EE 569: Homework #3

Issued: 10/13/2013 Due: 11:59PM, 11/3/2013

General Instructions:

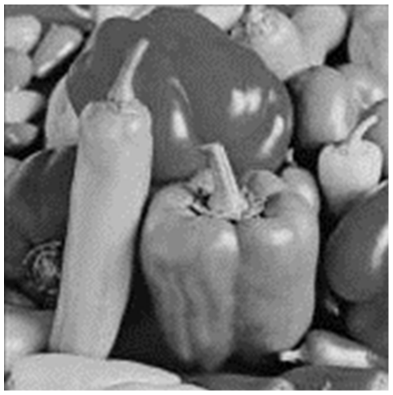
Please refer to Homework Guidelines and MATLAB Function Guidelines for more information about how to complete and submit the homework. Also, refer to the USC policy on academic integrity and penalties for cheating and plagiarism - these rules will be strictly enforced. **If you make any assumptions about a problem, please clearly state them in your report.**

Problem 1: Spatial Warping (30%)

In this problem, you will use the spatial warping technique to warp an image to another one of a desired shape.

1(a) Warping to Diamond Shape (Basic: 10%)

Design and implement a spatial warping technique that transforms an input square image into an output image of a diamond shape. An example is given in Figure 1.


(a) Original Image

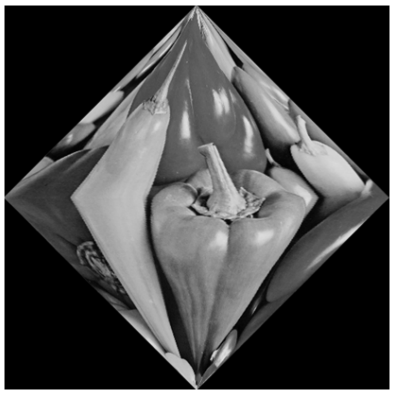
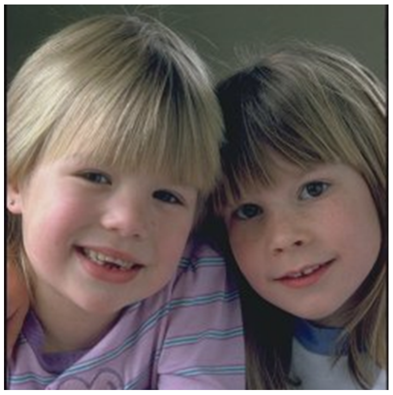

(b) Diamond-shaped Image

Figure 1: Warp the original image to a diamond-shaped image.

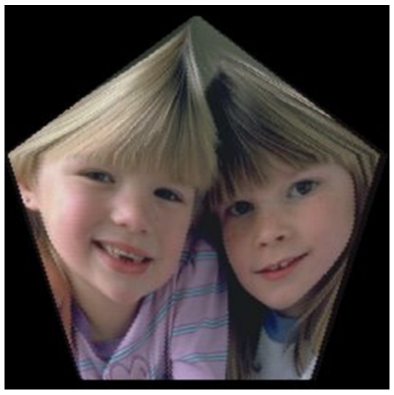
Apply the developed spatial warping algorithm to the 500x500 color image, "puppy.raw", as shown in Figure 4(a).

1(b) Warping to Pentagon Shape (Basic: 10%)

Design and implement a warping algorithm that maps a square image to a pentagon-shaped image as shown in Figure 2.



(a) Original Image



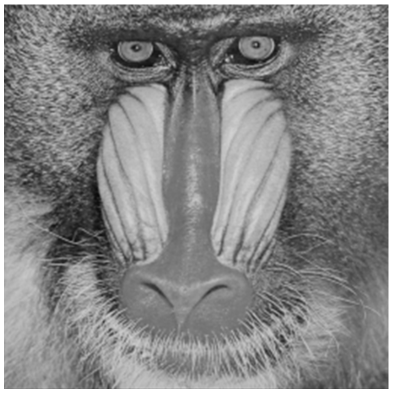
(b) Pentagon-shaped Image

Figure 2: Warp the original image to a pentagon-shaped image.

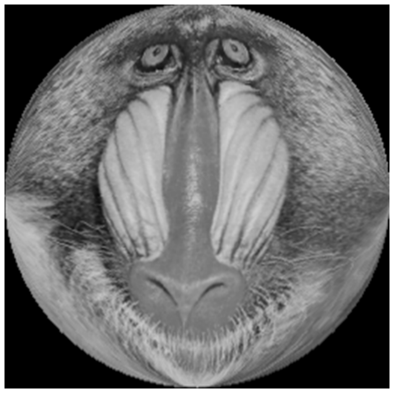
Apply the developed warping algorithm to the 500x500 color image, "cowboy.raw" as shown in Figure 4(b).

1(c) Warping to Circle Shape (Basic: 10%)

Lastly, design and implement a warping algorithm that maps a square image to a disk-shaped image as shown in Figure 3.



(a) Original Image



(b) Circle-shaped Image

Figure 3: Warp the original image to a disk-shaped image.

Apply the developed warping algorithm to the 500x500 color image, "transformmer.raw" as shown in Figure 4(c).

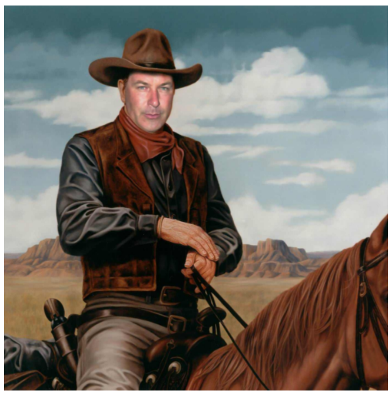
In parts (a)–(c), your three warped images should satisfy the following three requirements.

- Pixels that lie on boundaries of the square should still lie on the boundaries of the diamond, the pentagon, and the circle.
- The center of original images should be mapped to the center of warped images.
- The mapping should be reversible, *i.e.* it is a one-to-one mapping.

For Parts (a)–(c), please first describe your approach as clearly as possible and show the resulting images. Next, apply the reverse spatial warping to each warped image to recover its original image. Compare the

Prob 1(c) continued on next page...

(a) "puppy.raw"


(b) "cowboy.raw"

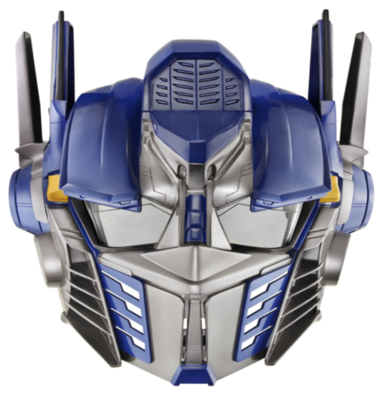

(c) "transformmer.raw"

Figure 4: Source images to be warped

recovered square image with the original square image. Is there any difference between two images? If any, explain sources of distortion in detail.

Problem 2: Perspective Transformation & Imaging Geometry (30% + 10% Bonus)

In this problem, you will use the perpective transformation and imaging geometry to capture a scene in the 3D world cordinates system and project it onto a 2D image plane.

A cube in the 3D world coordinates system is shown in Figure 5, where its center is located at the origin and each side is parallel to one of the XY-, XZ- and YZ-planes. One viewing camera has its lens centered at (5,5,5), viewing angle along the direction of (-1,-1,-1), and focal lenth f. Please show the image observed at the image plane in this problem.

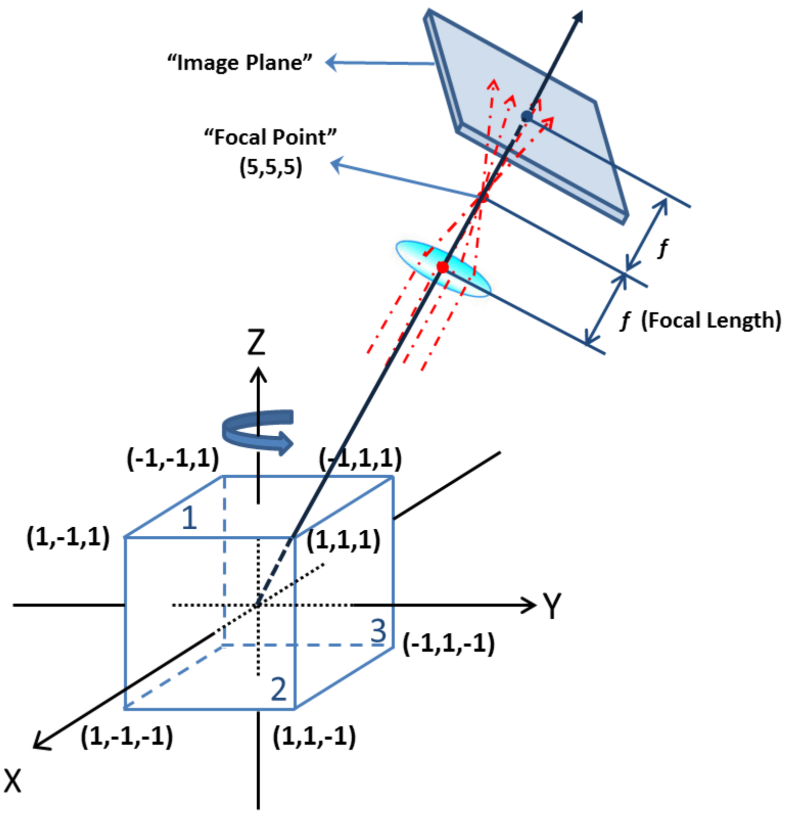


Figure 5: 3D Cube in the World Geometry.

To conduct this task, you need to understand and implement two coordinates transform matrices as given below. Generally, the image plane coordinates, $[x, \quad y]^T$, are related to the 3D world coordiates $[X, \quad Y, \quad Z]^T$ via

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

where K and $[R|t]$ are called the *intrinsic* camera matrix and the *extrinsic* camera matrix, respectively, and w is a scaling factor. Matrices K and $[R|t]$ are given below.

1. **Extrinsic Camera Matrix: from the World Coordinates to the Camera Coordinates**

To map the location of a point in the world coordinates to that of the image plane, we need to change it to the camera coordinates system (see Figure 6) as an intermediate step, and $[R|t]$ in Eq. (1) is used to accomplish this task.

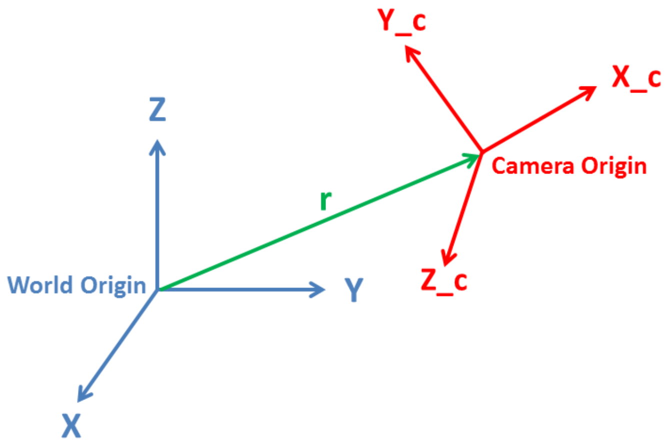


Figure 6: Relationship between the world coordinates and the camera coordinates.

It can be rewritten as

$$[R|t] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} = \begin{bmatrix} X_c^X & X_c^Y & X_c^Z & -\mathbf{r}\cdot\mathbf{X_c} \\ Y_c^X & Y_c^Y & Y_c^Z & -\mathbf{r}\cdot\mathbf{Y_c} \\ Z_c^X & Z_c^Y & Z_c^Z & -\mathbf{r}\cdot\mathbf{Z_c} \end{bmatrix},$$

where $\mathbf{r}$ is the vector from the origin of the world coordinates to the origin of the camera coordinates, and $\mathbf{X_c} = (X_c^X, X_c^Y, X_c^Z)^T, \mathbf{Y_c} = (Y_c^X, Y_c^Y, Y_c^Z)^T, \mathbf{Z_c} = (Z_c^X, Z_c^Y, Z_c^Z)^T$ are the unit vectors of the camera coordinates system with respect to the world coordinates system, respectively. Based on the configuration in Figure 5, we have

$$\mathbf{Z_c} = (-\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}})^T,$$

since the image plane faces the origin of the world coordinates and $\mathbf{r}= (5, 5, 5)$. Without loss of generality, we choose the following initial values for $\mathbf{X_c}$ and $\mathbf{Y_c}$ in matrix $[R|t]$:

$$\mathbf{X_c} = (-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0)^T, \qquad \mathbf{Y_c} = (\frac{1}{\sqrt{6}}, \frac{1}{\sqrt{6}}, -\frac{2}{\sqrt{6}})^T.$$

2. **Intrinsic Camera Matrix: From the Camera Coordinates to the Image Plane Coordinates**

Next, we map the location of a point in the camera coordinates to that of the image plane. In Figure 7, X_c, Y_c and Z_c are three axes of the camera coordinates, and $\mathbf{P}$ and $\mathbf{P}'$ are points in the camera coordinates and its projective point onto the image plane, respectively. Matrix K in Eq. (1) can be derived using the perspective image transformation as

$$\omega \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = K \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} f\cdot X_c + c_x\cdot Z_c \\ f\cdot Y_c + c_y\cdot Z_c \\ Z_c \end{bmatrix}$$

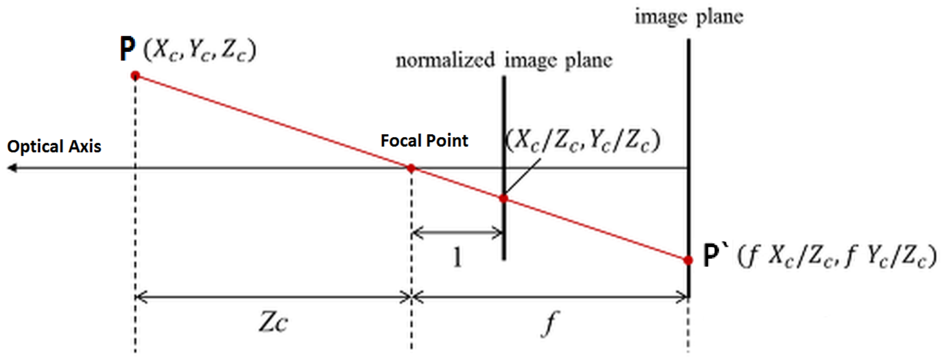


Figure 7: Relationship between the camera coordinates and the image plane coordinates.

where (c_x, c_y) is the image coordinates of the intersecting point between the optical axis and the image plan.

After the simplification, we have

$$x = \frac{f \cdot X_c + c_x \cdot Z_c}{Z_c} = f \cdot \frac{X_c}{Z_c} + c_x, \qquad y = \frac{f \cdot Y_c + c_y \cdot Z_c}{Z_c} = f \cdot \frac{Y_c}{Z_c} + c_y.$$

Note that $c_x = \frac{ImageWidth}{2}$ and $c_y = \frac{ImageHight}{2}$ based on the image coordinates convention. Also, by shiting the origin of the image plane to (c_x, c_y), one can verify that

$$x : X_c = f : Z_c \leftrightarrow x = f \cdot \frac{X_c}{Z_c}, \qquad y : Y_c = f : Z_c \leftrightarrow y = f \cdot \frac{Y_c}{Z_c}$$

as shown in Figure 7.

2(a) Pre-processing (Basic: 10%)

Figure 8 shows five 200x200 images (“image1.raw”−“image5.raw”) which will be placed on five faces of the cube as shown in Figure 8(f) (you can ignore the bottom face in this problem). With the configuration in Figure 5, the camera can observe images on face no. 1, no. 2 and no. 3.
Write a program to generate the location/intensity information of the five images by assuming the unit length of the world coordinates is 0.01. (because the resolution of the image is 200x200 and the side length of the cubic is equal to 2, we have $\frac{2}{200} = 0.01$). Explain the basic idea of your implementation. Please specify the input and the output of your program clearly.

2(b) Capturing 3D scene (Advanced: 20%)

In this part, you need to display the cube image on the image plane captured by the camera. By following the aforementioned steps, you can determine K and $[R|t]$.
Please first implement the forward mapping algorithm (from the world coordinates → the camera coordinates) with the following three fixed parameters:

- focal length $f = \sqrt{3}$,
- image width $w = 200$ pixels,
- image height $h = 200$ pixels.

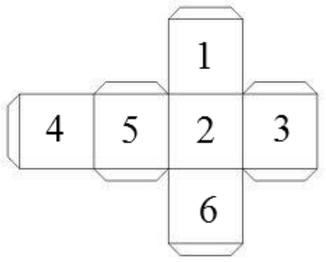
(a) "image1.raw"

(b) "image2.raw"

(c) "image3.raw"

(d) "image4.raw"

(e) "image5.raw"



(f) Unfolded Cube Image

Figure 8: Mapping five images to five faces of the cube.

There is one more parameter called the pixel density (namely, the number of pixels per unit length) for you to determine. Note that a poor choice of pixel density will result in a projected image either too small or too big. Please specify one good value of pixel density and provide the reason for your choice. Discuss the resulting projected image. If there are undesirable artifacts, explain the source of the distortions. Is there any way to improve the result? If so, provide your own algorithm (you do not need to implement it).
Then, with the set of parameters obtained above, please implement the reverse mapping (from the camera coordinates → the world coordinates). What are the challenges in the reverse mapping?

2(c) Rotating Cube (Bonus: 10%)

Now, the cube rotates along the Z-axis with an anti-clock direction as shown in Figure 5. Please capture the cube when the rotation angle is equal to $\theta°$, where $\theta = 1, 2, 3, ..., 90$. Then, please make a video clip that consists of the 90 images. To do this, you need to use the rotational transform with respect to a Z-axis of the world coordinates as follows.

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Problem 3: Texture Analysis and Segmentation using Laws Filters (40%)

In this problem, you will implement texture analysis and segmentation algorithms based on the 3x3 Laws filters discussed in the lecture or the 5x5 Laws filters constructed by the tensor product of the five 1D kernels in Table 1:

Table 1: 1D Kernels for 5x5 Laws Filters

NAME	KERNEL
L5 (Level)	[1 4 6 4 1]
E5 (Edge)	[-1 -2 0 2 1]
S5 (Spot)	[-1 0 2 0 -1]
W5 (Wave)	[-1 2 0 -2 1]
R5 (Ripple)	[1 -4 6 -4 1]

3(a) Texture Image Clustering (Basic: 12%)

In this part, please construct nine 5x5 Laws Filter using the following three filters:

$$E5 = \frac{1}{6}\begin{bmatrix} -1 & -2 & 0 & 2 & 1 \end{bmatrix}, \quad S5 = \frac{1}{4}\begin{bmatrix} -1 & 0 & 2 & 0 & -1 \end{bmatrix}, \quad W5 = \frac{1}{6}\begin{bmatrix} -1 & 2 & 0 & -2 & 1 \end{bmatrix},$$

Twelve sample images, "texture1.raw" to "texture12.raw" (size 128x128) are provided for texture image clustering. Samples of these images are shown in Figure 9.

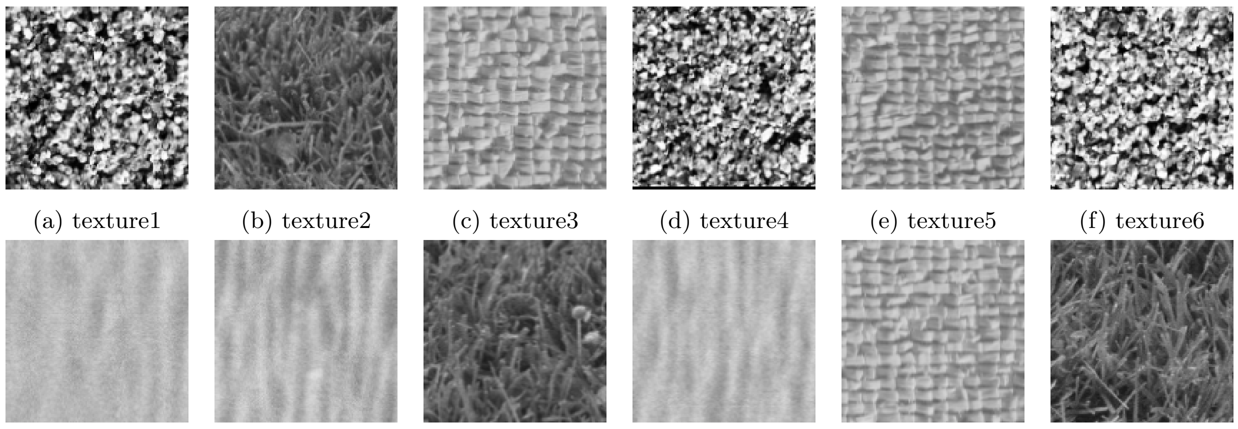

(a) texture1 (b) texture2 (c) texture3 (d) texture4 (e) texture5 (f) texture6
(g) texture7 (h) texture8 (i) texture9 (j) texture10 (k) texture11 (l) texture12

Figure 9: Samples of texture images.

Please cluster them into four different texture types with the following steps.

1. Feature Extraction: Use the nine 5x5 Laws Filters to extract feature vectors from each pixel in the image (use appropriate boundary extensions).

2. Feature Averaging: Average the feature vectors from the entire image, which should provide a 9-dimensional feature vector for each image.

3. Clustering: Use the K-means algorithm to perform image clustering using the feature vectors obtained in (2).

Report your results and compare them with the reality (by checking the texture images by eyes). Discuss any discrepancy.

Problem 3 continued on next page...

3(b) Texture Segmentation (Advanced: 18%)

In this part, apply the 3x3 Laws Filters to do texture segmentation for the image shown in Figure 10. Here, we use a different procedure from (a).

1. Apply all 3x3=9 convolution kernels to the input image and get 9 gray–scale images.
2. Energy computation: Use a windowed approach to computer the energy measure for each pixel in the results of (1). You should try different window sizes. The results are 9 feature images.
3. Normalization: Except for $L3^T L3$, all other kernels have zero-mean. Note that $L3^T L3$ is typically not a useful feature, and can be used for the normalization of all other features.
4. Segmentation: Use the k–means algorithm to perform segmentation on the composite texture image given in Figure 10. If there are K textures in the image, your output image will be of K gray levels, with each level represents one type of texture. For example, there are six textures in Figure 10, you can use six gray levels (0, 51, 102, 153, 204, 255) to denote six segmented regions in the output image.

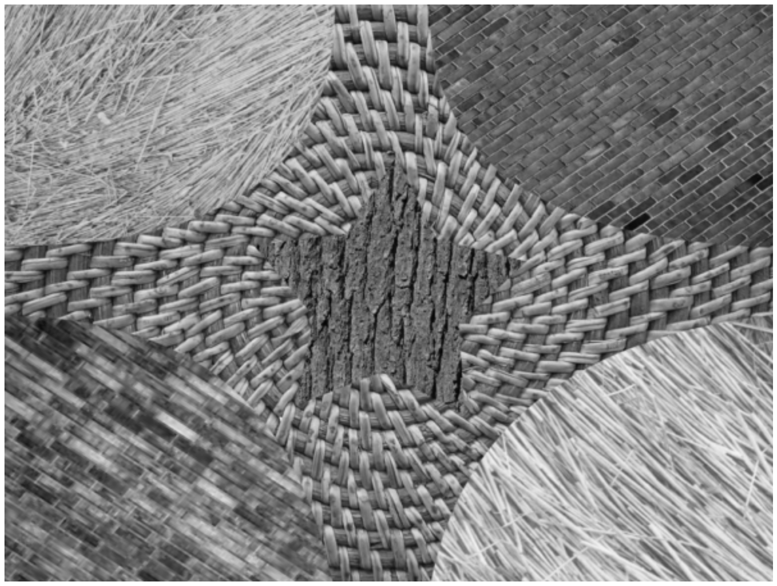


Figure 10: Composite Texture Image.

The input of your program consists of the input composite texture image and the K value. Please report your segmentation result. You can use provided C code, 'kmeans.c', to conduct K-means algorithm.

3(c) Advanced (10%)

1. Post-processing for Segmentation (3 points): Please develop some post-processing techniques to improve your segmentation results in Problem 3(b).
   [Hint] One simple way to improve the texture segmentation result is to exploit the fact that each texture is contained in one connected area.
2. Feature Reduction (7 points): You may adopt all twenty five 5x5 Laws Filters and use the Principal Component Analysis (PCA) method to reduce the dimensions of the feature vectors. Use these dimension reduced features to do texture segmentation of Figure 10. You may use built-in C/Matlab functions of PCA.

Appendix:

Problem 1: Spatial Warping

puppy.raw	500 × 500 × 3, 8-bit depth, inteleaved RGB image
cowboy.raw	500 × 500 × 3, 8-bit depth, inteleaved RGB image
transformmer.raw	500 × 500 × 3, 8-bit depth, inteleaved RGB image

Problem 2: Perspective Transformation & Imaging Geometry

image1–5.raw	200 × 200 × 3, 8-bit depth, inteleaved RGB image

Problem 3: Texture Analysis and Segmentation using Laws Filters

texture1–10.raw	128 × 128, 8-bit depth, grayscale image
comb.raw	600 × 450, 8-bit depth, grayscale image

Sample Codes Provided

readraw.m	MATLAB source code provided to read in grayscale raw image files
writeraw.m	MATLAB source code provided to output grayscale raw image files
kmeans.c	C code provided for k-means algorithm

Reference Images

All images used in this homework were downloaded either from Google image search or USC-SIPI image database.