HOMEWORK #3

University of Southern California

Lecturer: C.-C.JayKuo | Issued: 2/13/2019 | Due:3/3/2019

Report

1 Problem 1: Geometric Modification

1.1 Geometric Transformation

1.1.1 Abstract and Motivation

It is important to manipulate the image with transition, scaling, rotation method. In this section, I will recover the complete image with the three cut image patches.

1.1.2 Approach and Procedures

1. **Detect the four corners of each sub-image**
   By observing the image, we could conclude that the corner of the image should satisified some special pattern. For instance, the pixel on the left vertix must only have pixel values equal to 255 on three directions (right, up and down) while itself not equal to 255. So does the other vertices. I use this porperty to detect the vertices. For the rotated sub-image, the edges are kind of blury and the vertice is not sharp. I choose the pixel that *farest* from the image center as the vertices. To seperate from the similar corner pattern in the image, I choose the adjacent pixels with distance equal to 4 from the center pixels as the measurement.

2. **Detect the four corners of the hole**
   The edge of the hole in the *lighthouse.raw* is sharp and easy to be detect. First binary the image into black and white. Then design a patter that same as the corner pattern to scan the entire image to find the same pattern. The pattern are shown as follows.

$$left \quad up = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix} \tag{1}$$

$$right \quad down = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{2}$$

$$left \quad down = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{3}$$

$$right \quad up = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{pmatrix} \tag{4}$$

After finding the hole vertices, the width and height of the hold could be easily caculated.

3. **Compute the parameters of each sub-image**
After finding the four vertices, weight, height, central point and rotation orientation could be caculated.

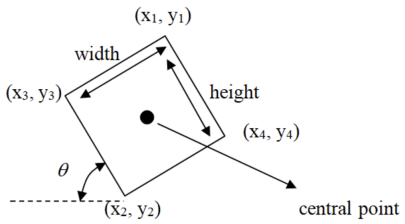

Figure 1: sub-image pattern

Set the four vertices index as (x_1, y_1), $(x_2, y_)$, (x_3, y_3), (x_4, y_4).

$$width = \sqrt{(x_1 - x_3)^2 + (y_1 - y_3)^2} = \sqrt{(x_4 - x_2)^2 + (y_4 - y_2)^2} \tag{5}$$

$$height = \sqrt{(x_1 - x_4)^2 + (y_1 - y_4)^2} = \sqrt{(x_3 - x_2)^2 + (y_3 - y_2)^2} \tag{6}$$

$$\theta = \tan^{-1}(\frac{y_2 - y_3}{x_2 - x_3}) \tag{7}$$

$$centeral \quad point = (\frac{x_3 + x_4}{2}, \frac{y_3 + y_4}{2}) or (\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2}) \tag{8}$$

To ensure the accuracy of each parameters, I caculate them with different vertices index and take average.

4. **Geometry manipulation**
In order to ensure that every pixels in the output image could be mapped from the input image. Instead of using forward mapping, we use inverse mapping from the output pixel (int) index to found the corresponding input pixel index (double). And estimate the pixel value by bilinear interpolation. First of all, the image index should be convert to cartisian coordinate by equation 9.

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & -0.5 \\ -1 & 0 & J+0.5 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} i \\ j \\ 1 \end{pmatrix} \tag{9}$$

Where J = height of the image. The geometry manipulation is done by matrix operation.

(a) **Rotation**

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \tag{10}$$

where θ is caculated by equation 7.
However, the rotation of the image is respect to the central of the entire image. To recover the original sub-image, it should be rotated above the central point of the sub-image patch. Thus, before rotating the image, the image should be shift to the central point (x_c, y_c). On the other hand, after image rotation, it should be shift back to the original position.
Thus, the matrix of rotating image by θ above (x_c, y_c) is:

$$\boldsymbol{R} = \begin{pmatrix} 1 & 0 & x_c \\ 0 & 1 & y_c \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -x_c \\ 0 & 1 & -y_c \\ 0 & 0 & 1 \end{pmatrix} \tag{11}$$

(b) **Scaling**

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \tag{12}$$

where $S_x = hole\_width/width$, $S_y = hole\_height/height$. Same as rotation, in order to scale the image above the central point of the sub-image patch, it should be shift to it and shift back after scaling.

$$\boldsymbol{S} = \begin{pmatrix} 1 & 0 & x_c \\ 0 & 1 & y_c \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -x_c \\ 0 & 1 & -y_c \\ 0 & 0 & 1 \end{pmatrix} \tag{13}$$

(c) **transition**

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \tag{14}$$

where $\begin{pmatrix} t_x \\ t_y \end{pmatrix} = \begin{pmatrix} X_c \\ Y_c \end{pmatrix} - \begin{pmatrix} x_c \\ y_c \end{pmatrix}$. (X_c, Y_c) is the central point of the corrsponding hole.

After finding the three forward mapping matrx, I multiply them and caculate the inverse matrix.

$$\boldsymbol{I} = \boldsymbol{RST}^{-1} \tag{15}$$

Use the inverse matrix to find the corresponding catisian coordinate of the input image index.

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \boldsymbol{I} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{16}$$

With the catisian coordinate, caculate the input image index.

$$\begin{pmatrix} i \\ j \\ 1 \end{pmatrix} = \begin{pmatrix} 0 & -1 & J + 0.5 \\ 1 & 0 & 0.5 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{17}$$

5. **Bilinear interpolation**
The input image index caculate by equation 18 is usually not a integer. To get the result image with less artifact, we use bilinear interpolation to estimate the pixel value of the double data type index.

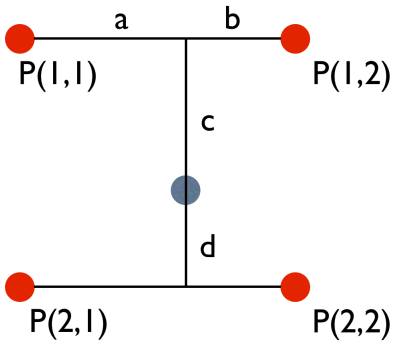

Figure 2: Bilinear Interpolation

$$P(i,j) = \frac{d}{c+d} \times \left(\frac{b}{a+b} \times P(1,1) + \frac{a}{a+b} \times P(1,2)\right) + \frac{c}{c+d} \times \left(\frac{b}{a+b} \times P(2,1) + \frac{a}{a+b} \times P(2,2)\right) \tag{18}$$

Where a, b, c, d represent the distance and $P(i,j)$ denote the pixel value. If the estimated pixel index fall at the edges (or the distance to edges is less than 1), we only use two pixels perpendicular to the edges to estimate the pixel value.

6. **Find the right direction**
   The sub-image patches are square so I cannot figure out which side should go up and which should go down. Thus, I caculated the average pixel value along a edge of sub-image pattern. Then I caculate average pixle values along the four edges of the corresponding hole. By comparing the average pixel value of the sub-image pattern and that of the hole edges, I pick up the one has least mean square error. I could finally settle down the correct direction of the sub-image.

7. **Eliminate the artifacts**
   Although the image pattern could be put into the hole correctly, there will still be some "white line" along the edges. It has similar property as the "pepper noise". To eliminate this "white line", I choose to use median filter.

1.1.3 Experimental Result

The vertice of *lighthouse1* is (118, 221), (182, 78), (80, 116), (223, 182)
The vertice of *lighthouse2* is (116, 218), (100, 15), (8, 125), (212, 108)
The vertice of *lighthouse3* is (43, 251), (211, 3), (3, 43), (251, 211)

Table 1: Parameters of Each Sub-image

	width	height	theta	central point
lighthouse1	110.4287	111.7273	0.35875 - $\pi/2$	[150.75, 149.25]
lighthouse2	144.7007	144.0509	0.8663 + π	[109., 116.5]
lighthouse3	211.8112	211.8112	0.1899	[127., 127.]

```
rotation_mtx:
 [[  0.35110873   0.93633469  -0.90959153]
 [ -0.93633469   0.35110873 208.18748102]
 [  0.           0.           1.        ]]
scaling_mtx:
 [[  1.44889839   0.         -67.22253451]
 [  0.           1.43205786 -45.25806097]
 [  0.           0.           1.        ]]
transition_mtx:
 [[  1.     0.    -7.25]
 [  0.     1.   169.75]
 [  0.     0.     1.  ]]
```

Figure 3: *lighthouse1.raw* forward mapping matrix

The coorfinates of the holes:

left hole : right up: (157, 221), left up: (157, 62), right down: (437, 316), left down: (278, 316). top hole : right up: (31, 437), left up: (31, 278), right down: (190, 437), left down: (190, 278). bottom-right hole : right up: (328, 485), left up: (328, 326), right down: (487, 485), left down: (487, 326).

The result image denoised by median filter:

```
rotation_mtx:
 [[ -0.64764417   0.76194293  81.14972953]
 [ -0.76194293  -0.64764417 330.52719393]
 [  0.           0.           1.        ]]
scaling_mtx:
 [[  1.10573014   0.         -12.37042683]
 [  0.           1.11071835 -16.22023812]
 [  0.           0.           1.        ]]
transition_mtx:
 [[  1.    0.  241. ]
 [  0.    1.  254.5]
 [  0.    0.    1. ]]
```

Figure 4: *lighthouse2.raw* forward mapping matrix

```
rotation_mtx:
 [[  0.98200645  -0.18884739  26.56106809]
 [  0.18884739   0.98200645 -21.76587113]
 [  0.           0.           1.        ]]
scaling_mtx:
 [[ 0.75538957  0.          31.18782924]
 [ 0.          0.75538957 31.43243966]
 [ 0.          0.          1.        ]]
transition_mtx:
 [[  1.   0. 279.]
 [  0.   1. -25.]
 [  0.   0.   1.]]
```

Figure 5: *lighthouse3.raw* forward mapping matrix

Figure 6: Filled *lighthouse.raw*

Figure 7: *lighthouse.raw* by median filter

1.1.4 Discussion

By observing the result image, we could see that the recovered sub-image is stll blury. That might be caused by the error in estimated parameters. Bilinear interpolation is used to generate the pixel value and it works well. Median filter efficiently remove the white line along the hole edges. However, there is still some artifacts remains.

1.2 Spatial Warping

1.2.1 Abstract and Motivation

It is useful to use spatial warping technique to creat some art effect. In this section, I first find the warping matrix of Figure 2 in homework 3. And I use that warping matrix to generate a similar matrix.

1.2.2 Approach and Procedures

By observing the Figure 2, it is easy to conclude that the the warping goes to four direction, which is non-linear operation. It is universally acknowledge that the matrix can only perform linear operation. Thus, we should cut the image into 4 triangles along the diagnoal. Then, select 6 control points for each triangle from the input image and output image. First cut the image into 4 parts as Figure 8. Then choose the control point as Figure 9:

Figure 8: warping

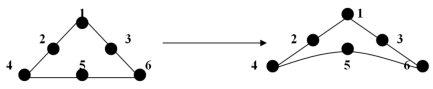

Figure 9: control points

Use the 6 control points to caculate the parameters matrix by equation 19.

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} a_0 & a_1 & a_2 & a_3 & a_4 & a_5 \\ b_1 & b_2 & b_3 & b_4 & b_5 & b_6 \end{pmatrix} \begin{pmatrix} 1 \\ x \\ y \\ x^2 \\ xy \\ y^2 \end{pmatrix} \tag{19}$$

Where (u, v) is the input image index under the cartisian coordinate, while (x, y) is the output image index under the cartisian coordinate.

Then apply the parameters matrix to *hat.raw* on the same triangle part. First of all, convert the image index into cartisian coordinate. Then solve the parameter matrix. That is,

$$\boldsymbol{U} = \boldsymbol{A}\boldsymbol{X} \tag{20}$$

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{X}^{-1} \tag{21}$$

Where, $\boldsymbol{U}$ is the 2×6 matrix that composed by input cartisian coordinate, $\boldsymbol{X}$ is the 6×6 matrix that composed by input cartisian coordinate. The reason that I pick 6 control points is that to make $\boldsymbol{X}$ a square matrix that have the inverse matrix. The result matrix $\boldsymbol{A}$ is the inverse mapping of warping operation.(from output to input). We can use it directly to find the coresponding index of the input image *hat.raw*. For instance, the result image of the up triangle is shown in figure.

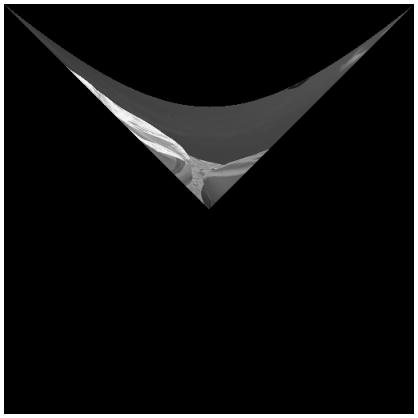

Figure 10: Up Triangle of the *hat.raw*

1.2.3 Experimental Result

1.2.4 Discussion

The control points I used are listed as follows:

Up triangel:

intput:[0,256],[0,512],[0,0],[256,256],[128,128],[128,384].

output:[128, 256], [0, 512],[0,0],[256,256],[128,128],[128,384].

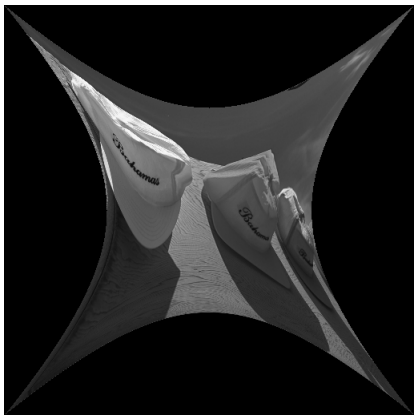

Figure 11: warping hat

The inverse mapping matrix is,

$$\begin{pmatrix} -6.1107 \times 10^{-13} & 1 & -4.8124 \times 10^{-13} & -3.9030 \times 10^{-16} & -6.4718 \times 10^{-16} & 9.9608 \times 10^{-16} \\ -4 & 4.0078 & -2.992 & -7.8125 \times 10^{-3} & 1.2275 \times 10^{-17} & 7.8125 \times 10^{-3} \end{pmatrix} \tag{22}$$

Bottom triangel:
intput:[512,0],[512,512],[512,256],[384,128],[384,384],[256,256].
output:[[512,0], [512, 512],[384,256],[384,128],[384,384],[256,256].
Left triangel:
intput:[256,0],[512,0],[0,0],[256,256],[128,128],[384,128].
output:[256,128], [512,0],[0,0],[256,256],[128,128],[384,128].

$$\begin{pmatrix} 4 & -3.0078 & -3.992 & 7.8125 \times 10^{-3} & 2.1709 \times 10^{-15} & -7.8125 \times 10^{-3} \\ 3.4992 \times 10^{-12} & -1.1047 \times 10^{-13} & 1 & 2.3065 \times 10^{-16} & -1.0404 \times 10^{-16} & -9.7774 \times 10^{-17} \end{pmatrix} \tag{23}$$

$$\begin{pmatrix} -1.3687 \times 10^{-13} & 1 & -1.5253 \times 10^{-13} & -8.6866 \times 10^{-17} & -3.5217 \times 10^{-1} & 5.8377 \times 10^{-16} \\ -4 & 4.0078 & 4.9921 & 7.8125 \times 10^{-3} & -5.4643 \times 10^{-15} & -7.8125 \times 10^{-3} \end{pmatrix} \tag{24}$$

right triangel:
intput:[256, 512],[0,512],[512,512],[256,256],[128,384],[384,384].
output:[256, 384], [0, 512],[512,512],[256,256],[128,384],[384,384]].

$$\begin{pmatrix} 4 & 5.0078 & -3.992 & -7.8125 \times 10^{-3} & -4.4319 \times 10^{-15} & 7.8125 \times 10^{-3} \\ -1.3204 \times 10^{-12} & 1.0095 \times 10^{-13} & 1 & -3.0703 \times 10^{-16} & 1.0254 \times 10^{-16} & 1.6719 \times 10^{-16} \end{pmatrix} \tag{25}$$

1.3 Lens Distortion Correction

1.3.1 Abstract and Motivation

The distortion cause the camera sometimes make people misunderstand the important information. In this section, I use a method to correct the distorted image.

1.3.2 Approach and Procedures

1. **Function mapping**
   Given the forward mapping from undistorted image to distorted image. We can use it as the inverse function of the input distorted imgae to output undistorted inmage to find the image index. The distortion function is show in Equation:

$$x_d = x(1 + k_1 \times r^2 + k_2 \times r^4 + k_3 \times r^6) \tag{26}$$

$$y_d = y(1 + k_1 \times r^2 + k_2 \times r^4 + k_3 \times r^6) \tag{27}$$

   Where, $r^2 = x^2 + y^2$, $k_1 = -0.3536$, $k_2 = 0.1730$.

   First of all, convert the output image index to the camera coordinates with equation 28 and 29:

$$x_c = \frac{i - u_c}{f_x} \tag{28}$$

$$y_c = \frac{i - v_c}{f_y} \tag{29}$$

   Where $u_c = 0.5 \times height$, $v_c = 0.5 \times width$, $f_x = f_y = 600$.

   Then use the mapping function to find camara coordinates of the input image, then it is easy to find the coresponding image coordinate of distorted image by:

$$u_d = f_x \times x_d + u_c \tag{30}$$

$$v_d = f_y \times y_d + v_c \tag{31}$$

   The result image is larger than the input image and some detail in the distored image is lost. To get a complete output image, I scale the image by 5/6.

2. **Linear regression** It is easy to conclude that the Equation 33 and 34 has no inverse fuction because it is not a linear transformation with polynomial degree. Thus, the inverse function of Equation 33 and 34 does not exist. However, we could use linear regression method to estimate the one-dimension model.

$$\begin{pmatrix} x_d \\ y_d \end{pmatrix} = \boldsymbol{A} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} r_x \\ r_y \end{pmatrix} \tag{32}$$

   The matrix $\boldsymbol{A}$ is the approximate inverse matrix we need to estimate. There is several way to do the linear regression. Such as PCA, SVM and etc. I choose the linear regression method to estimate matrix $\boldsymbol{A}$. By observing the image, I found that the image is twist into 4 direction. On the other hand, the Equation 33 and 34 indicate that the fuction differs in

the four quadrant. If we perform linear regression on the entire fuction, it is hard to get a accurate result. **Thus, it is resonable to perform linear regression in the four quadrant seperately and result in 4 linear function**. For the instance, the function lies in third quadrant is shown in Figure 12 and 13.

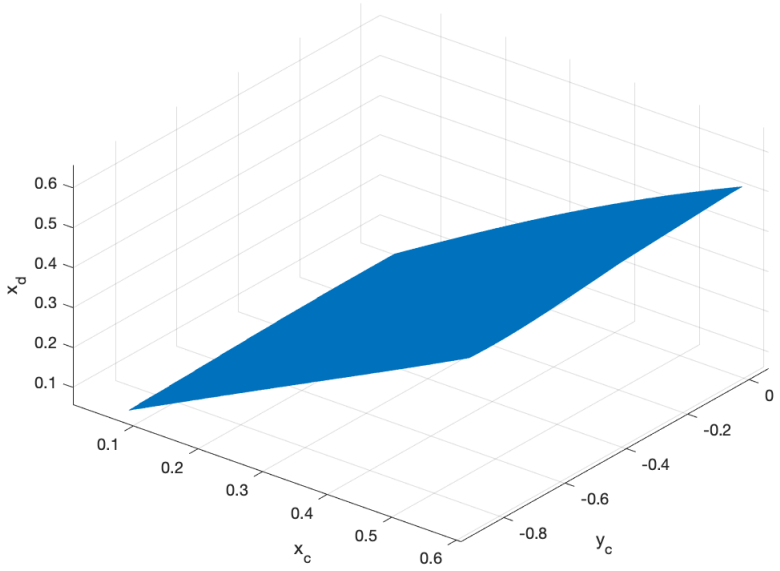

Figure 12: x_d vs. (x_c, y_c)

Thus, I finally got four matrix $\boldsymbol{A}_1$, $\boldsymbol{A}_2$, $\boldsymbol{A}_3$, $\boldsymbol{A}_4$ for 4 quadrant. To get a better visual result, I try several combination of the matrix obtain above. Finally, I choose the inverse matrix in the second and forth quadrant to recover the distored image. Because the right triangel in the image is usually crose this two quadrant.

$$x_d = x(1 + k_1 \times r^2 + k_2 \times r^4 + k_3 \times r^6) \tag{33}$$

$$y_d = y(1 + k_1 \times r^2 + k_2 \times r^4 + k_3 \times r^6) \tag{34}$$

The linear regression method is just a approximate estimate of the Equation 33 and 34. Thus, the result of linear regression method could not perform better than function mapping method.

1.3.3 Experimental Result

1.3.4 Discussion

Figure 14 shows that the right angle in the output recovered image turn out to be perpendicular. But the straight line clost to the corner of the image is still bending. Besides, there is some black

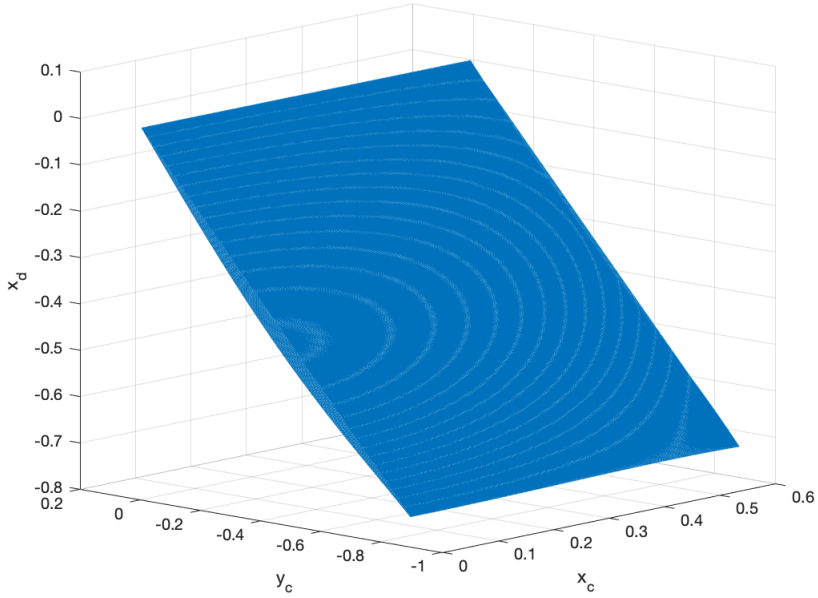

Figure 13: y_d vs. (x_c, y_c)

Figure 14: function mapping

Figure 15: function mapping

frame along the edges of the image and it is also bending. This is the main short coming of the function maping method.

When is comes to linear regression, the result image does not seems improved. That is because that the linear regress method loss much information of the function. The result image is worse than the function mapping method. But the frame of the image is more stright along the image edges. Thus, it maintain the most objects of the distorted image (like tapes and chairs). This is a good points of the regression method.

2 Problem 2: Morphological Processing

2.1 Basic Morphological Process Implementation

2.1.1 Abstract and Motivation

Morphological processes work much like spatial convolution in that pixel values are based on values of the neighboring pixels. Instead of constructing a dot product of a mask array and a pixel array, morphological processes use set theory operations such as intersection (AND), union (OR), and complement (NOT) to combine pixels logically into a resulting pixel value.

2.1.2 Approach and Procedures

The CHE topological attribute and LB topological attribute are shown in Table 2.

Table 2: Convex Set & Convex Hull of Binary pattrns

	Pattern 1	Pattern 2	Pattern 3	Pattern 4
C	1	1	1	1
H	0	0	0	1
E	1	1	1	0
L	0	0	0	1
B	0	4	1	0

The two basic morphological operations, dilation and erosion, plus many variants can be defined and implemented by a *hit-or-miss transformation*. For 3×3 stucturing element case, neighboring pixels are converted to a one-dimensional bit stream using the following pattern.

X_3	X_2	X_1
X_4	X	X_0
X_5	X_6	X_7

Figure 16: Nighbourhood Pixel

If the binary-valued pattern of the mask matches the state of the pixels under the mask (hit), an output pixel in spatial correspondence to the center pixel of the mask is set to some desired binary state (in our case, set to 1). For a pattern mismatch (miss), the output pixel is set to the opposite binary state (which is zero).

Hit-or-miss morphological algorithms are often implemented in digital image processing hardware by a pixel stacker followed by a look-up table (LUT).

1. **Shrinking and Thinning**
   Shrinking and Thinning is kind of similar. Shrinking is erasing pixels such that an object without holes erodes to a single pixel at or near its center of mass, and an object with holes erodes to a connected ring lying midway between each hole and its nearest outer boundary. While thinning is erasing pixels such that an object without holes erodes to a minimally connected stroke located equidistant from its nearest outer boundaries, and an object with holes erodes to a minimally connected ring midway between each hole and its nearest outer boundary.

   It is not possible to perform shrinking and thinning using a single stage 3×3 hit-or-miss transform. The 3×3 window does not provide enough information to prevent total erasure and to ensure connectivity. Thus we implement a two-stage algorithm with hit-or-miss transforms. In the algorithm, two concatenated 3×3 hit-or-miss transformations are performed to obtain indirect information about pixel patterns within a 5×5 window.

   In the first stage, the eight neighbour pixels are gathered in a bit stream. Following the Look-Up Table and generate a conditional mark M map for second stage. If the foreground and background pixels in the Conditional Look-Up Table exactly match foreground and background pixels in the image, then the pixel is marked M conditionally for erasure. In the second stage of the algorithm, the center pixel X and the conditional marks M in a 3×3 neighborhood centered about X are examined to create an output pixel. The output pixel value is:

   $$G(i,j) = X \cap [\bar{M} \cup P(M, M_0, M_1 ... M_7)] \tag{35}$$

   Where $P(M, M_0, M_1 ... M_7)$ is an erasure inhibiting logical variable equal to 0 or 1. It is defined by Hit-or-Miss transform in Unconditional Look-Up Table ($[P(M, M_0, M_1 ... M_7)] = 1$ if hit).

2. **Skeleton**
   A skeleton figure representation of an object can be used to describe its structure. The Unconditional Look-Up Table is slightly different from the shrinking and thinning. After the two-stage algorithm, it is necessary to perform bridging to restore the connectivety.

The automatic break of iteration is settled as follow. If the conditional mark M map are not erasured in the second stage, the algorithm will stop and return the latest image.

In summery, the main process are as follows:

1. Padding 1 pixel width in the image with zeros pad.
2. Generate the M-map by conditional Look-Up Table to mark the target pixel ready to be examed.
3. scan the M map and look up the unconditional Look-Up Table, if the pixel miss the Hit-N-Miss mask, erase it
4. repeat until the conditional mark M map are not erasured in the second stage.
5. return the latest image.

2.1.3 Experimental Result

First of all, enlarge the image with zeros padding.

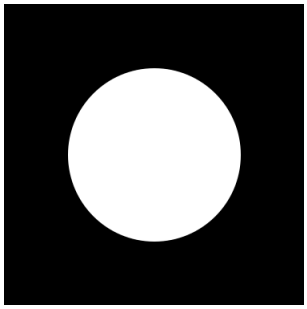

Figure 17: Pattern 1

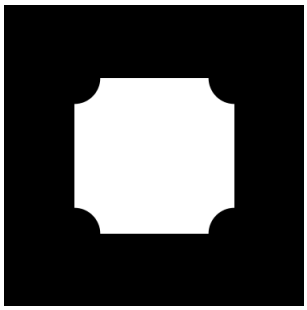

Figure 18: Pattern 2

2.1.4 Discussion

Shrink can be used to determine the isolate component and cluster. The pattern 1 and pattern 2 is shrink to the central point, while pattern 3 is a bit lower. That is because the heart's center of

Figure 19: Pattern 3

Figure 20: Pattern 4

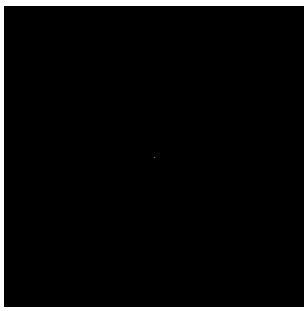

Figure 21: Pattern 1 Shrinking, iter times = 108

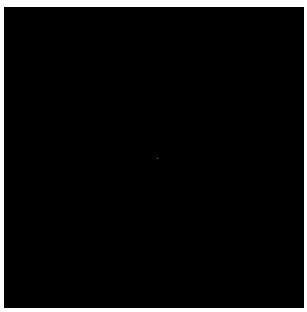

Figure 22: Pattern 2 Shrinking, iter times = 101

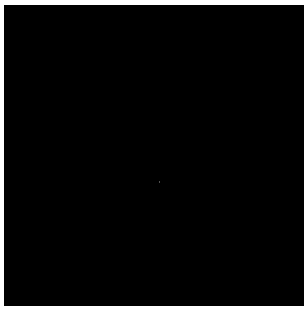

Figure 23: Pattern 3 Shrinking, iter times = 144

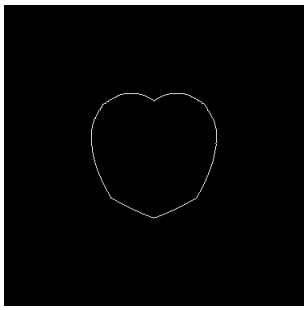

Figure 24: Pattern 3 Shrinking, iter times = 48

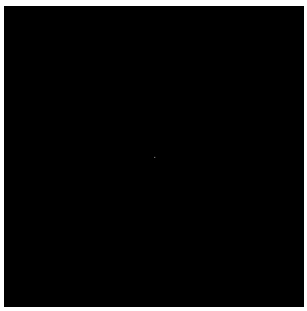

Figure 25: Pattern 1 Thinning, iter times = 109

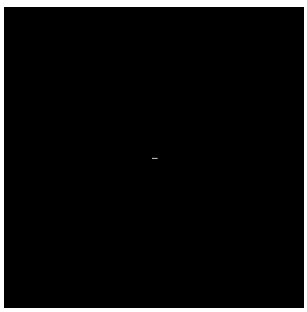

Figure 26: Pattern 2 Thinning, iter times = 99

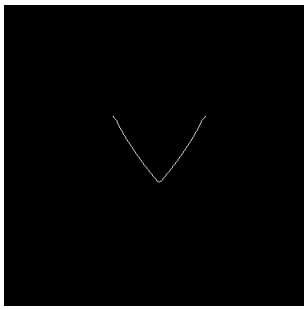

Figure 27: Pattern 3 Thinning, iter times = 87

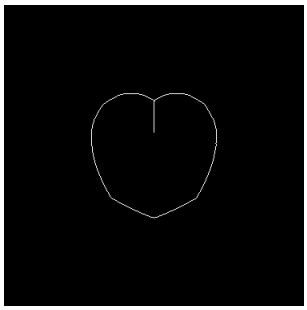

Figure 28: Pattern 4 Thinning, iter times = 50

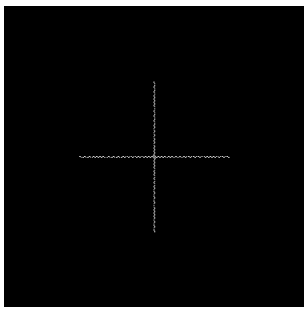

Figure 29: Pattern 1 skelentoning, iter times = 76

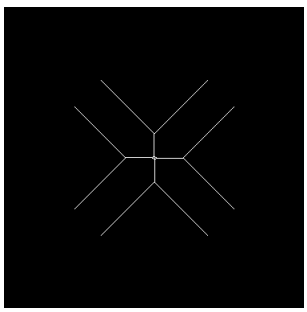

Figure 30: Pattern 2 skelentoning, iter times = 97

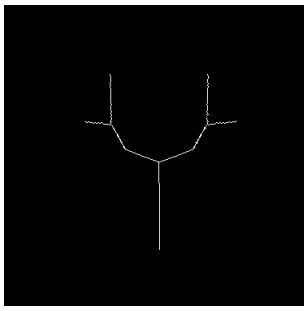

Figure 31: Pattern 3 skelentoning, iter times = 84

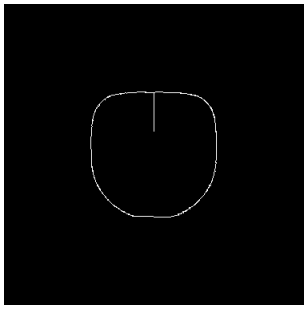

Figure 32: Pattern 4 skelentoning, iter times = 46

gravity is a bit lower, thus the shrinking point goes down. Pattern 4 is a ring pattern. There is a lake in the pattern. Thus, it shrink to the centeral line of the ring constructure.

The result of thinning must maintain the connectivity. This guarantees that the number of connected line structure equal to the number of connect regions in the original image. It contains the minimal number that maintain 8-connnectivity. For pattern 1, it is the same as the shrinking result. For pattern 2, the thinning result is a short line in the center of the image, which indicate it width is longer than the height (except the four bayes corner). The pattern 3 is thinned into a "V". The pattern 4 is thinned into a "apple".

While skelentoning processing returns the fundamental construct of the pattern but all redundant pixels should be removed. For pattern 1, it is skelentoning into a crossing patterns. For pattern 2, it returns the crossing line that could bulid up such a pattern. Because there is four bays in this morphological pattern, it goes into the center point. For pattern 3, it returns a "fork". While for the pattern 4 with a lake, it forms a close cycle afrer skelentoning. For all the 4 patterns, the skelentoning image center is stll located at the center of the image.

2.2 Defect Detection and Correction

2.2.1 Abstract and Motivation

Quality control is an important issue in the industry. On the other hand maintaining the defectless of production is also a major issue in manufacturing. It depends on purity of texture, accuracy of color, shape etc. In this section I perform a method to detect the defects and to recover the defectless binary image efficiently.

2.2.2 Approach and Procedures

1. **Shrink the Background**
   Before we start, the image should be zero padding with 1 pixels because there is some object pixels on the edge. If we shrink the object(deer), we could possibly get the following result image.

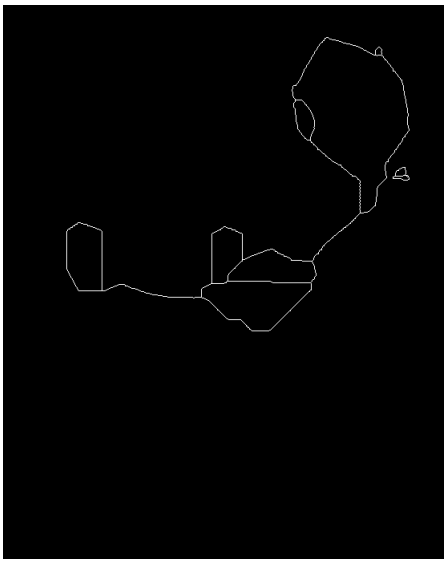

Figure 33: Shrinking deer, iter times = 354

   There is a way to detect the defect by counting the 4-connected black component in this image. But it is complicate and inefficient. I implement the method that shrinking the black background instead of the object. And the isolated dosts appears. The procedure is as follows. Firstly, convert the background to object and perform shrinking. That means the object (deer) will be expanded and both the defect region and the lakes of the object will be shrink to several isolate white pixels.

2. **Fill the defect region**
   To seperate defect region from the lakes in the origional image, the neighbour pixels need to be taken into consideration. Check the 8-connectivity of the white pixels obtain by shrinking.

3. **Check the 4-connected background**
   If the background pixel is 4-connected but not 8-connected, it should also be considered as a defect. Because if the pattern is enlarged into a larger scale, The hole will become larger and it will cause a big mistake in industry. Check every background pixel if it maintains the 4-connectivity. If the hole is 4-connected, fill it with 255.

2.2.3 Experimental Result

Defect pixel $P(i, j)$: (207,498), (280,93), (284,275), (335,334), (352,331), (284,275), (335,334), (352, 331).

Figure 34: Shrinking background, iter time = 429

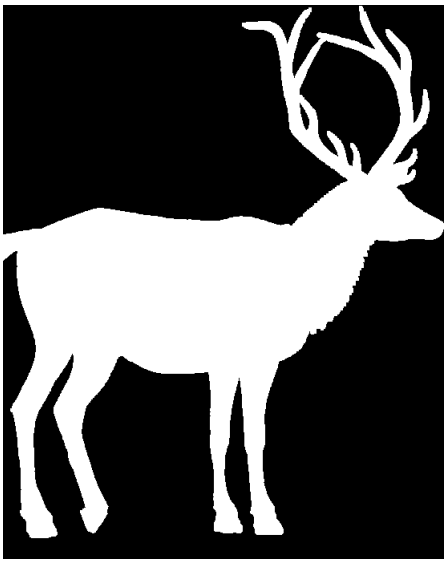

Figure 35: defectless deer

2.2.4 Discussion

The deer image is not defectless. The Defect region $P(i, j)$ is (207,498), (280,93), (284,275), (335,334), (352,331), (284,275), (335,334), (352, 331). The first five dot is detected by shrinking, while the last three dots is detected by checking the four connected area. Shrinking cannot detect the eight-connected black dot. Thus, the other detect method must be taken into consideration.

2.3 Object Analysis

2.3.1 Abstract and Motivation

Object analysis plays an important role in image processing. It extract items in the image into fundamental binary pieces and perform mophological processing to extract important information from the input image. In this section, I analysis the input image *rice.raw* to figure out the number of rices in the picture and the size of the different kind of rice.

Figure 36: *rice.raw*

2.3.2 Approach and Procedures

1. **Turn into binary image**
   First, I turn the color image *rice.raw* into grayscale image with equation 36:

$$WB(i, j) = 0.2989 \times R(i, j) + 0.5870 \times G(i, j) + 0.1140 \times B(i, j) \tag{36}$$

   Then I draw the histogram of the grayscale image. The histogram is shown in Figure 37. By observing the histogram of *rice.raw*, there is two peak occure in this figure. The higher peak might be the backgroud's pixel value. The lower peak might be rice's pixel value. To seperate the background and the rice grains, set the threshold equal to 85. The grayscale image is turned into binary image. Figure 38 shows the binary image.

2. **Connected component labeling**
   In order to counting the rice grains, I use the connected component labeling method to collect the 8-connected region. The programm is build on c++. The algorithm is designed as follows.
   Algorithm:

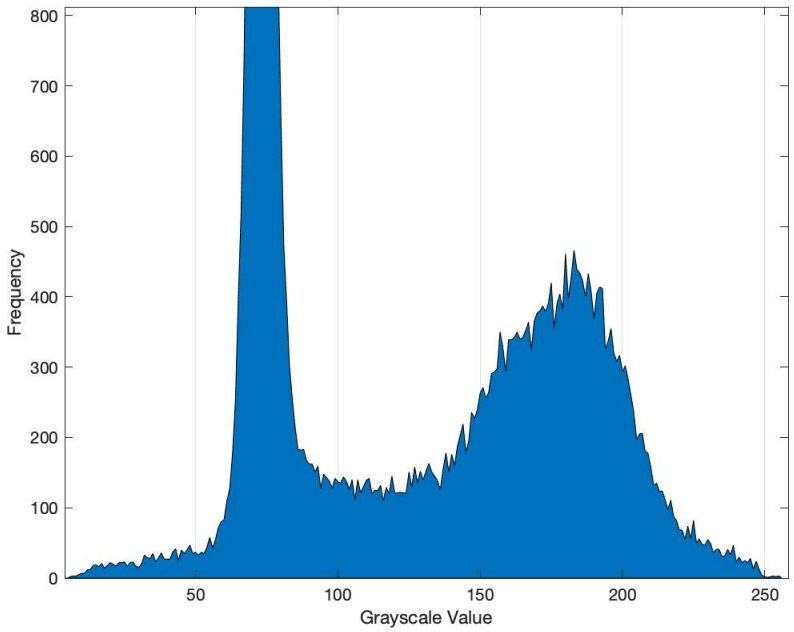

Figure 37: *rice.raw* histogram

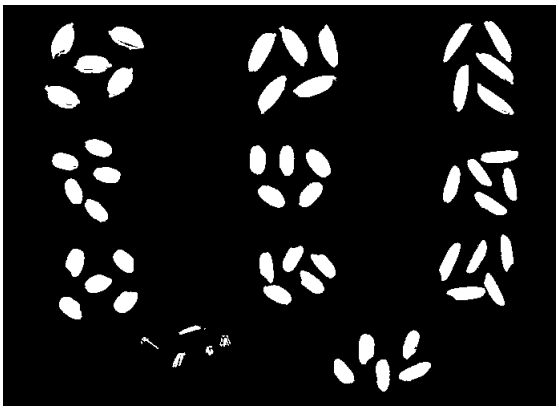

Figure 38: *rice.raw* binary ver.

set $label = 1$

for each pixels $I(i,j) \in I(A)$

if ($I(i,j)$ is object and $I(i,j)$ is not labeled):

$I(i,j) = label$

push $I(i,j)$ into the queue q

while (q is not empty):

pop the first element $I(m,n)$ in the q

if the 8-connected neighbours $I(a,b)$ of $I(m,n)$ is an object and not labeled

$I(a,b) = label$

push $I(a,b)$ into the queue q

end if

end while

$label = label + 1$

end if

end for

Then the connected component are labeled into several regions. To filter the noise, set a threshold = 10. That means, if the number of the pixels in a connected area are less than 10, it will not be considered as a "rice grain". The connected component are marked in figure 39 (There is an area that marked nearly white and it is hard to be seen in the piture).

Figure 39: *rice.raw* connected component ver.

It is difficult to tell which rice is larger and which is smaller. Thus, I use the number of pixels of the same kind of rice grains as the measurement to rank the size of different kinds of grain. To measure the size of each kind of rice grains, I use k-means method to cluster each grains into 11 groups (assume that each cluster in the image indicate that the grain belongs to the same type). I use Matlab build-in toolbox to do the k-means.

1. **Threshold the rice grains and fill the hole and erase the noise**
   In order to clearly show the grains in the image, I use two threshold 66 and 78 to thres the grayscale image into binary image. There will be some holes and noise remains in the image. I use **Ranking the connecting** method to fill the hold and erase the noise. Firstly, check the connectivity of each pixels in the image. Then, use the following equation:

$$rank = 2 \times \ 4\_connected + 1 \times 8\_connected \tag{37}$$

   If the rank is lower than 6 and it is a object, erase it. If the rank is higher than 6 and it is a background, fill it. Then I got the figure num. (I try my best but it still looks not perfect:( )

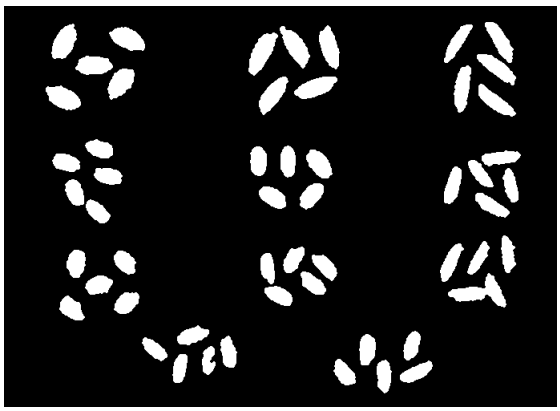

Figure 40: *rice.raw* denoised ver.

2. **build cluster**
   Use the binary image obtained above as the input. Collect the index of the object (whose pixel value is not equal to background pixel value) and form the database. The first column is row index, the second column is column index.

3. **perform kmeans on the detabase** Classify the pixels into 11 clusters. Sum the pixels with the same label and sort them in increasing order. The cluster result is shown in Figure 41.

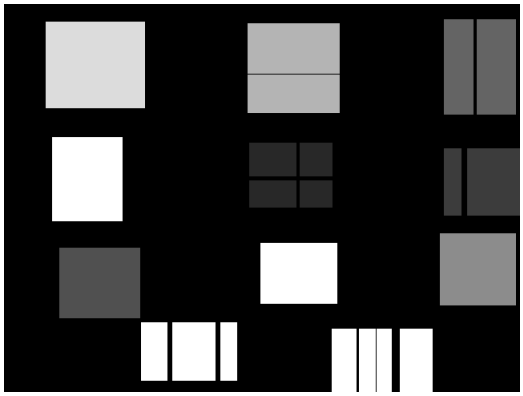

Figure 41: 11 Clusters

2.3.3 Experimental Result

There are 55 rice grains in the image. The rank of size for each grain are shown in 42 in increasing order:

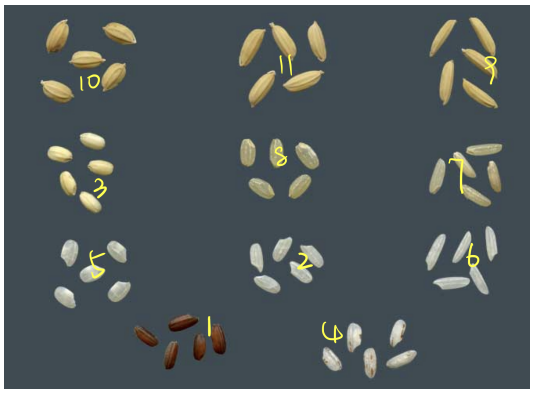

Figure 42: *rice.raw* denoised ver.

2.3.4 Discussion

The pixel value of black rice in the left-down corner is partially darker than the background. Therefore, I cannot get a both complete black rice grain and other rice grains under the same threshold. In order to compare the size of the grains, I need to use two threshold and capture the grain pixels. On the other hand, the edge of the black rice is darker than the background color. Which means, when I use the threshold to obtain the rice grains, some part of the black grain was erased. In visual comparision, the black rice (rank 1) is larger than rank 2 rice grain. The average size of each piece of rice grains are in Table 3.

Table 3: Size of the Rice grain

	1	2	3	4	5	6	7	8	9	10	11
size	545.8000	587.4000	593.8000	618.6000	627.2000	628.2000	642.8000	675.4000	830.0000	890.2000	957.6000

The size show in 3 is definely larger than the actual size. Because I add some artifact around the object as well as filling the hole.