As Rigid as Possible Shape Interpolation

Team 19

CHAITANYA GARG

1 INTRODUCTION

As rigid as possible shape interpolation technique is employed for morphing. Morphing is a technique used to transform one graphical object into another and create a smooth transition for the same. Given a source and a target image, the method aims to find a rigid region undergoing the least change and hence remains static and an elastic area undergoing a transition. The part is rigid means that it consists of techniques such that it is least distorting. The primary aim is to morph two images with similarities and devise an algorithm to blend two objects with different shapes using minimal human interaction. It can also be extended to morphing two contradictory images, like generating a transition from U to T. Minimal human interaction prevents further changes due to human error, so they retain their configuration and similarities as much as possible. The reason this technique is used is to produce more natural-looking and smooth morphs, to reduce human interaction as much as possible and to improve morphing accuracy as much as possible.

2 LITERATURE REVIEW

The research paper by Marc Alexa et al.[2] aims to venture into the realm of metaphysical entities by stating that "Most of the cited object-space morphing techniques are concerned with the correspondence problem while simply linearly interpolating the corresponding vertices, not taking into account that the blended shapes are implicitly representing metaphysical entities." This attempt comprises blending the interior rather than the boundaries to get a sequence of in-between shapes that are locally least distorting.

The process can be broken down into the following stages: getting the set of vertices, triangulating the set of vertices, applying a least distorting constraint on the triangulated vertices, and optimizing the position of the vertices to minimize the object's deformation. The steps above are for 2-dimensional objects, and to apply them to 3-dimensional objects, we use tetrahedralization instead of triangulation, which can be extended for higher dimensions. First, we input two images, source and target images. Then, we get a set of points from those images using an algorithm of choice like cloud sampling, which extracts a random collection of points or using a surface reconstruction algorithm. The paper has yet to go into much detail on this and left it to the user's choice.

Then, the paper applies triangulation using the Delaunay algorithm, which selects triangles to maximize the minimal angle for each triangle. An alternative is to put a constraint on the maximum side of the triangle, which improves the quality but is slower.

Once triangulation is done, we arrive at the primary purpose, which is triangle rotation, such that it is the least and the path is simple. First, we handle rotation for a single triangle. So to convert the source vertices of P($\overrightarrow{p}_1$,$\overrightarrow{p}_2$,$\overrightarrow{p}_3$) to target Q($\overrightarrow{q}_1$,$\overrightarrow{q}_2$,$\overrightarrow{q}_3$), they use the following affine tranformation :

$$A\overrightarrow{p}_i + \overrightarrow{l} = \begin{pmatrix} a1 & a2 \\ a3 & a4 \end{pmatrix} \overrightarrow{p}_i + \begin{pmatrix} l1 \\ l2 \end{pmatrix} = \overrightarrow{q}_i, \quad i \in \{1, 2, 3\} \tag{1}$$

When dealing with triangulations, we consider a source triangles $P_{i,j,k}$= ($\overrightarrow{p}_i$, $\overrightarrow{p}_j$, $\overrightarrow{p}_k$) and its corresponding target triangles $Q_{i,j,k}$= ($\overrightarrow{q}_i$, $\overrightarrow{q}_j$, $\overrightarrow{q}_k$). To do this, a matrix is $A_{i,j,k}$ is computed as in Eq.1. Since, we a set of vertices

Author's address: Chaitanya Garg, chaitanya1248@iiitd.ac.in.

is related to more than 1 traingle in general, a mapping cannot be constructed

$$V(t) = (v_1(t), \dots, v_n(t)), \quad t \in [0,1] \tag{2}$$

where,

$$V(0) = (p_1, \dots, p_n), \tag{3}$$

$$V(1) = (q_1, \dots, q_n), \tag{4}$$

They have defined $B_{i,j,k}$(t) to be the matrix in affine transformation from $P_{i,j,k}$ to $\overrightarrow{v}_i$(t),$\overrightarrow{v}_j$(t),$\overrightarrow{v}_k$(t) as

$$B_{i,j,k}(t)\overrightarrow{p}_f + \overrightarrow{l} = \overrightarrow{v}_f, \quad f \in \{i,j,k\} \tag{5}$$

Once this is handled, the purpose is to minimise the error function E_V, where V(t) is the intermediate shape obtained. E_V is expressed as a quadratic function (T is the set of triangulations and ||.|| is the Frobenius norm)as

$$E_V = \Sigma_{i,j,k \in T} ||A_{i,j,k}(t) - B_{i,j,k}(t)||^2 \tag{6}$$

Minimizing E_V is further solved by setting the gradient of E_V over the free variables to 0 which yields the following solution :

$$V(t) = -H^{-1}G(t) \tag{7}$$

Upon applying, the above equations they were able to enforce the constraints and obtain the sequence of matrices pertaining to the morphification of a source image into a target image.

3 APPROACH

Image Reading

The first step towards doing image processing is to read the image and be able to display it on one's screen. For this task OpenCV setup was done on the virtual machine.Then, the image was read which was handled with the help of imread() and displayed on the screen with imshow(). Challenge faced in setup, opencv module was not getting connected to the pkgconfig file so, made a file named opencv.pc and exported its location in the file /./bashrc

Point Extraction

Once, image reading and showing became possible we arrive at the next main step which is to extract the points from the image.Since, it is not possible to directly apply extract the points the following procedure is applied:

MedianBlur -> GreyScaling -> SobelFilter -> NonMaximalSuppression -> SetBoundaryto0 -> minDistancePoints

Lets take a glance at all the methods used in the procedure:

Median Blur

We select a suitable n to make a nxn kernel such that n is odd and the center of the kernel coincides with the respective row and col position of a particular pixel. We then create an array and add all the elements coinciding with the kernel, not adding those which lie out of bounds and then, we sort the array. The corresponding row,col pixel is set to the middle value of the array and the algorithm is extended for all 3 color channels,i.e, rgb. Also, incase the array is of even length then, the corresponding row,col pixel is set to the avg of the two middle values of the array.

GreyScaling

For each corresponding pixel the value of each of the 3 color channels is set as the average of the 3 color channels.

$$r = (r + g + b)/3 \tag{8}$$

$$g = (r + g + b)/3 \tag{9}$$

$$b = (r + g + b)/3 \tag{10}$$

This has to be done simultaneously otherwise wrong value might settle.

Sobel Filtering

The image is traversed pixel by pixel and appropriate gradients are calculated by the use of 3x3 kernels Gx and Gy.

$$\mathrm{G}_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix},$$

$$\mathrm{G}_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

For each pixel we have to ensure that the center of the kernel correspond with the respective row and col value of the pixel.Intially, xVal=0 and yVal=0 then, we apply the following equation:

$$xVal += Gx(weight) * img[row][col] \tag{11}$$

$$yVal += Gy(weight) * img[row][col] \tag{12}$$

where Gx(weight) and Gy(weight) denote the corresponding weight upon overlapping the kernel over the corresponding choice of row and column.Finlly the pixel value is set as: (for all 3 color channels once again!)

$$img[row][col] = (|xVal| + |yVal|)/2 \tag{13}$$

Non Maximal Suppression

We traverse the pixels in an orderly fashion and look at the neighbouring ones. In my case I have used a 5x5 kernel centered at the respective position of the row,col value and if that pixel has the highest value, it is selected else not considered. Once, this is done all the pixels are traversed once and those below a certain threshold are converted to 0 while others are set to white color.

SetBoundaryto0

We simply traverse along the boundary the set the boundary values to 0 to remove the boundary noise.This is necessary since if not handled, unnecessary triangles will be obtained as noise.

MinDistancePoints

We traverse all the pixels and select those which satisfy a min distance criterion, in my case I have taken this value to be 5 for now. This is necessary since not doing this will lead to very small triangles being generated and nothing meaningful could be obtained.

Delaunay Triangulation

I have used the Bowyer Watson algorithm for implementing Delaunay triangulation. Implemented the Point2dim and Tringlee classes to manage points and Triangles respectively. Point2dim takes in the float values of the x, y of an arbitary point while Trianglee handles 3 such points made by Point2dim. A major challenge was that while forming the classes, 'arbitary class encountered error was occuring' which was due to the fact that the names I was choosing like Point2d, Triangle had already existing implementations in opencv which were clashing with my classes and producing error. To handle this challenge, I thought a bit out of the box and chose, understandable yet similar names for the classes and the situation got resolved.

Next the workflow of the algorithm is follows :

PointsInput -> SuperTriangleMade(which is basically a very big triangle capturing all the points in it) ->make an empty trinagles array and add supertriangle to it -> EachPoint iterated one by one ->

Point Procedure:

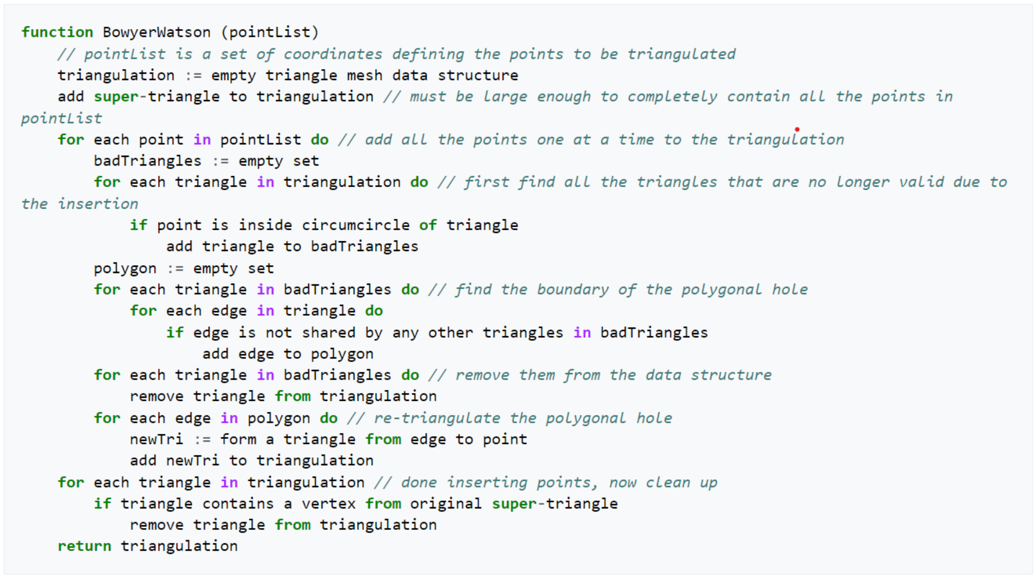
for each point check if it lies in the circumcircle of an already existing triangle, if it does add that triangle to the badTriangle array else, nothing -> for the bad triangles construct a polygon with the edges of the badtriangles ensuring no repetition(used set for this) -> remove all the triangles from the existing one's array -> using the points of the subsequent edge from the created Polygon make triangles -> once all is done proceed over to the next point .

Once, all points are iterated over, go over all the triangles and remove those which share even one edge with the super trinagle since the three vertices of the super triangle are redundant.

The major challenge was to handle the procedure of steps and construct appropriate data structures for the same while at the same time making sure that the complexity doesn't shoot up.

To handle if Point is in the circumcircle of a given triangle, a function is implemented which checks the same by the use of determinants. This was a challenge as the algorithm requires extensive multiplication which causes the algorithm to give segmentation fault, to tackle it, I opted to scale and rescale my values by 100.

Followed the following procedure for Bowser Watson's algorithm of Delaunay triangulation.



```
function BowyerWatson (pointList)
    // pointList is a set of coordinates defining the points to be triangulated
    triangulation := empty triangle mesh data structure
    add super-triangle to triangulation // must be large enough to completely contain all the points in
pointList
    for each point in pointList do // add all the points one at a time to the triangulation
        badTriangles := empty set
        for each triangle in triangulation do // first find all the triangles that are no longer valid due to
the insertion
            if point is inside circumcircle of triangle
                add triangle to badTriangles
        polygon := empty set
        for each triangle in badTriangles do // find the boundary of the polygonal hole
            for each edge in triangle do
                if edge is not shared by any other triangles in badTriangles
                    add edge to polygon
        for each triangle in badTriangles do // remove them from the data structure
            remove triangle from triangulation
        for each edge in polygon do // re-triangulate the polygonal hole
            newTri := form a triangle from edge to point
            add newTri to triangulation
    for each triangle in triangulation // done inserting points, now clean up
        if triangle contains a vertex from original super-triangle
            remove triangle from triangulation
    return triangulation
```

Fig. 1. Bowyer Watson's Algorithm steps.
(https://en.wikipedia.org/wiki/Bowyer%E2%80%93Watson_algorithm).

Mapping triangulation

Now that we have created the triangulation for both the source and the target, we need to map the two triangulation so that we could further perform the interpolation.

The basic idea was taken from [1] by Aronov et el. wherein the author gives the idea on how to map 2 polygons based on the minimal vertice distance calculated by means of euclidean distance. I extend this idea to apply them to apply my triangulations and arrive at 2 possible methods:

• Euclidean distance on Vertices of triangles
• Euclidean distance on Centroid of triangles

I tried both the algorithms on a number of images and found Centroid method to be way more consistent than the vertices one as it takes into the average of the vertices giving an optimal fit as per my understanding.

I encountered the problem of having a very high time complexity in my mapping algorithm. Initially, it was $O(n^2)$ with just iteration over the source and target triangulation but it was missing out the case wherein the last object gets mapped to the first one but it was more optimal with the second one making my intial approach sub-optimal. So, to tackle the problem I placed the unmapped source triangulation in a queue and then mapped to best one if that is not mapped and in case a mapping exists checking which is the better suitor if its the existing one then, leave it be and iterate to the next. In case this was not the case, the current triangle will be assigned the target triangle and the one already mapped will be placed back in the queue. However, this algorithm showcased an expotential time complexity and hence, I changed my approach. I create a map with source Triangle as the key and a priority queue(Min heap) as the pair(euclidean distance,final Triangle) as the value. Once this, was pre-calculated, I ran the algorithm to iterate over the objects once again assigning them the desired triangle if it was unmapped and checking the better fit in case of mapped. If the existing combination is the better fit then we iterate over the next set of values else we change the mapping and assign the source triangle as unmapped and push it into the queue. Algorithm is performed till the queue is not empty. In case, the heap for a triangle becomes empty that key value pair is removed (generally happens when source triangulation has more triangles than the target triangulation)

Once the mapping is done, the best fit alignment of the points is check by the means of euclidean distance on the vertices and aligning along the least distance one. This is necessary as the movement might cause an inversion in case this is not performed.

Enforce movement in a simple path

I followed the approach followed by Marc Alexa et el.[1], that was to break the initial and final into SVD forms and then generate the desired position for movement and shift the triangulation accordingly.

Problem faced, well the paper suggests to make a combined matrix having all the edges however this gives a matrix of vector 3-D vector with 3n rows and 3 cols which leads to segmentation fault or memory lapses for a large value of n. Even if that isn't the then Singular Value Decomposition formed takes a lot of time and space, which slows down the algorithm. To tackle this issue at hand, I opted to perform the SVD method a triangle a time to make sure and the corresponding change is stored in a map dp with key as the point and array of resultant points as the value. Once, this is done, the point is then translated to the mean of the resultant point.

The use of SVD itself enforces a simple path as stated by the paper [2]

Obtaining Points

To iteratively perform the algorithm, 2 approaches can be followed repeatedly perform the enforce simple path movement on the new resultant until convergence. This works but no difference observed from the 4-5th steps. Another one is to recompute the triangulation with the current triangles in hope for triangulation then, to map it and finally, apply simple path movement. This method was observed to work much better than the standard iterative approach.

The algorithm is converge once no change in arap energy difference is observed.

Obtaining the intermediate state

After each iteration the resultant mesh can be displayed over the source and final images to observe and compare the pattern of movement.In my algorithm the red meshes mean they are rigid and shouldn't be moved much while the green ones are tangible and should be moved to get the desired result.

4 RESULTS

The procedural pics were found to be as good as those on wikipedia and on google. The process of applying filters has been showcased through the ubuntu's logo image. Comparison of my code's sobel filter image and wikipedia's image is done.

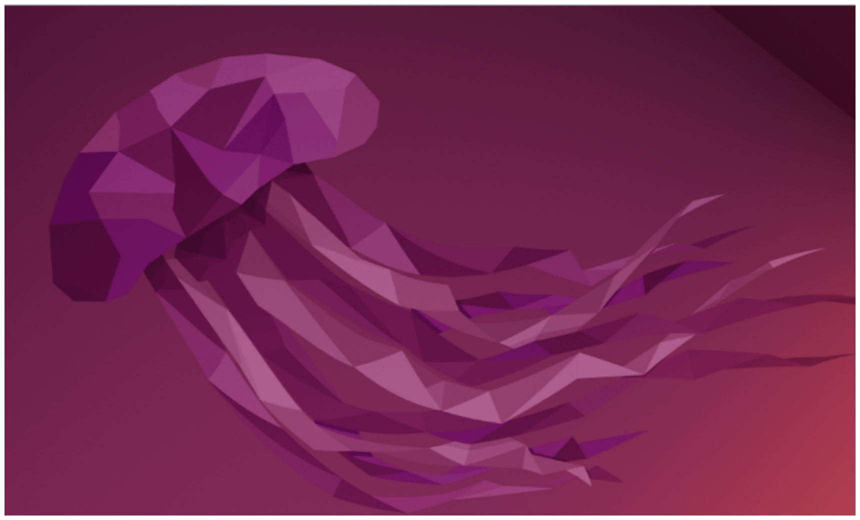


Fig. 2. Original/Initial Pic

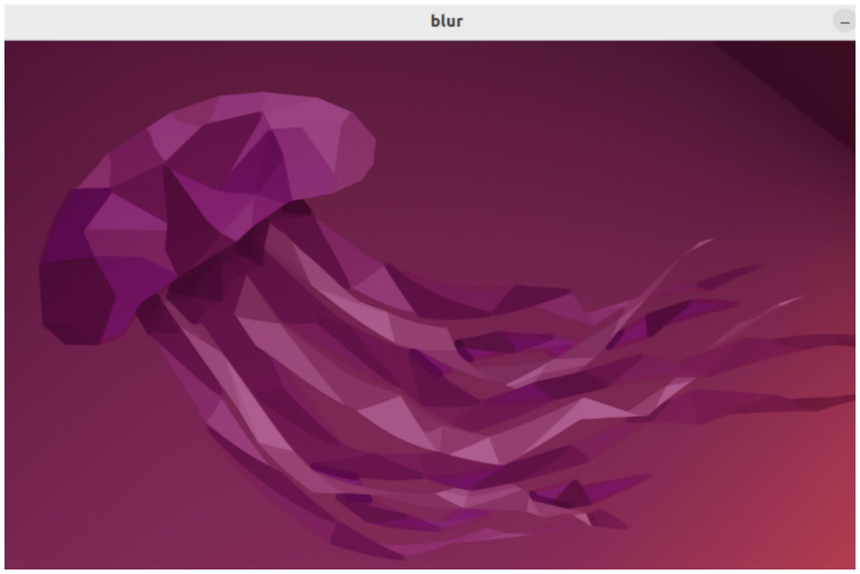


Fig. 3. Median Blur Pic, if you compare the central peaks with fig1 you will see a visible change

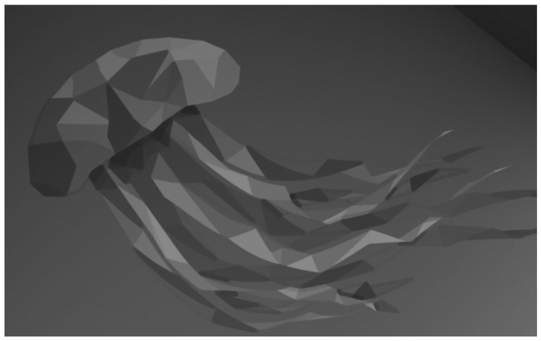


Fig. 4. GreyScaling

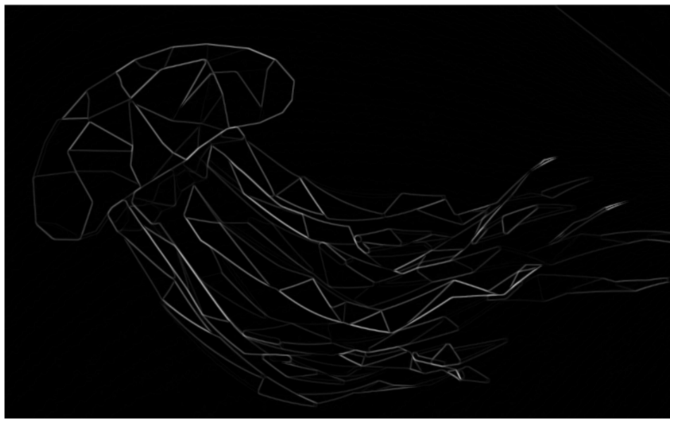


Fig. 5. Sobel Filtering

S. No.-Algorithm	My Algorithm's Time	Library Implementation Time
1-Median Blur	5.02s	0.055s
2-Grey Scale	0.0042s	0.012s
3-Sobel Filtering	0.1193s	0.032s

The results of the my implementation and library implementation didn't differ much in median blur and grey scale, since they are standardised algorithms. A bit difference was noted in Sobel Filter output which mine is assigned the gradient color while the library implementation makes them white for better viewing.

5 CONCLUSION

The As-rigid-as-Possible algorithm is quite accurate and fast for the triangular meshes. The meshes do differ by some amount so, in a real world scenario wherein fast computations are needed instantly then, we could use it as a baseline model with other model.

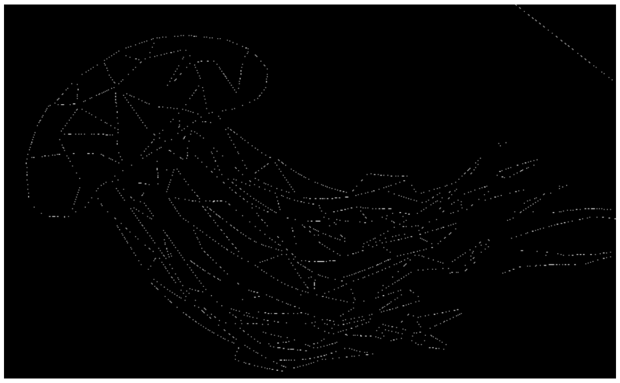


Fig. 6. Non Maximal Suppression

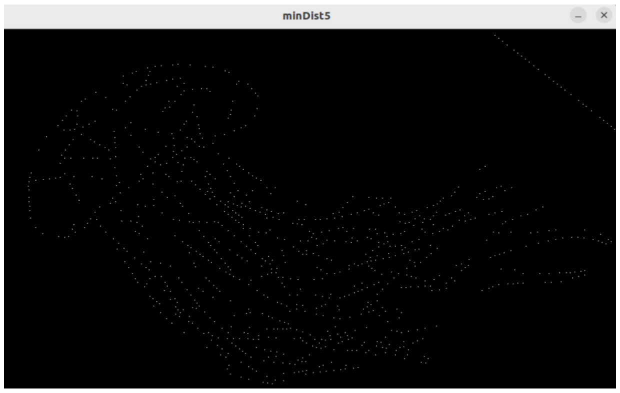


Fig. 7. min Point Distance Enforced

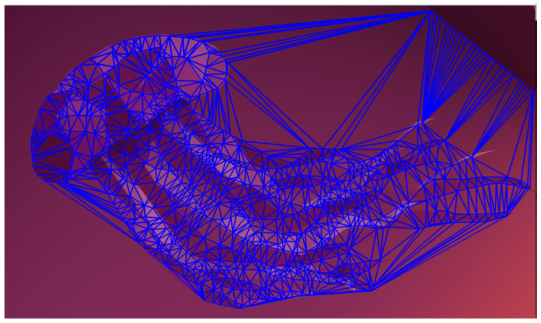


Fig. 8. Final Delaunay Triangulation Obtained after performing steps Fig2-7 and Bowyer Watson Algorithm

The algorithm itself has scope for improvement those being: To carry out triangle flips in Delanuay triangulation
To have better mapping function
To have a more sophisticated way for point extraction

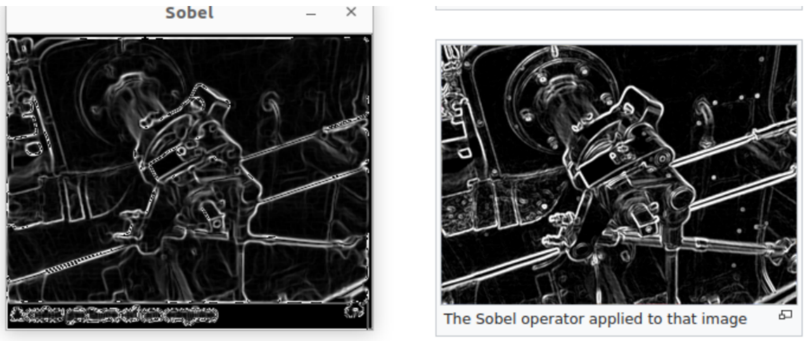


Fig. 9. On the left, the sobel filter applied by my algorithm and on the right is the one according to Wikipedia (https://en.wikipedia.org/wiki/Sobel_operator)

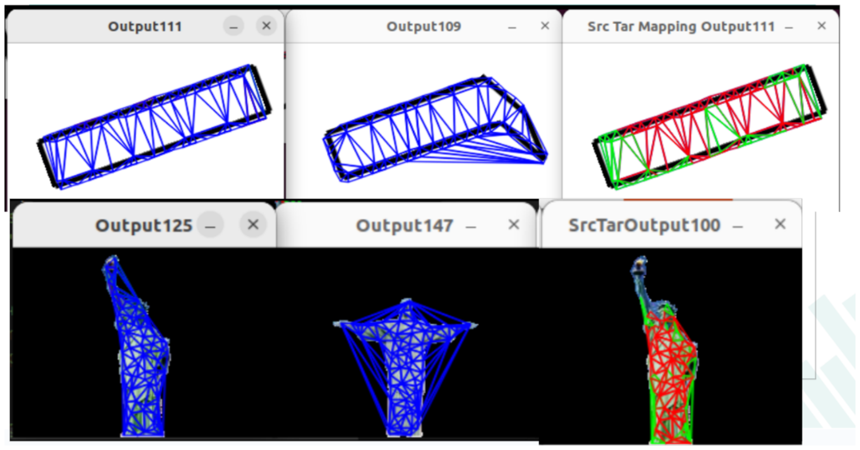


Fig. 10. The first row shows the case of a slanted box wherein the 1st image is the source, 2nd is the target, 3rd is the centroid Mapping Similarly follows for the second row

)

To carry out point interpolation in a go

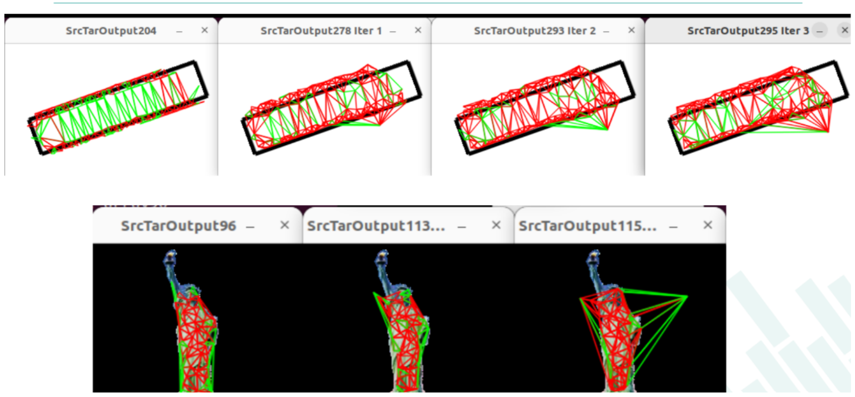


Fig. 11. The first row shows the case of a slanted box wherein the 1st image is the source with mapping, 2nd is the 1st is the resultant on the 1st iter, 3rd is the resultant on the 2nd iteration and vice versa. Similarly follows for the second row
)

6 MILESTONES

S. No.	Milestone	Member
	Mid evaluation	
1	Image inputing and processing-Completed	Chaitanya
2	Getting set of points from the given image/mesh-Completed	Chaitanya
3	Applying Delaunay Triangulation on set of points-Completed	Chaitanya
	Final evaluation	
4	Constraint triangle movement for rigid and elastic-Completed	Chaitanya
5	Enforce movement in a simple path-Completed	Chaitanya
6	Optimize vertex position for minimal deformation-Completed	Chaitanya
6	Obtain the sequence of images/meshes from source to target-Completed	Chaitanya

REFERENCES

[1] Raimund Seidel Boris Aronov. 1992. On compatible triangulations of simple polygons. https://www.sciencedirect.com/science/article/pii/0925772193900285?via%3Dihub
[2] David Levin Marc Alexa, Daniel Cohen-Or. 2000. As-Rigid-As-Possible Shape Interpolation. https://www.cs.tau.ac.il/~levin/arap.pdf