Name: Remaldeep Singh
UID: u1143744
Email: u1143744@utah.edu

This project has 2 parts:

- Image morphing.
- Atlas creation.

The main.m file in the project directory is used to call the morph() and the atlas() functions.

Image Morphing

This part is under Step 1 of main.m. **morph()** method is called for image transformation. morph() takes in input a parameters file. morph() calls the readParams() method.

1) readParams() method:
   The morph method starts by reading the parameters file with the function **readParams().** This method takes in parameters file and decodes the following info:
   1. The no of points, no of images.
   2. Location of each control point and creates an A matrix (from Ax=b) after reading all the control points.
   3. Name of each input file and puts it inside the array **inputFiles.**
   4. The output file names.
   5. The **tValue** i.e the step size for the morphing.
   6. The **stdParam** i.e the kernel width for the radial basis functions.
   7. The **kernelNo**. This is the index that specifies which kernel will be used for radial basis function.
      a. 01 is for gaussian.
      b. 02 is for thin plate splines.
      c. 03 is for inverse quadratic.

   After reading the parameters from the file, the A (from **Ax=b**) matrix is made by first creating the submatrix B. Then sub matrix B is replicated at diagonal places of matrix A. To create the submatrix the method **createSubMatrix** is used. This function takes in **control points**, the **stdParams** and the **kernelNo** and returns the matrix B. In order to calculate the **basis matrix** (phi matrix), createSubMatrix method calls phiMatrix method. **phiMatrix** takes in controlPoints, newPoint, stdParam and the kernelNo. Based on the kernelNo it creates the phi matrix and returns it. The createsubMatrix method also sets the control points to its respective positions.
   The readparams() method returns:
      1) Two A matrices, one for each images.
      2) Two sets of control points.
      3) The step size for morphing.

2) morph() method:
   After getting the output from readParams method, morph method starts iterating over each step size in the range [0,1]. Based on the t value new controlPoints are made between the two images with interpolation. The two **xVector** are made (one for each image) using the **pinv** of the A matrix and the **bVector** **(x = A^{-1}B).** bVector is made from the interpolated control points.

In order to find out the size of the canvas to which the points are transformed to, the method doForwardWarping is used. The method **doForwardWarping()** takes in the following inputs:

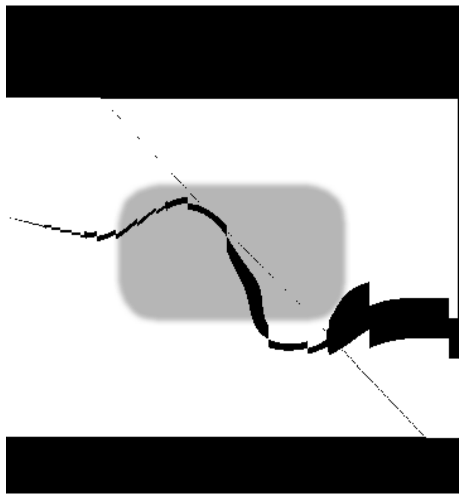
1) control points.
2) Input file.
3) X vector.
4) Std parameter. (The kernel width parameter.)
5) kernelNo.

doForwardWarping iterates over each pixel and transforms it with the xVector to the new coordinates. It makes and A matrix for each pixel coordinate and does "aMatrix * xVector" to get the final position. Then it calculates the **maximum** and the **minimum** of x and y based on **all the transformed points**. I did the transformation for each pixel rather than just the corners because it can happen that, for a pixel inside the image, due to transformation it went outside the boundary of the canvas made by just the corners. The doWarpingMethod returns the range of x and y.

Based on the x and y range, morph method creates an intermediate image of the same size. It also **offsets** the control points (**controlPointsB**) so that none of the values map to negative. Now to do inverse warping the control points are reversed and a new A matrix is made with interpolated control points. Based on that inverse vectors are calculated (xVector1, xVector2) for each image. While iterating over each pixel in the intermediate image, morph method creates the A matrix (**aMatrixP**) for each pixel coordinate and calculates the final position in each image. Based on the **number of channels** of the image corresponding bilinear interpolation is called. The intermediate image takes the return value from the **bilenearInterpolation** method and puts it in the **(i,j)** index. All these intermediate images are saved as individual steps.

There are **two** methods for **bilinear interpolation**. One is for single channel and the other one is for 3 channels. Regardless both of them have similar code. A **caveat** I found in bilinear interpolation is that if the transformation maps to a perfect integer then the term (x2-x1) or (y2-y1) or both will become 0 making the pixel value at that place undefined. Hence there is a check for that and it rounds the x and y to the nearest integer and returns the pixel value at the rounded x and y. An example of interpolation going wrong and after the fix.

Wrong

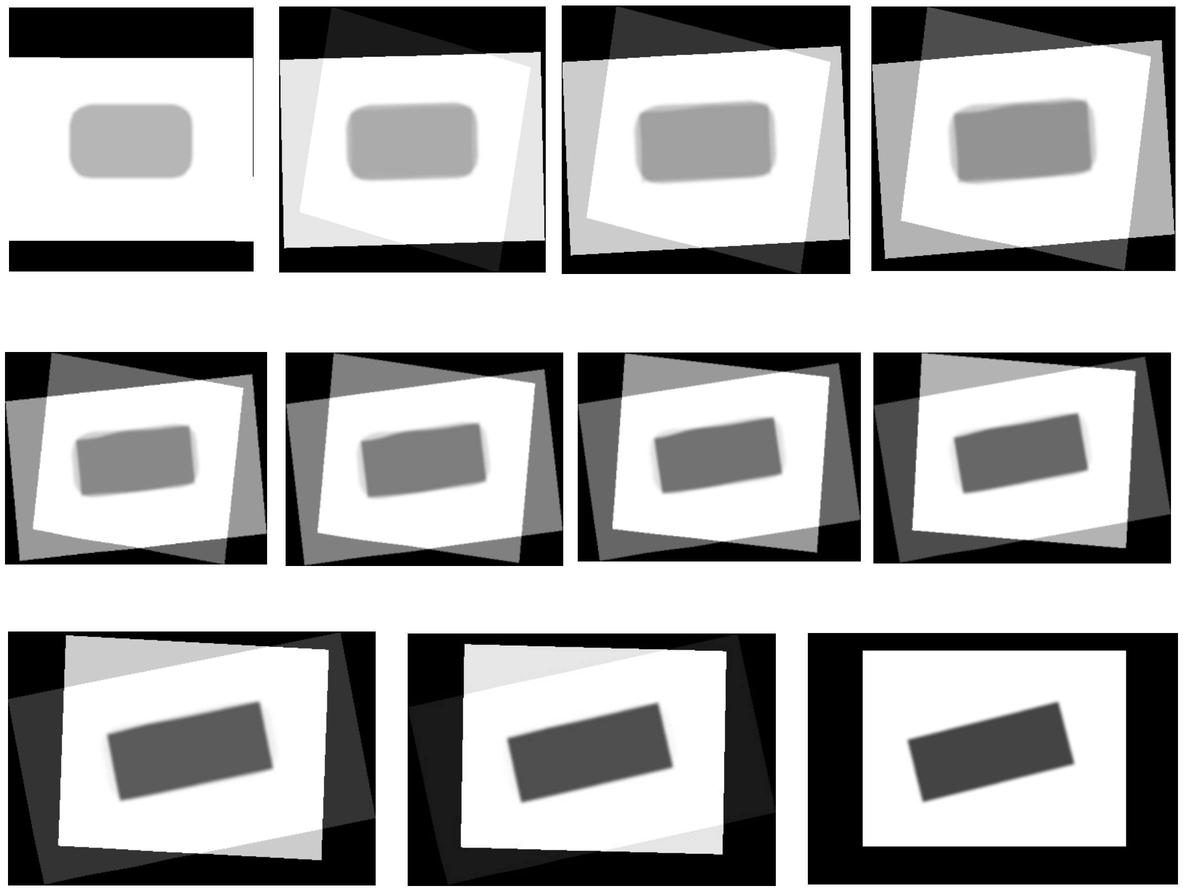


Fixed

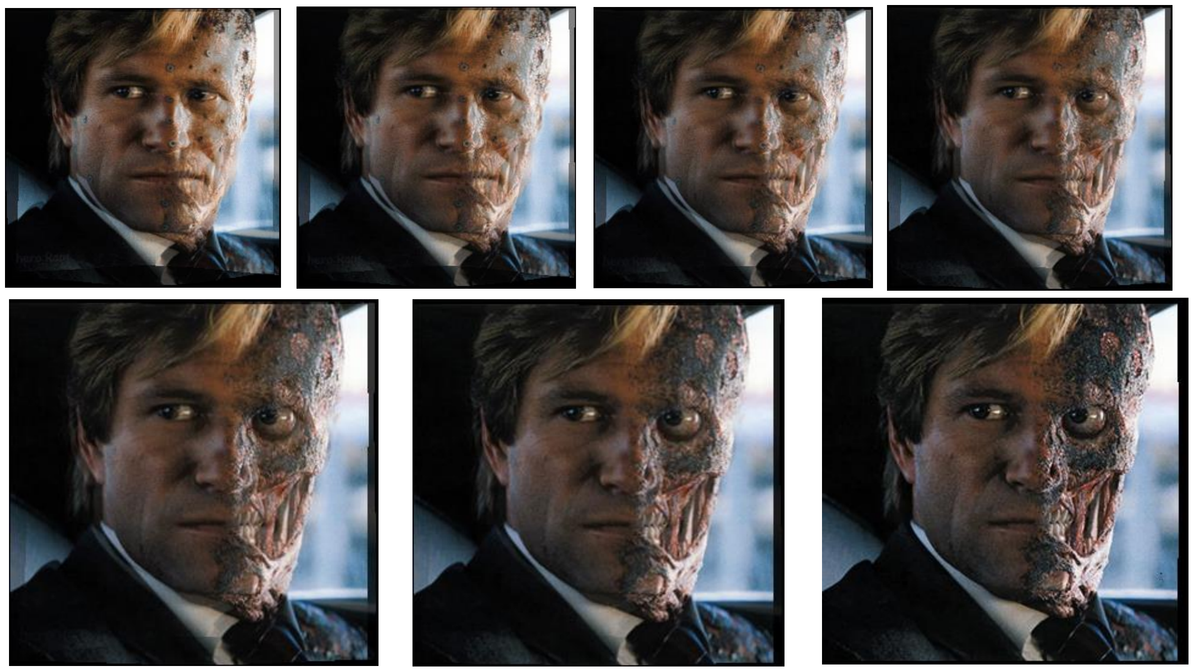
A method **ceilFix()** is used in a couple of places. It basically rounds the double away from 0. So a positive number towards infinity and negative number towards negative infinity. Used this to round the x and y range of the intermediate image canvas.

Some examples of morphing:

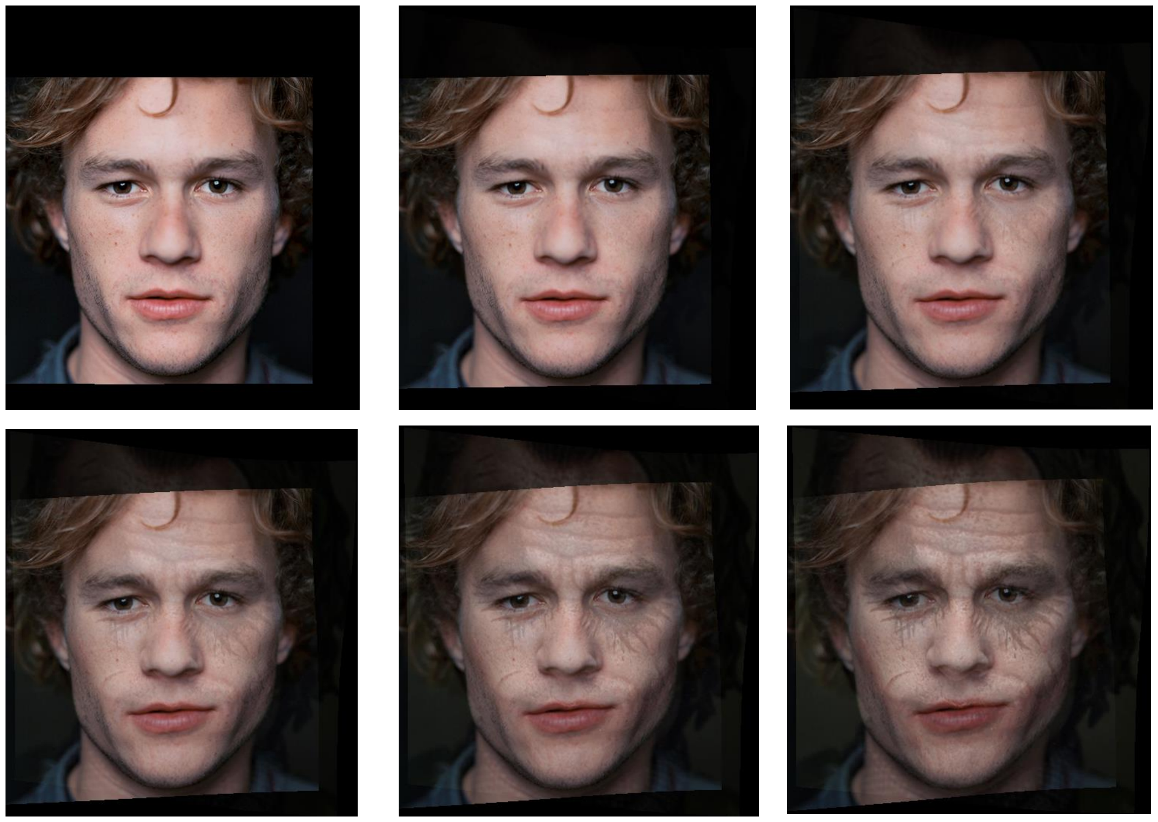
Shapes morphing:



Harley Morphing:



Joker Morphing:



Atlas Creation.

This part is under Step 2 of main.m. **atlas()** method is called for atlas creation transformation. tlaas() takes in input a parameters file. atlas() calls the readParams() method. This code is similar to morphing hence, some of the below details are the same.

1) readParams() method:

   The atlas method starts by reading the parameters file with the function **readParams().** This method takes in parameters file and decodes the following info:

   1. The no of points, no of images.
   2. Location of each control point and creates an control point matrix having all the control points for all the images.
   3. Name of each input file and puts it inside the array **inputFiles.**
   4. The output file name.
   5. The **stdParam** i.e the kernel width for the radial basis functions.
   6. The **kernelNo**. This is the index that specifies which kernel will be used for radial basis function.
      a. 01 is for gaussian.
      b. 02 is for thin plate splines.
      c. 03 is for inverse quadratic.

   After reading the parameters from the file, the **mean of the control points** is made by averaging the x and y. readParams then returns the set of all control points, mean control points, stdparam, kernelNo, input files and output file.

the A (from **Ax=b**) matrix is made by first creating the submatrix B. Then sub matrix B is replicated at diagonal places of matrix A. To create the submatrix the method **createSubMatrix** is used. This function takes in **control points**, the **stdParams** and the **kernelNo** and returns the matrix B. In order to calculate the **basis matrix** (phi matrix), createSubMatrix method calls phiMatrix method. **phiMatrix** takes in controlPoints, newPoint, stdParam and the kernelNo. Based on the kernelNo it creates the phi matrix and returns it. The createsubMatrix method also sets the control points to its respective positions.
The readparams() method returns:

4) Two A matrices, one for each images.
5) Two sets of control points.
6) The step size for morphing.

2) morph() method:
After getting the output from readParams method, the morph method iterates over each inputFile and calls **doForwardWarping()**. The method **doForwardWarping()** takes in the following inputs:
    1) control points.
    2) Input file.
    3) X vector.
    4) Std parameter. (The kernel width parameter.)
    5) kernelNo.

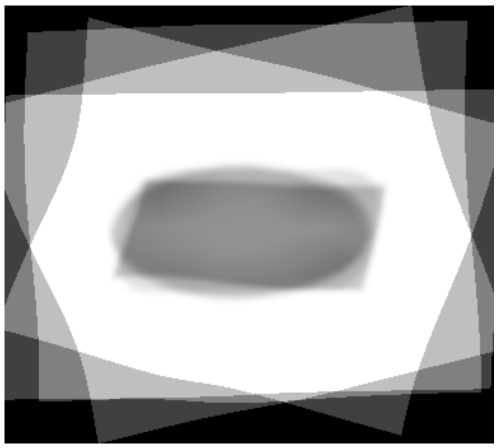
doForwardWarping iterates over each pixel and transforms it with the xVector to the new coordinates. It makes and A matrix for each pixel coordinate and does “aMatrix * xVector” to get the final position. Then it calculates the **maximum** and the **minimum** of x and y based on **all the transformed points**. I did the transformation for each pixel rather than just the corners because it can happen that, for a pixel inside the image, due to transformation it went outside the boundary of the canvas made by just the corners. The doWarpingMethod returns the range of x and y.

The morph method then calculates the minimum x and y across **all** the images. The method **ceilFix()** is used to round the double away from 0 for x and y values. Then a new **canvasImage** is created with the size of **imageSize** (which is made from new x and y range). The control points are then **offset** so that values don’t map to negative. The offset has 1 added to it since indexing in matlab starts with 1. For this canvas image and based on mean control points a **inverse matrix A** is created to map the points back to each image.

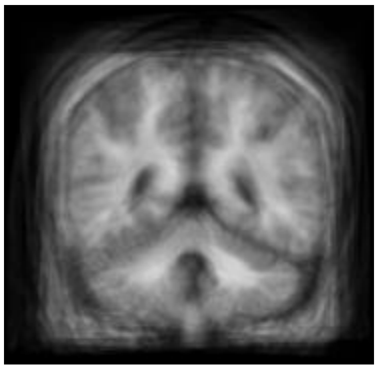
The morph method iterates over each image, **adaptive histogram equalizes** it, creates the inverse vector x and then iterates over each pixel. For each pixel, A matrix is calculated and the final position is calculated by multiplying it by inverse vector. To get the pixel value at final position bilinear interpolation is used. Then the pixel value is added to the canvas. In the last image the canvas values are divided by number of images and hence averaged.

Some examples of atlas:

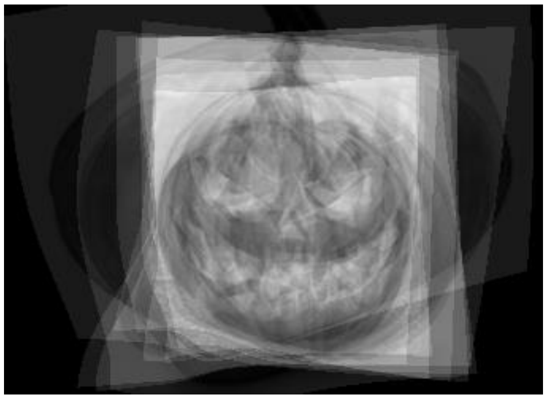
Shapes image atlas:



Brain images atlas:



Halloween Pumpkin atlas:

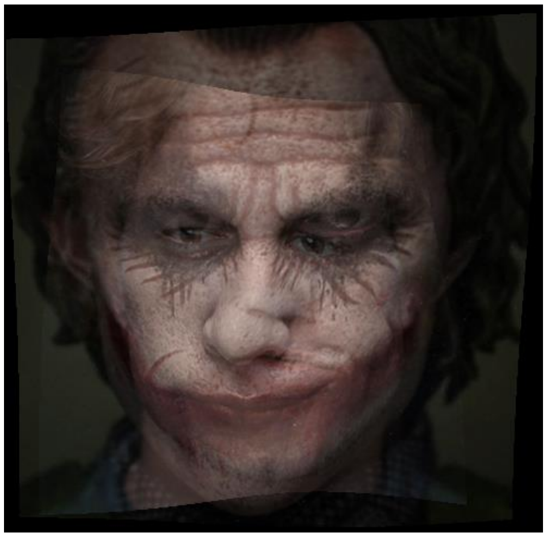
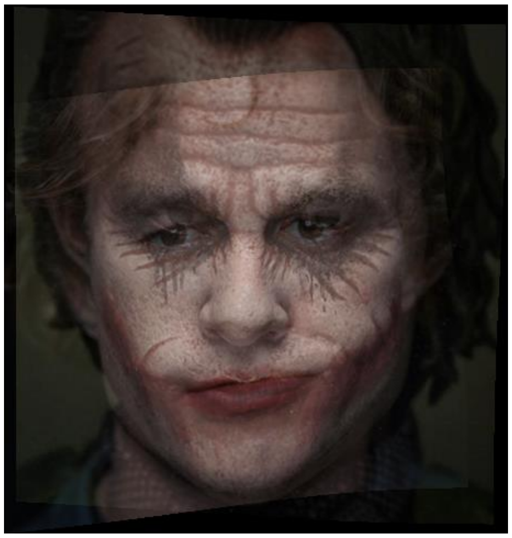


For the pumpkin images the atlas came out alright because some of the inputs had eyes of completely different shape than the rest. Nonetheless I was curious for the output.

Experiments:

Morphing:

Experimented shapes with corner point correspondences and and then changing it to some point lying inside the image.

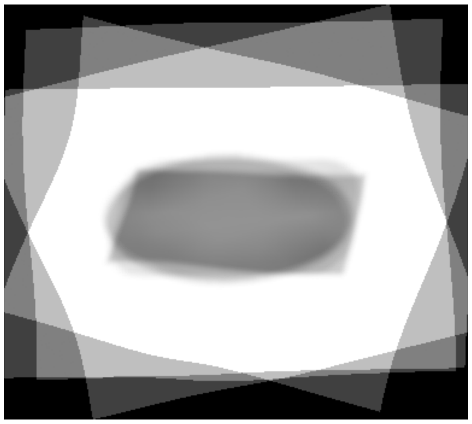



The above images had morphing done with a bit correspondence strategy as you can see the ont on the right has much better outcome than the left.

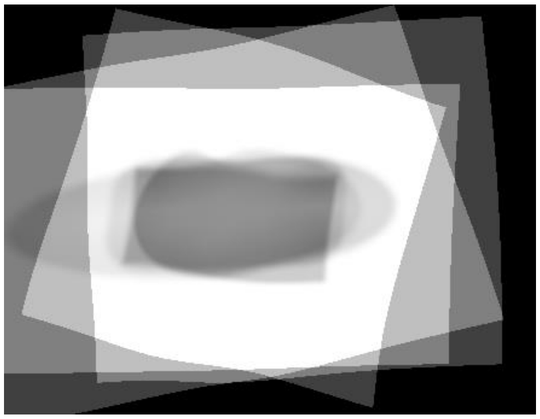
Atlas:

By changing the control points for the images, the atlas image had a drastic change. Basically, the spline instead curving at the edges went inside the rectangle and gave a wrong approximation.

Right



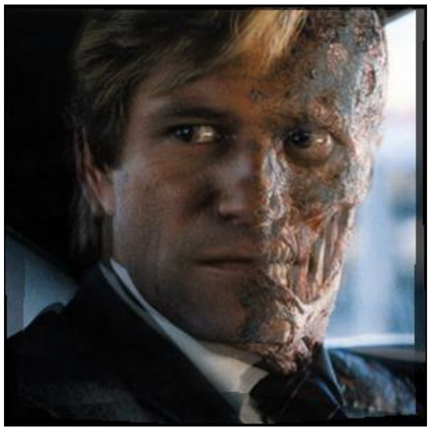
Wrong



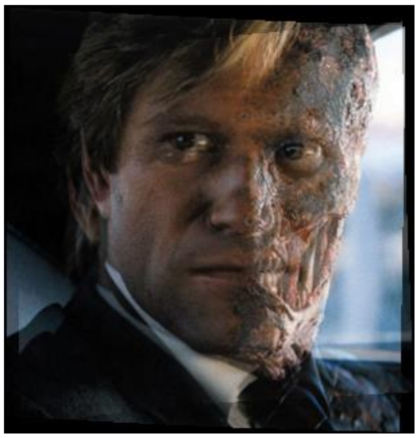
Questions:

1) With less number of control points the quality of the **morphing** decreases. The reasoning behind this is that the weights vector (x) now has less correspondences to learn the weights (x vector) from hence it is not good at generalizing the image detail. An example of Harvey morphing with less correspondences is below:

More correspondences



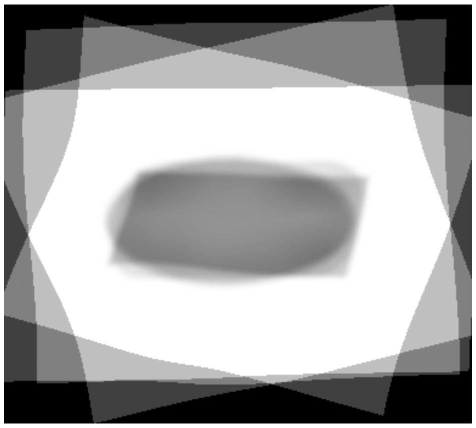
Less correspondences



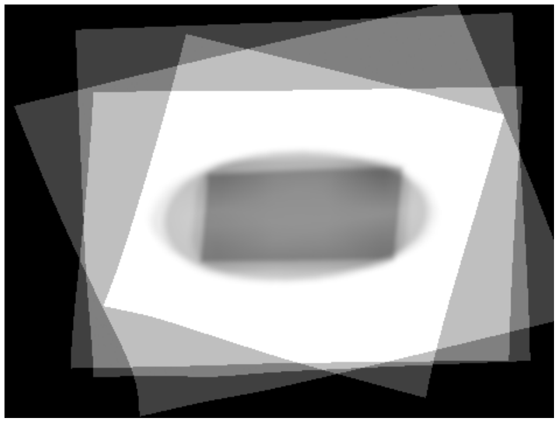
In the above images with less correspondences you can see that at step 7 the images are not overlapping better than the one with high correspondences. Hence this will not lead to better morphing.

For **atlas** as well the quality of final atlas will decrease since the x vector (the weights) will be less generalized to the anatomy of the shape. An example is given below:

More correspondences



Less correspondences

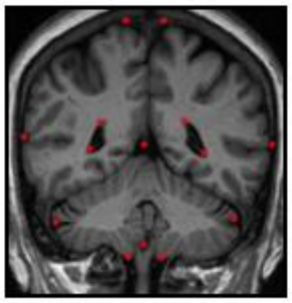


The atlas to the right doesn't correctly depict all the shapes that were present in the inputs due to less correspondences.

2) The quality of the morphing or atlas is very similar in both cases. The only caveat is that for morphing other than the thin plate splines we need an external parameter kernel width and depending upon that the output varies. An example below:

First one is processed with gaussian and the second one with thin plate splines bot h of them are pretty much the same.

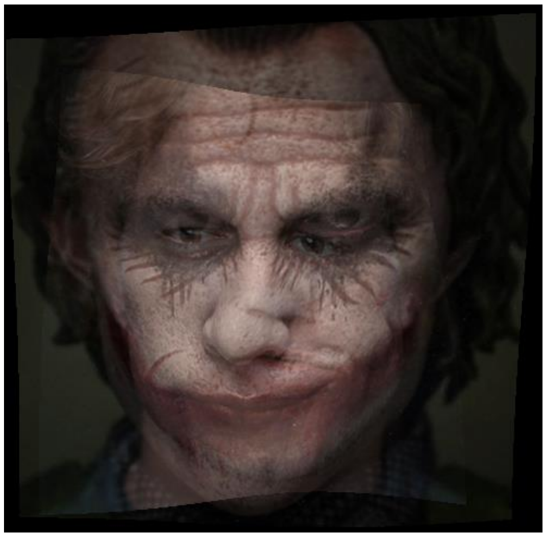
3) For the atlas example the points easily found among the brain images are marked in red:



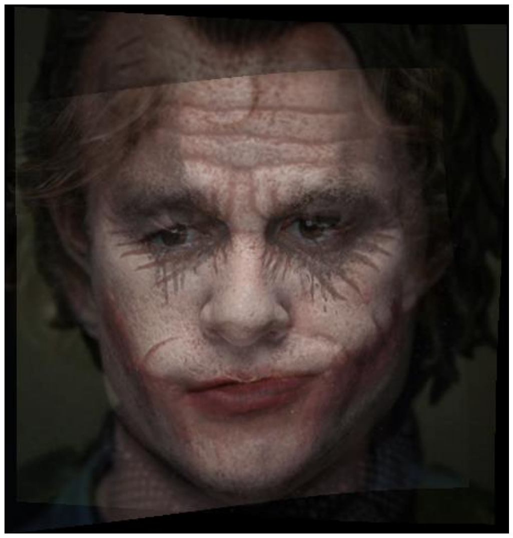
I have selected all those points that have been changing across different images of the brain MRI. These are also the points with edges, which have high tendency of changing between the images.

4) The accuracy of the control points affects the images a lot. For example if we don't keep the eyes of the face as a control point then eyes won't match in the morphing with the final image. The mapping function will not be appropriate at all. An example is given below:

Wrong control points for eye. t=0.7



Correct control points. t=0.7



The above images have only two different control points, both of them are eyes and that led to a drastic change in the morphing.

References:

https://www.mathworks.com/help/fixedpoint/ref/floor.html
https://www.mathworks.com/help/matlab/ref/fix.html
https://www.mathworks.com/matlabcentral/answers/77062-how-to-store-images-in-a-single-array-or-matrix
https://www.mathworks.com/help/images/examples/contrast-enhancement-techniques.html
http://www.mathworks.com/help/matlab/ref/eps.html;jsessionid=bbb1c62f701ad9b7b389526caf98
https://www.mathworks.com/matlabcentral/answers/213034-plotting-points-on-top-of-image-imshow-uiaxes
https://www.google.com/search?rlz=1C1JZAP_enUS758US758&biw=1227&bih=546&q=no+of+channels+in+an+image+matlab&oq=no+of+channels+in+an+image+matlab&gs_l=psy-ab.3..33i22i29i30k1l2.378.1502.0.1551.7.6.0.0.0.0.130.259.0j2.2.0....0...1.1.64.psy-ab..5.1.130....0.KAPvlaCDz20
https://www.mathworks.com/matlabcentral/answers/10578-coordinates-of-manually-selected-point