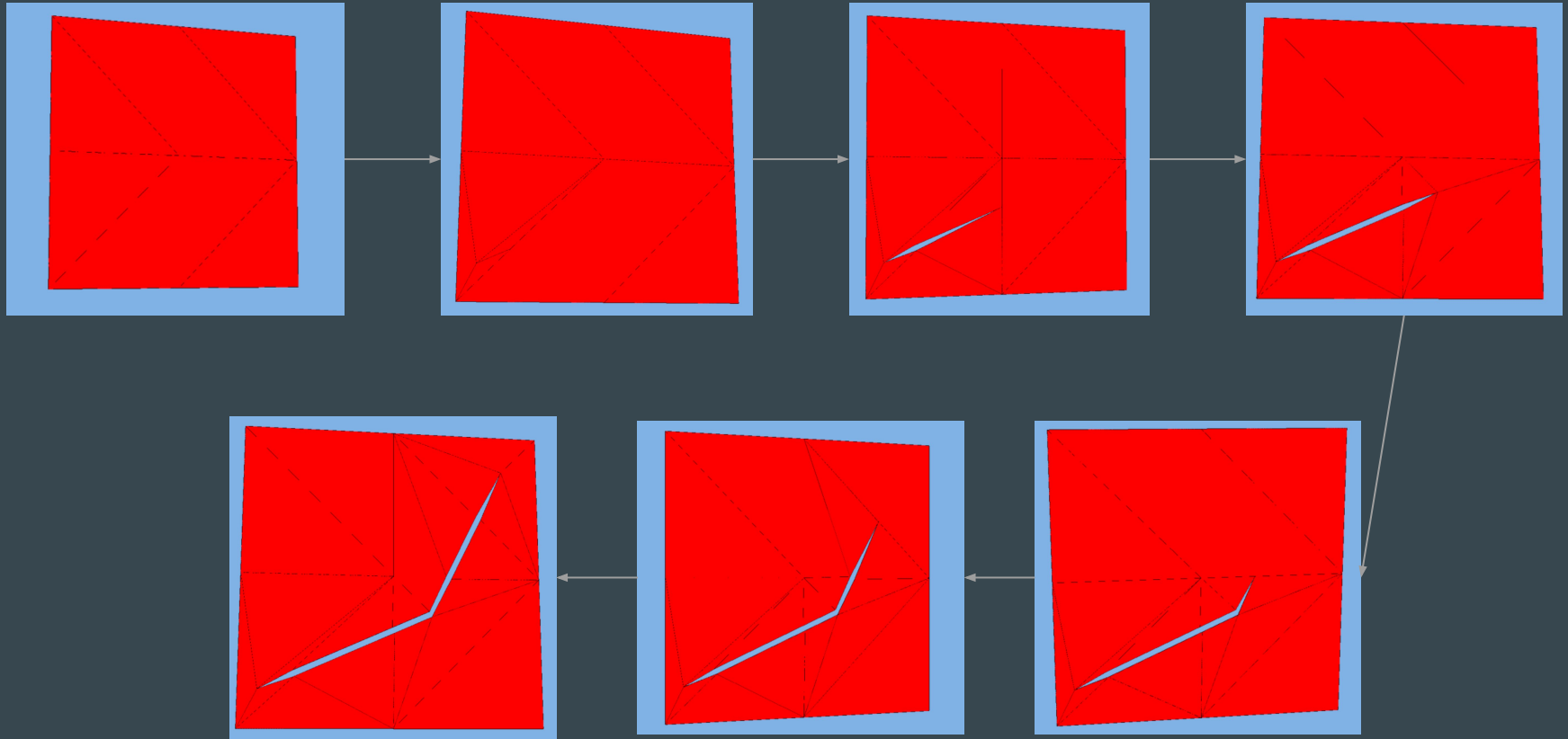
1. Small Mesh, Single-Segment Cut, 1



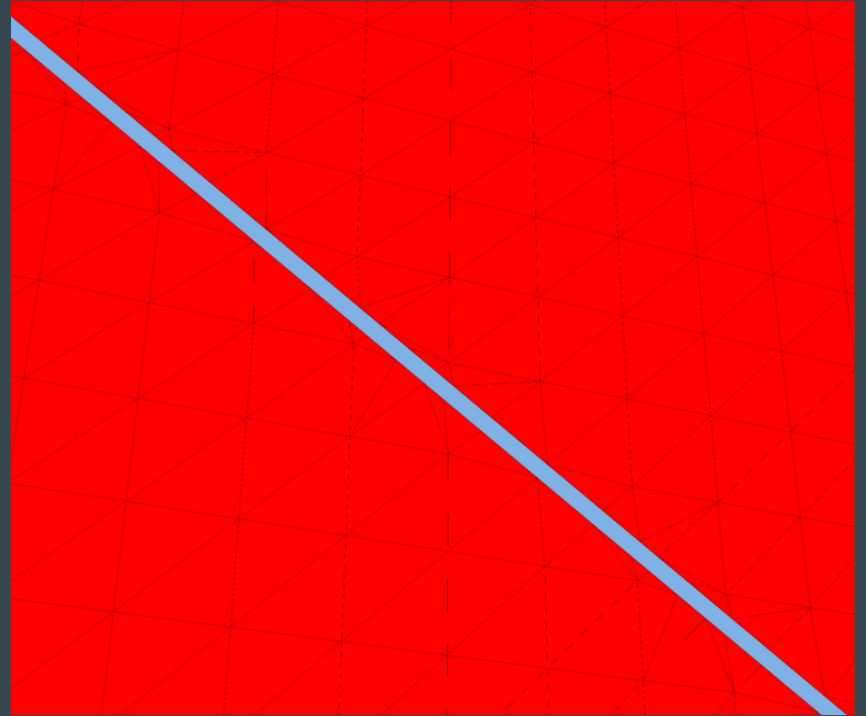
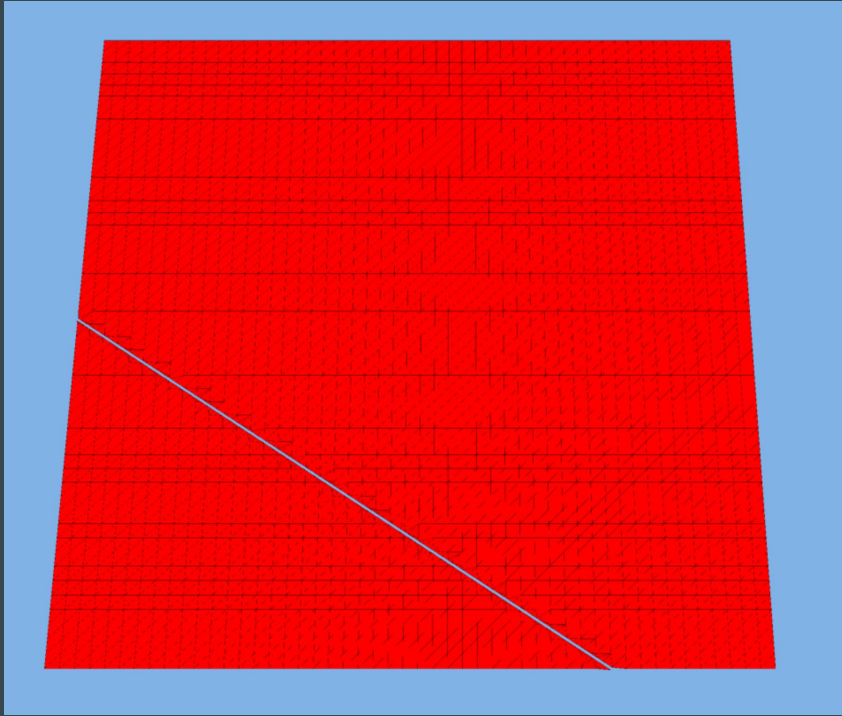
2. Small Mesh, Single-Segment Cut, 2



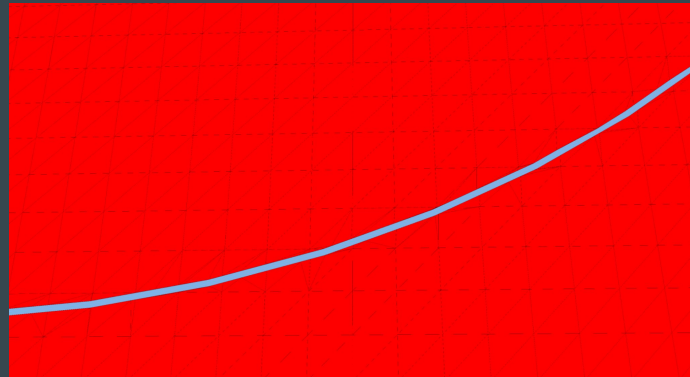
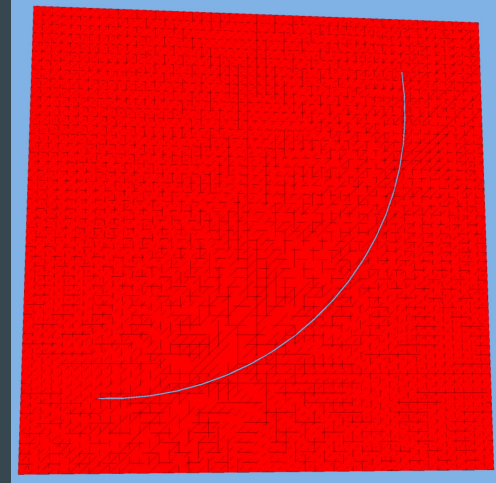
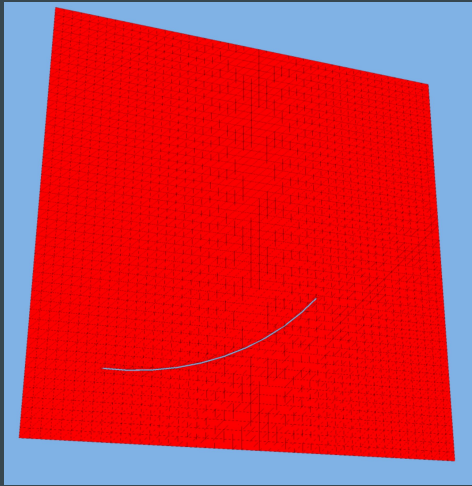
3. Small Mesh, Multi-Segment Cut



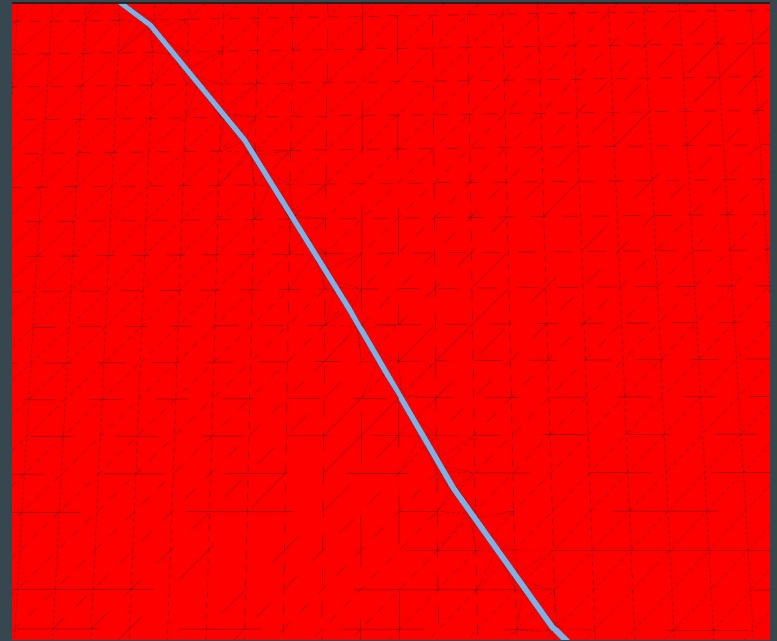
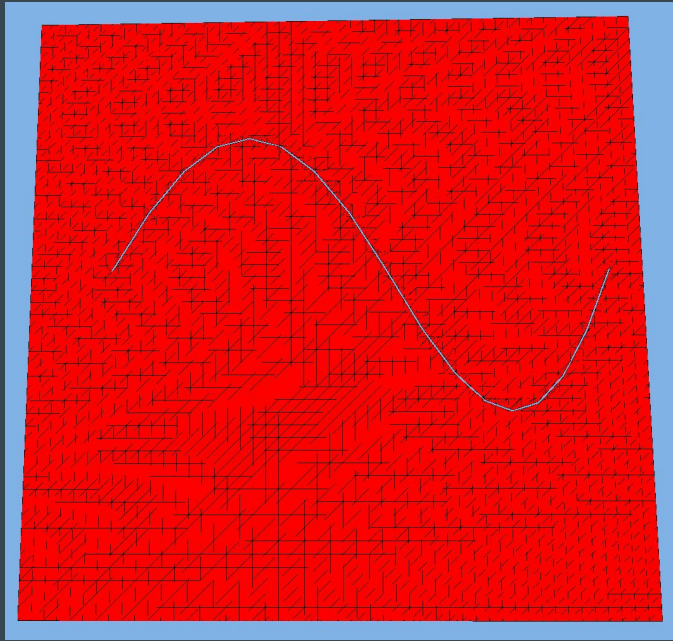
4. Large Mesh, Single-Segment Cut



5. Large Mesh, Multi-Segment Cut



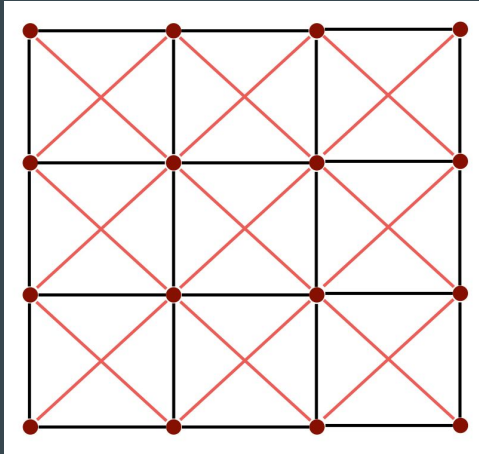
6. Large Mesh, Multi-Segment Cut, General Bezier Curve



Future Work(Integrating Physics)

1. Modelling the membrane

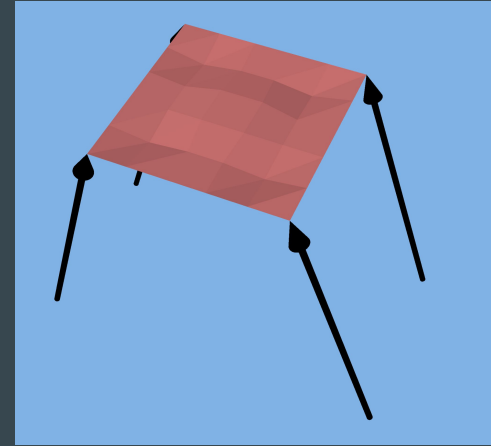
- As a ball-spring system(like a cloth) to incorporate its elastic nature
- Represented as an undirected graph with **particles as vertices** and **springs as edges**.
- At the moment, these vertices/edges are different from the vertices/edges of the mesh.



Structural Springs(side edges)

Shear Springs(diagonal edges)

Source: [4]



6 x 6 ball- spring system
with four corners clamped

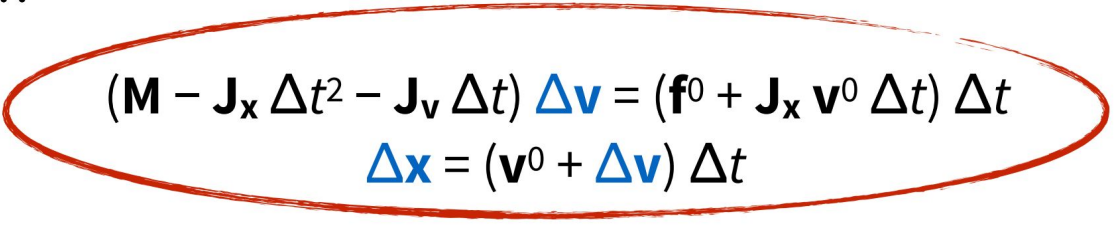
2. Simulating the Membrane

- Backward Euler(with damping) for time integration

$$\mathbf{F}_i(t)/m_i = d\mathbf{v}_i(t)/dt = (\mathbf{v}_i(t) - \mathbf{v}_i(t - \Delta t)) / \Delta t$$

$$\mathbf{v}_i(t) = d\mathbf{x}_i(t)/dt = (\mathbf{x}_i(t) - \mathbf{x}_i(t - \Delta t)) / \Delta t$$

- Equation solving using Newton's Method(number of iterations = 1)


$$(\mathbf{M} - \mathbf{J}_x \Delta t^2 - \mathbf{J}_v \Delta t) \Delta \mathbf{v} = (\mathbf{f}^0 + \mathbf{J}_x \mathbf{v}^0 \Delta t) \Delta t$$
$$\Delta \mathbf{x} = (\mathbf{v}^0 + \Delta \mathbf{v}) \Delta t$$

Solving Backward Euler using Newton's Method (Source: [4])

where $\mathbf{J}_x = d\mathbf{f}/d\mathbf{x}$ and $\mathbf{J}_v = d\mathbf{f}/d\mathbf{v}$ are the Jacobians of the forces

References

- [1] Manos Kamarianakis, Nick Lydatakis, Antonis Protopsaltis, John Petropoulos, Michail Tamiolakis, Paul Zikas, George Papagiannakis. "Deep Cut": An all-in-one Geometric Algorithm for Unconstrained Cut, Tear and Drill of Soft-bodies in Mobile VR. arXiv:2108.05281.
- [2] Manos Kamarianakis and George Papagiannakis. An All-In-One Geometric Algorithm for Cutting, Tearing, and Drilling Deformable Models. arXiv:2102.07499
- [3] Geometry Processing Algorithms. Jerry Yin, Jeffrey Goh.
- [4] Lecture Slides(COL865, Physics Based Animation, IIT Delhi). Prof. Rahul Narain.
- [5] Baby B, Singh R, Singh R, Suri A, Arora C, Kumar S, Kalra PK, Banerjee S. A Review of Physical Simulators for Neuroendoscopy Skills Training. World Neurosurg. 2020 May;137:398-407. doi: 10.1016/j.wneu.2020.01.183. Epub 2020 Jan 31. PMID: 32014545.
- [6] Lecture Slides(COL781, Computer Graphics, IIT Delhi). Prof. Prem Kalra.

THANK YOU