16811 - Math Project Report

Aditya Agarwal (adityaa2), Sarthak Ahuja (sarthaka)

Assistive Sketching and Animation Using Shape-Aware Moving Least Squares Deformations

Abstract

In this project we design an end-to-end sketching platform, which assists an artist to make complex scenes and characters by interpolating and extrapolating their tooltip. For this purpose, we build tools for creating simple shapes and also use bezier curves to smoothen their strokes. Next, we perform automatic rigging (skeletonization) of the scene with joints and handles to create pivots for animation. Rigging or skeletonization is a process for extracting connectivity of the original image while throwing away other foreground pixels. This can be seen as the inverse of what we have learned in class - given a line/curve find the important points that characterize the shape and can serve as potential pivots. Finally, we will explain and integrate an implementation of Shape-Aware Moving Least Squares deformations to give animation controls to the artist and allow them to deform the characters and the scene. A highlight of this closed-form method is that it tends to preserve deformations around point handles as well as line handles through appropriate weighting (using an intrinsic distance metric) in a manner that preserves the created shapes.

1 Introduction

Deforming 2-D images has many applications in a range of domains such as animation, medical imaging, and graphic design. We primarily address the usecase of animation and graphic design in this report to motivate the need for the upcoming algorithms and our designed platform. Traditional animation techniques can be imagined to be similar to a flip book where each frame of the animation sequence would have to be designed individually with slight variations, so that when one flips through the pages of the book quickly, the scene appears to be in animated (as shown in figure 1) [14].

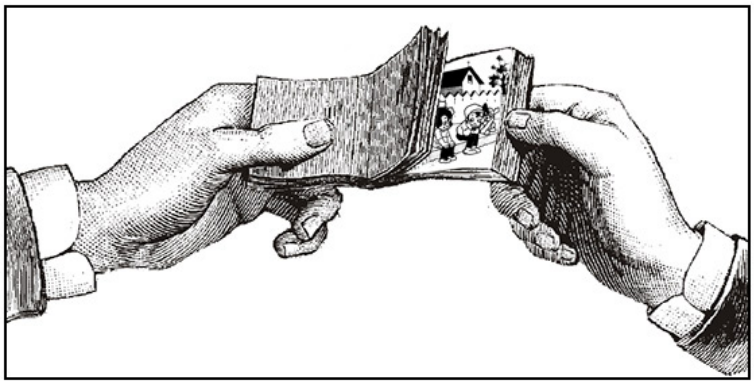


Figure 1: Flip Book

As machines such as scanners and xerox came into existence, instead of designing each page individually from scratch, one could now scan a scene and trace it's variations to save time. Later with the introduction of more powerful computing and the advent of computer graphics, we could represent these images digitally in the form of pixel matrices and make animation sequences even more conveniently. As one would now start to appreciate, an important stage in this process was and continues to be deforming a template (representing the scene or a character) frame by frame programmatically to make the design process easier. It is extremely crucial for the image to deform in an intuitive and realistic fashion to minimize the amount of manual correction needed from an animator.

Softwares such as Adobe Photoshop, Adobe Illustrator, etc. are the most common tools of trade in this field and allow animators to carry out such deformations. The most common deformation tools in these softwares operate on the assumption that the template in context is a convex structure. Example, they would encapsulate the template in a bounding box (hull) and deform the box thereby also deforming the inner template. While this may work reasonably well for convex shapes, for more complicated shapes (non-convex) we would require to move a certain area of the template (for example, the arm of a human silhouette) while restricting any deformation in the remaining areas 2. This property is what we refer to as being "Shape-Aware" and this process usually involves allowing the user to add a set of handles/pin points/bones in the template image and deform the image about these pivots. In our project we go a step further and automate this process of having the user to add these pivot points by implementing "skeletonization" - a process of finding these potential candidates for these pivot points. We also elaborate on the origins of a recent algorithm [23] that combines an interesting interior distance metric [20] with a moving least squares shape aware deformation [21] and integrate it into our platform.

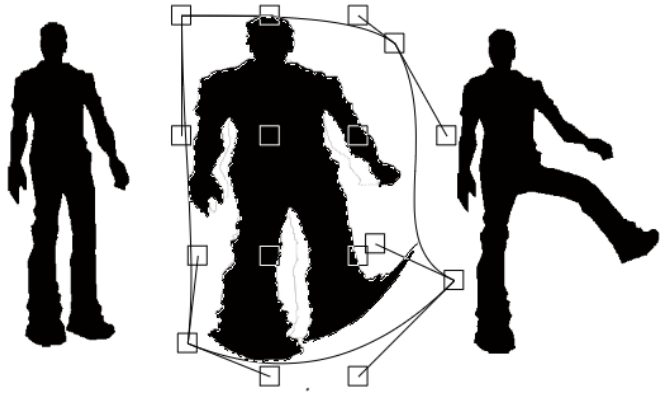


Figure 2: Left - Original Image, Middle - Deformation using a Bounding Box (Non-Shape Aware), Right - Shape Aware Deformation

In this report we will first go through a literature survey of existing techniques that we have used to design the platform. We will then elaborate on some of the necessary derivations and math behind these techniques as well as showcase results generated by our overall algorithm incorporated in the tool. Towards the end we will present a conclusion and scope for future work in enhancing the tool. We have named our tool ***Andy*** in honour of the famous Pittsburgh artist Andy Warhol [1].

2 Literature Survey

2.1 Skeletonization

Skeletonization is a process, in which we extract the skeletal remnant of an object in an image. Skeletonization preserves the connectivity and spread of the original object while discarding most of the other foreground pixels [19]. For example, in case of an image of a human body or animal, the skeletonization output will represent a human skeleton or the corresponding animal skeleton as shown in Figure 3. This concept can be extended to images of general objects as show in Figure 4. For these general objects we can think of the image skeleton as the point at which fires lit all along the boundary of the object meet each other [19].
Computation of an image skeleton can be done by several methods [18] :

1. Thinning : Thinning or erosion is the process of eroding foreground pixels in an image. In this method we keep thinning the image till we are left with only 1 pixel length of the skeleton.

2. Voronoi Skeleton : Voronoi diagram of a set of points is made by partitioning the space into cells so that each cell contains only 1 generating point.

Figure 3: Skeletonization of an animal image [4]

Figure 4: Skeletonization of a general image
[13]

If we consider points on the boundary of the image as a set of points, then their Voronoi diagram will converge to a skeleton as the number of boundary points go to infinity as shown in Figure 5. However the Voronoi method is computationally expensive and can have a high runtime.

3. Distance Transform : The distance transform of an image is an image where each pixel represents the distance to the closest boundary point. In our boundary on fire analogy, if the boundary of the object is on fire, then the distance transform can be made by noting the time it takes the boundary fire to reach interior points in the image. Once we have the distance transform, we can compute the skeleton by extracting local extremes or points furthest from the boundary as shown in Figure 6. Computation of distance transform of an image is fast and can be done in O(n) time where n is the number of pixels in the image.

Because of their connectivity and shape preserving nature, image skeletons serve

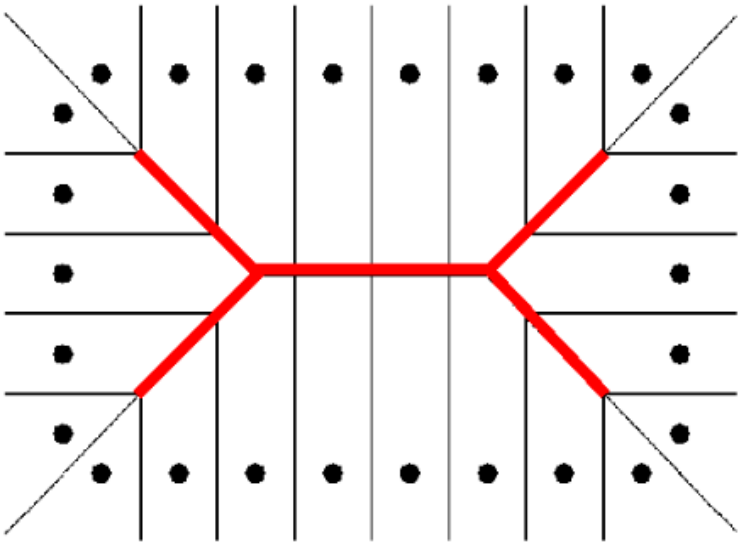


Figure 5: Voronoi diagram of boundary converging to skeleton [18]

as an ideal candidate for line or polyline handles that provide an artist an intuitive way of animating an object in an image [23]. More specifically, following properties of the skeleton make it appropriate to be used as a means of providing control to artists when animating objects [18]:

1. Topology : The skeleton preserves the topology or the shape of the object
2. Geometry : The skeleton is present in the middle of the object and is invariant under various image transformations

Figure 7 shoes control handles added to manipulate or animate a curve in the software Maya. In most softwares these control handles need to be added manually for animation. Another thing that needs to be taken into account when animating objects in this way is shape deformation. Shape deformation refers to the changes in an object's image when its stretched or rotated (transformed) through control handles for the purpose of animation.

2.1.1 Harris Corner Detection

Harris corner detection is used to find sudden changes in intensity in an image [5]. Traditionally it has been used to detect corners in an image which can then be used for feature matching across multiple images of the same scene. Corners are ideal candidates for image features as they are invariant to translation, rotation and illumination. Figure 8 shows how corner regions corresponding to significant changes in intensity in all directions are detected by Harris corner detection.

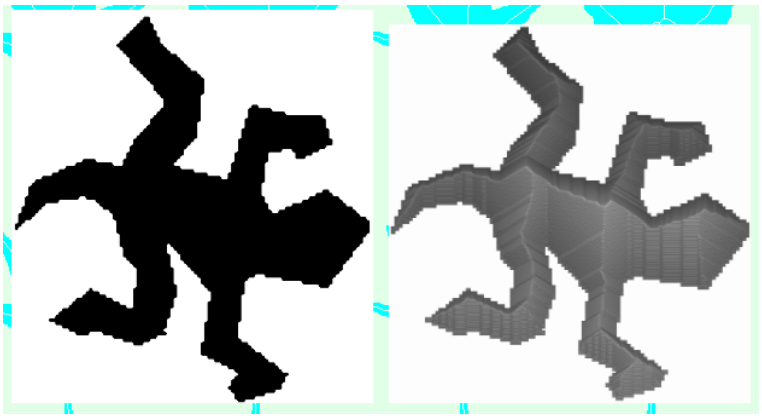


Figure 6: Distance transform of an image with the ridge showing the local extremes or the skeleton [18]

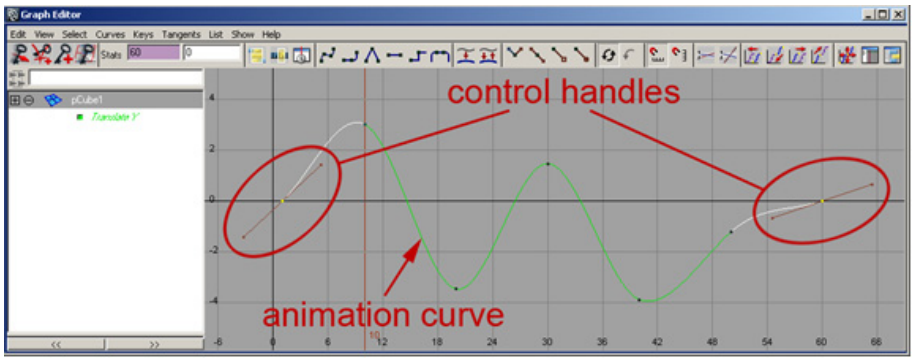


Figure 7: Animation control handles in Maya software [3]

Harris corner works by shifting a window function $w(x, y)$ or kernel over the image by (u, v) in 2D and calculating the change in intensity :

$$E(u, v) = \sum_{x,y} w(x, y)[I(x + u, y + v) - I(x, y)]^2 \tag{1}$$

Using first order Taylor's approximation for 2D functions and billinear approximations [7] we can obtain :

$$E(u, v) = \sum_{x,y} w(x, y)[u^2 I_x^2 + 2uvI_xI_y + v^2 I_y^2] \tag{2}$$

$$E(u, v) = \sum_{x,y} w(x, y) \begin{pmatrix} u & v \end{pmatrix} \times \begin{pmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{pmatrix} \times \begin{pmatrix} u \\ v \end{pmatrix} \tag{3}$$

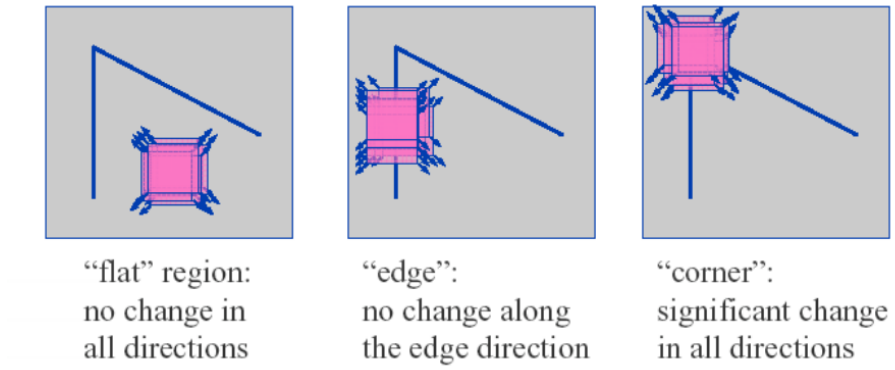


Figure 8: Harris Corner Detection [7]

From the above approximation we can obtain the parameter M and corner response R [7]:

$$M = \sum_{x,y} w(x,y) \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix} \tag{4}$$

$$R = detM - k(traceM)^2 \tag{5}$$

Once the response R has been computed for every image location, we chose a threshold and select all locations above that threshold, these become our corner or feature locations.

2.1.2 Hough Transform

Hough transforms are used to detect lines or circles or other parametric curves in an image. Given a collection of points, the hough transform operator returns all possible curves that can fit the given collection of points. Given a thresholded edge image as input, hough transform works in the following way :

1. Represent the equation of a line passing through point (x, y) in parametric form :

$$\rho = x\cos\theta + y\sin\theta \tag{6}$$

2. Divide the parameter space (ρ, θ) into bins with a fixed step size of each ρ and θ. This creates an accumulator array as shown in Figure 9

3. Now each pixel in the image casts a vote for every pair of (ρ, θ) that denote a line passing through that point.

4. After every point has voted, we select the parameters that receive maximum votes and they correspond to the lines present in the image

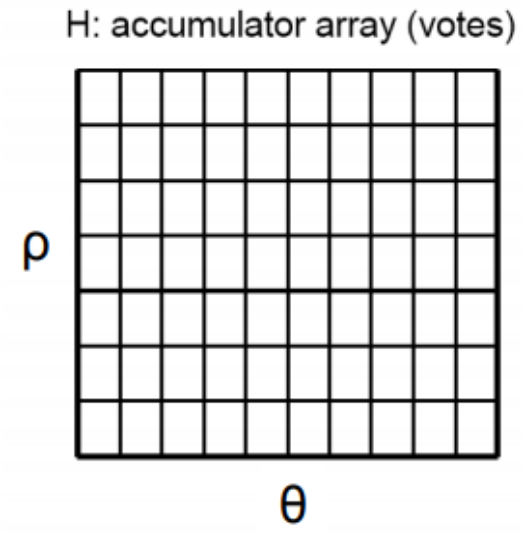


Figure 9: Hough Transform Accumulator [8]

2.2 Bezier Curves

Bezier curves represent a class of curves which can be easily be created through line handles and are widely used in computer graphics to create smooth curves while sketching objects. The equation representing Bezier Curves is given by :

$$B(t) = \sum_{i=0}^{n} C_i^n (1-t)^{n-i} t^i P_i \tag{7}$$

In above equation if we set $n = 3$ we get Cubic Bezier Curves that can be created using 2 anchor points and 2 control points. If we connect each anchor point to each control point we obtain 2 line handles as show in Figure 11. The

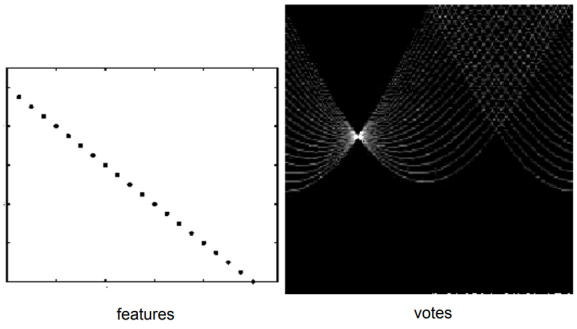


Figure 10: Hough Transform Voting [8]

slope of line handles at the ends of curves remains tangential to the curve at those points and this allows us to manipulate the curve easily.

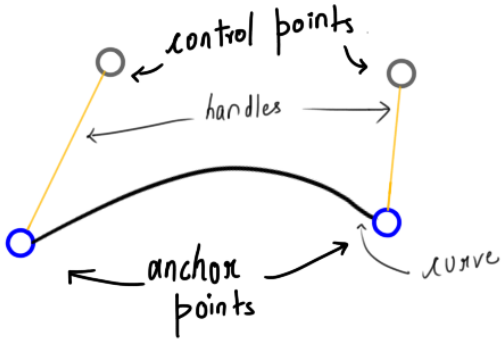


Figure 11: Line handles and control points in a Bezier Curve [10]

2.3 Shape Deformation

2.3.1 Grid Based Techniques (Free Form Deformation) - Seidberg and Parry

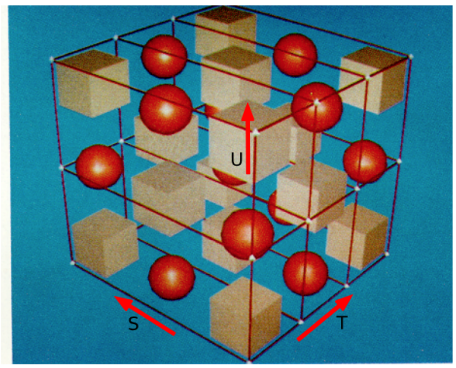


Figure 12: Example of how a boxoid with control points would look like. The box encloses all the objects that will be effected by any deforms done on the boxoid using the control points [22]

Free-Form Deformation (FFD)[22] is a technique in Computer Graphics used to model simple deformations of rigid objects. It can be extended to deformation in 2-D spaces as well. It is based on the idea of enclosing an object within a bounding box (or hull), and transforming the object as the hull is deformed.

Deformation of the hull is based higher-dimensional analogs of parametric curves such as Bezier curves. In 3-D space, it can be formally stated as a tensor product trivariate Bernstein polynomial (Bernstein polynomial is a polynomial that can be expressed as a linear combination of Bernstein basis polynomials ($b_{v,n}(x) = \binom{n}{v} x^v (1-x)^{n-v},\ v = 0, .., n$)). This is similar to how we express Cubic Bezier Curves in 2-D with n = 3 as a Univariate Bernstein Polynomial $B(t) = \sum_{v=0}^{3} b_{v,3}(t) P_v$ (note that it is defined over 4 points, hence it is cubic and it is defined only on one variable t, hence it is univariate). However in this case we define a bounding boxoid in space using 3 orthogonal vectors (S, T, U) defined by with a fixed number of control points (l, m, n) on each face arranged as a grid. An example of the same is shown in figure 12. Without going into much detail we can jump to the fact the a point x defined by $x = sS + tT + uU$ in the boxoid frame of reference, will have it's deformed form defined as follows -

$$B(s,t,u) = \sum_{i=0}^{l} \sum_{j=0}^{m} \sum_{k=0}^{n} b_{i,l}(s) b_{j,m}(t) b_{k,n}(u) P_{ijk} \tag{8}$$

Advantages -

1. It's simple! The formulation shown above is reasonably straightforward (as an extension to high dimensional Bezier curves) to both understand and implement and allows for a quick and fast way to find the deformed points.

Disadvantages -

1. Requires manual alignment of all the gridlines corresponding to control points to the image features in case of complicate non-convex shapes. It may also involve the user to break the non-convex shape into convex components and apply FFD on each separately. Overall, highly inconvenient.

2.3.2 Integrate Shephard's Interpolation on to the Grid - Beier and Neely

Shephard's interpolation is a method to assign a value u to a point x, based on an inverse distance weighting of existing points x_i with available values $u(x_i)$. It can be depicted as follows -

$$u(x) = \frac{\sum_{i=1}^{N} w_i(x) u(x_i)}{\sum_{i=1}^{N} w_i(x)} \tag{9}$$

$$w_i(x) = \frac{1}{d(x, x_i)^p} \tag{10}$$

where d is a distance metric and p is control variable (in out case u would really be the coordinates themselves). Beier and Neely in their work [15] use this distance metric to find the deformation among image pixels given the deformation among a set of user provided line handles.

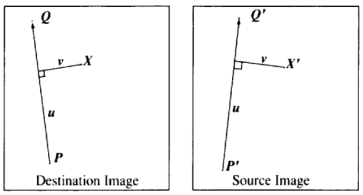


Figure 13: X' is the location in the source image (undeformed) for the pixel at X in the destination image (deformed). [15]

Note that earlier we were talking about control handles as points and now we are talking about control handles as lines. Refer to figure 13 and we can define the algorithm to find the deformed image as follows (from [15]) -

1. Find the corresponding vectors u and v using straight forward vector algebra projections.
2. Find the X' in the source image for that u and v.
3. destinationImage(X) = sourceImage(X')

When extending to multiple line handles, as would be the general case, inverse weighting similar to that of the Shephard's interpolation will be incorporated to find the value of a pixel in the deformed image and the weights will be defined as follows -

$$w = (\frac{length^p}{(a + dist)})^b \tag{11}$$

where, a,b and p are control parameters, dist is the perpendicular distance of a point to the line, and length is the length of the line.

Advantages -

1. This is much better than FFD as now we can use line handles to control deformations in non-convex templates.
2. The use of the inverse distance metric is novel and forms the basis for the future algorithms that we will talk about shortly.

Disadvantages -

1. The authors report ghost warps that occur in this formulation and they require the user to add very precise lines to get desired outputs. In short, the algorithm is very noisy as it is based on direct euclidean distance and there are no constrained transformations (such as affine, similarity, etc) overlooking the deformations.

2.3.3 Thin Plate Splines - Bookstein

The TPS (Thin Plate Splines) algorithm tries to fit a function $f(x)$ between given K control points locations, undeformed (x) and deformed (y) that minimizes -

$$E_{tps,smooth}(f) = \sum_{i=1}^{K} ||y_i - f(x_i)||^2 + \lambda \int\int [(\frac{\partial^2 f}{\partial x_1^2})^2 + 2(\frac{\partial^2 f}{\partial x_1 \partial x_2})^2 + (\frac{\partial^2 f}{\partial x_2^2})^2] dx_1 dx_2 \tag{12}$$

The latter half of the above equation is meant to add a smoothness constraint on function f. It is also provided to us that a closed form unique solution to the above minimization exists in the paper [16] and further we learn that this function is parameterized by α which contains 2 matrices (c and d). For a point z in the undeformed image we can get the deformed coordinates as -

$$f_{tps}(z, \alpha) = f_{tps}(z, d, c) = z.d + \phi(z).c = z.d + \sum_{i=1}^{K} \phi_i(z) c_i \tag{13}$$

where, ϕ is a RBF kernel defined as - $\phi_i(z) = ||z - x_i||^2 log||z - x_i||$ and d represents an affine transformation.

Advantages -

1. This method addresses the drawbacks of previous Beier and Neely algorithm by explicitly taking into account the smoothness component of the deformation. f can always be decomposed into a global affine and a local non-affine component.

2. This method also does not require any additional control parameters.

3. Due to using the RBF kernel, it is infinitely differentiable.

Disadvantages -

1. The template goes under local non-uniform scaling and shearing which is often undesirable.

2. Also as we see, a global affine transformation though leads to smooth deformations, they don't look that real.

2.3.4 As Rigid As Possible - Igarashi

This paper [17] proposes a point-based image deformation approach where the amount of local scaling and shearing is minimized by explicitly finding the rotation part and a scale part of the transformation function each handled by an independent quadratic error function and combining them thereafter. The first step of this algorithm is to create a triangular mesh over the input template. This is done to make computation feasible. As shown in 14, the input to the algorithm is the xy-coordinates of the handles on the mesh vertices and the

output is the xy-coordinates of the remaining free vertices that minimize the deformation error associated with all triangles in the output mesh.

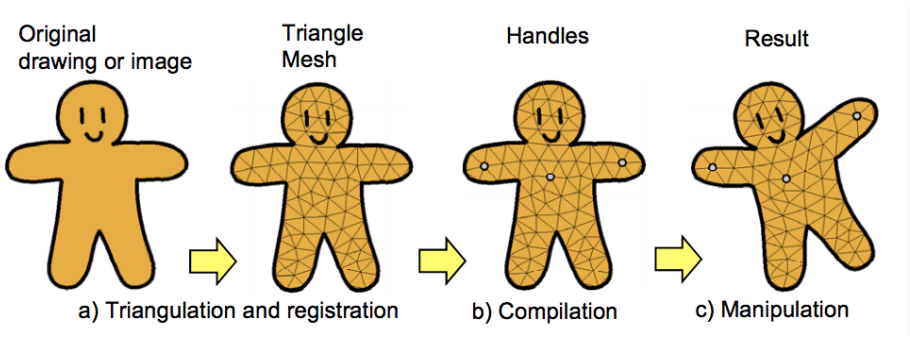


Figure 14: Overview of the "As Rigid As Possible" Algorithm [17]

It is interesting to note how they compute the deformation error between two mesh triangles. For the two triangles, they align one of the edges of both triangles against each other and compute the eucliden distance error between the third vertex as depicted in figure 15. They do this for all three vertices and sum them together as -

$$E_{v_0,v_1,v_2} = \sum_{i=1,2,3} ||v_i^{desired} - v_i'||^2 \tag{14}$$

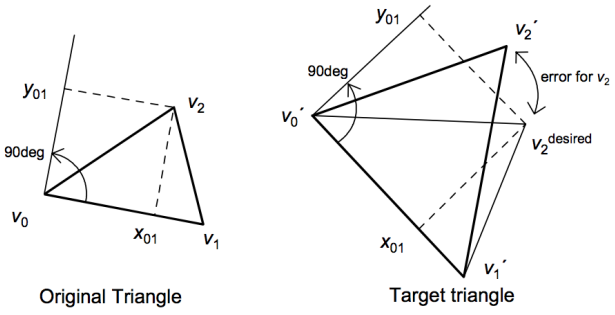


Figure 15: Deformation Error Between Triangles in a Mesh [17]

The error is then summed over all mesh triangles and minimized over the free points (non-handle) to find a closed form solution.

Advantages -

1. This method addresses the drawbacks of previous TPS method and produces as rigid as possible deformations which look much closer to reality than affine deformations.

Disadvantages -

1. The method is extremely slow and does not scale well for larger images. This forces the algorithm to use a limited density of triangulation which in turn leads to non-smooth deformations.

2.3.5 Moving Least Squares - Schaefer

In this work [21] the authors construct a deformation function f that is parameterized on each point v in the image. If given p as the set of control handle points and q as their deformed positions, we solve for l_v that minimizes -

$$\sum_i w_i |l_v(p_i) - q_i|^2 \tag{15}$$

$$w_i = \frac{1}{|p_i - v|^{2\alpha}} \tag{16}$$

Since, the weights are dependent on the point of evaluation v, this is called the Moving Least Squares Minimization. The authors provide closed form expressions for obtaining l_v additionally adding constraints of $l_v(x) = xM + T$ where M is a rigid body transform. The authors further provide an extension from control handles as points to line segments.

Advantages -

1. This method addresses the drawbacks of previous method by significantly improving the performance time. This is because instead of solving a large set of linear equations like [17], this method solves only a small set of many equations for each point which is highly parallelizable.

2. The method focuses on identifying rigid transforms which are more reaslistic in nature. A comparison of the output of the 3 previously discussed methods is shown in figure 16.

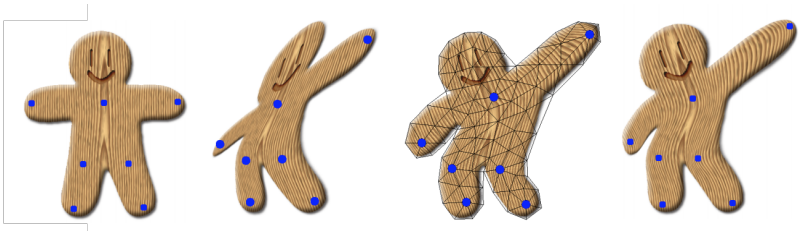


Figure 16: Original, TPS, ARAP, MLS[21]

Disadvantages -

1. Their are some corner cases where this method fails, which are addressed as an improvement in the following work.

2.3.6 Shape Aware Moving Least Squares using Interior Distance - Sharma

Extending on the previous MLS Algorithm this paper [23] proposes to use a more suitable distance metric to use in the weight formulation. The proposed metric is based on interior distances using baricentric coordinates.[20]. The idea is to define a volumetric mapping, for embedding the boundary mesh to a higher dimension, that preserves the boundary distances of a mesh while at the same time mapping the interior of the mesh to the same dimension using barycentric coordinates. This is depicted in figure 17.

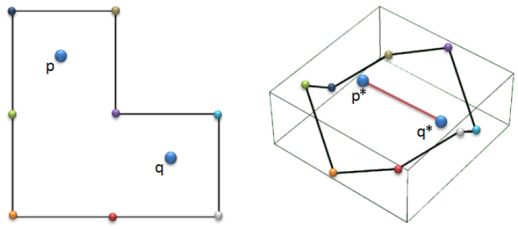


Figure 17: Mapping the boundary to an higher dimension. In the higher dimension the euclidean distance between p and q defines the interior point distance. [20]

Let the original distance between mesh vertices v_i and v_j be $d(v_i, v_j)$. The distance function between two interior points p and q with barycentric coordinates $b(p)$ and $b(q)$ in this higher dimension can be depicted as -

$$\hat{d}^2(p,q) = (b(p) - b(q))^T A(b(p) - b(q)) \tag{17}$$

A is the gram matrix of dot products of the higher dimension embeddings v_i^* (corresponding to actual vertices v_i). A special property of this embedding is that $|v_i^* - v_j^*|$ approximates $d(v_i, v_j)$. The paper endorses tools from Multi-Dimensional Scaling to obtain the gram matrix A from a matrix of square distances $D_{ij} = d^2(v_i, v_j)$. In the current paper this distance metric is taken to be the shortest interior distance by computing vertex visibility graphs within the template.

The effect of this change can be visualized in figure 18. We can clearly observe that the distances appear much more coherent when using higher dimensional metric.

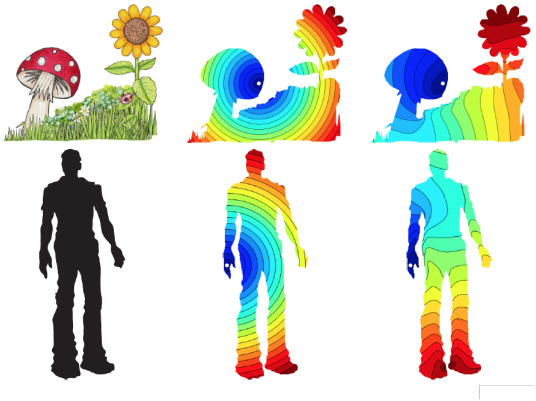


Figure 18: Original, Euclidean Distance Metric, Interior Distance Metric[20]

After this modification the new weight formula becomes -

$$w_i = \frac{1}{(\hat{d}(p_i, v))^{2\alpha}} \tag{18}$$

3 Implementation and Results

3.1 Drawing Canvas

Figure 19 shows the drawing canvas of our tool *Andy*. It has been developed in QT [12], a framework for developing cross-platform software applications and OpenGL [11], a graphics rendering engine. The drawing canvas is enriched with the following features :

1. Multiple Shapes : The canvas can draw multiple rigid shapes such as a variable sided polygon, circle and straight lines which can be used to form closed shapes.

2. Free Hand Sketching : The canvas also provides a way for freehand curve drawing which currently allows users to use the mouse and draw curves on screen

3. Blank and Filled Sketches : The canvas can be used to draw shapes with colors filled or it can be used to draw blank sketches with just shape outlines

4. Bezier Curves [2] : In our canvas, we have implemented cubic Bezier Curves which can be created using 2 lines handles or 4 control points. Figure 20 shows a Bezier curve in our drawing canvas. A line is first created using the mouse and then the mouse can again be used to manipulate the curve into the desired shape.

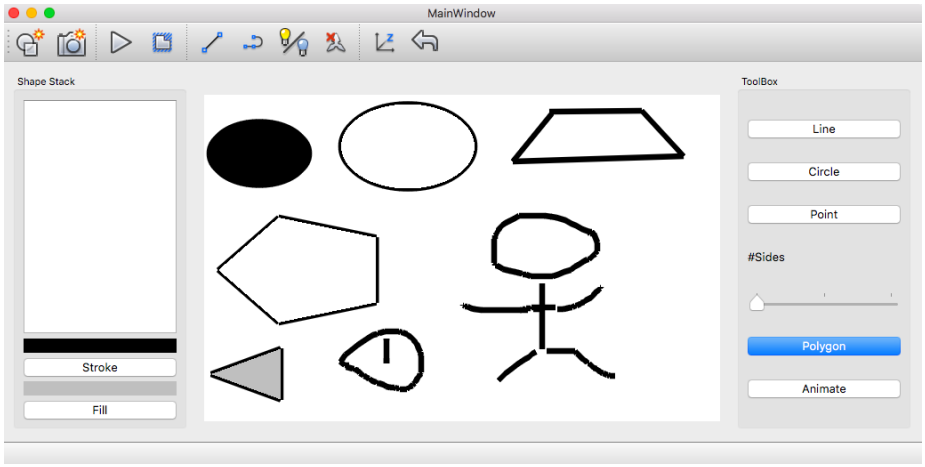


Figure 19: Drawing Canvas in *Andy*

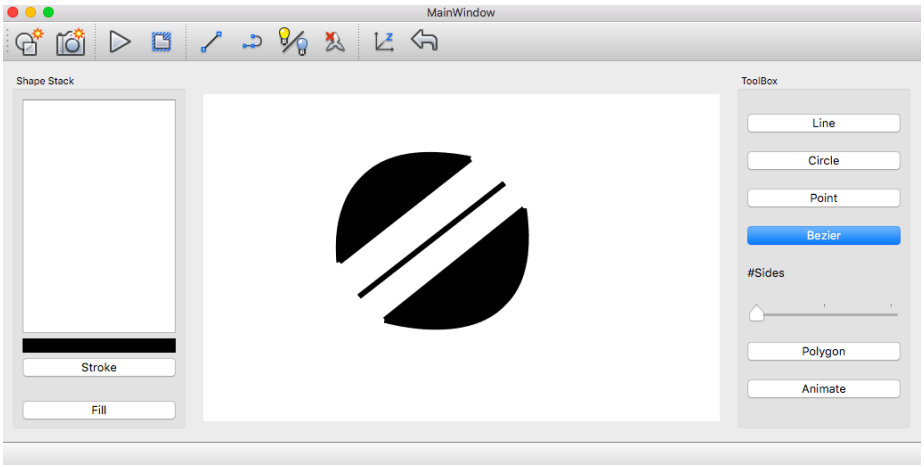


Figure 20: Bezier Curves in Andy Canvas

3.2 Skeletonization

In order to compute the skeleton, we take the following steps. We use OpenCV library functions to implement our algorithm where possible. For a solid sketch filled with some color we do :

1. Invert the background : Here we make the image background black in color or basically fill the background pixels with 0 pixel values. This is done because 0 pixel values are ignored by distance transform operator that is used in future steps (so that distance transform of pixels outside the object is not computed). To invert the background, we find pixels which are white in color as they correspond to the white background (assuming a filled colored sketch) and color them black.

2. Laplacian [9]: Laplacian derivative of the image is taken to sharpen and increase the response on boundaries of the object. This happens because, in areas where the image has a constant intensity, the laplacian response will be zero. In the vicinity of a change in intensity, however, the laplacian response will be positive on the darker side, and negative on the lighter side. As a result only the boundaries of objects are preserved.

3. Distance transform : As described in the section earlier, we compute the distance transform of the image using OpenCV distance transform functions. The result is an image wherein the intensity of each pixel denotes distance from nearest boundary. The skeleton is represented by points furthest from the boundary, so these pixels have maximum intensity.

4. Harris corner : We use OpenCVs Harris corner detection [6] to isolate peaks in the distance transform image and extract the points corresponding to the skeleton. Harris corner detection also finds points on the image

boundary, we filter these out by thresholding since pixels on the boundary will have low intensity in the distance transform. The reason harris corner detection works so well for extracting the skeleton in our case is because along the skeleton, there is an intensity change in all directions. So essentially every point on the skeleton behaves like a corner and we are able to get points corresponding to the skeleton easily.

5. Hough Transform : Once points corresponding to the skeleton have been obtained, we use OpenCV Hough Transform operator to get edges corresponding to these points. These edges correspond to the line handles needed for animating the object.

The overall flow for a triangle shape is shown in Figure 21. In addition to skeletonization of a single (color) filled object, we have to consider the following corner cases since an artist is likely to encounter them while using *Andy* :

1. Blank Sketches : So far in our skeletonization flow we have considered solid shapes only, however blank sketches such as in Figure 22(a) are a common requirement in any sketching tool. However such sketches create a problem for us because now we can't distinguish between background and foreground. This is required by Step 1 of our skeletonization flow above to make the background pixels black.

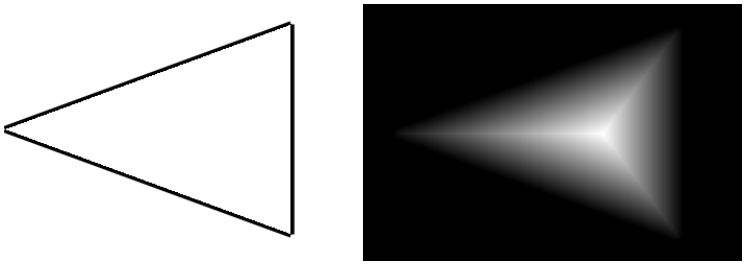
2. Closed Shapes in Single Object : This is again represented in Figure 22(a) where a figure is drawn using two closed shapes. These shapes may be overlapping or may have common boundaries. However to compute the right skeleton we cant treat them as separate objects. We need to create this compound shape as a single object and compute the skeleton.

3. Multiple Objects in Scene : As shown in Figure 22(a) as user may sketch multiple things and then may use to animate one of them. Here we need to ensure that the skeleton of each is computed individually so that they can be animated individually as well.

We deal with these cases by efficiently doing background and foreground separation before running the skeletonization flow. For this flow :

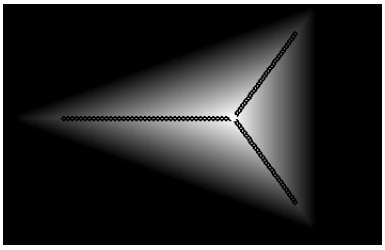
- We find all closed contours in the image, the contours correspond to the overall shapes of all objects in the image. This accounts for blank sketches, objects built using compound shapes and multiple objects in the same scene.

- Once the contours have been found, we re-color every pixel that lies inside any of the contour. This is shown in Figure 22(b). This is only done for our algorithm and the original image on the canvas is not modified. It allows us to separate background and foreground and the image can be plugged into Step 1 of the skeletonization algorithm where all background pixels will be covered black and then distance transform will be computed for foreground pixels with respect to their respective boundaries. The output is show in 22(c)

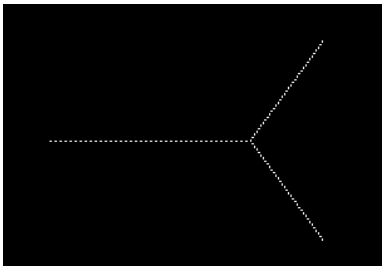


(a) Sketched Shape

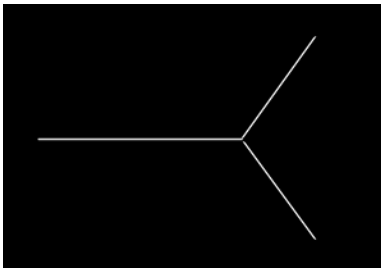
(b) Computed Distance Transform



(c) Harris Corner Peaks showing skeleton points



(d) Extracted Skeleton

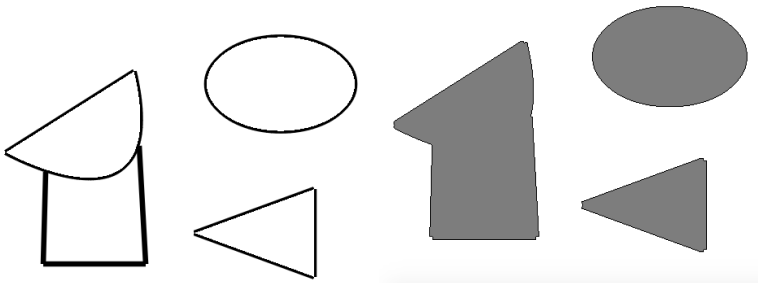


(e) Extracted skeleton edges

Figure 21: Skeletonization Flow

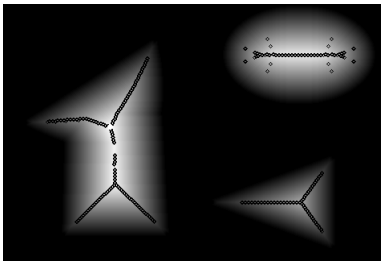
3.3 Shape Aware Deformation

Given the short time available to implement this project, we contacted authors of "Shape-aware MLS deformation for line handles" [23] and obtained their code for shape aware deformation. We moulded and integrated their interior distance computation and creation of line handles components from this code after going through it in detail (as described in section 2.3.6) with our drawing canvas and skeletonization algorithm to generate the results we showcase in the Figures 23, 24, 25, 26.



(a) Multiple Sketched Shapes

(b) Foreground identified containing interior of all shapes



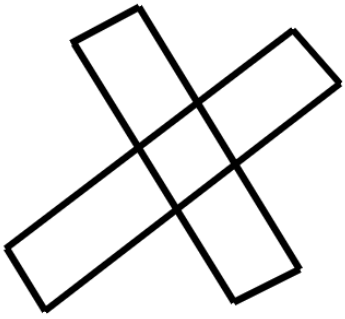
(c) Extracted skeletons from skeletonization flow

Figure 22: Foreground/Background Separation Flow

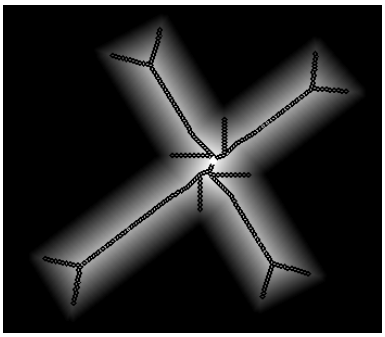
4 Conclusion

We have presented an end-to-end automated pipeline for sketching convex as well as non-convex shapes and animating them. Our work is a good starting point for anyone looking to explore development of software in this field or looking to understand the extent of research that has been done in this area so far. On a technical level we can think of the following improvements that can be made to our tool :

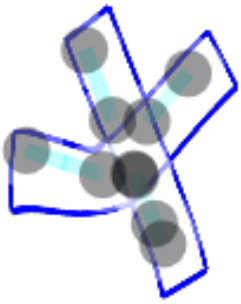
1. Extending to a touch or stylus based platform
2. Improving connectivity extraction from the obtained skeleton to work for more complex shapes. Currently our line fitting works for basic shapes but using methods like RANSAC which can filter out outliers, we can get better line fitting



(a) Sketched Shape

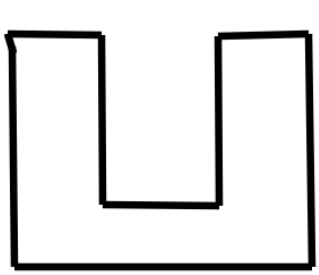


(b) Skeleton Points

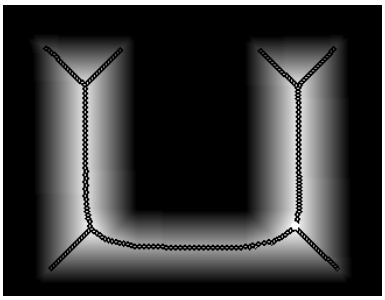


(c) Control Handles and Deformed Shape

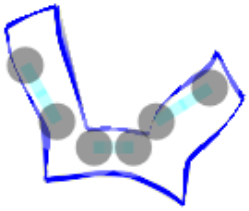
Figure 23: Shape Deformation for given shape



(a) Sketched Shape

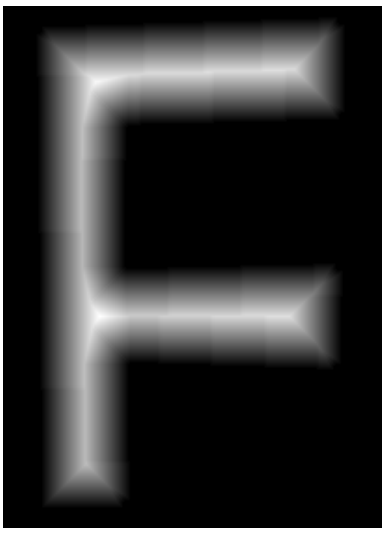


(b) Skeleton Points

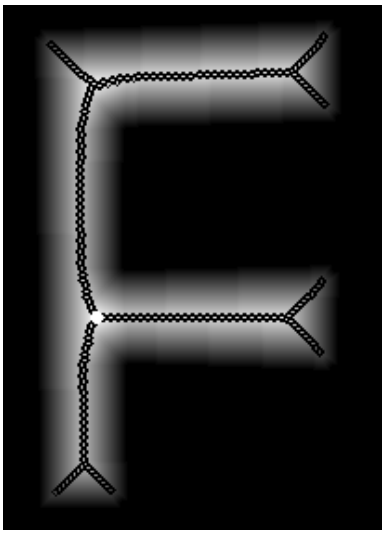


(c) Control Handles and Deformed Shape

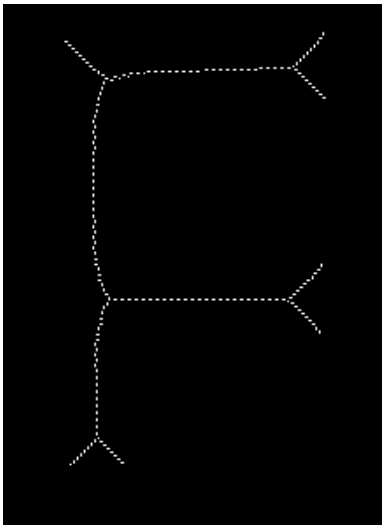
Figure 24: Shape Deformation for given shape



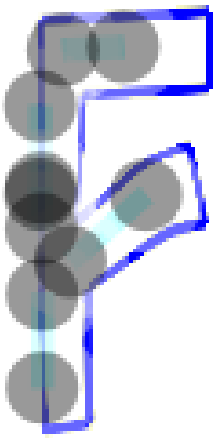
(a) Computed Distance Transform



(b) Skeleton Points

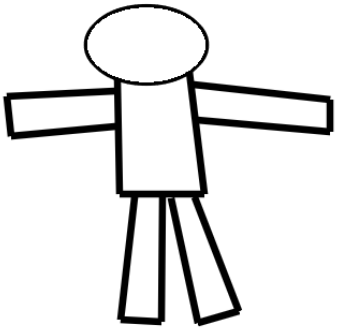


(c) Extracted skeleton



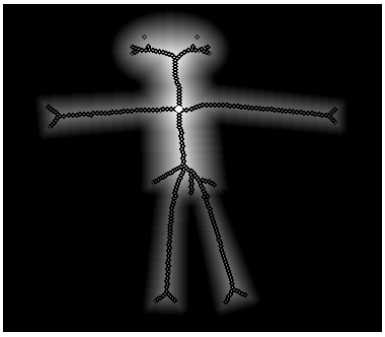
(d) Control Handles and Deformed Shape

Figure 25: Shape Deformation for given shape

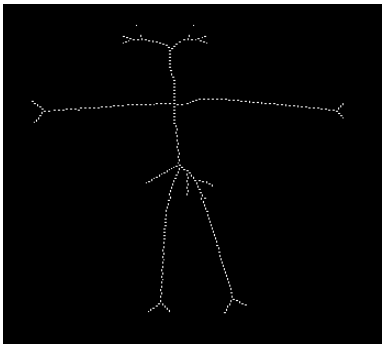


(a) Sketched Shape of Human

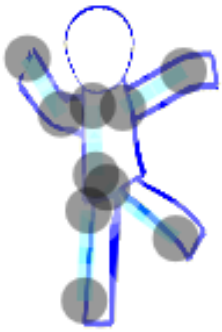
(b) Computed Distance Transform of compound shape



(c) Extracted skeleton points from harris corner



(d) Extracted skeleton



(e) Shape deformation using control handles

Figure 26: Shape Deformation for a complex compound shape

References

[1] Andy warhol. Link.

[2] Bezier curve. Link.

[3] Creating loopable animations. Link.

[4] Find contours function. Link.

[5] Harris corner detector. Link.

[6] Harris corner function. Link.

[7] Harris corner lecture. Link.

[8] Hough transform lecture. Link.

[9] Laplacian/laplacian of gaussian. Link.

[10] Nerding out with bezier curves. Link.

[11] Opengl. Link.

[12] Qt framework. Link.

[13] Topological skeletonization. Link.

[14] Traditional animation. Link.

[15] Thaddeus Beier and Shawn Neely. Feature-based image metamorphosis. In *ACM SIGGRAPH Computer Graphics*, volume 26, pages 35–42. ACM, 1992.

[16] Fred L. Bookstein. Principal warps: Thin-plate splines and the decomposition of deformations. *IEEE Transactions on pattern analysis and machine intelligence*, 11(6):567–585, 1989.

[17] Takeo Igarashi, Tomer Moscovich, and John F Hughes. As-rigid-as-possible shape manipulation. *ACM transactions on Graphics (TOG)*, 24(3):1134–1141, 2005.

[18] K. Palagyi. Skeletonization. http://www.inf.u-szeged.hu/ palagyi/skel/skel.html.

[19] A. Walker R. Fisher, S. Perkins and E. Wolfart. Skeletonization/medial axis transform. http://homepages.inf.ed.ac.uk/rbf/HIPR2/distance.htm, 2003.

[20] Raif M Rustamov, Yaron Lipman, and Thomas Funkhouser. Interior distance using barycentric coordinates. In *Computer Graphics Forum*, volume 28, pages 1279–1288. Wiley Online Library, 2009.

[21] Scott Schaefer, Travis McPhail, and Joe Warren. Image deformation using moving least squares. In *ACM transactions on graphics (TOG)*, volume 25, pages 533–540. ACM, 2006.

[22] Thomas W Sederberg and Scott R Parry. Free-form deformation of solid geometric models. *ACM SIGGRAPH computer graphics*, 20(4):151–160, 1986.

[23] Ojaswa Sharma and Ranjith Tharayil. Shape-aware MLS deformation for line handles. In *SIGGRAPH ASIA 2015 Technical Briefs on - SA '15*, pages 1–4, New York, New York, USA, 2015. ACM Press.