EE569 Homework #3

Xiang Gao
9216-3489-10
xianggao@usc.edu

November 3, 2013

1 Spatial Warping

1.1 Motivation

Image warping is used to distort original image and then create desired special effect. During warping, pixels are mapped to different position without changing color.[1] In this task I will implement three image warping effects: diamond, pentagon and circle. These transformation should also satisfy the following requirements:(1)Boundaries should still lie on the boundaries.(2) Center is mapped to center.(3) The mapping should be reversible, i.e. it is a one-to-one mapping.

1.2 Approach

1.2.1 Warping to Diamond Shape

In this task, we want to transform the original image to a diamond-shaped image with affine transformation. The first step is to find triangles which are transformed separately with different transformation matrices. In the approach, the image is divided into 8 sub-triangles as is shown in figure 1.

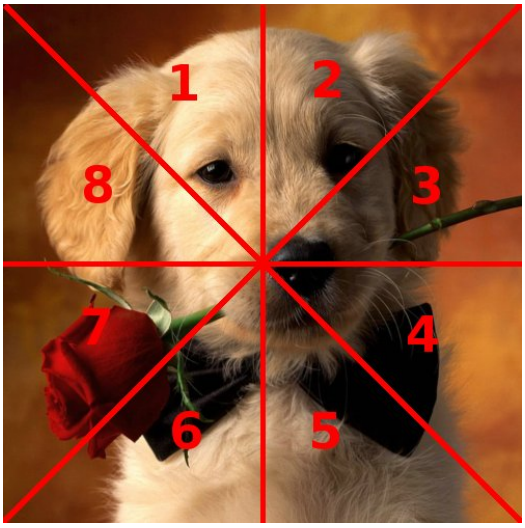


Figure 1: 8-triangles

For a affine transformation, we have the transformation matrix forms like:$T = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$ and the forward transformation(from original image to desired):$\begin{bmatrix} U \\ V \end{bmatrix} = T * \begin{bmatrix} X \\ Y \end{bmatrix}$. However, in image warping, we'd like to use backward transformation(to find original pixel) to avoid non-integer samples. That is $\begin{bmatrix} X \\ Y \end{bmatrix} = T_{inv} * \begin{bmatrix} U \\ V \end{bmatrix}$. Inverse transformation matrix used in each triangle is :

$$T_1 = \begin{bmatrix} 1 & 1 & -250 \\ 0 & 2 & -250 \\ 0 & 0 & 1 \end{bmatrix}, T_2 = \begin{bmatrix} 1 & -1 & 250 \\ 0 & 1.992 & -248 \\ 0 & 0 & 1 \end{bmatrix}, T_3 = \begin{bmatrix} 2 & 0 & -250 \\ -0.992 & 1 & 248 \\ 0 & 0 & 1 \end{bmatrix},$$
$$T_4 = \begin{bmatrix} 1.992 & 0 & -248 \\ 0.992 & 1 & -248 \\ 0 & 0 & 1 \end{bmatrix}, T_5 = \begin{bmatrix} 1 & 0.992 & -248 \\ 0 & 1.992 & -248 \\ 0 & 0 & 1 \end{bmatrix}, T_6 = \begin{bmatrix} 1 & -0.992 & 248 \\ 0 & 2 & -250 \\ 0 & 0 & 1 \end{bmatrix},$$
$$T_7 = \begin{bmatrix} 1.992 & 0 & -248 \\ -1 & 1 & -250 \\ 0 & 0 & 1 \end{bmatrix}, T_8 = \begin{bmatrix} 2 & 0 & -250 \\ 1 & 1 & -250 \\ 0 & 0 & 1 \end{bmatrix}$$

For each pixel in output image, we find its corresponding pixel in input image. Note that X, Y may not be integer. We use bilinear interpolation to solve this problem. $F(p^*, q^*) = (1-a)(1-b)F(p,q) + (1-a)bF(p,q+1) + a(1-b)F(p+1,q) + abF(p+1,q+1)$, where $a = p^* - p,\ b = q^* - q$.

However, the result shows that this affine method doesn't give us a satisfying output since the image is greatly distorted. I used another mapping

method to do the task, that is every horizontal line in the original image is mapped to a shorter line. The ratio is based on the position to make sure that we can get a diamond shape. This will give us less inner distortion. If we set the length of top and bottom of output image more than 1 pixel, the mapping still can be reversible.

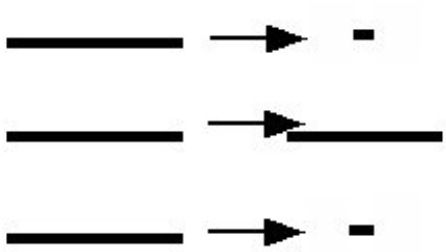


Figure 2: Method 2

1.2.2 Warping to Pentagon Shape

In doing this task, I use similar method as the first one. First we use a affine transformation method and use 7 triangles in this question.

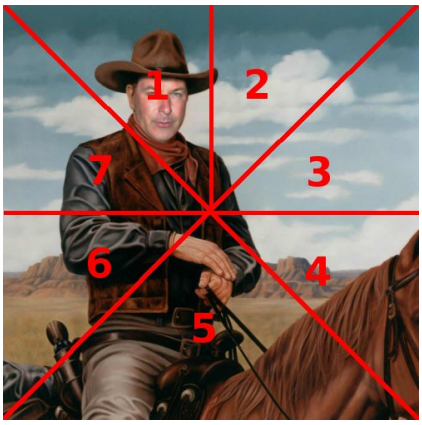


Figure 3: 7-triangles

The affine transformations are similar with Q1 and we just use different transformation matrix for each triangle. We should also use interpolation to deal with non-integer coordinates. Obviously, this method will also gives great distortion.

The second method I used is the same as previous task. The challenge is to find the region properly.

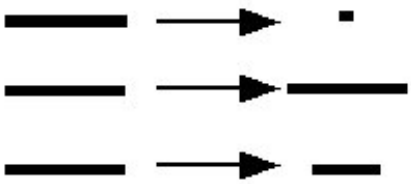


Figure 4: Method 2

1.2.3 Warping to Circle Shape

To deal with circle shape, I used polar coordinates and scale each line across center so that they have the same length.

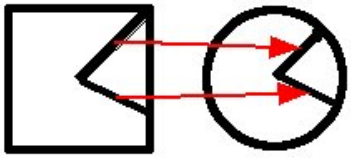
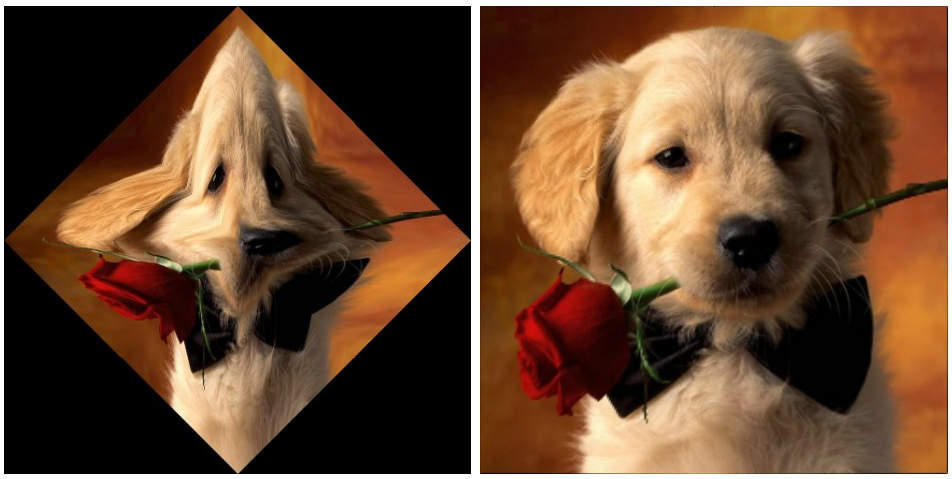


Figure 5: Transform

The above figure shows a transformation example. The line in the original image is shrinked to the line in the output image and these two lines share the same angle with horizontal coordinate.

1.3 Results and Discussion

1.3.1 Warping to Diamond Shape



(a) Affine transformation (b) Reverse

Figure 6: Warping to diamond shape

By using triangles, we can garantee the transformation is one-to-one and can be reversed. The result looks not satisfying, because the transformation is applied based on triangles. The diagonal is shrinked greatest and thus makes the distortion happen. The reversed image is a little bit blurry in some place, that's because when we are doing transformation, we used bilinear interpolation to deal with non-integer coordinates. The colors in the reversed image are no longer the original colors.

(a) direct transformation (b) Reverse

Figure 7: Warping to diamond shape

The warping result looks better if we use direct line mapping, where I map each line in the original image to the output image. This makes the distortion only happens in horizontal line. It should be noted that this transformation is one-to-one transformation if the first and last row are mapped to more than one point respectively. From the reverse result we can see the distortion after reversion is larger using this method than affine transformation, especially around the upper and lower boundary. That's because those lines are transformed to few pixels and thus lose information.

1.3.2 Warping to Pentagon Shape

(a) Affine transformation (b) Reverse

Figure 8: Warping to pentagon shape

The result has large distortion and is not satisfying.

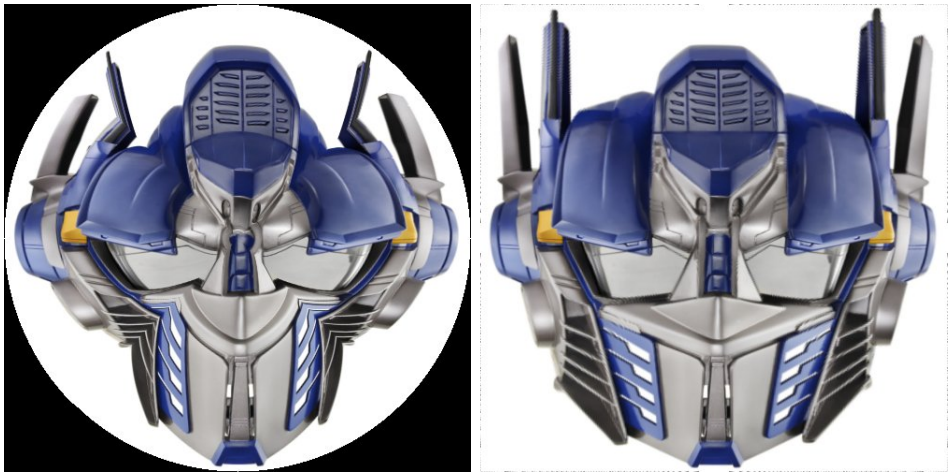
(a) Direct mapping (b) Reverse

Figure 9: Warping to pentagon shape

The warping result contains less distortion with direct mapping since it keeps relationship between pixels next to each other horizontally. The drawback of this method is around the top and bottom boundary in the

reverse image. Because it's a discrete image and because of the effect of interpolation, the output image is blurry.

1.3.3 Warping to Circle Shape



(a) Warping (b) Reverse warping

Figure 10: Warping to circle shape

Using this warping method, the diagonals are shrinked with a larger ratio than horizontal lines. This leads to distortions of the shape (create new angles from no where).

1.3.4 About distortions in reverse image

All the above reverse images contains distortions. These distortions are because the image is discrete. Suppose we are dealing with continuous images, the transformation transforms every pixel to a new position and for each pixel in the transformed image we can find its original pixel. Thus when doing reverse transformation, we can recover all pixels (if the warping transformation is reversible). However because the image is discrete, when we are doing warping, some pixels may not be transfromed to integer index. In that case we use interpolation method to get new pixel values. Another case is that when we transform a line to a shorter line, we will lose information because it's not continuous. That why the reverse image is not the same as the original one.

2 Perspective Transformation

2.1 Motivation

Perspective transformation is widely used in computer graphics and computer vision. In this task, I will implement a method to mimic a camera capturing a cube. Each surface of the cube is an image and we want to gain the image captured by the camera. Doing this we should use two kind of camera matrices: extrinsic matrix and intrinsic matrix.

2.2 Approach

Extrinsic matrix is used to convert world coordinates to camera coordinates. This matrix is a combination of rotation and translation. The basic approach to get image plane coordinates from world coordinates is through following equation:

$$w \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

After we get the image plane coordinates, again we have to convert it to image index based on pixel density.

2.2.1 Pre-processing

In this task, we map image to each surface of the cube. When user gives a location (world coordinates), program should feedback with RGB values. The method is simple. We first determine which surface is the "location" on and then get pixel index with world coordinates. Then search for that pixel in the image data.

2.2.2 Capturing 3D scene

In this task, we get a capture image from a 3-D cube. Since $w \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = K[R|t] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$, we can get the image plane coordinates and hence get image index. In forward mapping algorithm, we want to get image index from world

coordinates. Thus for each pixel in the total 5 images, we can get its world coordinates first and then get its corresponding capture image index based on camera matrix and pixel density. Because this is a 3D to 2D transform, several points on the cube may be mapped to the same pixel on the output image. We have to determine which pixel we want to use. I do this by using a depth image. This depth image stores the distance from pixel point to focal point. We want to choose the pixel which is neareat to the focal point because others would be hidden by this one.

The difficult of reverse mapping comes from the depth information(Z). When we are doing the forward mapping, we only care about x and y. However w contains information of depth which can not be gained from a single 2d image. Another problem is that even we know the depth of each point on the image, we lose many pixels that is hidden. Without knowing all the depth information, we can hardly recover the 3D coordinates.

2.2.3 Rotating Cube

To get captures of rotating cube, we can first get the rotated world coordinates of a point. Since the focal point and direction of camera doesn't change, we can use the same camera matrix above. For each point of the object, we first apply the rotation matrix to it, then use the camera matrix. Note that, in this part, the depth of pixel should be the depth after rotation. Thus, we get a video clip for each degree and combine all clips to get video.

2.3 Results and Discussion

2.3.1 Pre-processing

For example, the center pixel of image 3 has a location in the world coordinates:(0,1,0):

```
Process A begins...
(X Y Z):0 1 0
(R G B):123 31 34
Process A ends...
```

115,63,71	34,17,20	11,5,6	
132,70,78	45,22,23	36,4,5	
147,63,68	87,33,35	61,5,6	
126,60,63	105,35,38	42,9,9	
83,29,31	123,31,34	62,15,13	
84,10,12	117,29,32	124,29,26	
64,9,7	56,19,19	116,34,36	
115,90,51	87,66,38	128,69,48	1
219,178,103	219,179,100	165,107,64	1
150,90,52	185,132,76	105,33,20	1

Figure 11: location/intensity

This shows the mapping is correct.

2.3.2 Capturing 3D scene

With the specifications: $f = \sqrt{3}, w = 200, h = 200$, we can get the image plane coordinates. To get image index we also need pixel density. We change pixel density and compare the results.

(a) Pixel Density=100 (b) Pixel Density=200 (c) Pixel Density=400

Figure 12: Captures

We can see the effect of pixel density. If it's too small, the object appears to be too small and we can not see the details. On the other hand, if it's too big, the image (200x200) can't contain the object and we can only see part of the object. Pixel density around 200 gives us a good result. Note that this specification is corresponding to the focal position(5,5,5). If the focal point as well as the camera direction is fixed, object's camera coordinates are fixed. These coordinates are in meters(length metric). We should get corresponding pixel position based on both camera coordinates and pixel density.

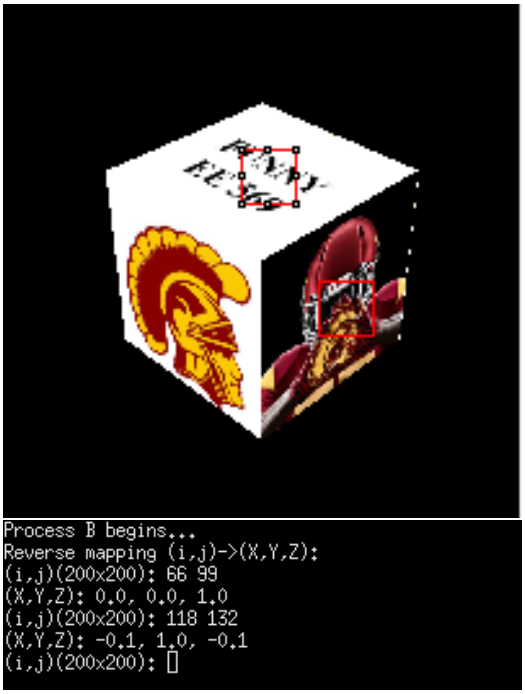


Figure 13: Reverse mapping

As I have mentioned in the approach part, when doing reverse mapping, we have to use depth information. Since $Z_c = depth$, we can get the camera coordinates of the pixel seleted: $\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}$. Considering the relationship of world coordinates and camera coordinates:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

To get X,Y,Z, we can simply use the inverse of the matrix [R—t]. That is:

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}$$

2.3.3 Rotating Cube

(a) $\theta = 1°$ (b) $\theta = 45°$ (c) $\theta = 90°$

Figure 14: Rotating Cube

3 Texture Analysis and Segmentation

3.1 Motivation

Laws' filter is a simple but useful method in texture analysis and segmentation. It applies a bank of filters on the neighborhood of a pixel. In frequency domain, these filters actually seperate frequency domain into several bands. Outputs of filters are used as features to help us determine or seperate textures. In this task, I will implement texture clustering and texture segmentation method. Texture clustering is that we want to seperate a bunch of texture images (each contains one texture) into several clusters so that images contain the same texture fall into the same cluster. Texture segmentation is that we want to seperate a single image into several parts so that each part contains one texture.

3.2 Approach

3.2.1 Texture Image Clustering

In this problem, I use nine 5x5 Laws' filter, thus we will get 9 features for each pixel. By averaging all pixels' features, we get a 9 dimension feature vector for each image. The dimension of feature space is thus 9. We use K-means method to cluster all feature vectors. It should be mentioned that it's not guaranteed that K-means will give us the optimal solution[2]. Result of K-means method depends a lot on the initial centers.

3.2.2 Texture Segmentation

In texture segmentation, we use one image and divide it into segments with different textures. The first step is to apply 9 Laws' filters to this image and compute energy of each pixel's neighborhood. This gives us a 9-D feature vector for each pixel. We still use K-means method to do clustering. Here we set the cluster number to be 6. After clustering, we set the graylevel to each pixel based on its label.

3.2.3 Advanced

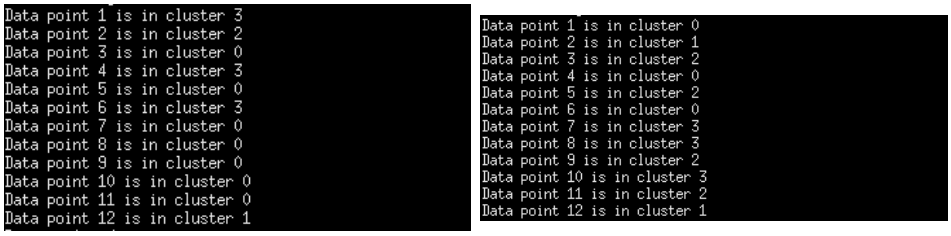
I tried to do some improvement to the result. (1) We can see that the difference within each feature is actually in different scale. Therefore,when seperating them in feature space with Euclid distance, features naturally have different weights. So I tried to normalize features. (2) Noting that each texture is contained in a connected region, I tried to add location information to the feature vector.

Second we want to use all twenty five 5x5 Laws Filters and use the Principal Component Analysis (PCA) method to reduce the dimensions of the feature vectors. More filters give us more detailed representation of features but the dimension of feature space grows. We employ PCA to extract those most significant features and construct a lower dimensional feature space.

3.3 Results and Discussion

3.3.1 Texture Image Clustering

Using random-selected initial centers, the clustering result is:



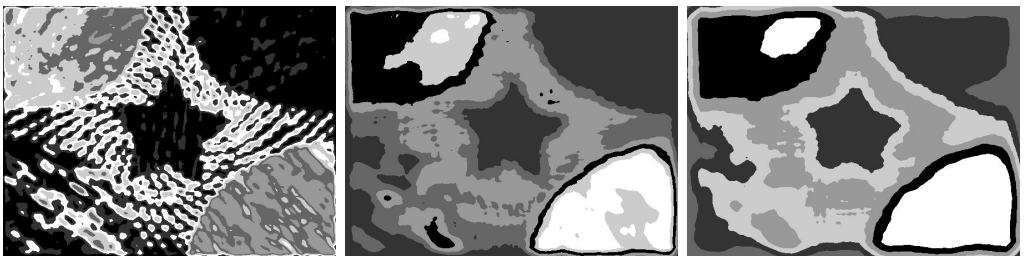
(a) Random-selected (b) Best

Figure 15: Clustering

The result is [3 2 0 3 0 3 0 0 0 0 0 1]. We can see "texture1""texture4""texture6"are correctly clustered. 3,5,7,8,10,11 are clustered as the same texture. One possible reason is that these textures are similar to each other and their feature vectors are close to each other in this 9 dimension feature space.

If I manually set initial center as "texture1""texture2""texture3""texture7", as the right figure shows, only one texture is mis-clustered. This is showing that clustering result of K-means method depends largely on the selection of initial centers.

3.3.2 Texture Segmentation



(a) Window size=11 (b) Window size=31 (c) Window size=51

Figure 16: Segmentation

We can see from the segmentation result that as the window size grows, the region becomes smoother but as a tradeoff the edge becomes thicker. Region is smoother because larger window contains more texture information. A larger window can not seperate textures correctly at edge parts.

3.3.3 Advanced

(1) Window Size = 31:

(a) Use normalized features

(b) Add location features

(c) Window Size=51

Figure 17: Segmentation: improvement

Doing these adjustment gives us a quite different result. The segments become much smoother and pixels near to each other tends to be segmented into same texture. Another fact is that after doing this, the edge effect is not

that significant as the window size grows. We can see textures in the four corners are segmented better while the drawbacks lie between the central star texture and the texture above and below it. They all contains a large amount of vertical components. Now I tried to solve the problem above. Since before normalization, the central star texture and the texture above and below it are seperated better. The problem is caused by normalization. Take a look at the feature vector we can see the values of first element are ususally large. After normalizing, we just give each feature the same weight in calculating different. Now I put large weight on the first element, this gives results below:

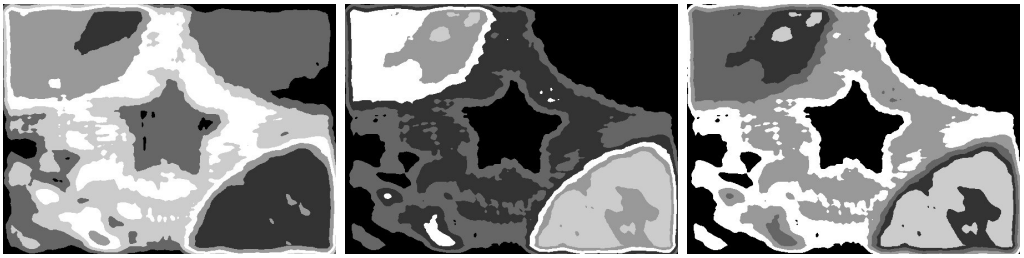
(a) Weight=5 (b) Weight=8 (c) Weight=10

(d) Window=51, weight=10

Figure 18: Different weights of 1st feature

Editing the weight of features gives us an improvement on the result.

(2)Use twenty five 5x5 Laws Filters and use the Principal Component Analysis (PCA) method to reduce the dimensions of the feature vectors. Window Size=31:



(a) Feature vector dimension: 1 (b) Feature vector dimension: 10 (c) Feature vector dimension: 15

Figure 19: Segmentation: 25 features with PCA

If we select enough features, higher dimension will not give us better performance but more computation load. An interesting fact is that even I reduced feature dimension to 1, the segmenting result is not very bad. That means centers of different textures in the feature space are acturally sparse. Even we project them to a line, they still locate far from each other.

References

[1] [Online]: `http://en.wikipedia.org/wiki/Image_warping`

[2] [Online]: `http://en.wikipedia.org/wiki/K-means_clustering`