Manan Vyas
USC ID: 7483-8632-00 Email: mvyas@usc.edu

EE-569 HOMEWORK # 3

Issue Date: 10/13/13 Due Date: 11/03/13

Problem #1: SPATIAL WARPING

Motivation: In this problem, we will be learning how to warp the given image into a desired shape. Many camera applications have this in-built functionality to alter the image according to the user input, and this problem gives us an insight into one such fun application.

Task: To warp the given 'puppy.raw','cowboy.raw' and 'transformer.raw' images into diamond shape, pentagon and circle shape. The desired images of such transformations are given in the problem and we must achieve our output of our warping on similar lines.

1. A WARPING TO DIAMOND SHAPE: (FORWARD AND REVERSE)

Code Written in: C++ / Matrix Operation and Equation solving and evaluating 2D polynomial co-efficients using MATLAB.

Approach and Implementation: The most common method to warping images, is using the control points and divide the image into various triangles using these control points. The control points are 'master' and the other points within the triangular region are 'slaves. The slaves follow the master and any mapping on control point will produce a similar effect on its neighbourhood pixels.

However, on inspection, it was found that the desired images (diamond warping on peppers ; pentagon warping on girl and the circle warping on mandrill images are not completely warped by using the triangle method).

To avoid getting the fractional values of pixel co-ordinates in the output warped image, the reverse mapping was used. It was assumed that the output image pixel co-ordinates were only integral and the corresponding input image pixels location was found. If the pixel co-ordinate location turned out to be fractional, Bilinear Interpolation was used to approximate the mapping of input image pixels to warped image pixel intensity.

Bilinear interpolation uses the 4 nearest integral neighbours pixel intensities to interpolate the intensity of the pixel that has fractional co-ordinates.

$$F(p',q') = (1-a)*(1-b)*F(p,q) + (1-a)*b*F(p,q+1) + a*(1-b)*F(p+1,q) + a*b*F(p+1,q+1).$$

On inspection the image1(b) given in homework; it is found that the pixels that are near the top and bottom are warped in the direction indicated by the arrows in the image. And the pixels near the middle stay on the same horizontal line as that of the original image.

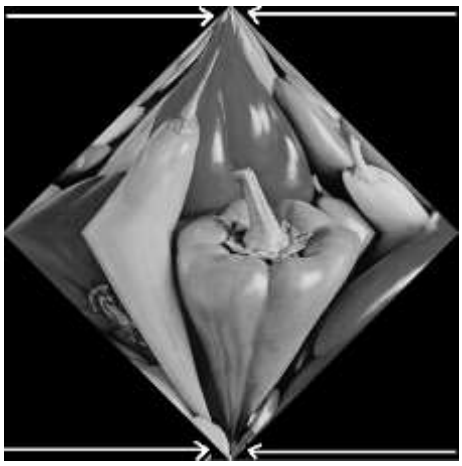

Figure 1.1: Given peppers warped to diamond image [1].

Hence, it was seen that the warping must be done as shown below.

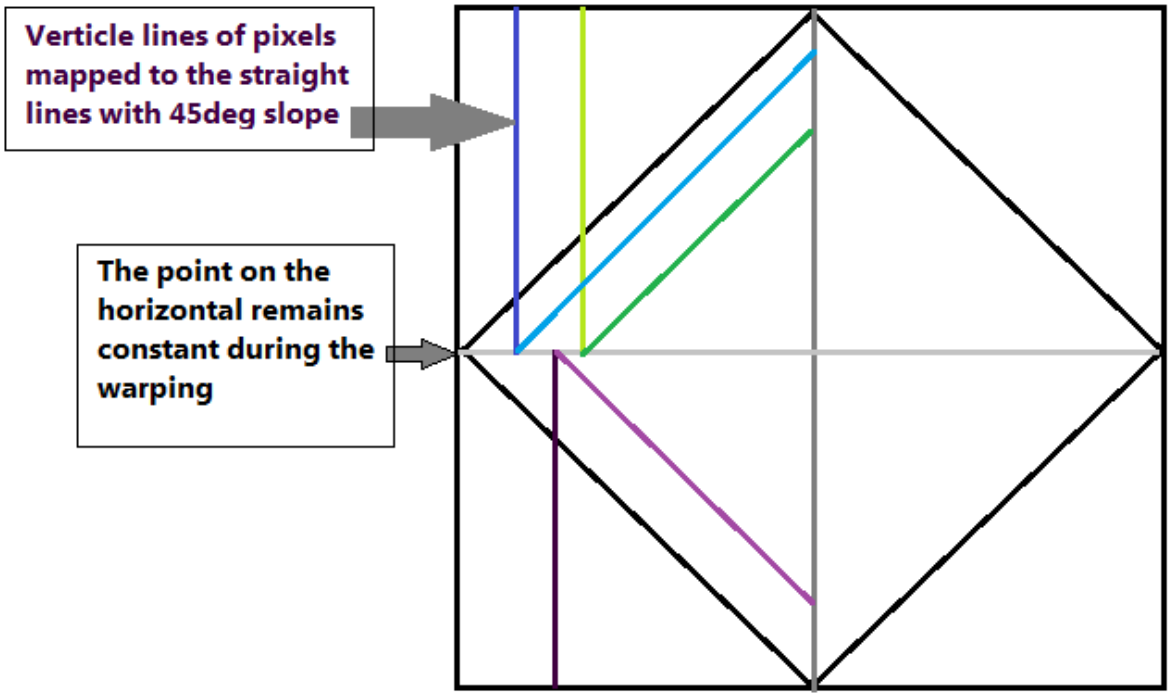

Figure 1.2: Graphical representation of diamond warping approach.

Hence, for each of the four quadrant the equation for straight line was found that warped the vertical into the desired slope. This process was done for the vertical lines of the input image in all the four quadrants of the input image. Thus mapping equations were found for the 4 quadrants of the input image.

For reverse mapping, the same procedure in reverse order was implemented, hence we get four new sets of equations for the four quadrants that maps the straight lines with 45 deg slope into vertical straight lines in all the four quadrants.

To avoid getting the fractional values of pixel co-ordinates in the output warped image, the reverse mapping was used. It was assumed that the output image pixel co-

ordinates were only integral and the corresponding input image pixels location was found. If the pixel co-ordinate location turned out to be fractional, Bilinear Interpolation was used to approximate the mapping of input image pixels to warped image pixel intensity.

Bilinear interpolation uses the 4 nearest integral neighbours pixel intensities to interpolate the intensity of the pixel that has fractional co-ordinates.

$$F(p',q') = (1-a)*(1-b)*F(p,q) + (1-a)*b*F(p,q+1) + a*(1-b)*F(p+1,q) + a*b*F(p+1,q+1).$$

All the straight lines and slope calculation was done using the Cartesian co-ordinates. Hence to convert the image to Cartesian co-ordinates the formula used was;

$$uq = q - 0.5 \; ; \; and \; ; \; vp = 500 + 0.5 - p$$

Results and Discussion:
The following warping results are obtained on 'puppy.raw' image.

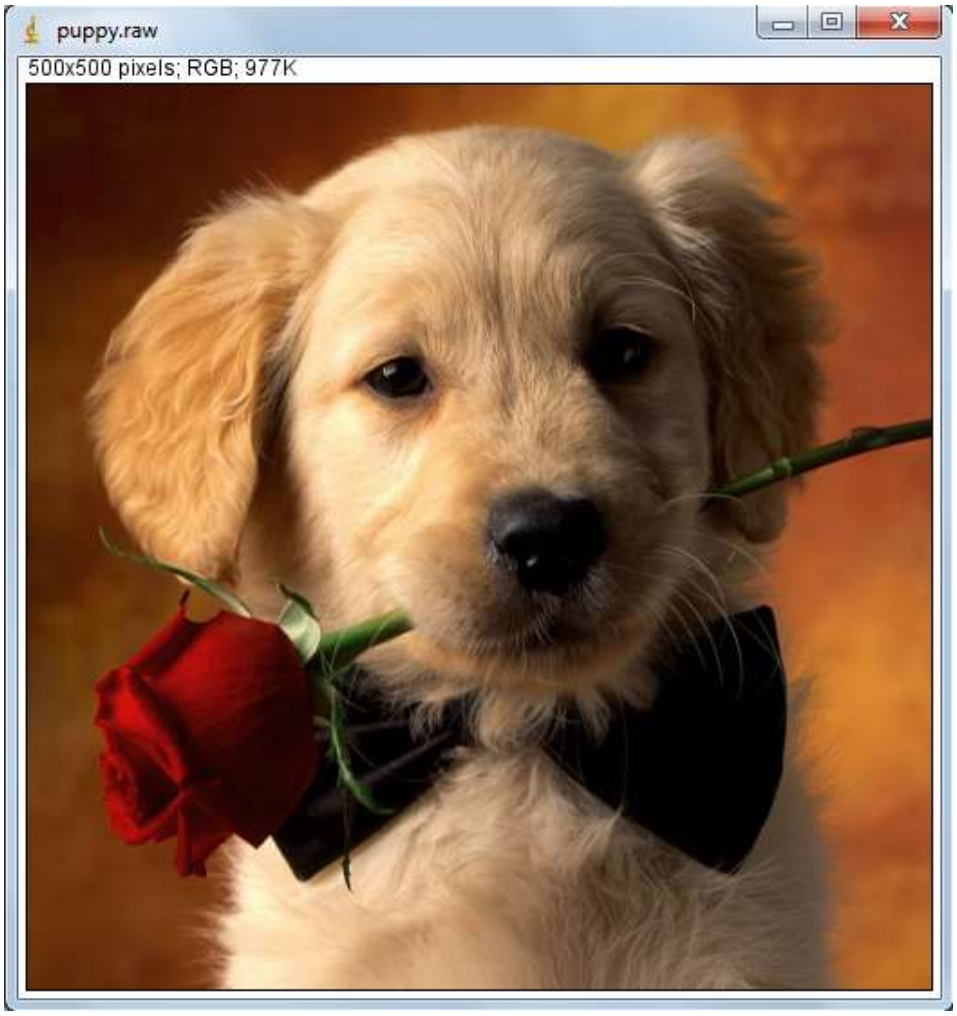

Figure 1.3: Given 'puppy.raw' image.

Figure 1.4: Diamond warped 'puppy.raw' image with bilinear interpolation.

Thus, it can easily be seen that the center of the original image maps to the center of the warped image. The boundary pixels are mapped to the boundary of the diamond shape and the mapping is clearly unique (one-one). Thus, all the design requirements are fulfilled in this warping.

The following is the output, where the original image is recovered from the warped puppy image. (reverse mapping output).

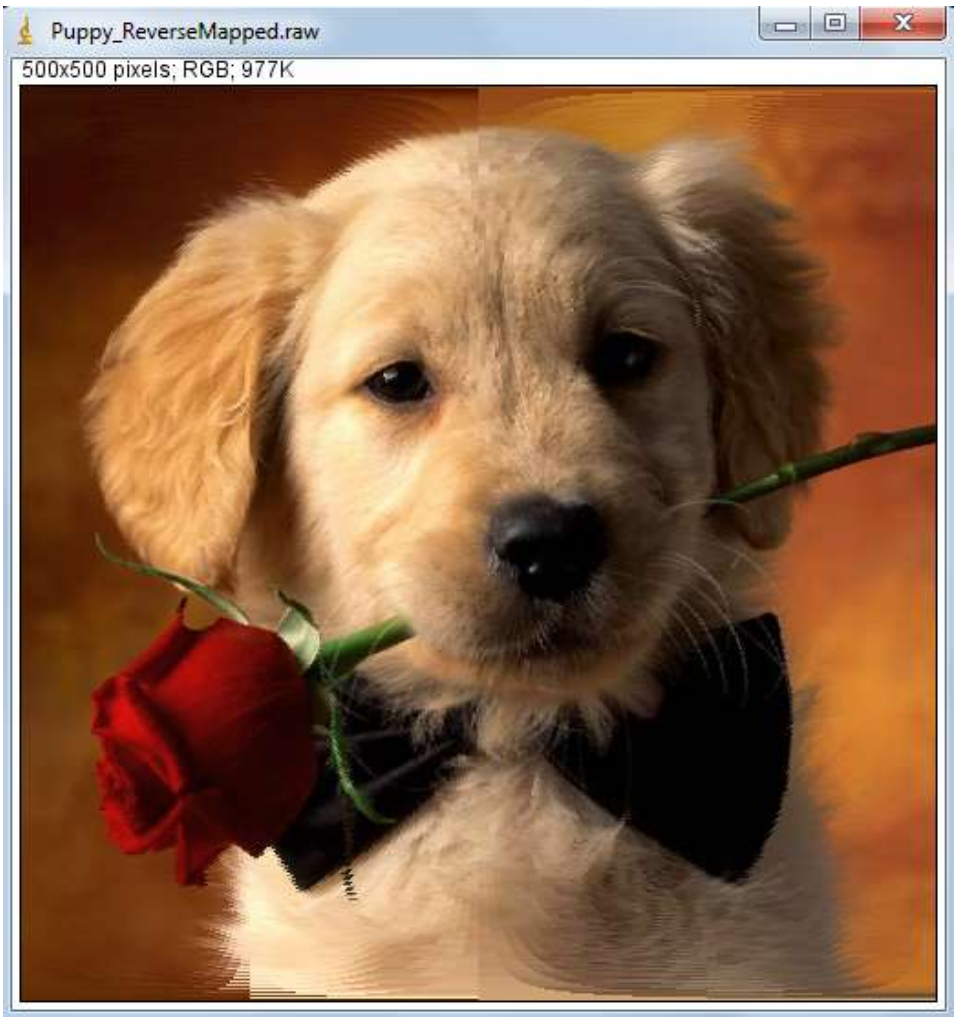

Figure 1.5: Reverse Mapping of Diamond warped 'puppy.raw' image with bilinear interpolation.

We can also notice, that as we move away from the image center, more and more pixels are approximated to pixel co-ordinates that are very close to each other and hence, their intensities combine to produce the warped pixel intensity. This happens maximum near at the boundaries of the diamond and it is minimum at the center of the warped.

Source of Artifacts in diamond warping:

Due to this, the artifacts are seen in the recovered image especially as we move away from the image center near outer parts and the boundaries of the image. More and more pixel intensities are combined to very nearby values after bilinear interpolation to form one representative intensity of all the pixels that were mapped to that co-ordinates. Thus while recovering, the original pixel intensity was lost and instead, we got the combined intensity that was mapped to the recovered image. This is one of the major source of artifacts/distortion in the recovered image.

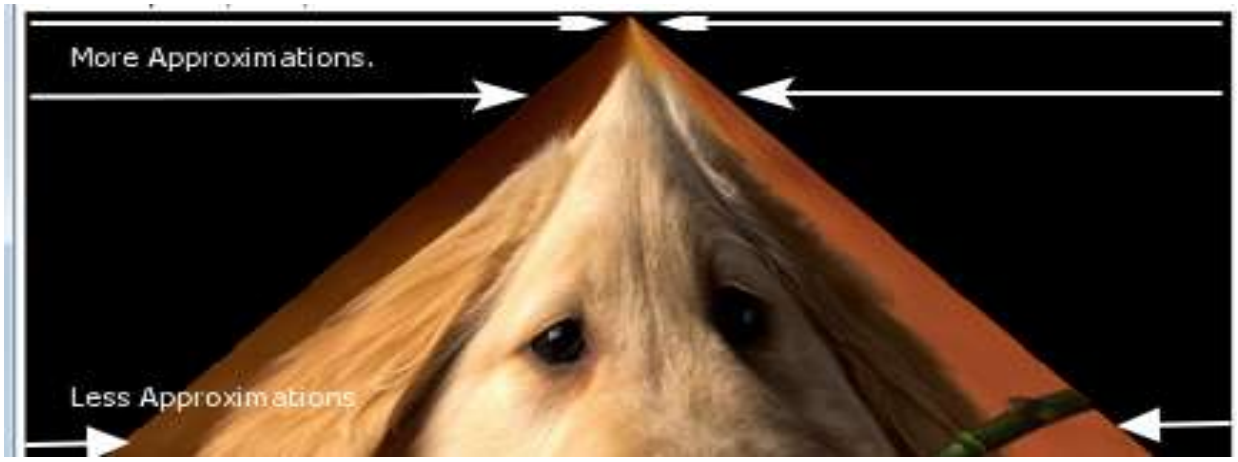

Figure 1.6 Source of artifacts in Diamond warped 'puppy.raw' image with bilinear interpolation

Comparision between original and recovered image : Artifacts seen near the top and bottom boundaries of the recovered image.

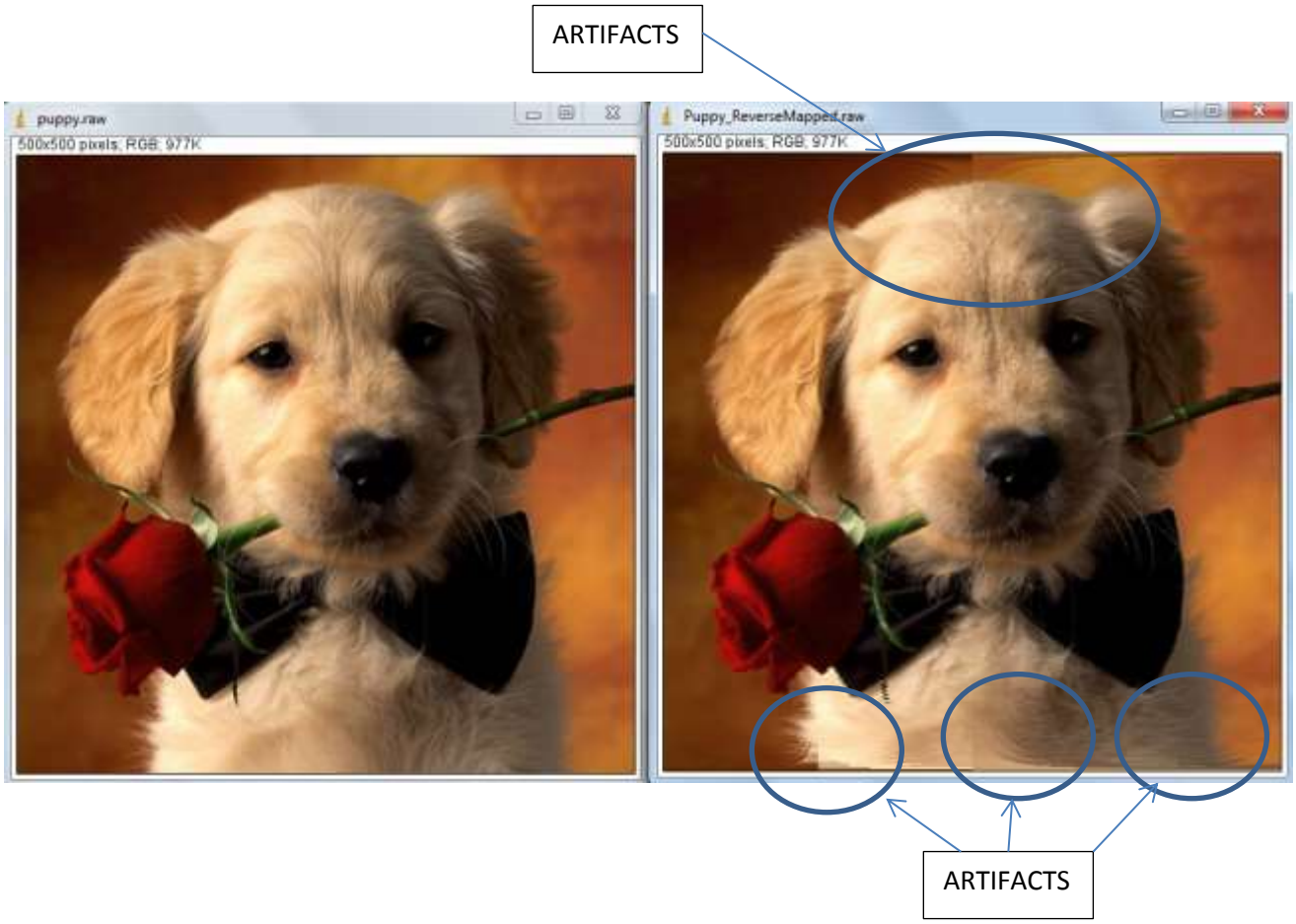

Figure 1.7 Original puppy image (Left).Puppy Recovered image (right)

Manan Vyas
USC ID: 7483-8632-00 Email: mvyas@usc.edu

1. B WARPING TO PENTAGON SHAPE: (FORWARD AND REVERSE)

Code Written in: C++ / Matrix Operation and Equation solving and evaluating 2D polynomial co-efficients using MATLAB.

Approach and Implementation:

Generating the regular pentagon: To generate the pentagon, one of the vertexes of the pentagon was fixed to the point (250, 0) and then using the basics of geometry, the co-ordinates of the other 4 vertices was found for the pentagon. It was also assumed that the center of the image is the center of the pentagon for the sake of simplicity in calculations.

The girls image warped to pentagon was inspected, and it was found the lower part of the image was warped through triangulation and the upper half of the warped using an approach similar to the diamond warping. However, the difference being that the co-ordinates of the horizontal points no longer remained constant but followed the line joining the center of the pentagon and the vertex. Thus instead of just determine the equation of the line (its slope), it was also required to trace the straight line joining the vertex and the center of the pentagon.

To avoid getting the fractional values of pixel co-ordinates in the output warped image, the reverse mapping was used. It was assumed that the output image pixel co-ordinates were only integral and the corresponding input image pixels location was found. If the pixel co-ordinate location turned out to be fractional, Bilinear Interpolation was used to approximate the mapping of input image pixels to warped image pixel intensity.

Bilinear interpolation uses the 4 nearest integral neighbours pixel intensities to interpolate the intensity of the pixel that has fractional co-ordinates.

$$F(p',q') = (1-a)*(1-b)*F(p,q) + (1-a)*b*F(p,q+1) + a*(1-b)*F(p+1,q) + a*b*F(p+1,q+1).$$

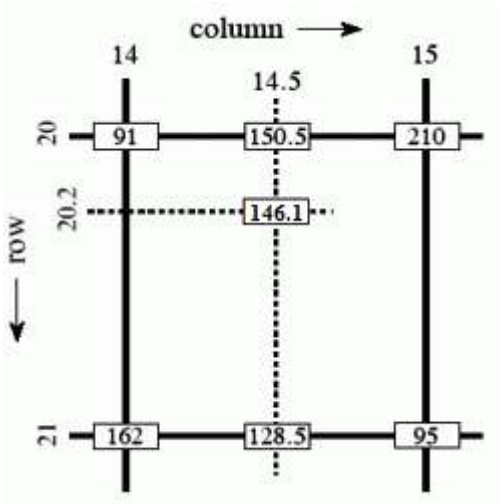

Figure 1.8 Bilinear Interpolation[2]:

Manan Vyas
USC ID: 7483-8632-00 Email: mvyas@usc.edu

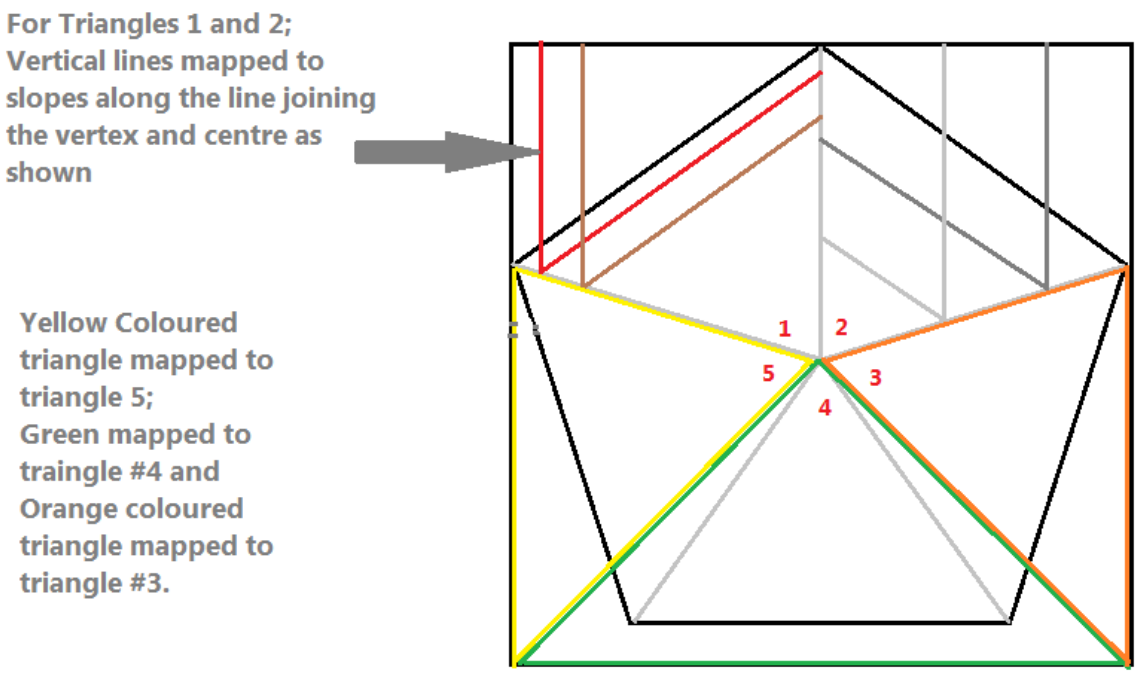

Figure 1.9: Graphical representation of pentagon warping approach

For triangle method, barycentric co-ordinate system [3] were used to determine whether the pixel is within the triangle or outside it. Pixels with +ve eigen values are said to be within the triangle. If inside the then warp using co-efficients 'c0,c1,c2' and 'd1,d2 and d0' coefficients of the generalized linear transform matrix.

$$\begin{bmatrix} uq \\ vp \\ 1 \end{bmatrix} = \begin{bmatrix} C1 & C2 & C0 \\ D1 & D2 & D0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Xk \\ Yj \\ 1 \end{bmatrix}$$

All the straight lines and slope calculation was done using the Cartesian co-ordinates. Hence to convert the image to Cartesian co-ordinates the formula used was;

$$uq = q - 0.5 \; ; \; and \; ; \; vp = 500 + 0.5 - p$$

Thus we have;

$$q = 0.5 + (C1 * Xk) + (C2 * Yj) + C0$$
$$p = 500 + 0.5 + (D1 * Xk) + (D2 * Yj) + D0$$

The same operation in reverse was used to recover the original image from the warped image. The slopes were mapped to vertical lines and the triangles 3,4,5 were mapped to those indicated in orange, green and yellow colours in the above diagram to get the original image from warped image.

Results and Discussion:

The following are results obtained for pentagon warping;

Figure 1.10: Original cowboy.raw image

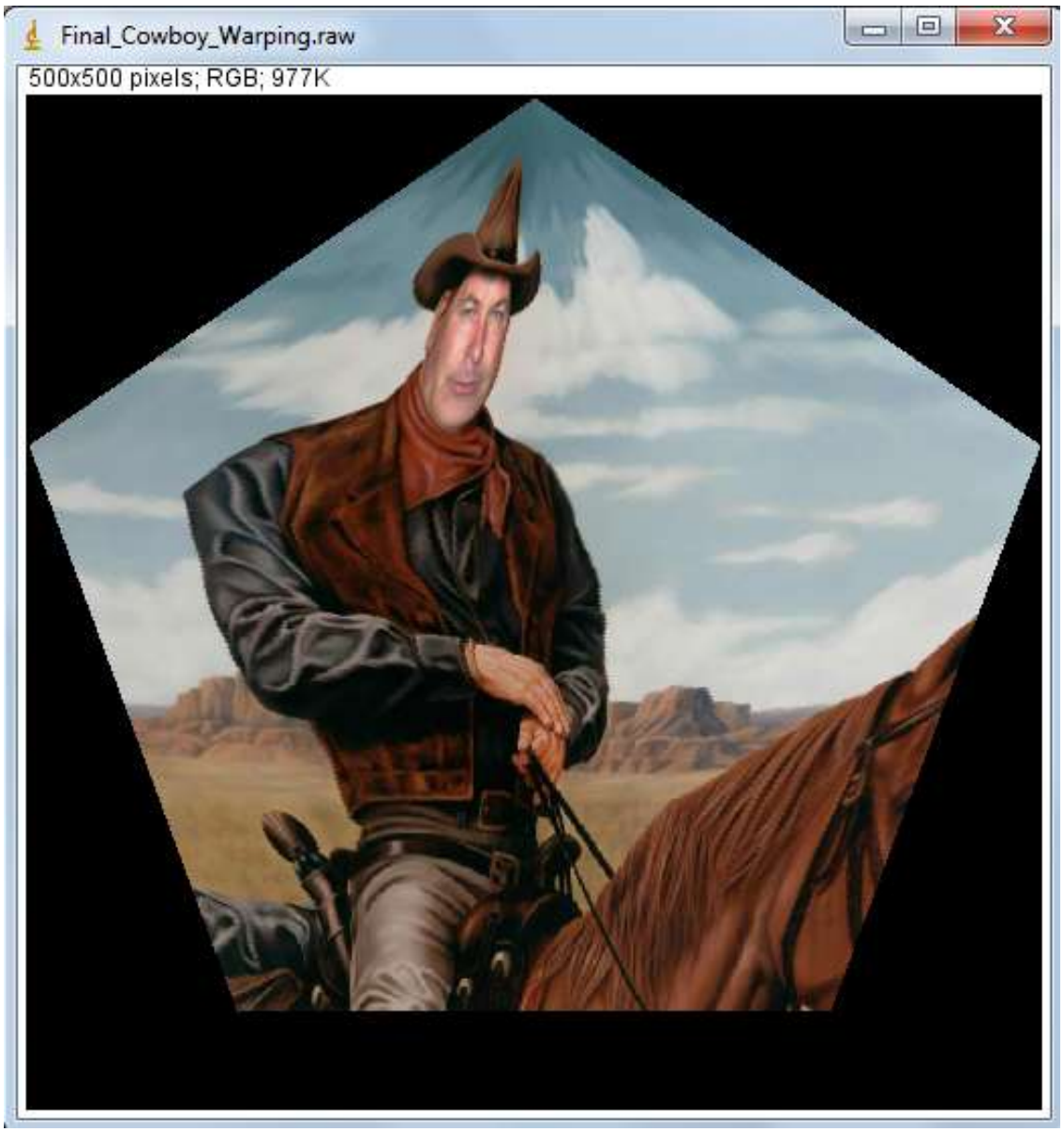

Figure 1.11: Pentagon Warped cowboy.raw image

Thus, it can easily be seen that the center of the original image maps to the center of the warped image. The boundary pixels are mapped to the boundary of the diamond shape and the mapping is clearly unique (one-one). Thus, all the design requirements are fulfilled in this warping.

The following is the output, where the original image is recovered from the warped puppy image. (reverse mapping output).

Figure 1.12: Recovered (reverse mapped) cowboy.raw image.

It is noticed, that as we move away from the image center toward the vertices of the pentagon, more and more pixels are approximated to pixel co-ordinates that are very close to each other.

Hence, their intensities combine to produce the warped pixel intensity. This happens maximum near at the vertices and the sides of the pentagon and it is minimum at the center of the warped image.

Thus we see that the corners and the borders have lost intensity information in the reconstructed image.

It is also observed that the reconstructed image gets slightly blurred with respect to the original image and it lacks the crisp edges that were present in the original image.

Manan Vyas
USC ID: 7483-8632-00
Email: mvyas@usc.edu

Comparision with original image:

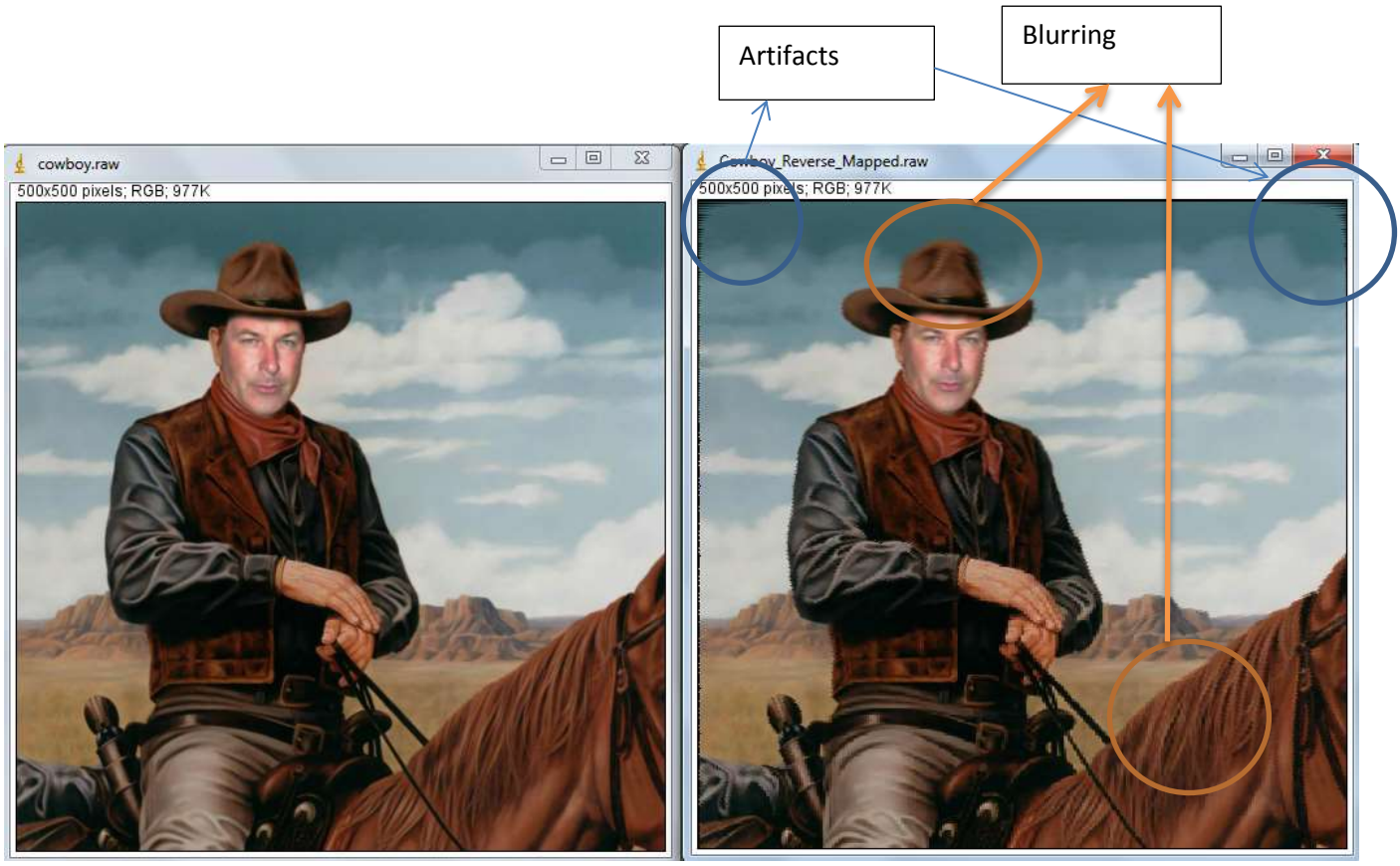

Figure 1.13 Original Cowboy image (Left).Cowboy Recovered image (right)

Manan Vyas
USC ID: 7483-8632-00 Email: mvyas@usc.edu

I. C WARPING TO CIRCLE SHAPE: (FORWARD AND REVERSE)

Code Written in: C++ / Matrix Operation and Equation solving and evaluating 2D polynomial co-efficients using MATLAB.

Approach and Implementation: To warp into circular shape, we take into account the concept of similiarity of triangles in order to the map the pixels on the edges/boundary of the image to the circumference of the circle. And consequently all the points in a given quadrant to the quarter circle. Firstly, the image co-ordinates we changed to polar form with r = 250 for the circle.

$$x = rcos\theta \; and \; y = rsin\theta$$

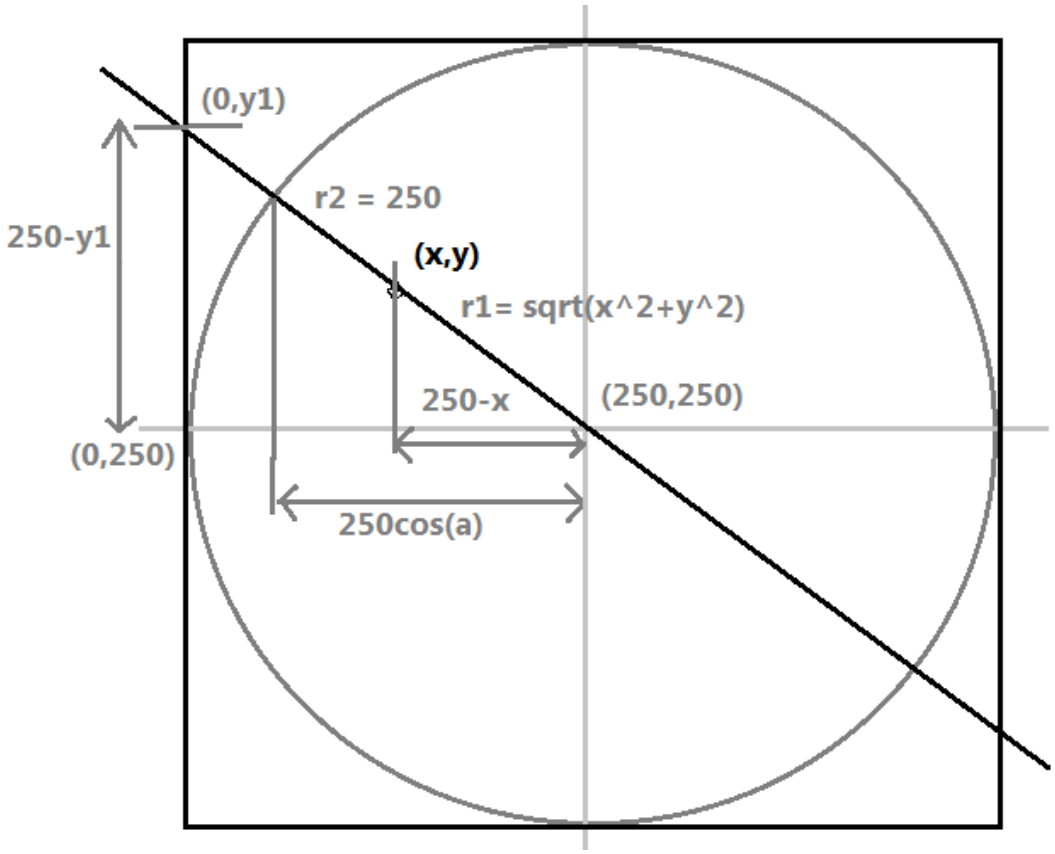

Figure 1.14: Graphical representation of circle warping approach

And now using the concept of similiarity of triangles to find out the ratio in which the image pixels are warped to circular shape. This ratio will be constant for only a particular value of θ. Once, the angle changes, this ration needs to be recalculated constantly with every change in θ.

Thus we get the ratios from the similiarity of triangles that ;

$$\frac{\sqrt{((250-x)^2)+((250-y)^2)}}{250-x} = \frac{250}{250cosa} = \frac{\sqrt{((250)^2)+((250-y1)^2)}}{250}$$

Since we are performing the reverse mapping (from warped co-ordinates (integers) we are finding the image co-ordinates) we know the values of (x,y) and thus we can find the value of y1 for a particular value of θ. To get good results, I divided the circle into 8 parts instead the 4 quadrants.

Thus we get the following equation for θ [0-45deg range].

$$p = 250 - (tan\theta * \sqrt{x^2 + y^2})$$
$$q = 250 - (\sqrt{x^2 + y^2})$$

Similiary the equations for all the ranges for θ [0-360deg] were found for warping.

To avoid getting the fractional values of pixel co-ordinates in the output warped image, the reverse mapping was used. It was assumed that the output image pixel co-ordinates were only integral and the corresponding input image pixels location was found.

If the pixel co-ordinate location turned out to be fractional, Bilinear Interpolation was used to approximate the mapping of input image pixels to warped image pixel intensity.

Bilinear interpolation uses the 4 nearest integral neighbours pixel intensities to interpolate the intensity of the pixel that has fractional co-ordinates.

F(p',q') = (1-a)(1-b)*F(p,q) + (1-a)*b*F(p,q+1) + a*(1-b)*F(p+1,q) + a*b*F(p+1,q+1).*

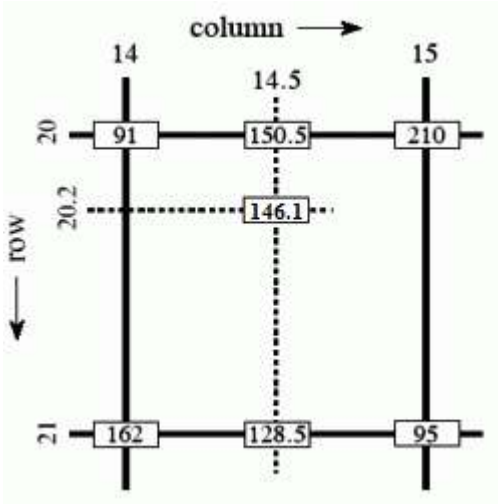

Figure 1.15 Bilinear Interpolation[2]:

For recovering the original image from warped image: To get the original image , I used the same similiarity ratios, the difference being now we know the values of x1,y1 and we need to find the co-ordinate (x,y).

Results and Discussions:

The following results were obtained;

Figure 1.16: Original Transformer Image.

Figure 1.17: Circular Warped transformer image.

Figure 1.17: Reverse Mapped- Recovered transformer image

- Thus it is observed that the reconstructed image has lost pixel information and edge crispeness by blurring the edges of the transformer image.

- This is due to the non linear scaling effect that is produced by our warping transformation along every angle.

- The blurring effect is more pronounced along lines of 45 deg angle as observed in the reconstructed image.

- The boundary of the reconstructed image has some black pixel which were not supposed to be there. This may have happened because of the warping technique applies and some of the blurred data not present on the edges of transformer may have wrapped around due to unsigned char variable used to generate the output image.

Comparision with original image :

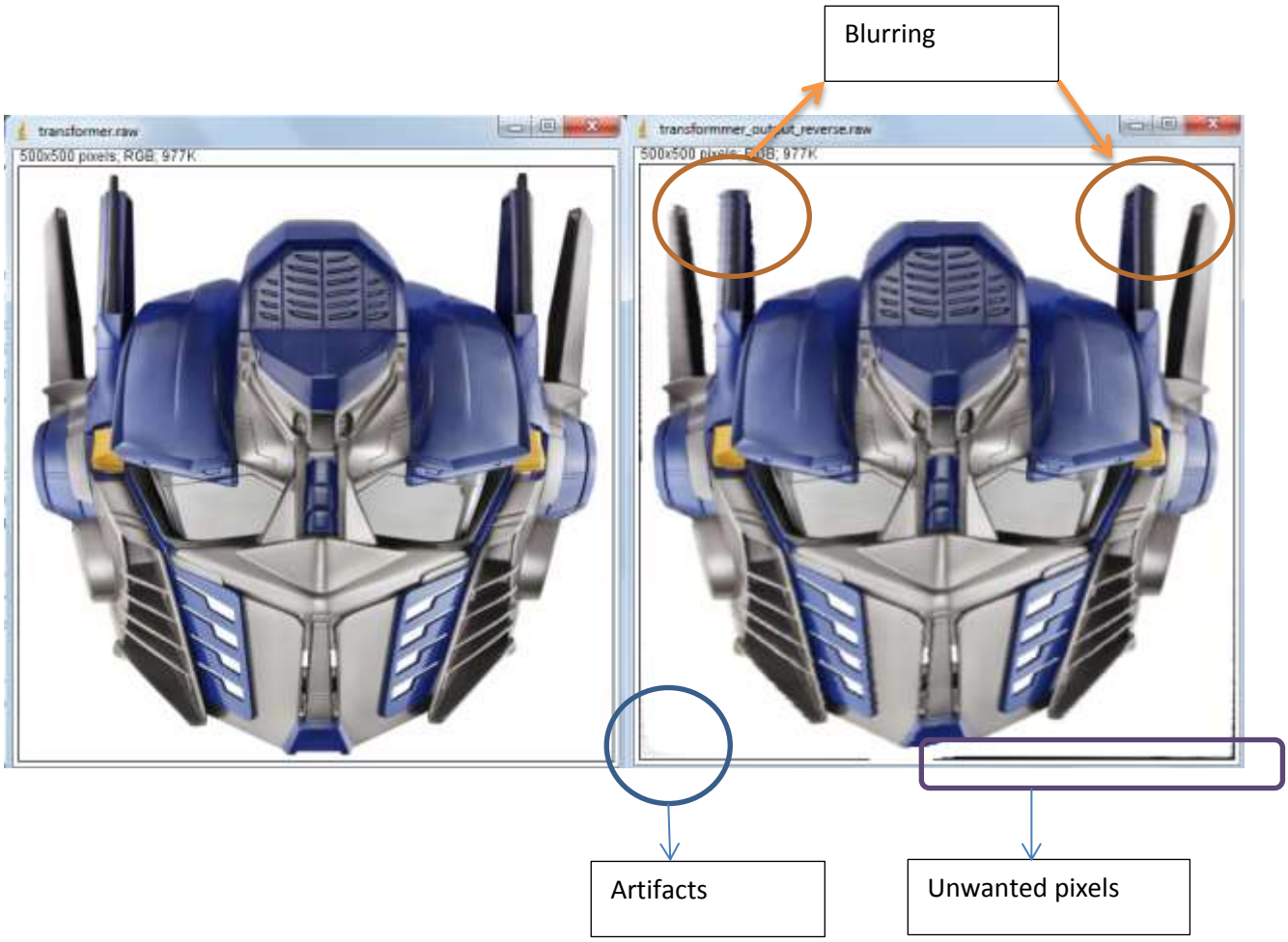

Figure 1.18 Original Transformmer image (Left). Transformmer Recovered image (right)

Manan Vyas
USC ID: 7483-8632-00 Email: mvyas@usc.edu

Problem # 2: PERSPECTIVE TRANSFORMATION AND IMAGING GEOMETRY.

Motivation: Perspective transformation and imaging geometry form the basis of 3D graphics and rendering. Imaging geometry forms the basis of real time 3D world capturing using imaging lens and then generating a corresponding 2D image. This problem gives us an insight into both the techniques.

Code Written in: PROCESING IN C++/ MATRIX CALC. IN MATLAB (inversion).

2. A: PRE-PROCESSING.

Task: To generate the location and intensity information on the surface of the cube on which the images are going to be projected.

Approach and Implementation: The given images are of the dimensions 200x200 which are needed to map to the surface of the cube of side length 2 with each X,Y,Z of the cube varying from [-1 to 1]. Thus the unit length distance mapping from 200x200 image onto the cube of side length 2 will be 2/100 = 0.02, as stated in the homework problem. This means that a unit length of 1 in the image maps to a unit length of 0.02 on the cube. For this mapping, the range mapping formula was used [4].

According to range mapping ;

"*If your number X falls between A and B, and you would like Y to fall between C and D, you can apply the following linear transform:*

$$Y = \frac{X-A}{B-A}(D-C) + C \quad [4]$$ "

In our case we want the Y to fall in [-1 1] and our X is in the range [0 200] for the width and height of the images. The following equations were derived for mapping;

Cube Surface	Projected Co-Ordinates (p,q)

z=1	p = (x+1)*100.0; //P = cubeX
	q = (y+1)*100.0; //Q=cubeY

x=1	p = ((-1)*(z-1)/2.0)*199.0; //P=cubeZ
	q = ((y+1)/2.0)*199.0; // Q=cubeY

y=1	p = ((-1)*(z-1)/2.0)*199.0; //P=cubeZ
	q = ((-1)*(x-1)/2.0)*199.0; // Q=cubeX

y=-1	p = ((-1)*(z-1)/2.0)*199.0; //P=cubeZ
	q = ((-1)*(x-1)/2.0)*199.0; // Q=cubeX

x=-1	p = ((-1)*(z-1)/2.0)*199.0; //P=cubeZ
	q = ((-1)*(y-1)/2.0)*199.0; // Q=cubeY

As instructed in the problem z=-1 plane was neglected as we do not proje any image on that surface	

Hence the input may be any point (X,Y,Z) on the surface of the cube and we would get the mapping that returns the corresponding pixel co-ordinates and the intensity that would be projected on that surface at that X,Y,Z co-ordinate.

Results and Discussion:

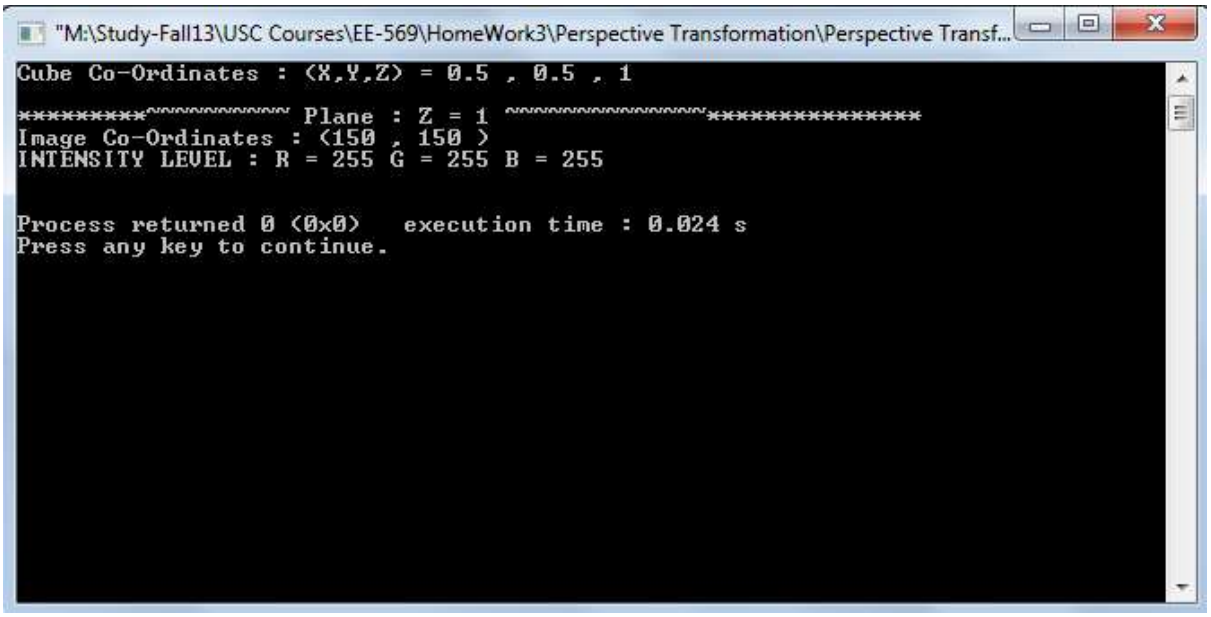

Figure 2.1 Mapping for (X,Y,Z) = (0.5,0.5,1) on command prompt.

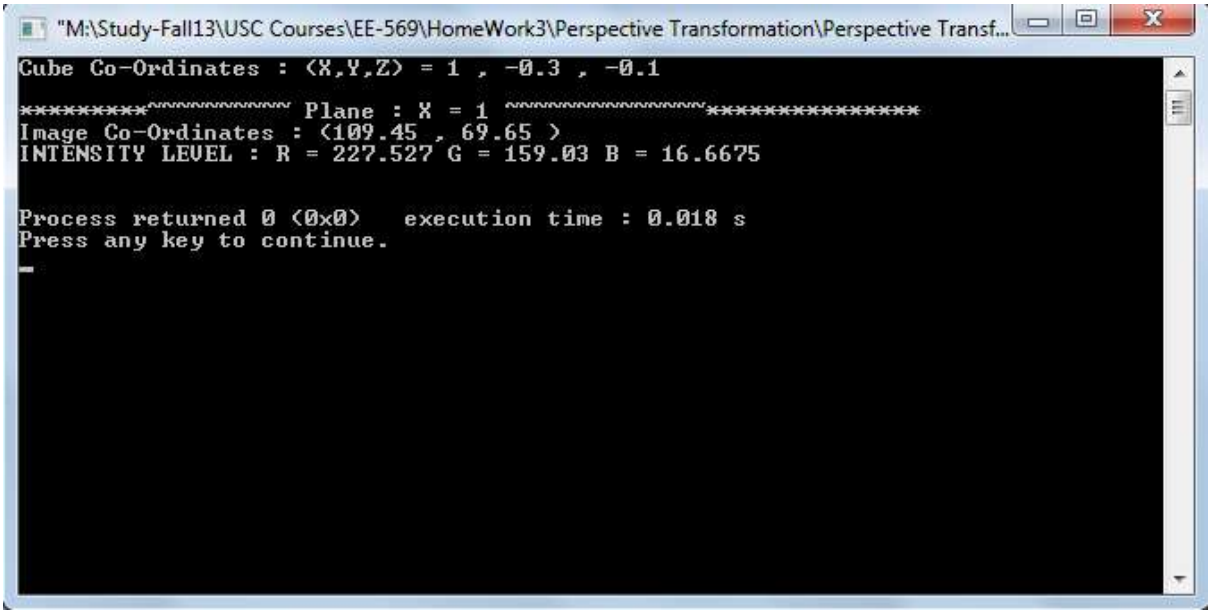

Figure 2.2 Mapping for (X,Y,Z) = (1,-0.3,-0.1) on command prompt

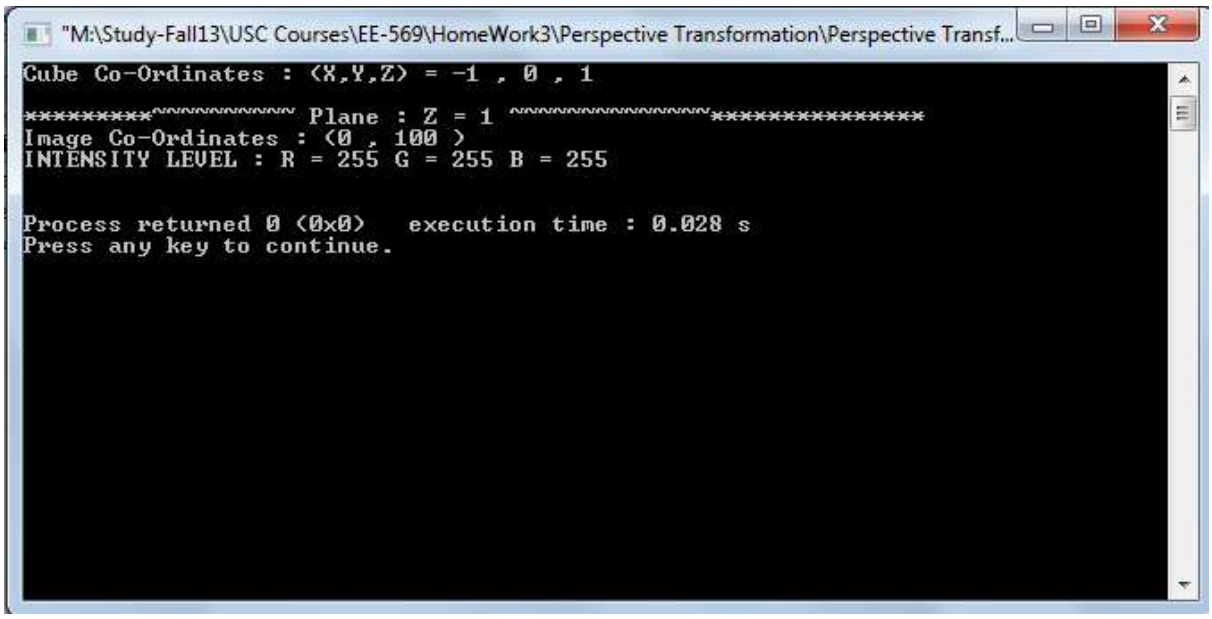

Figure 2.3 Mapping for (X,Y,Z) = (-1,0,1) on command prompt

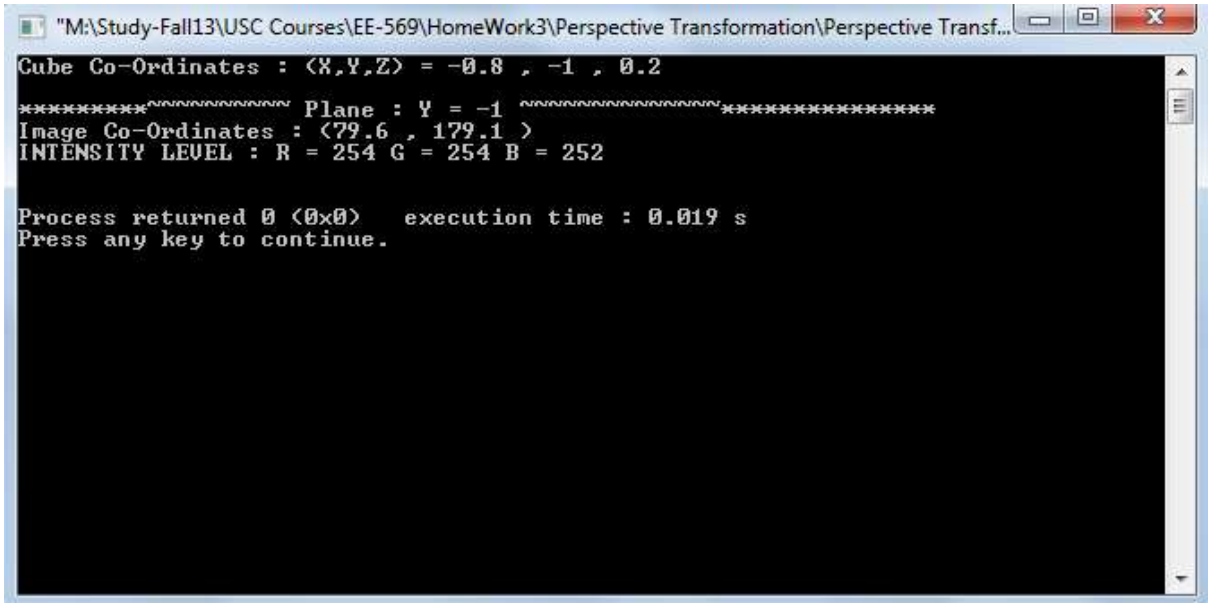

Figure 2.4 Mapping for (X,Y,Z) = (-0.8,-1,0.2) on command prompt

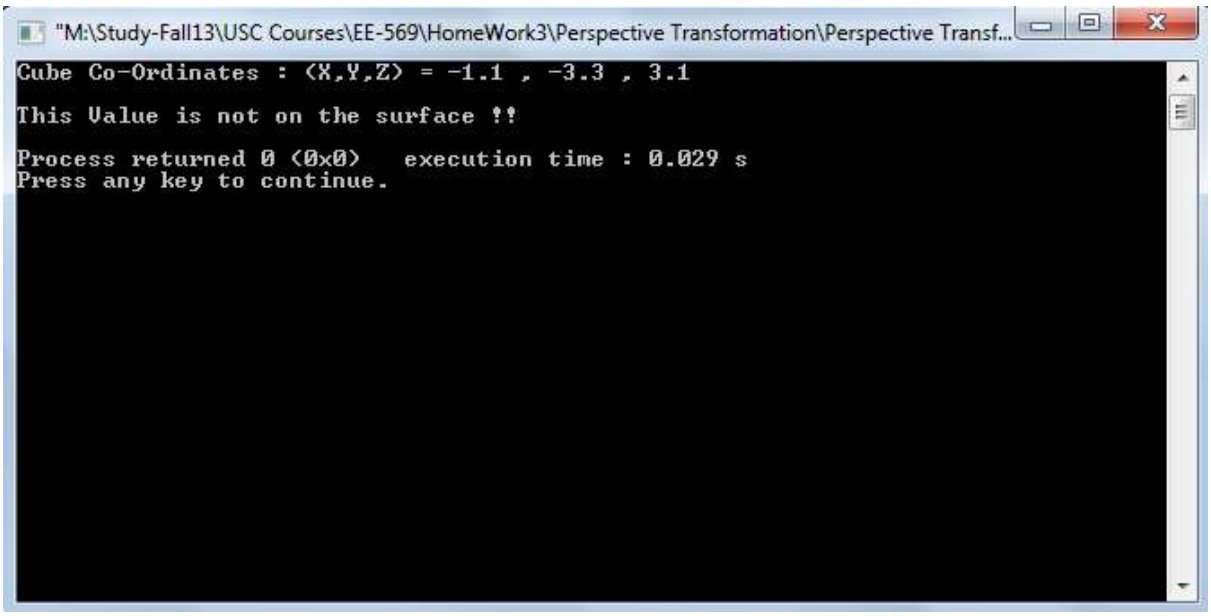

Figure 2.5 Mapping for a point not on any surface of the cube on command prompt

- Thus, it is inferred that the mapping function written as pre-processing is correct. And it also decides whether a point lies on the surface of the cube or not.

- If it doesn't lie on any of the faces of the cube, the mapping is not evaluated.

Manan Vyas
USC ID: 7483-8632-00 Email: mvyas@usc.edu

2 .B: CAPTURING A 3D SCENE:

Task: To project the given images on the surfaces of the cube and then display the cube in 2D image plane. Thus a 3D cube will be projected on 2D image plane with each surface of the cube containing the corresponding image.

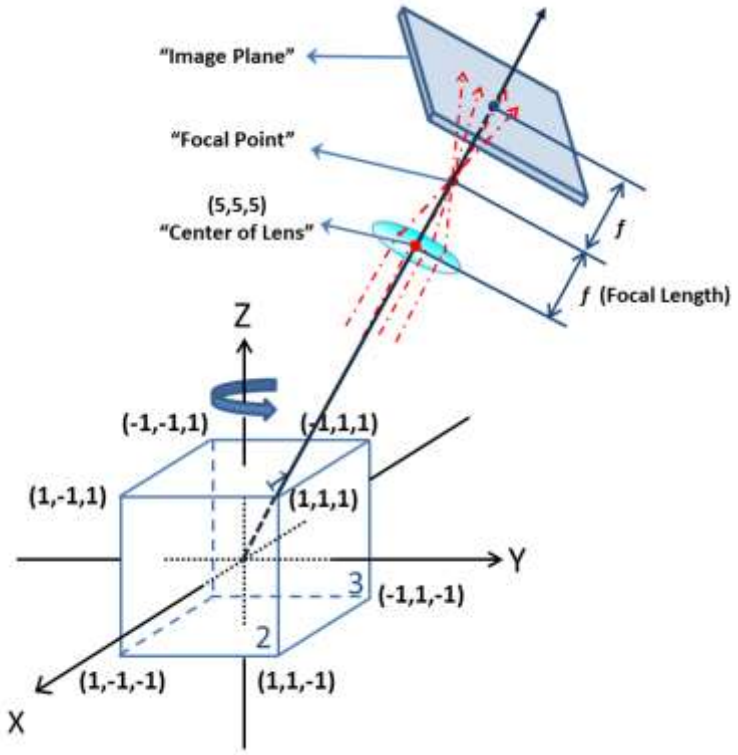

Figure 2.6 Visualization provided in HW#3 Image5 [5].

Approach and Implementation:

As instructed in the homework, firstly the forward mapping was implemented (i. e: from World Co-Ordinates -> the Camera Co-Ordinates).

The extrinsic camera matrix was evaluated with the r = 5i+5j+5k as given the homework problem. The vectors Xc,Yc and Zc are given as;

The values of the Xc, Yc and Zc vectors given in the problem were used. And the dot product of –r.Xc ,-r.Yc and –r.Zc was calculated to get the intrinsic camera matrix.

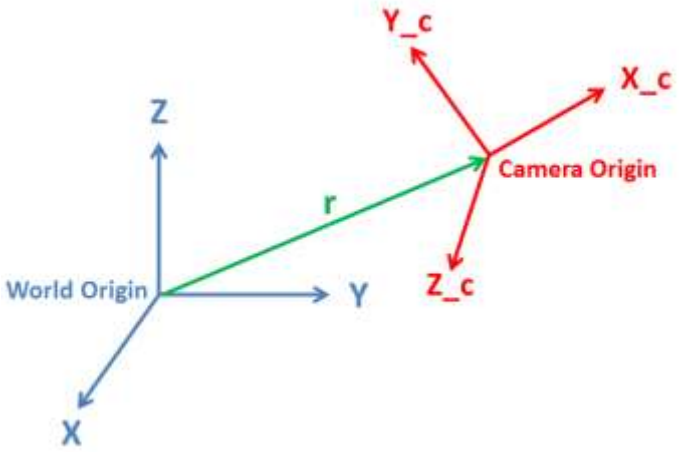

Figure 2.7 World to Camera Co-Ordinate Mapping[6].

The extrinsic camera matrix is given as ;

$$[R|t] = \begin{bmatrix} Xcx & Xcy & Xcz & 0 \\ Ycx & Ycy & Ycz & 0 \\ Zcx & Zcy & Zcz & -5\sqrt{3} \end{bmatrix}$$

Once the intrinsic camera matrix is obtained, we need to project this onto the image plane. This is done using the Intrinsic camera matrix. The focal length of the lens is given as $f = \sqrt{3}$ .the image dimesions are 200x200. Thus the intrinsic matrix was evaluated as;

$$K = \begin{bmatrix} \sqrt{3} & 0 & 100 \\ 0 & \sqrt{3} & 100 \\ 0 & 0 & 1 \end{bmatrix}$$

To project the cube, loop through all points on the surface of the cube, and using the intrinsic and extrinsic matrix, calculate the image points on which the points on the surface gets mapped to.

The resolution of the looping is the pixel density of the projected image. Eg: We can loop through -1 to 1 with increments of 0.01 indicating that the pixel density is 0.01.

There is normalizing factor w which will divide all the terms to get the mapping from camera to image.

Thus, I calculated the expression as;

[x y 1] = (1/w)* [K][R|t][X Y Z];

To evaluate the image co-ordinates and the pixel intensity that must be projected at the points got from the matrix operation, I used the pre-processing technique of 2a.

Thus, we get (ximg, yimg) from [x y 1] and corresponding RGB values at (ximg,yimg).

If (ximg,yimg) turned out to be fractional, the Bilinear Interpolation was used to calculate the RGB value at that point using the 4 nearest neighbours to approximate the pixels intensity value.

The effect of pixel density is further discussed in the results and discussion.

As instructed in the homework, now the reverse mapping was implemented (i. e: from Camera Co-Ordinates -> the World Co-Ordinates).

In reverse mapping a free variable needs to be introduced into the mapping. Since the image to camera is 2D to 3D and then camera to world co-ordinate is 3D. Hence, the free variable needs to be introduced in order to take care of the depth information. For this implementation, different free variables were used.

Now to calculate the reverse mapping the result of [K][R|t] was inversed using the MATLAB function inv(mat). And the different free variables were plugged into the matrix operation to get the reverse mapping.

Results and Discussion for Forward Mapping:

For projecting the images, the center of the image was fixed to one of 4 vertex of the cube surface.

- For Image1 : (I,j) = (0,0) was kept at (-1,-1,1) on z=1 plane.
- For Image2 : (I,j) = (0,0) was kept at (1,-1,1) on x=1 plane.
- For Image3 : (I,j) = (0,0) was kept at (1,1,1) on y=1 plane.

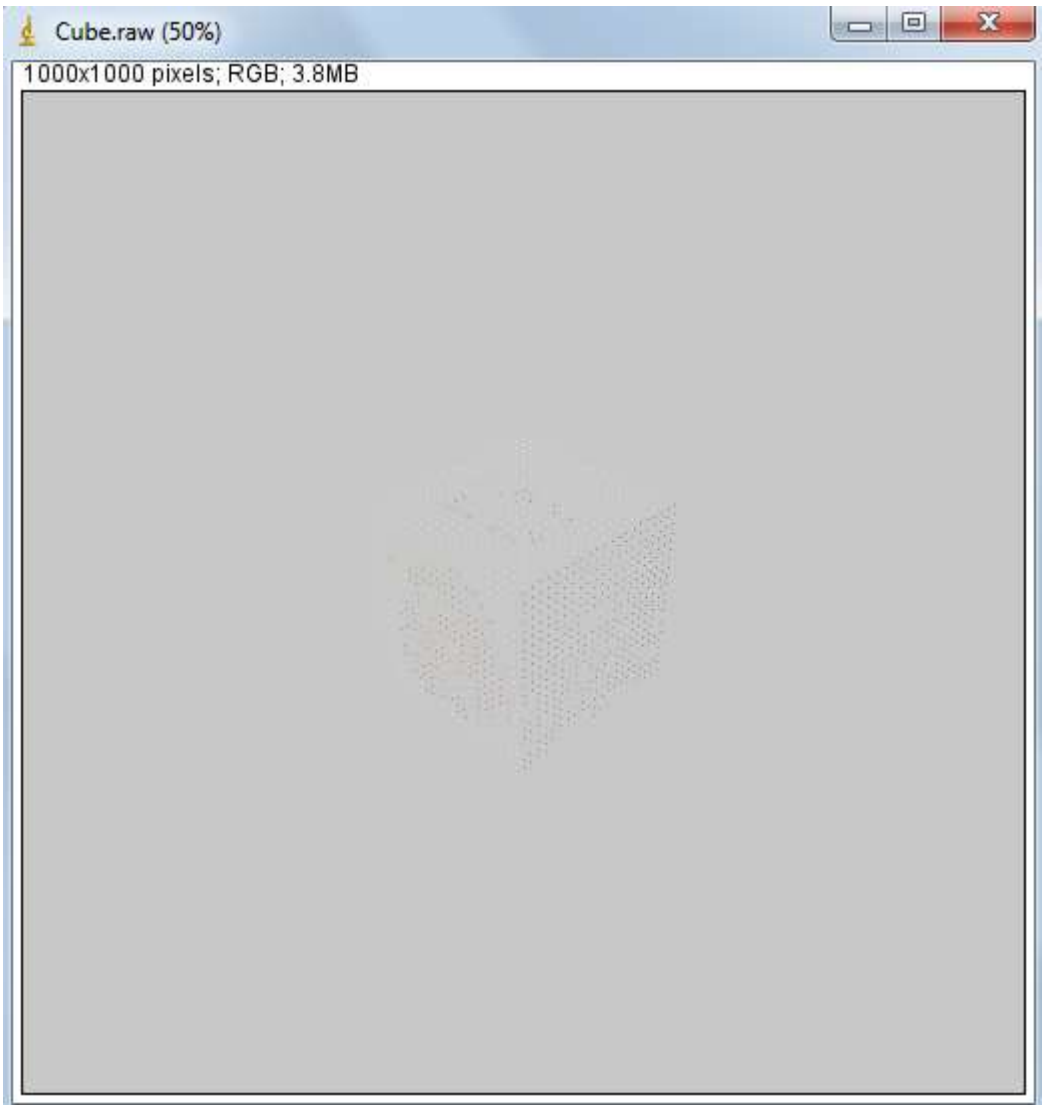

Figure 2.8 Cube captured with pixel density of 0.1

The pixel density = 0.1 is a very poor choice. It seems as no cube is projected.On looking closely, very few pixels are mapped. Hence pixel density need to be decreased furthermore.

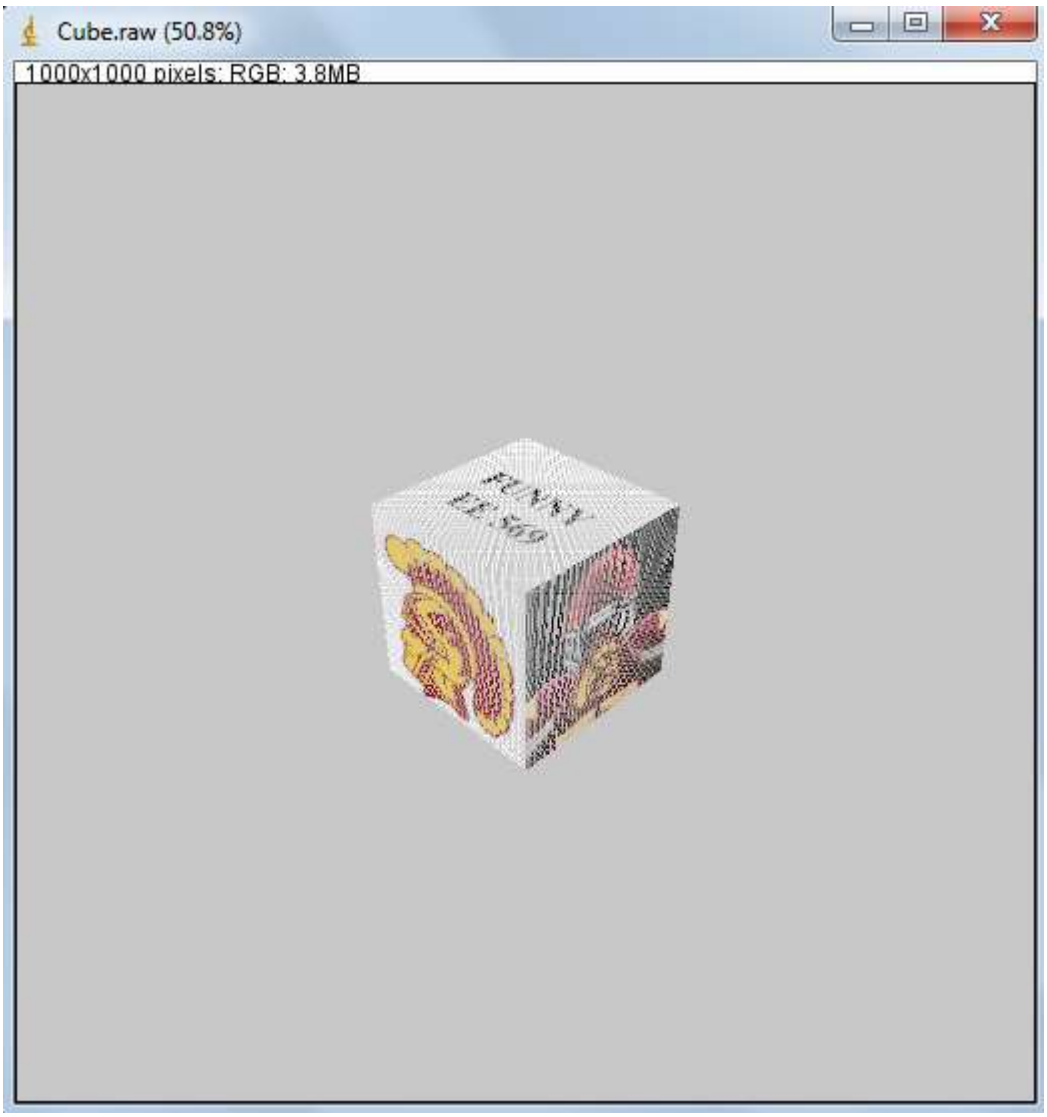

Figure 2.9 Cube captured with pixel density of 0.02

This is also not an optimum pixel density value as the visibility of the images on the surfaces is still poor. Futher decreasing the pixel density.

Figure 2.10 Cube captured with pixel density of 0.01.

This is the perfect cube image. The corresponding pixel density is 0.01. There are no artifacts visible in the image and all edges are crisp and clearly visible. The image content are also true to the original and distortion is visible.

Results and Discussion for Reverse Mapping:

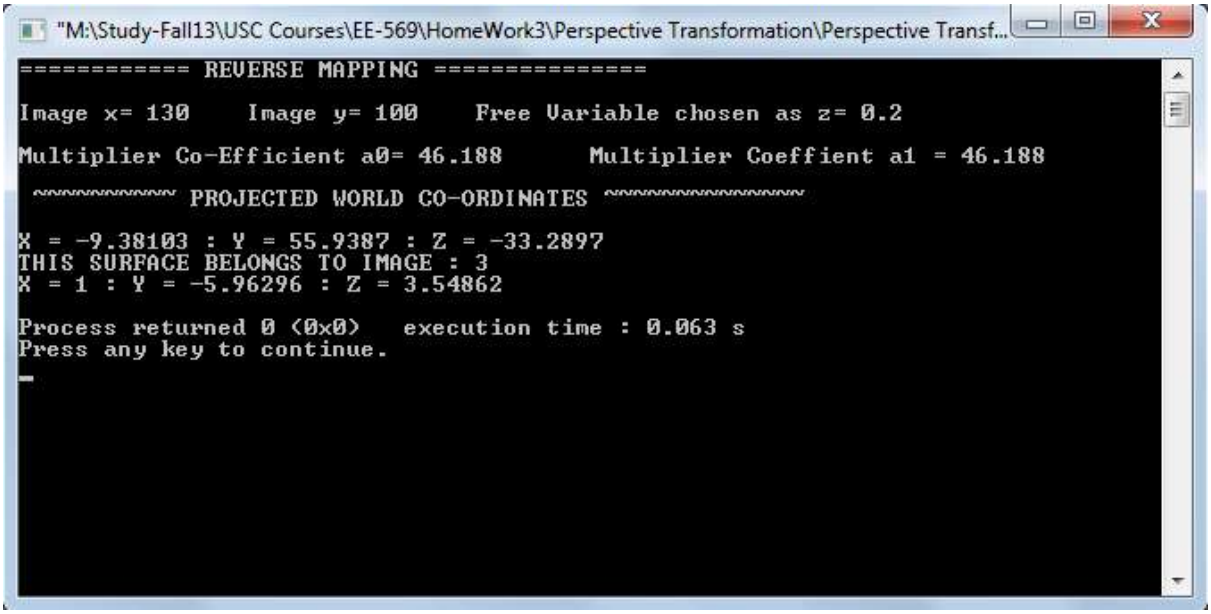

Figure 2.10 Reverse mapping with free variable 0.2 and image co-ordinates (130,100).

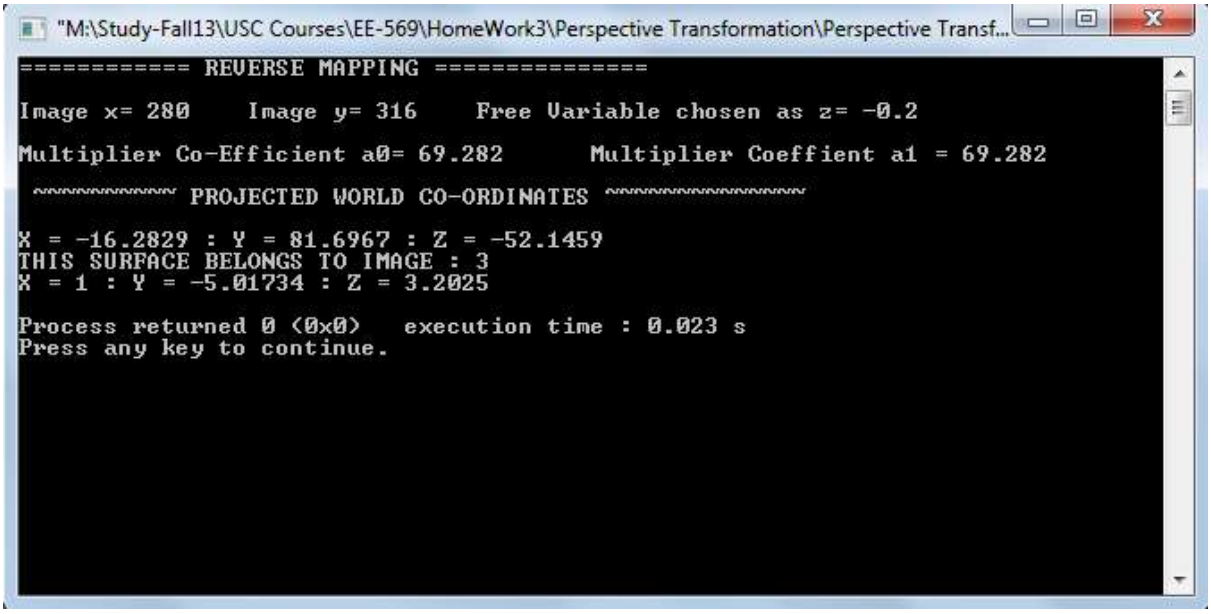

Figure 2.11 Reverse mapping with free variable -0.2 and image co-ordinates (280,316).

However, with this the reverse mapping at least one of the X,Y,Z comes out as outside the surface of the cube. Hence, the main challenge of reverse mapping is to maintain the mapping boundary for the projected images on the 3D object. Hence, this needs it be further improved upon.

Manan Vyas
USC ID: 7483-8632-00 Email: mvyas@usc.edu

2 .C: ROTATING CUBE:

Task : To rotate the cube along the Z-axis along the anti-clockwise direction and capture frames at angles 0-90deg with increments of 1deg. Combine these 90 frames to form a video of cube rotation upto 90deg.

Approach and Implementation:

To perform rotation of cube in anti-clockwise direction, the following rotation matrix was used;

$$Rot = \begin{bmatrix} \cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Important Consideration: Since we have projected the 3D scene to 2D, simply rotating the cube by a certain angle will create distortion. Hence, all the images need to be aligned properly.

Another important aspect is that after rotation by some angle in anticlockwise direction, the surface #5 will be visible, hence, the image5 will will visible after approximately 55 degrees. Hence I projected the image5 on the surface 5 before performing the rotation.

- When the image 2 on the surface#2 will rotate, it will wraparound and produce distortion and interfere with the image#1 and both the images will be partially visible which is not desired. Hence we need to supress the image that will not be visible after rotating through some angle and keep the one that will still be visible.
- For this inside the main rotation loop, I used a check_flag image to make sure the wrap around does not occur and the image that is not supposed to be visible is indeed not visible.
- I firstly, keep set all the pixels in the check_fill image to be 0. Then calculate all the necessary image projection and the transformations to generate the cube as decribed in part(2) of the problem and then rotate it by the above matrix transformation by and arbitrary angle. Now the ximg and yimg co-ordinates obtained from forward projection of the camera to image co-ordinates.
- Now check the corresponding (ximg,yimg) co-ordinates in the check_flag image. If it is zero then copy the intensity locations from the corresponding input image1-5 to the projection and set that pixel location (ximg,yimg) to 1 in the check_flag.
- For the first rotation from 0deg to 1deg this condition is always satisfied but there after it helps in giving the perfect output.
- Loop this process for angles varying from 0 to 90deg incremental steps of 1.

Result and Discussion:

The following are the different frames obtained after rotation by a certain angle.

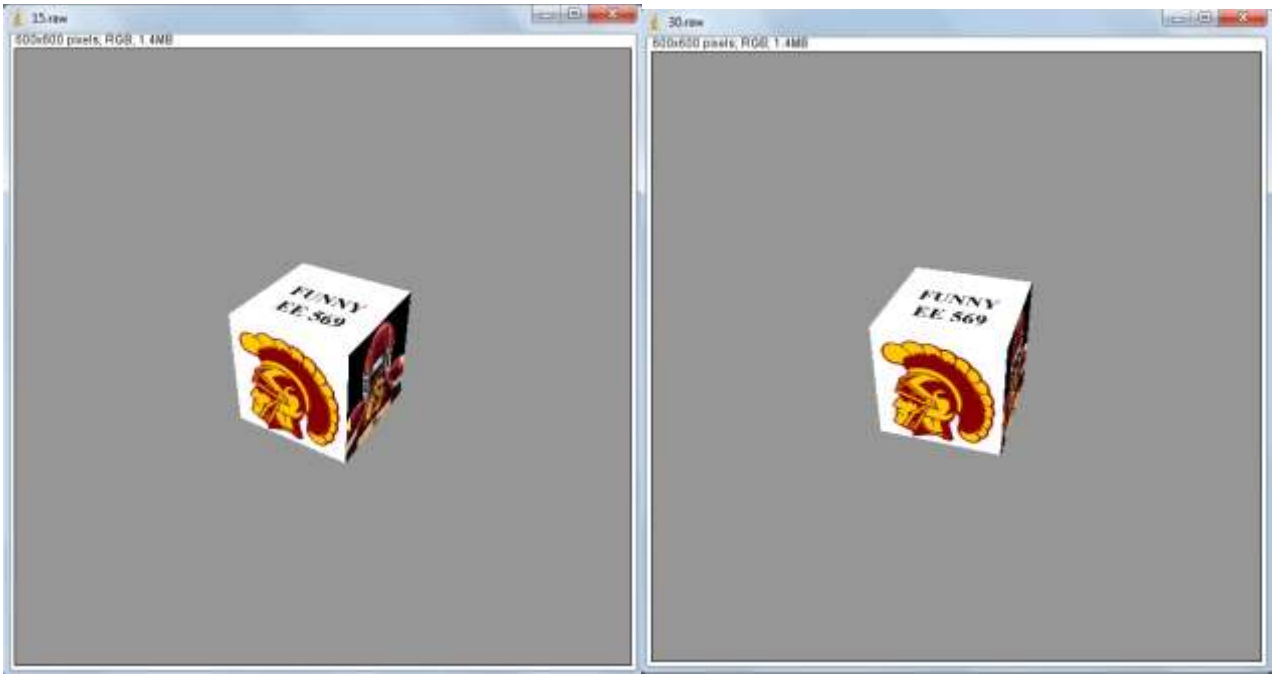

Figure 2.12 Cube rotated by 15deg (LHS) and 30deg (RHS) in anticlockwise direction along the Z-axis

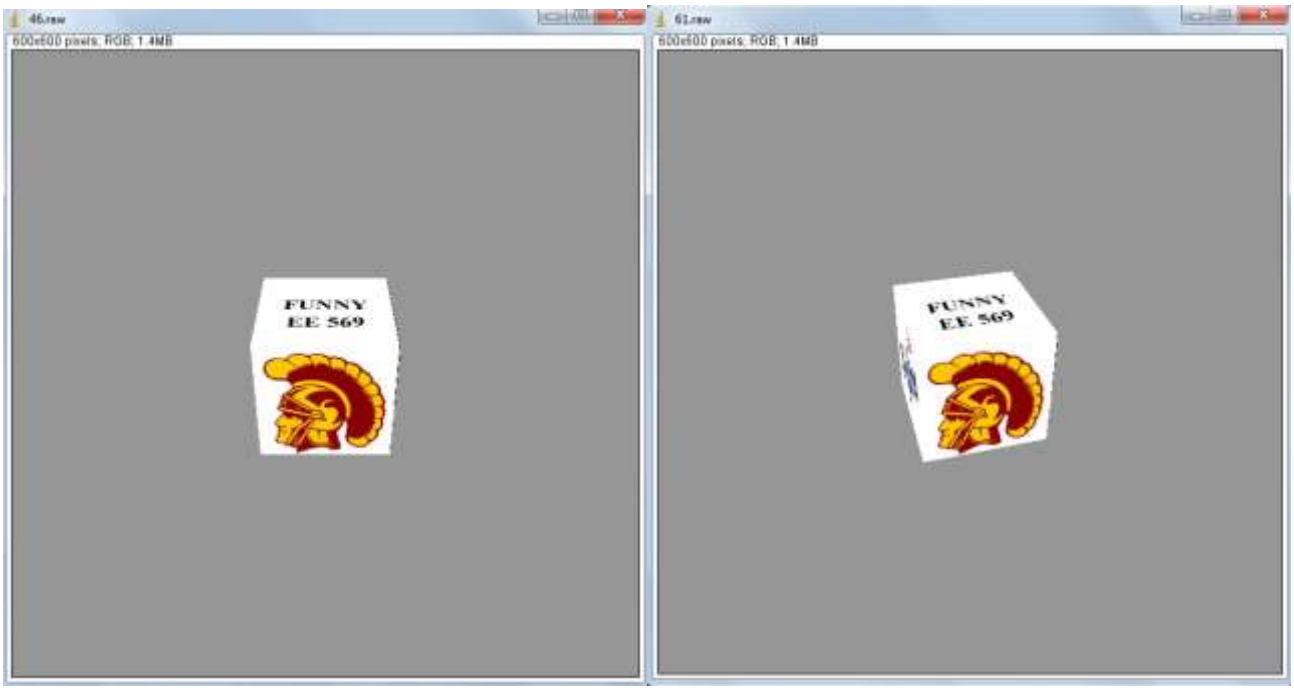

Figure 2.13 Cube rotated by 45deg (LHS) and 60deg (RHS) in anticlockwise direction along the Z-axis

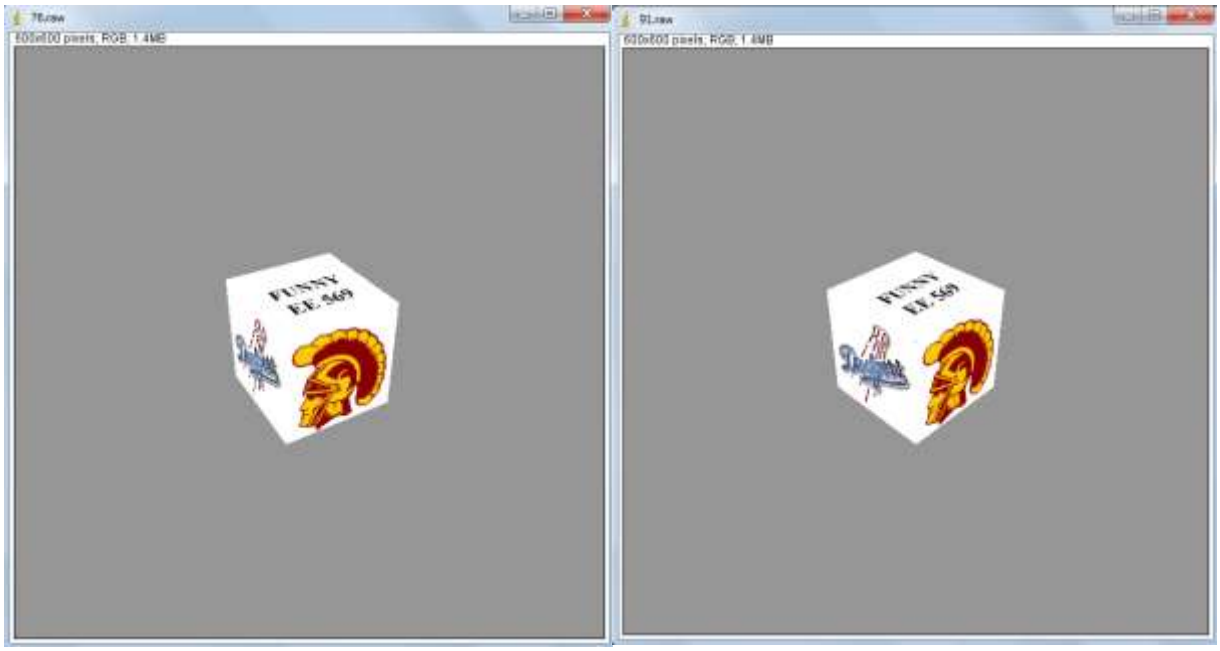

Figure 2.14 Cube rotated by 75deg (LHS) and 90deg (RHS) in anticlockwise direction along the Z-axis

The following is a of screenshots of the video playing;

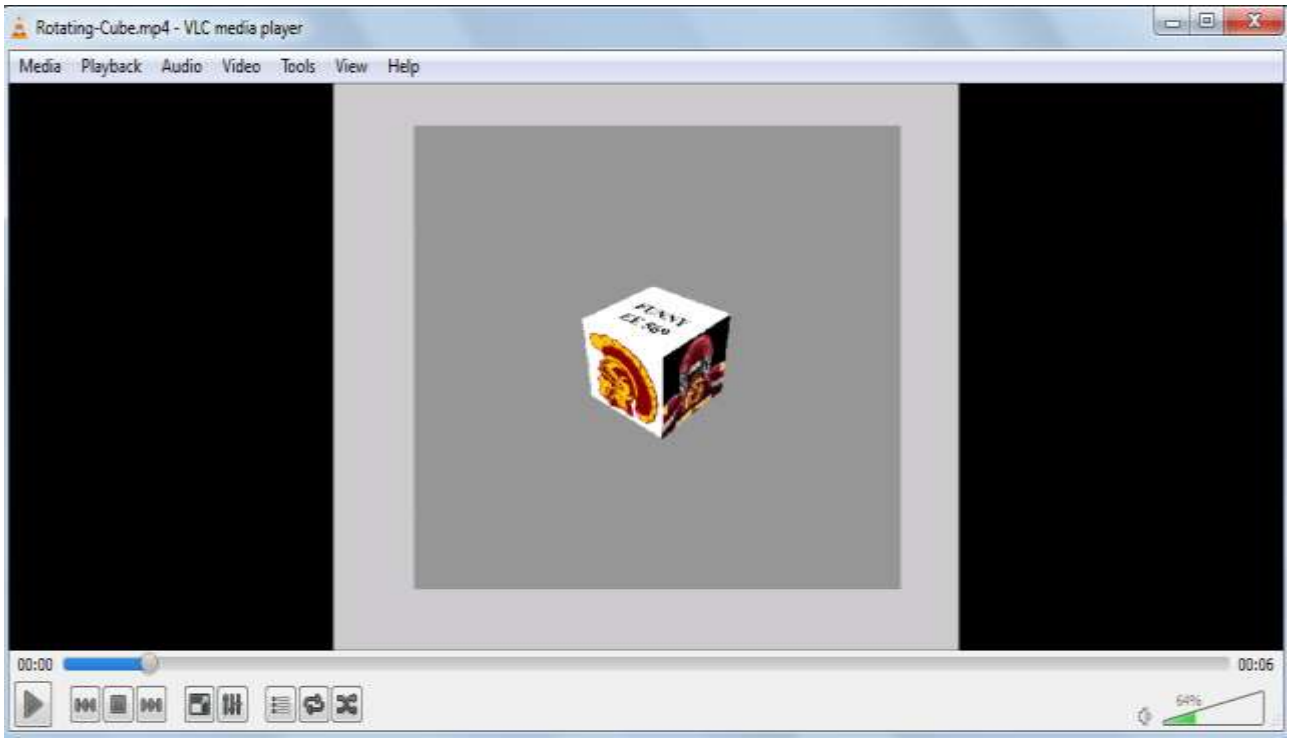

Figure 2.15 Screenshot of Cube rotation video playing in VLC

Manan Vyas
USC ID: 7483-8632-00 Email: mvyas@usc.edu

Problem # 3: TEXTURE ANALYSIS AND SEGMENTATION USING LAWS FILTERS.

Motivation: Texture processing is now becoming an important part of image processing domain. One application of this would be using texture image processing on the images sent by unmanned drones used to sweep through areas on the earth to detect the changes in landscapes, forests, soil composition, human habitation etc. These areas have different textures looking from up above and texture classification and image segmentation becomes an important aspect of analysing such data.Texture Clustering and segmentation are important aspect of texture analysis. Laws filters are one of many filters used for texture classification in image proessing.

3. A: Texture Image Clustering

Task: To construct the 9, 5x5 Laws filters corresponding to edge spot and wave and perform feature extraction on the 12 given texture images using these 9 Laws filters. Once we have the features and their averages, use K-means to cluster and label each texture.

Code implemented in: All Processing in C++/ Only Kmeans output shown in Matlab.

Approach and Implementation:

To generate the nine 5x5 Laws filters, the tensor product of each Filter with itself and one another needs to be evaluated. The given 3 Laws filters are;

$$E5 = \frac{1}{6}[-1\ -2\ \ 0\ \ 2\ \ 1]\,; S5 = \frac{1}{4}[-1\ \ 0\ 2\ 0\ -1]; W5 = \frac{1}{6}[-1\ 2\ 0\ -2\ 1]$$

Now using tensor products to calculate the EE5,ES5,EW5 ….. WS5 and WW5.

So, EE5 = E5 $\otimes$ E5^T , ES5 = E5 $\otimes$ S5^T and so on….

Now apply these laws filters on the texture images provided to get the feature vectors for each image. Thus we will have 9 feature vectors for each image corresponding to the filter EE5,ES5 …. WW5.

Since these filters are 5x5 in dimensions, each texture image was boundary stretched by amount 2 .For boundary extension, the first row and column we copied twice at each boundary of all the texture image .***Thus, Pixel copying was used for Boundary Extension.***

Thus feature vector (I,j) of a particular texture image is given as;

$$Feature[i,j][k] = \sum_{i,j=2}^{i,j=height-3,width-3} \sum_{m,n=-2}^{m,n=2} LawsFilter[m+2][n+2] * TextureImg[i+m][j+n]$$

Thus we have total 12x9 feature vectors, 9 feature vectors for one image and there are total 12 texture images.

Now for one particular image, each feature vector has to be averaged. Thus the whole featureVector[I,j] will be average to just one value. Thus we will have 9 such 1D average values for one image and there are total 12 images.

$$Average\ Feature\ Vector\ [i = 0..8] = \frac{\sum_{i.j=0}^{i,j=Image\ Dimensions}(FeatureVector[i,j][k = 0..8])\text{^}2}{128 * 128}$$

And then K-means was applied on these average feature vectors of all the 12 images.

The Kmeans in MATLAB was used and the initial centroids were given as 'samples', meaning MATLAB takes random samples of initial centroids. The clustering was replicated over 10 iterations so that the cluster with maximum inter cluster variance was given in the end.

Since there are total 4 visibly different textures in the given images, the number of clusters will be 4. Thus MATLAB output will consist of label assign to each image cluster.Thus it was expected that kmeans would help classify the similar texture with same label.

Results and Discussion:

The following output was obtained after 10 replications[7] of kMeans with 4 clusters and selecting the one with minimum classification error. (using 'replicates'), and initial centroids selected at random (default for Kmeans function of Matlab).

Figure 3.1: Output of Kmeans Labelling

Thus it is observed that (1,4,6) are similar with label 3 , (3,5,11) are similar with label 1, (2,9,12) are similar with label 2 and (7,8,10) are similar with label 4.

The following output was obtained after 10 replications of kMeans with 4 clusters with no replications.

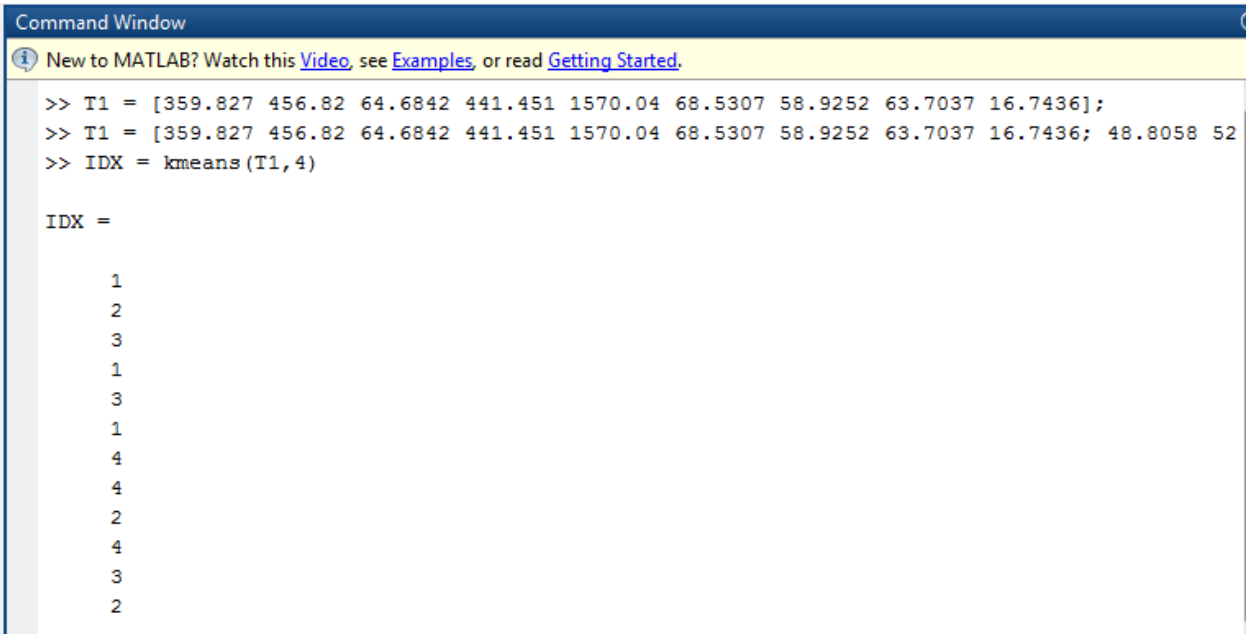

Figure 3.2 : Output of Kmeans Labelling

Thus it is observed that (1,4,6) are similar with label 1 , (3,5,11) are similar with label 3, (2,9,12) are similar with label 2 and (7,8,10) are similar with label 4.

Now the given 12 textures are as follows;

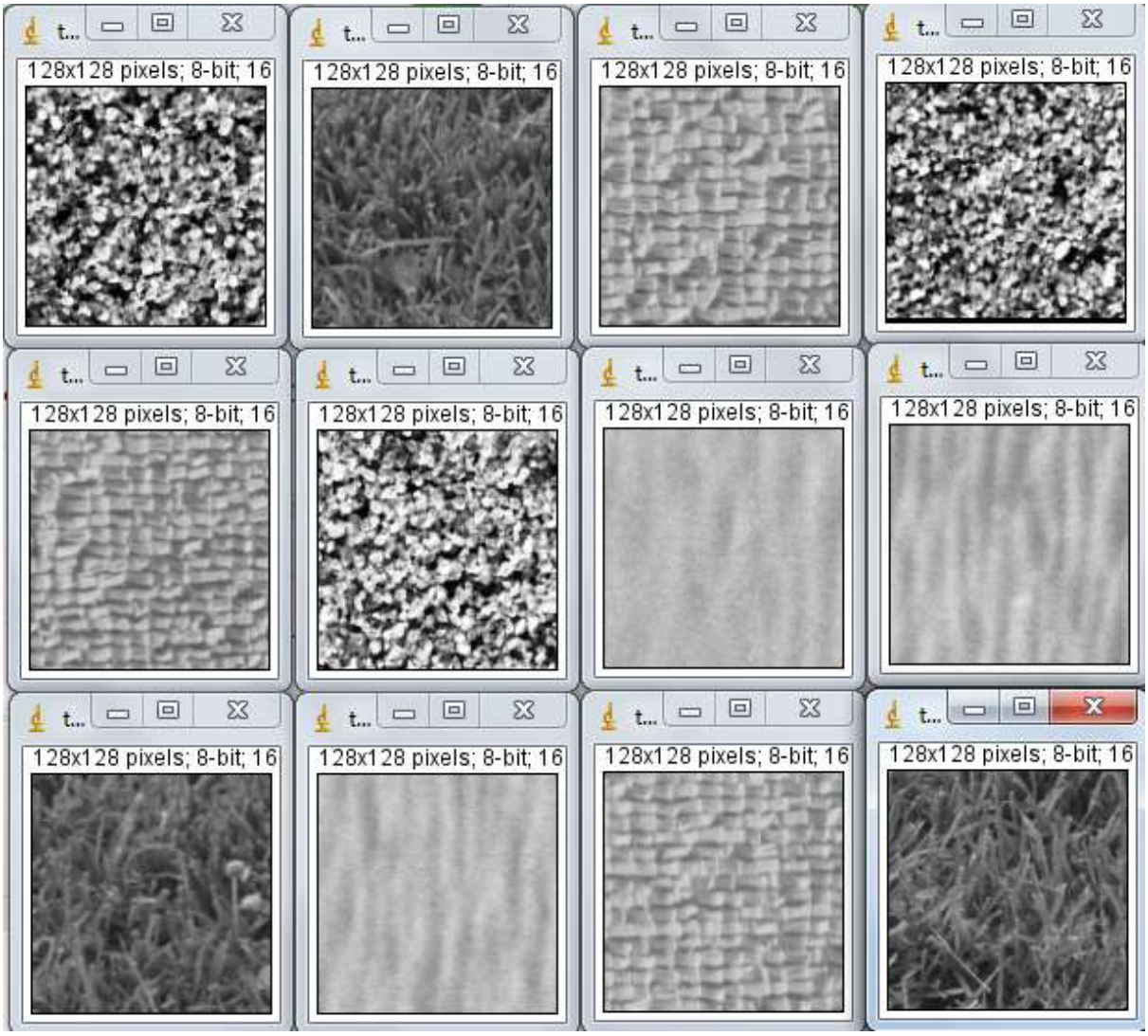

Figure 3.3 : 12 Texture Images given in order
1234 , 5678 , 9 10 11 12

Thus on inspection it can be easily found that (1,4,6) are similar , (3,5,11) are similar, (2,9,12) are similar and (7,8,10) are similar.

Thus the Kmeans has perfectly labelled and clustered similar texture and there is no discrepancy obtained in the output.

Manan Vyas
USC ID: 7483-8632-00 Email: mvyas@usc.edu

3. B: TEXTURE SEGMENTATION

Task: To construct the 9, 3x3 Laws filters corresponding to edge spot and local average and perform energy computation on the given composite texture images using these 9 Laws filters. Once we have the energies normalized with respect to local averaging kernel of the Laws filter, use K-means to cluster and label each texture within the image using 6 different grey scale.

Code implemented in: All Processing in C++ / Kmeans and Output Image generated in MATLAB.

Approach and Implementation:

To generate the nine 3x3 Laws filters, the tensor product of each Filter with itself and one another needs to be evaluated. The given 3 Laws filters of dimension 3 are;

$$L3 = \frac{1}{6}[1 \;\; 2 \;\; 1]\,; E3 = \frac{1}{2}[-1 \;\; 0 \;\; 1]; S3 = \frac{1}{2}[\,1 \;\; -2 \;\; 1]$$

Now using tensor products to calculate the LL3,LE3,LS3 ….. SE3 and SS3.

So, LL3 = L3 $\otimes$ $L3^T$, LE3 = L3 $\otimes$ $E3^T$ and so on….

Now apply these laws filters on the texture images provided to get the feature vectors for each image. Thus we will have 9 feature vectors for each image corresponding to the filter LL3,LE3 …. SS3.

Since these filters are 3x3 in dimensions, each texture image was boundary stretched by amount 1 on all the sides .For boundary extension, the first row and column we copied above/below/left and right of each boundary of all the texture image .***Thus, Pixel copying was used for Boundary Extension.***

Thus feature vector (I,j) of a particular Composite texture image is given as;

$$Feature[i,j][k] = \sum_{i,j=1}^{i,j=height-2,width-2} \; \sum_{m,n=-1}^{m,n=1} LawsFilter[m+2][n+2] * TextureImg[i+m][j+n]$$

Using an appropriate Kernel, the energies of the feature vectors are then calculated for each feature 0...8 for the composite texture image.

Energy was calculated using the following formula;

$$Energy(Feature(i,j)[k]) = \sum_{Image\ Dimensions\ Stretched} \; \sum_{Window\ Size} Feature(i,j)[k] * Feature(i,j)[k]\,; k = 0 \ldots .8\,;$$

Since the local average provides little insight into the feature of the image, it was advised in the homework to normalize the energies of all the feature vectors using the energy of the feature vector [0] = LL3 feature vector.

$$Normalized\ Energy\ (Feature(i,j)[k]) = \frac{Feature(i,j)[k]}{Feature(i,j)[0]}\ ; Feature\ 0\ corresponds\ to\ LL3$$

Thus the nine 1D normalized energies corresponding to each feature vector is obtained with dimension width*height of the composite texture image.

Now, the Kmeans was applied on an array containing these 9x(width x height) features of the image. ***Since there are 6 different texture visible to the naked eye in the composite image, the number of clusters chosen was 6. Hence the Kmeans will label all the similar feature vectors within a cluster with a unique number between 1 to 6***.

To construct the output image, a simple if else condition was used to generate the image with different grey levels corresponding to each label. The representative intensities were [0, 51, 102, 153, 204, 255].

Thus the expected output will be image that contain areas of different texture segments filled with these 6 representative intensities.

Results and Discussion:

The following results were obtained,

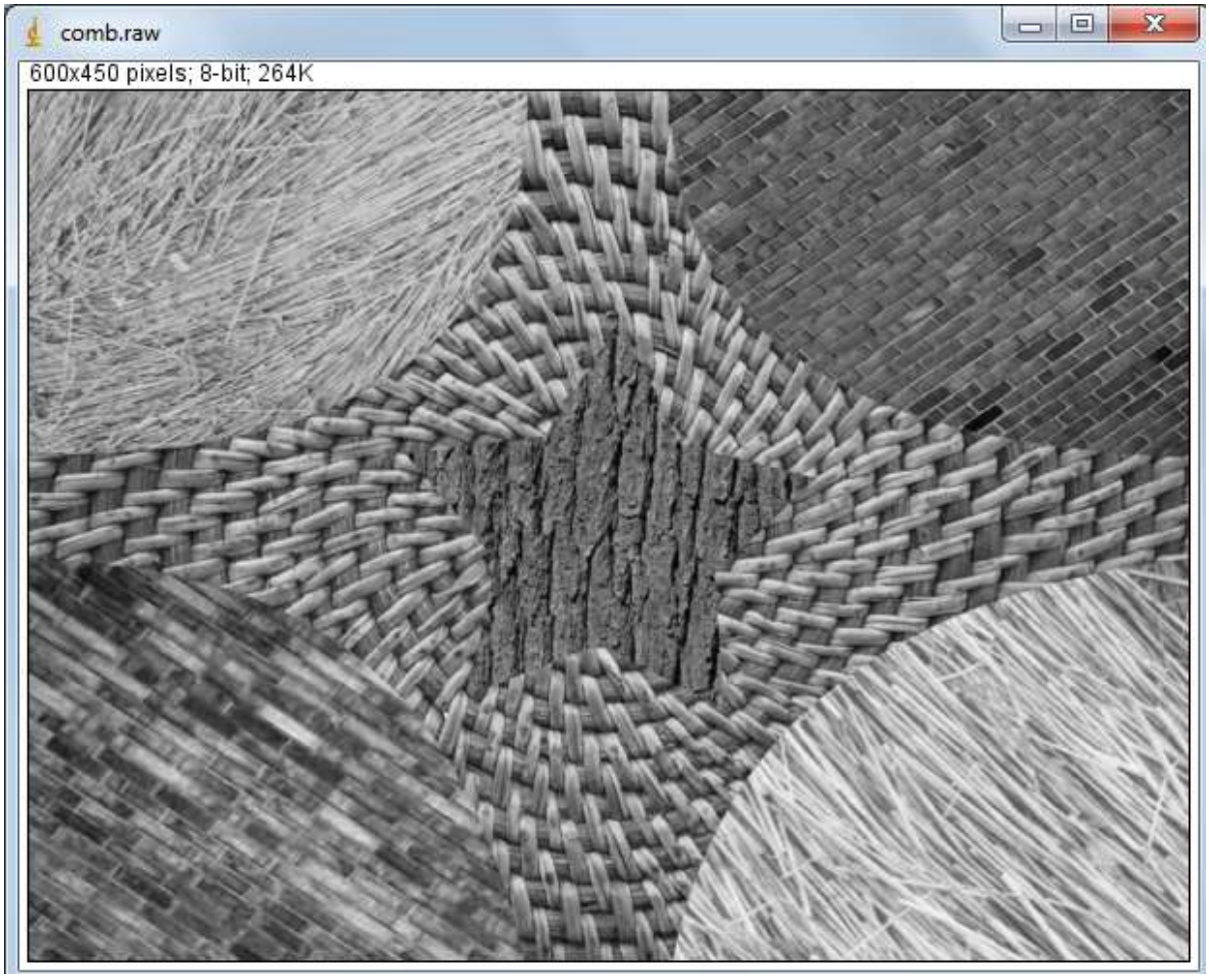

Figure 3.4 : Given composite texture image

Firstly a 13x13 kernel was used to compute the energies of all the feature vectors.And random centroids were initialized by the MATLAB Kmeans function.

The following output was obtained;

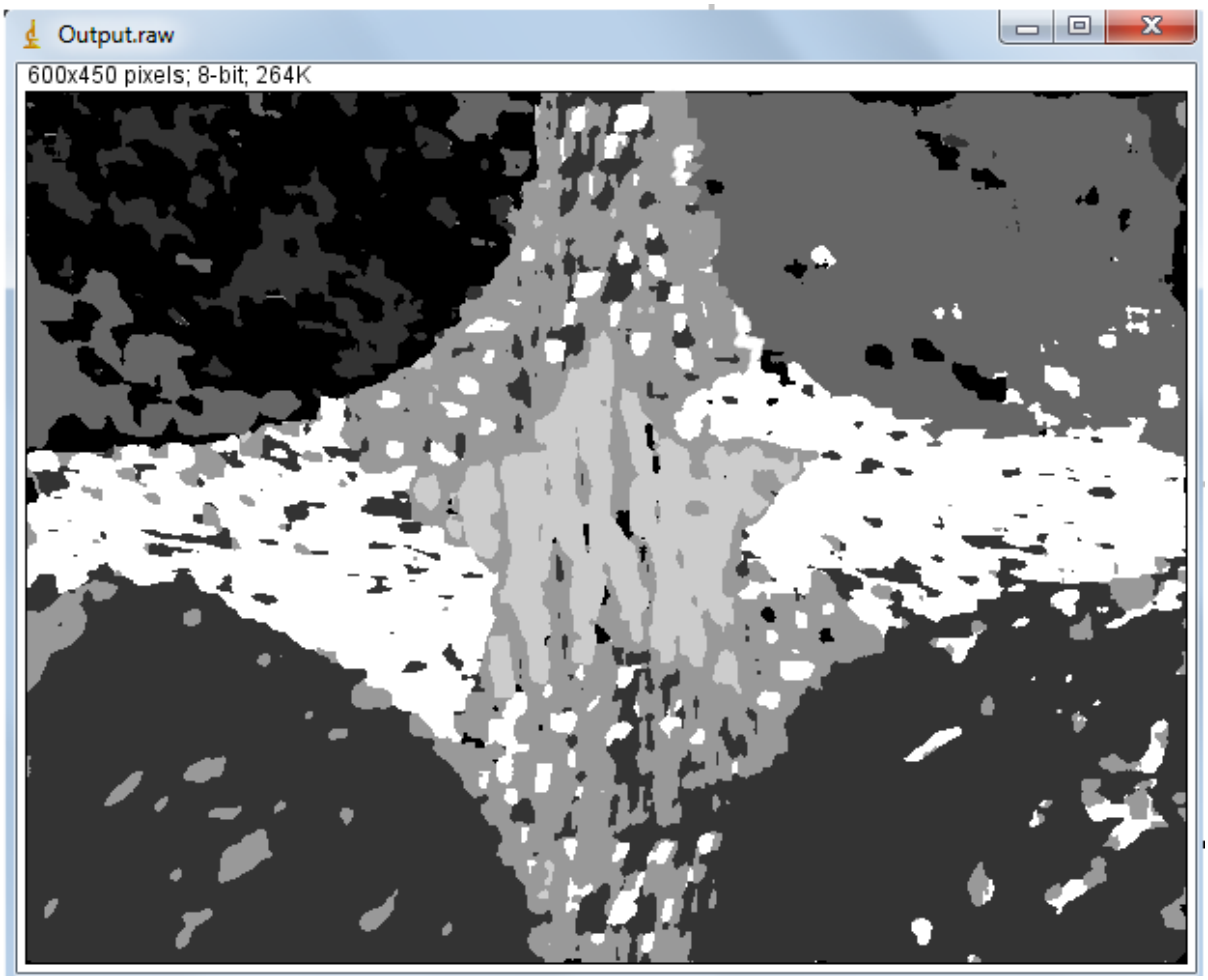

Figure 3.5: Texture segmentation using the 13x13 energy window and random initla centroids of Kmeans.

- Thus it is observe that some texture are misclassified or classified in correctly due to similar nature of the texture in neighbourhood. Also there is mi-classification within the one brick and one straws texture patterns in the bottom left and bottom right corners. The segmentation is not distinct as in the the output contains overlapping segments of clusters and not a distinct region containing only one intensity level
- Straws texture in top left corner classified almost distinctly than others.
- Vertical texture of center star was mis-classified as its neighbourhood texture also containing many vertical segments.
- Sporadic mis-classification occurs within the four corner texture clusters and most of them confused with the adjacent texture.
- Instead of classifying the bricks and straws in the botoom, it labels them as same texture. And uniform texture around the centre star was classified using different labels for horizontal and vertical directions even though the texture is same.

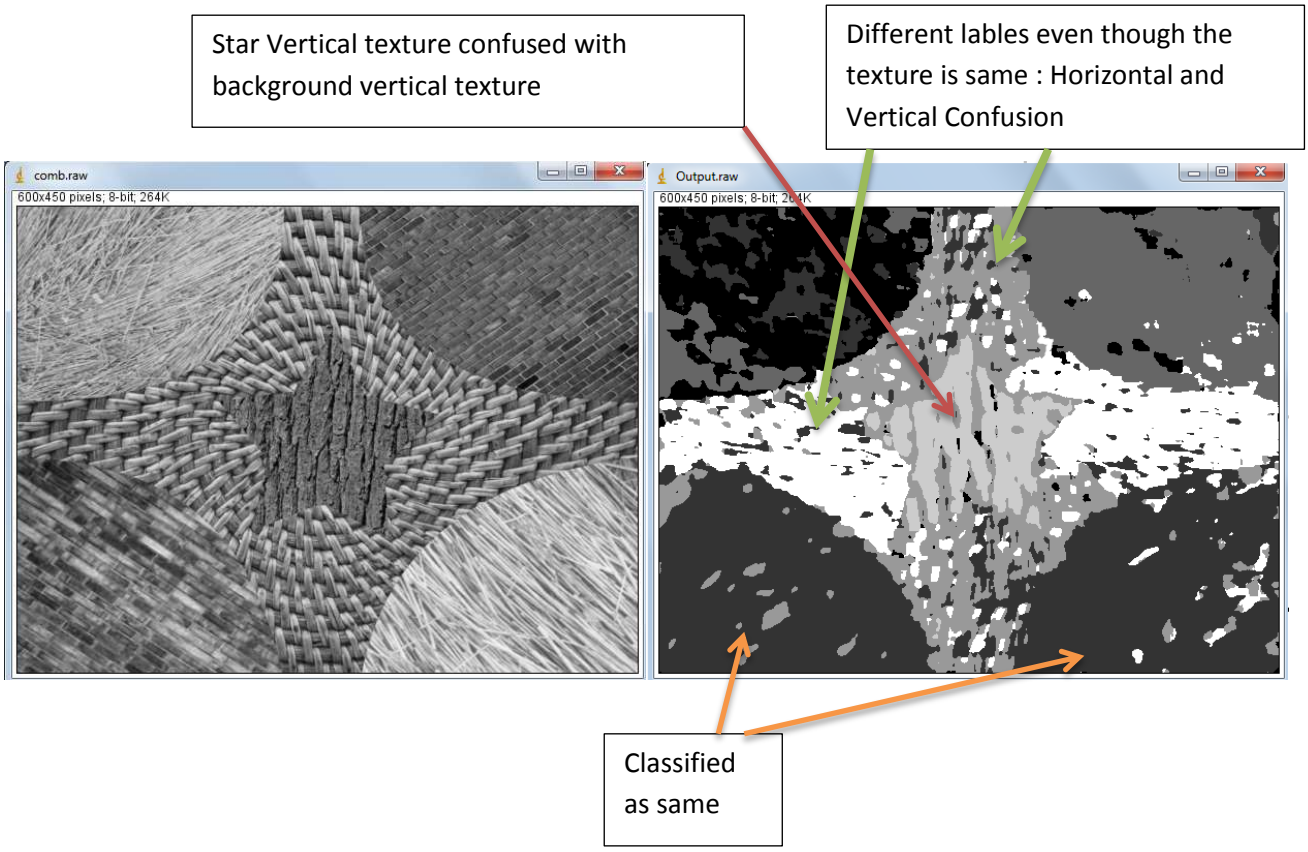

Figure 3.6: Texture segmentation Analysis.

Firstly a 39x39 kernel was used to compute the energies of all the feature vectors.And random centroids were initialized by the MATLAB Kmeans function.

The following output was obtained;

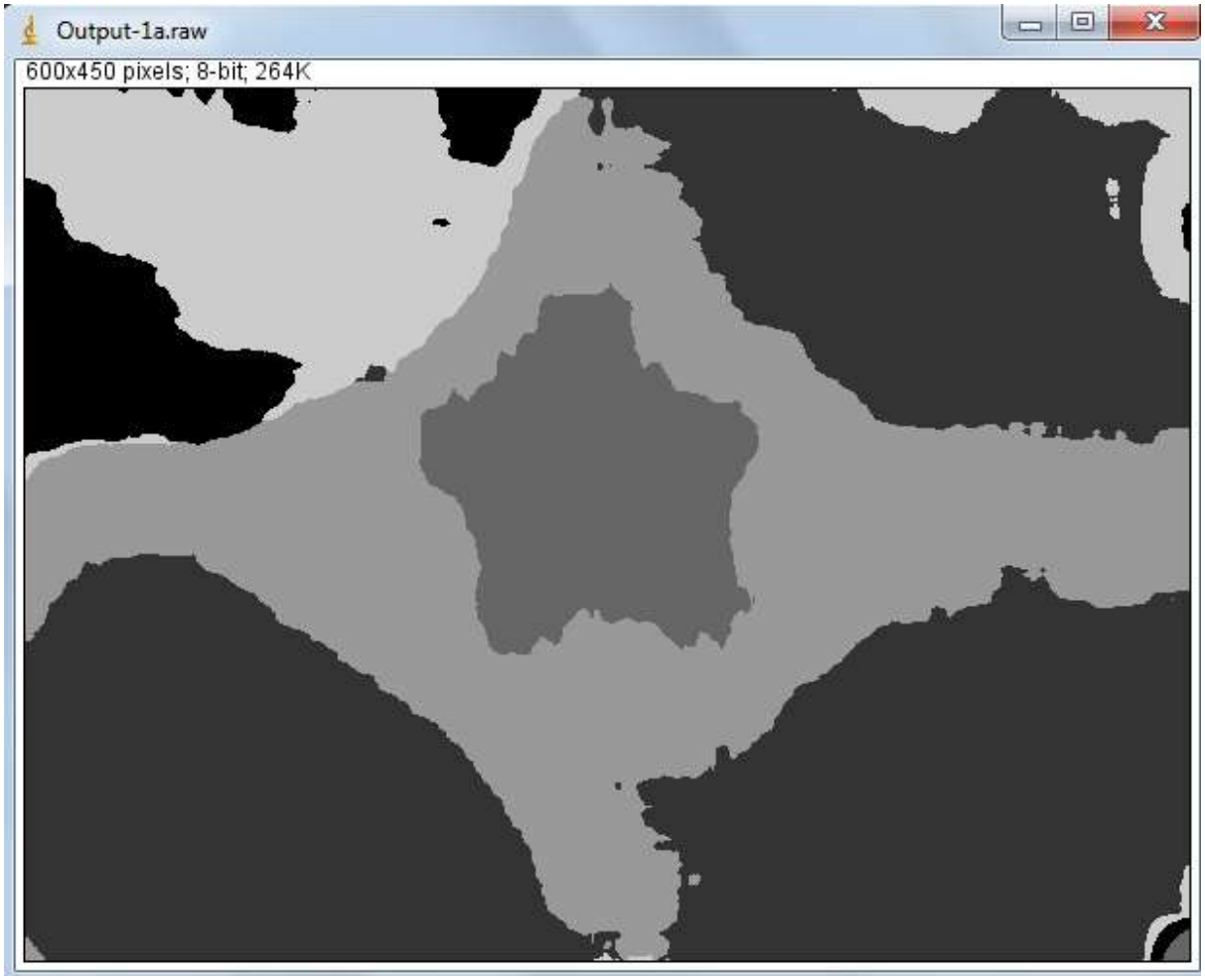

Figure 3.7: Texture segmentation using the 39x39 energy window and random initial centroids of Kmeans.

- Star in the centre classified perfectly.
- Bottom two corner textures still identified as same.
- Misclassification in the top left corner texture. Even though the region has uniform texture, there is error in labelling.
- The texture in the background is almost identified correctly and uniformly, there is no vertical and horizontal confusion as previous case.

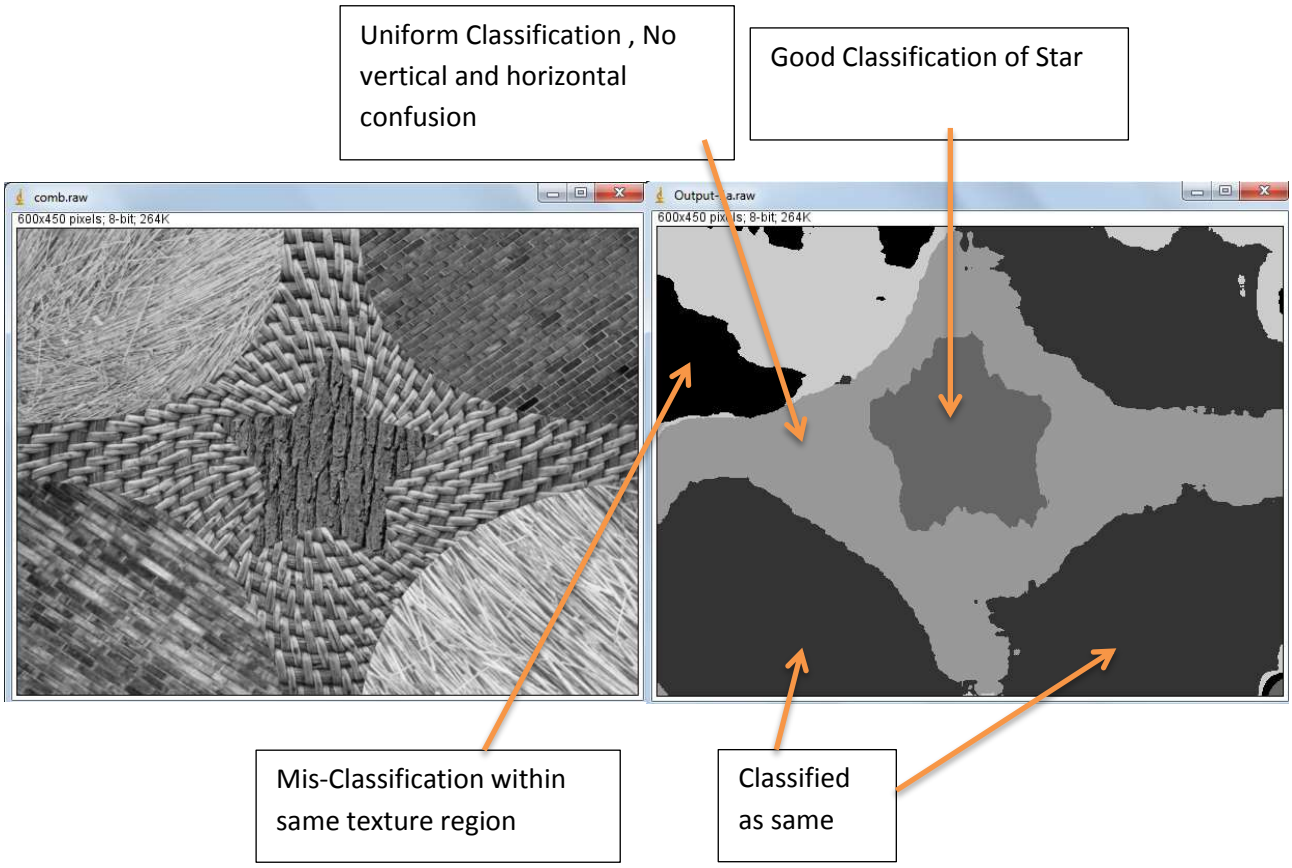

Figure 3.7: Texture segmentation using the 13x13 energy window and random initial centroids of Kmeans.

Conclusion: *What seems to be simple to our eyes (identifying textures), is difficult to emulate it from the point of view of image processing and computer vision.*

Manan Vyas
USC ID: 7483-8632-00 Email: mvyas@usc.edu

3. C: ADVANCED SEGMENTATION:

Task: To implement some post processing techinique to improve the segmentation result. Also perform Principal Component Analysis (PCA) and reduce the dimension of the feature vector and then perform the kmeans classification for segmentation of textures.

Code written in: All Processing in C++ / Kmeans PCA and output image generation in MATLAB.

Approach and Implementation:

Post Processing Technique: As advised in the homework "One simple way to improve the texture segmentation result is to exploit the fact that each texture is contained in one connected area". Therefore in the output of previous problem, I check the 4- connected adjacent pixel intensity of all the pixels in the whole image. If the right adjacent pixel has the same intensity. If 3 out of 4 have the same intensity and the middle has a different intensity than the majority intensity the center pixel intensity is changed to the majority pixel intensity.

Results and Discussion for post processing :

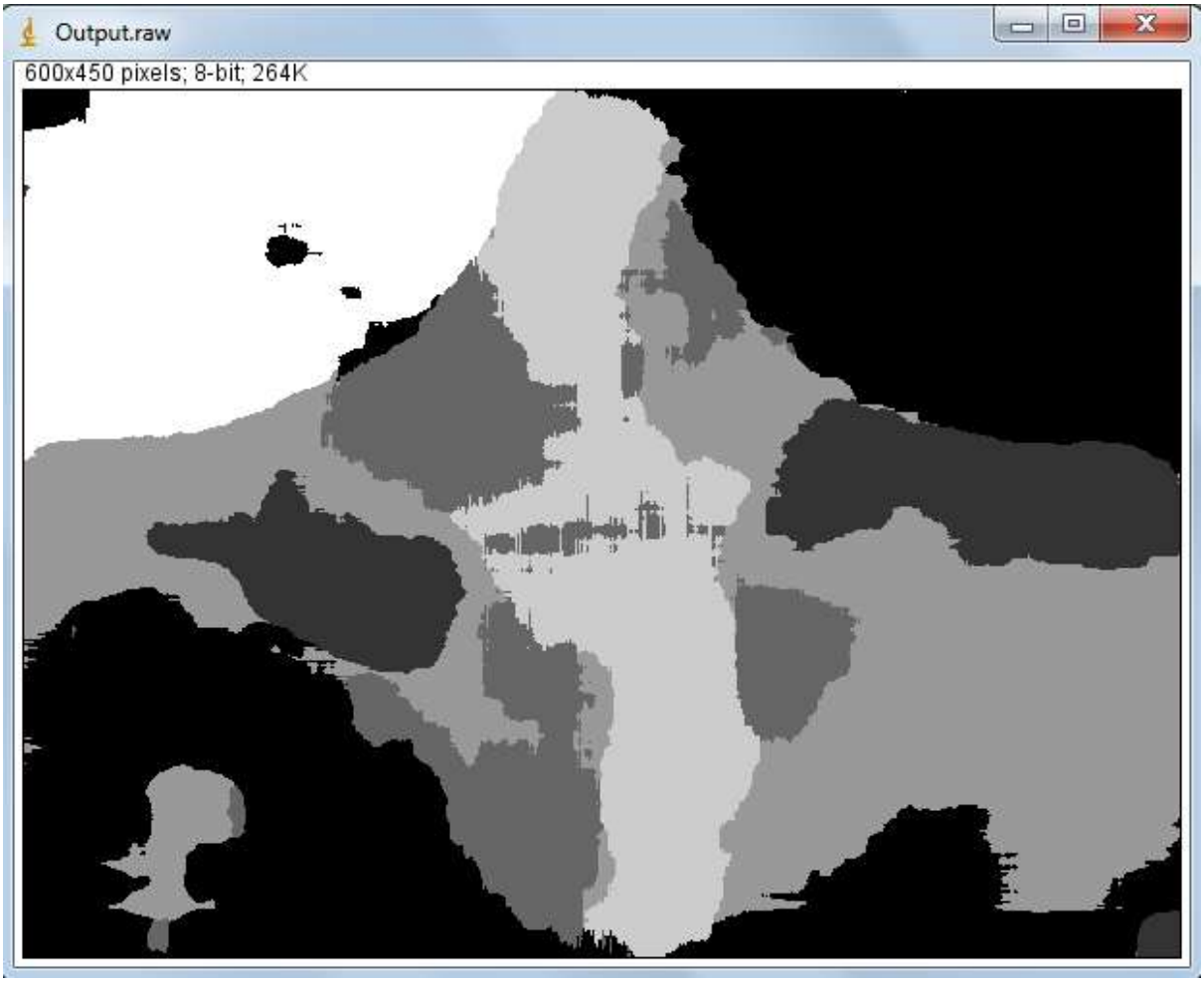

Figure 3.8: Texture Segmentation after post processing.

The output obtained through the post processing techniques fails to improve the results obtained in the previous problem. Hence, some other algorithm needs to be implemented. One another way to improve the result is using the connected component labelling on the output of previous problem and then distributing the intensities depending upon the labels assigned to the pixels. However, due to time constraint the proposed technique was not implemented.

Approach and Implementation Principal Component Analysis (PCA);

The initial Laws filtering processing for PCA is similar to that implemented in the previous problem 3.a. The only difference being that all the 25, 5x5 Laws filters are used.

All the 5 Law Filters are given by;

$$E5 = \frac{1}{6}[-1\ -2\ 0\ \ 2\ \ 1]\,; S5 = \frac{1}{4}[\,-1\ \ 0\ 2\ 0\ -1]; W5 = \frac{1}{6}[\,-1\ 2\ 0\ -2\ 1]$$

$$L5 = \frac{1}{10}[\,1\ \ 4\ 6\ 4\ 1]; W5 = \frac{1}{10}[\,1\ -4\ 6 - 4\ 1]$$

Once all 25 filters are obtained, then extract the 25 feature vectors corresponding to all the 25 Laws Filters.

Boundary Extend the 25 features vectors by appropriate amount and use the kernel approach to compute the energy of all the feature vectors. This process is similar to that implemented in the problem 3.B.

Normalize all the energies by the LL5 (Local average feature vector).

Apply PCA to the 25 normalized energy 'images' to reduce the dimensions of the feature data.

Apply K-Means to this reduced dimension data and perform the same steps which were performed in problem 3B.

For PCA the same 39x39 kernel was used for windowing and energy computation;

Results and Discussion for PCA:

The following result is obtained after PCA analysis on the comb.raw image;

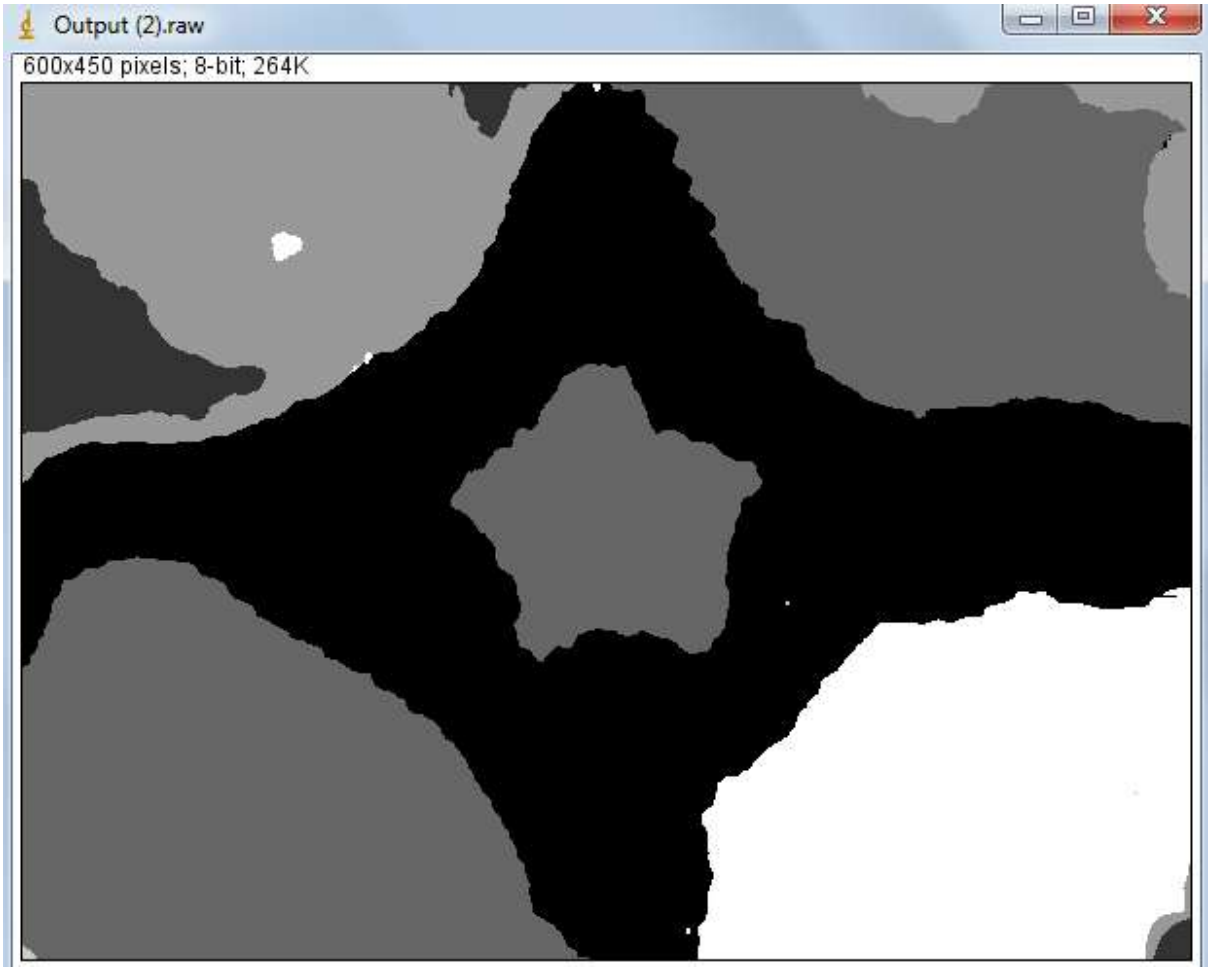

Figure 3.8: Texture Segmentation after PCA and Kmeans Clustering.

Thus the output obtained after PCA has definitely improved segmentation result. There is still some misclassification in top left corner, top right corner and bottom right corner. But overall the performance of the kmeans has improved after applying the PCA to the data before assigning labels to them and clustering them.

References:

[1]. EE569 HW#3 Fall 2013. Page 1. Figure #1.

[2]. (online) < http://upload.wikimedia.org/wikipedia/commons/a/a7/Bilin3.png > dated 11-03-2013

[3]. (online) < http://en.wikipedia.org/wiki/Barycentric_coordinate_system > dated 11-03-2013

[4] (online) <http://stackoverflow.com/questions/345187/math-mapping-numbers. > dated 11-03-2013

[5] EE569 HW#3 Fall 2013. Page 4. Figure #5

[6] EE569 HW#3 Fall 2013. Page 5. Figure #6.

[7] (online) < http://www.mathworks.com/help/stats/k-means-clustering.html> dated 11-03-2013.