Reconstruction of 3D Solid Ball by X-Ray Image

[1,*] *Yu-Yan Li*（李昱言）, [1]*Chiou-Shann Fuh*（傅楸善）,

[1]Department of Computer Science and Information Engineering,
National Taiwan University, Taipei, Taiwan,
[*]E-mail: b08902122@ntu.edu.tw fuh@csie.ntu.edu.tw

ABSTRACT

This paper presents a study on 3D reconstruction of solid ball by X-ray images using Algebraic Reconstruction Technique (ART).

Keywords: *3D Reconstruction, SART*

1. INTRODUCTION

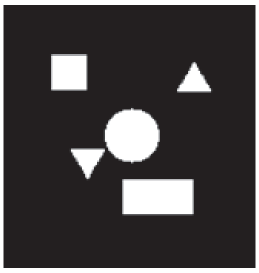


Fig. 1 Original 2D image we try to reconstruct.

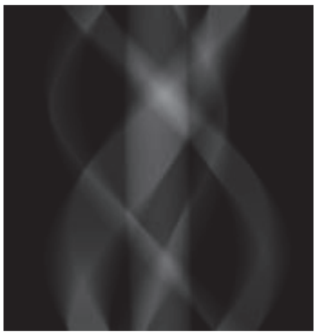
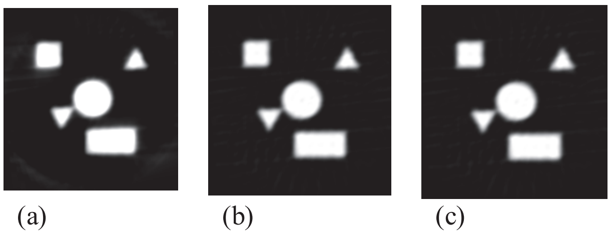


Fig. 2 Sinogram generated from the original 2D image.


(a) (b) (c)
Fig. 3 Reconstructed from sinogram using 32 projections and addition algebraic reconstruction technique with (a) 1, (b) 100, (c) 500 iterations.

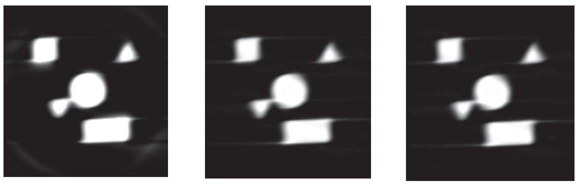
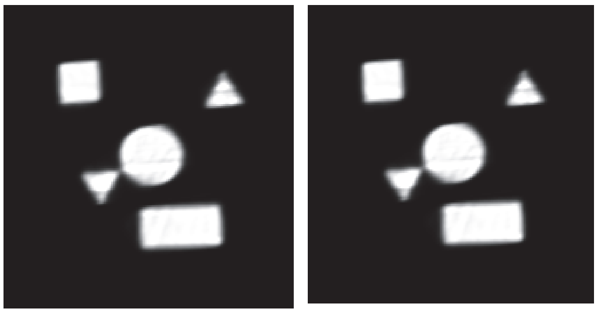
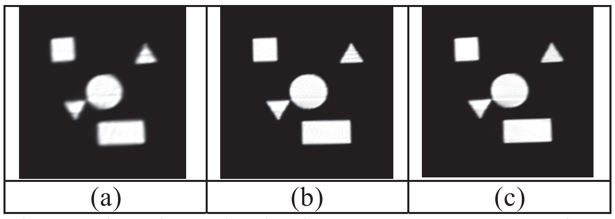

(a) (b) (c)

Fig. 4 Images reconstructed from sinogram using 180 projections and addition algebraic reconstruction technique with (a) 1, (b) 100, (c) 500 iterations. We can see that the image we reconstruct contains more error than the image used 32 projections.


(a) (b)
Fig. 5 The above is the image we reconstruct from sinogram using 32 projections and multiplication algebraic reconstruction technique with (a) 100 and (b) 500 iterations.



(a)	(b)	(c)

Fig. 6 The above is the image we reconstruct from sinogram using (a) 32 and (b) 64 and (c) 128 projections and multiplication algebraic reconstruction technique with 500 iterations. Note that we first rescale the original

image from 164 x 171 to 512 x 512 pixels to improve the resolution.

The following is the code we use to produce sinogram:

```
import cv2
import numpy as np

def rotate (image, angle):
        h, w = image.shape[:2]
        center = (w/2, h/2)
        M = cv2.getRotationMatrix2D(center, angle,
1.0)
        rotated = cv2.warpAffine(image, M, (w, h))

        return rotated

def getSinogram(image):
        colAddImage = np.ndarray(shape=(180,
image.shape[0]))
        for angle in range(0, 180):
                rotatedImage = rotate(image, angle)
                colAddImage[angle] =
np.sum(rotatedImage, axis=1) / image.shape[1]
        return colAddImage
```

The following is the code we use for addition algebraic reconstruction technique:

```
def addition(sinogram, lastresult, projections):
        anglePerPhoto = 180/projections
        for angle in np.arange(0, 180, 180/projections):
                colSum = sinogram[int(angle)]
                newColSum = np.sum(lastresult,
axis=1) / lastresult.shape[1]
                diff = colSum - newColSum
                for i in range(diff.shape[0]):
                        lastresult[i] += diff[i]
                lastresult = rotate(lastresult,
anglePerPhoto)
        lastresult = rotate(lastresult, 180)
        return lastresult
```

The following is the code we use for multiplication algebraic reconstruction technique:

```
def multiplication(sinogram, lastresult, projections):
        anglePerPhoto = 180/projections
        for angle in np.arange(0, 180, 180/projections):
                colSum = sinogram[int(angle)]
                newColSum = np.sum(lastresult,
axis=1) / lastresult.shape[1]
                for i in range(colSum.shape[0]):
                        if newColSum[i] == 0:
                                newColSum[i] = 1;
                        lastresult[i] *= (colSum[i] /
newColSum[i])
                lastresult = rotate(lastresult,
anglePerPhoto)
        lastresult = rotate(lastresult, 180)
        return lastresult
```

2. Filtered Back Projection

We try FBP on Matlab to reconstruct the original image. First, we perform Radon transform on the original image. Then, we apply filter on the transformed image. Last, we perform Inverse Radon transform to get the reconstructed image.

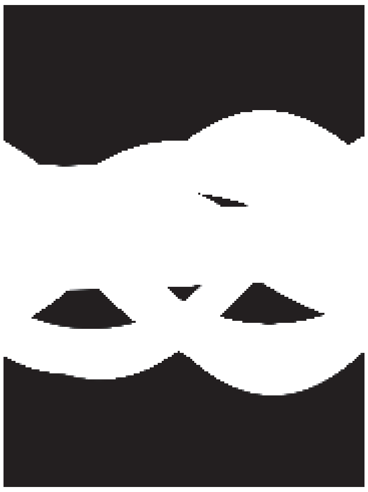


Fig. 7 The above is the sinogram after we perform Radon transform.

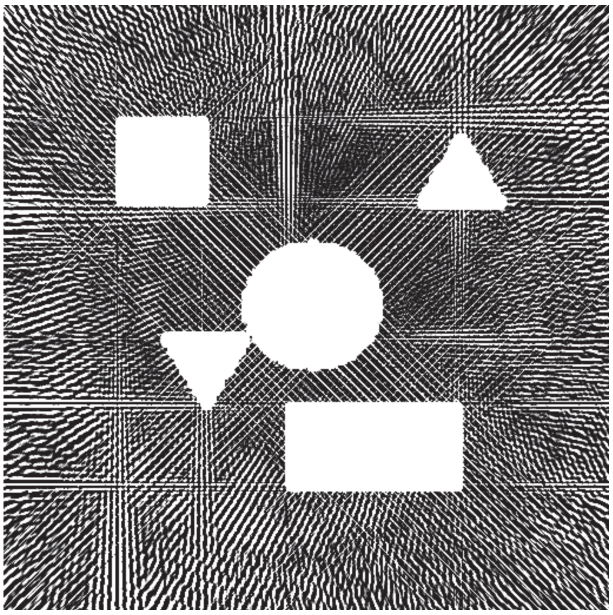


Fig. 8 The above is the image we reconstruct using Ram-Lak filter.

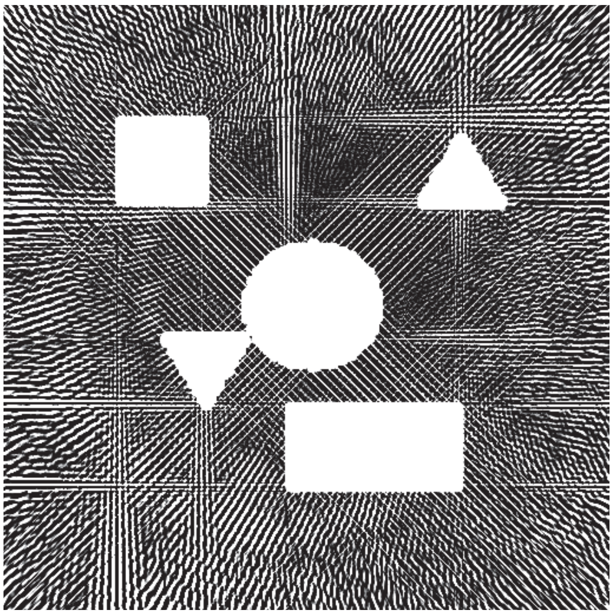


Fig. 9 The above is the image we reconstruct using Shepp-Logan filter.

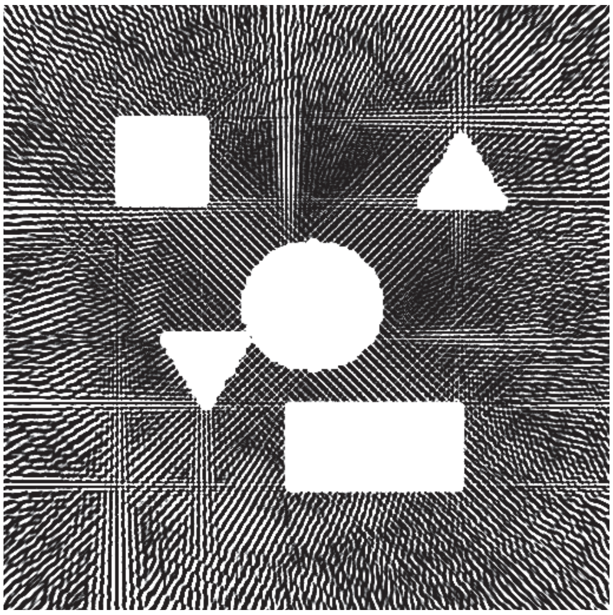


Fig. 10 The above is the image we reconstruct using Cosine filter.

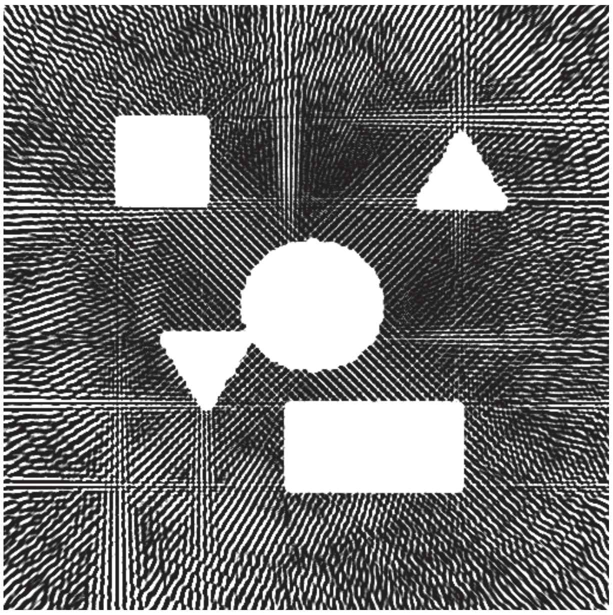


Fig. 11 The above is the image we reconstruct using Hamming filter.

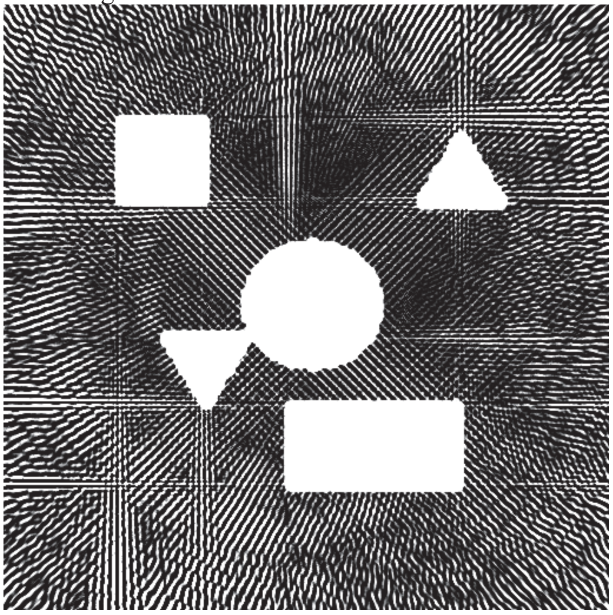


Fig. 12 The above is the image we reconstruct using Hann filter.

3. SART

$$f_j^{(i)} = f_j^{(i-1)} + \frac{\sum_{p_i \in P_\varphi} \left[\lambda \frac{p_i - \sum_{k=1}^{N} f_k^{(i-1)} w_{ik}}{\sum_{k=1}^{N} w_{ik}} \right]}{\sum_{p_i \in P_\varphi} w_{ij}} w_{ij} \quad j = 1, 2, \ldots, N$$

N : total number of pixels.
λ : relaxation factor.
p_i : the projection value of the i-th ray.
w_{ik} : the weight of the k-th pixel on the i-th ray.

$f_j^{(i)}$: the j-th pixel value after being updated by the i-th ray.

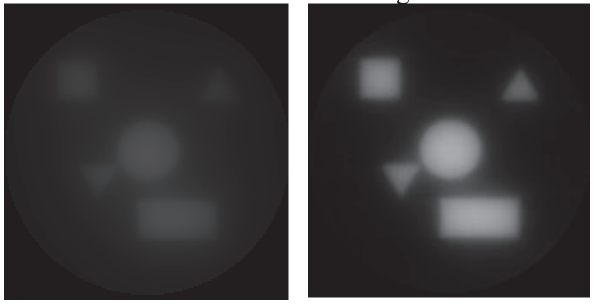
$\sum_{k=1}^{N} f_k^{(i-1)} w_{ik}$: the current projection value of the i-th ray.

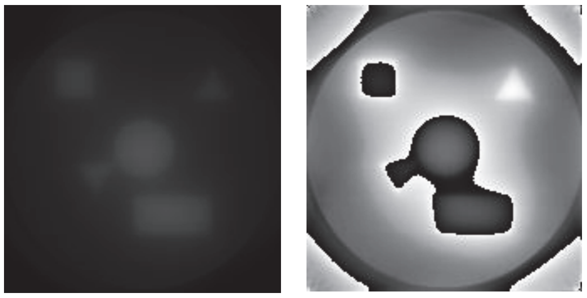
P_φ : All projection rays.

Simultaneous Algebraic Reconstruction Technique considers all degree in one iteration.
The following is the python code we use to perform Simultaneous Algebraic Reconstruction Technique.

```
def SART(sinogram, lastresult):
   lastsino = getSinogram(lastresult)
   diff = sinogram - lastsino
   h, w = lastresult.shape[:2]
   center = (w/2, h/2)
   for i in range(lastresult.shape[0]):
      for j in range(lastresult.shape[1]):
         value = np.array([], dtype=np.float64)
         for angle in range(180):
            M = cv2.getRotationMatrix2D(center, angle, 
1.0)
            coor = np.array([j, i, 1])
            new_coor = np.dot(M, coor)
            if new_coor[0] >= 0 and new_coor[0] < w and 
new_coor[1] >= 0 and new_coor[1] < h:
               value = np.append(value, 
diff[angle][int(new_coor[1])])
         if len(value) == 180:
           avg = np.average(value)
           lastresult[i][j] += avg
   return lastresult
```

Note that we ignore points that do not have 180 projection rays passed. We think that it can help us concentrate on the center of the image.


(a) (b)
Fig. 13 The above is the images we reconstruct using simultaneous algebraic reconstruction technique with (a) 1 and (b) 5 iterations.


(a) (b)
Fig. 14 The above is the images we reconstruct using simultaneous algebraic reconstruction technique with (a) 1 and (b) 5 iterations without ignoring points that do not have 180 projection rays passed. Note that there are noises in the corner of the image.

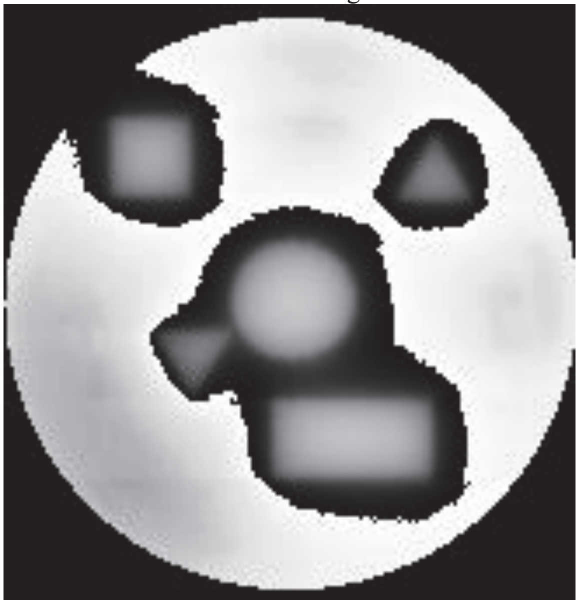


Fig. 14 The above is the image we reconstruct using simultaneous algebraic reconstruction technique with 10 iterations. Notice that there are noises in the image.

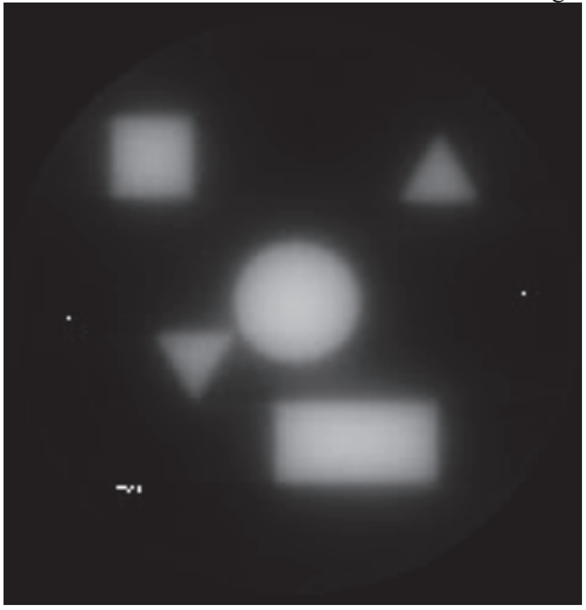


Fig. 15 The above is the image we reconstruct using simultaneous algebraic reconstruction technique with 7

iterations. We check the output image of each iteration, and we discover that the noise starts appear in seventh iteration.

As the above, the image we reconstruct using SART with 10 iterations contains a lot of noises. We believe that the noise comes from overflow or underflow of pixel values. To eliminate the noises, we first fix the SART function.

```
def SART(sinogram, lastresult):
    lastsino = getSinogram(lastresult)
    diff = sinogram - lastsino
    h, w = lastresult.shape[:2]
    center = (w/2, h/2)
    for i in range(lastresult.shape[0]):
        for j in range(lastresult.shape[1]):
            value = np.array([], dtype=np.float64)
            for angle in range(180):
                M = cv2.getRotationMatrix2D(center, angle,
1.0)
                coor = np.array([j, i, 1])
                new_coor = np.dot(M, coor)
                if new_coor[0] >= 0 and new_coor[0] < w and
new_coor[1] >= 0 and new_coor[1] < h:
                    value = np.append(value,
diff[angle][int(new_coor[1])])
            if len(value) == 180:
              avg = np.average(value)
              lastresult[i][j] += avg
              if lastresult[i][j] < 0:
                lastresult[i][j] = 0
              elif lastresult[i][j] > 255:
                lastresult[i][j] = 255
    return lastresult
```

Note that we check the pixel value is overflow or underflow right after we update its value. Also, we declare the result image be an array of 64-bits float number instead of 8-bits unsigned integer.

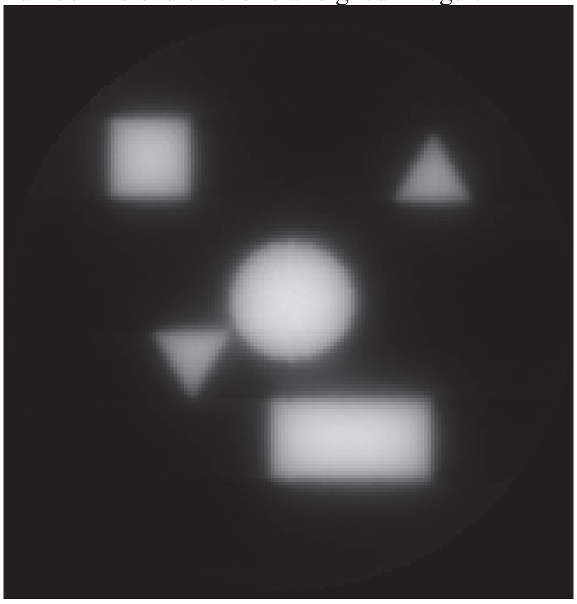


Fig. 16 The above is the image we reconstruct using simultaneous algebraic reconstruction technique with 10 iterations after we fixed the SART function.

4. 3D

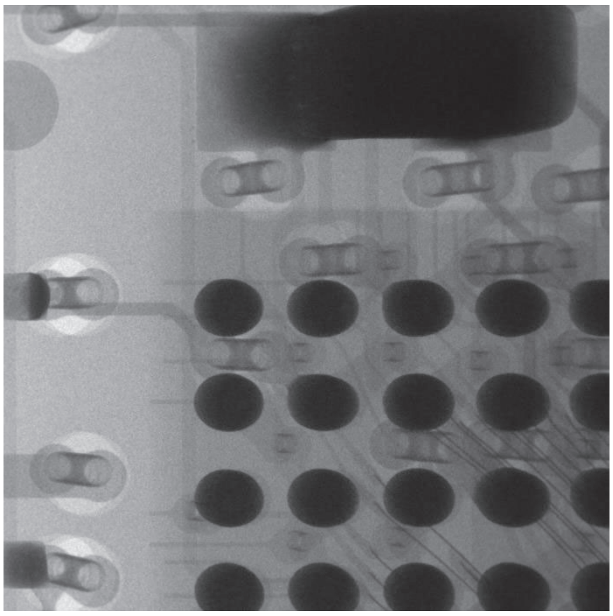


Fig. 17 The above is the X-Ray projection image of Printed Circuit Board (PCB) (totally 128 images) we try to reconstruct from.

We use the code from [2] to reconstruct the 3D image of the PCB.

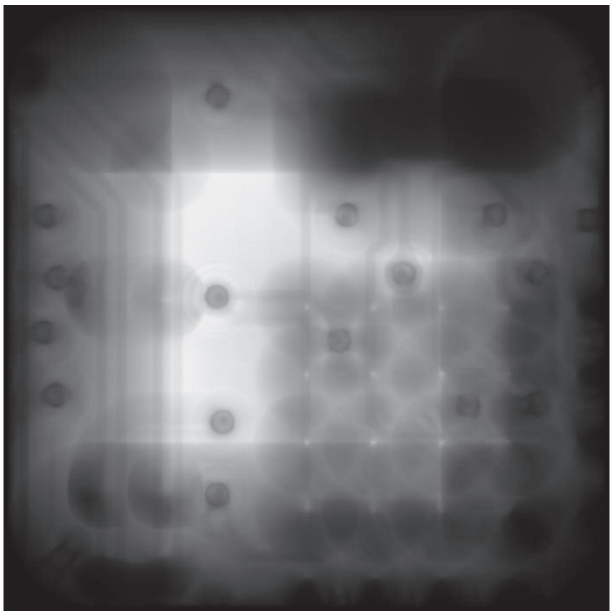


Fig. 18 The above is the image we reconstruct with z=10.

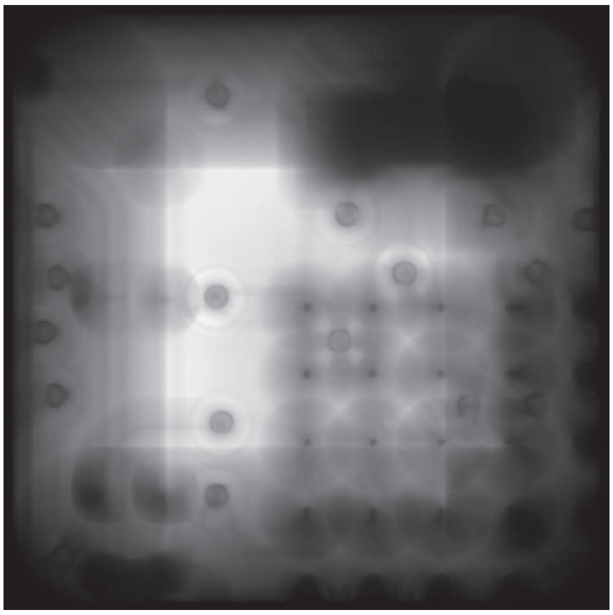


Fig. 19 The above is the image we reconstruct with *z*=20.

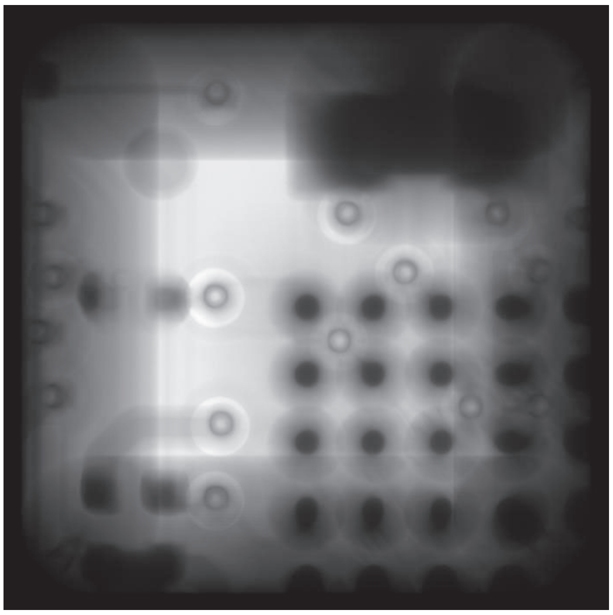


Fig. 20 The above is the image we reconstruct with *z*=40.

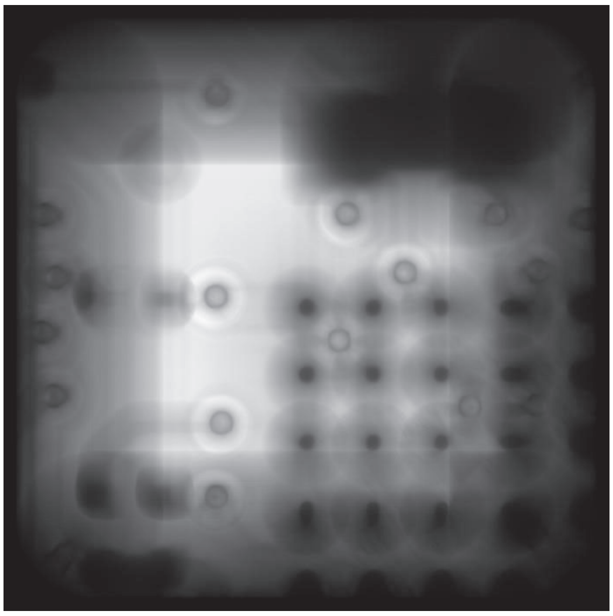


Fig. 20 The above is the image we reconstruct with *z*=30.

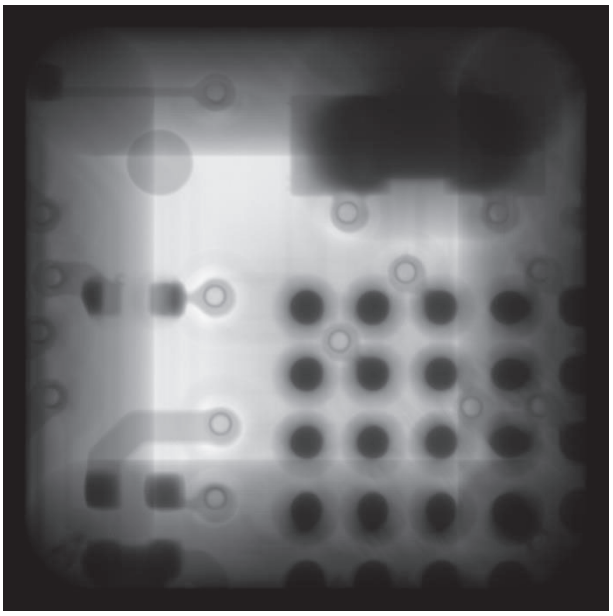


Fig. 21 The above is the image we reconstruct with *z*=50.

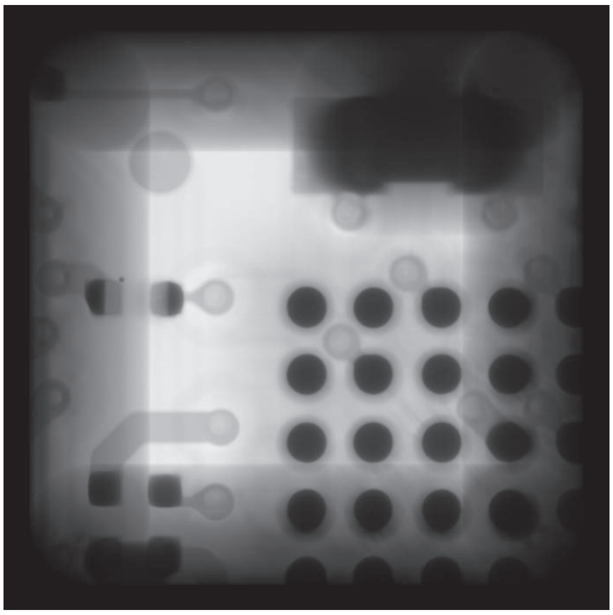


Fig. 22 The above is the image we reconstruct with *z*=60.

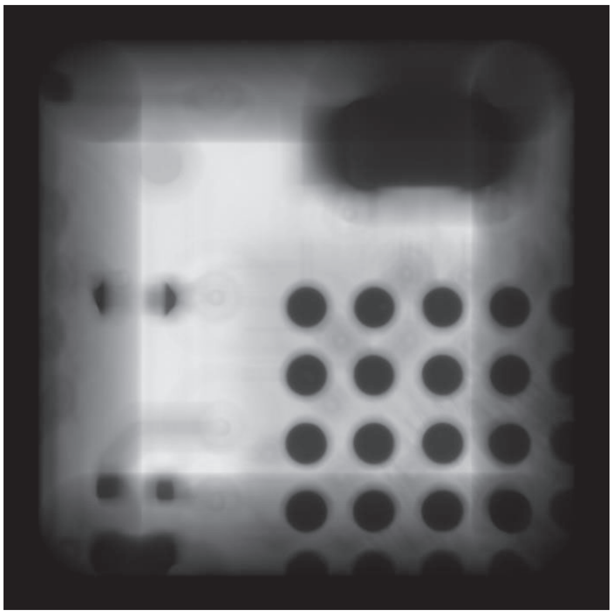


Fig. 24 The above is the image we reconstruct with *z*=80.

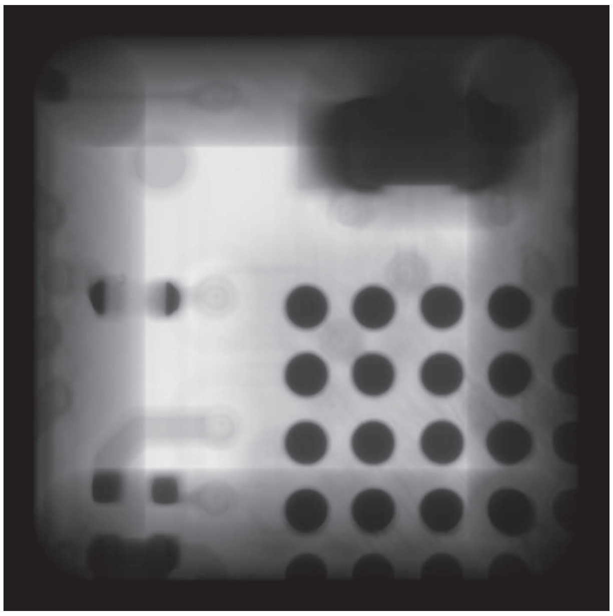


Fig. 23 The above is the image we reconstruct with *z*=70.

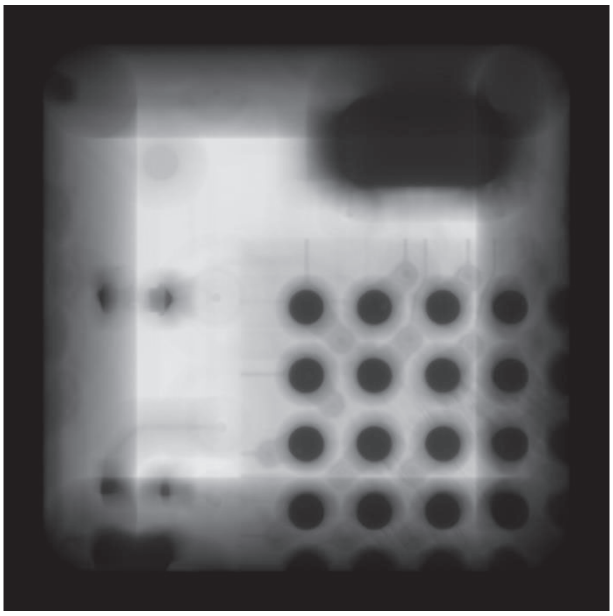


Fig. 25 The above is the image we reconstruct with *z*=90.

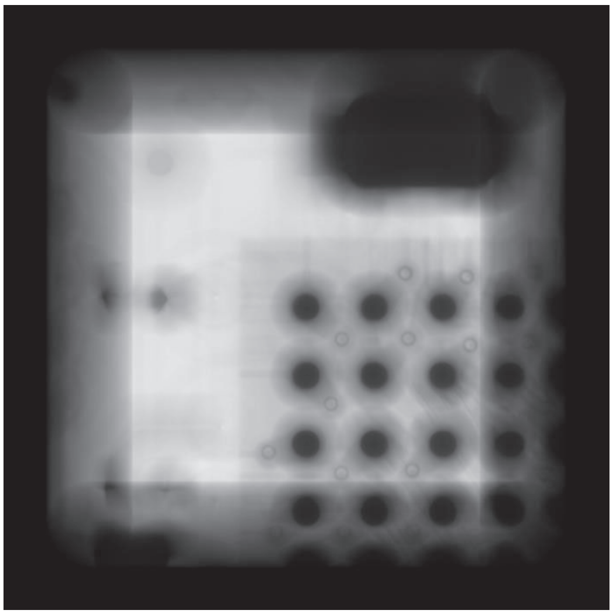


Fig. 26 The above is the image we reconstruct with *z*=100.

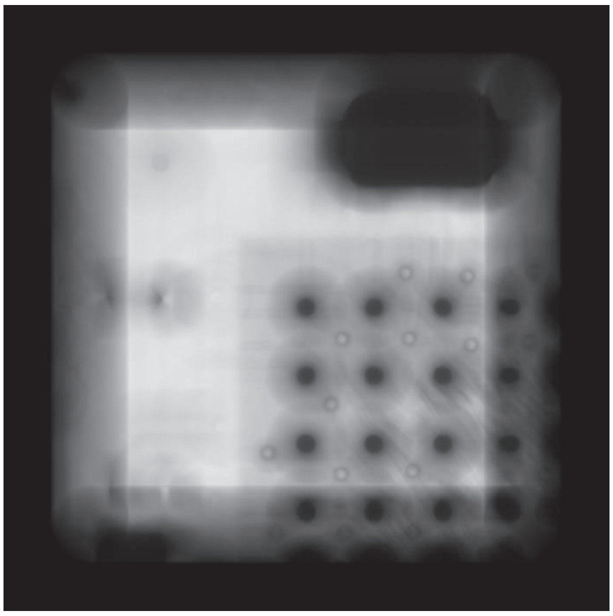


Fig. 27 The above is the image we reconstruct with *z*=110.

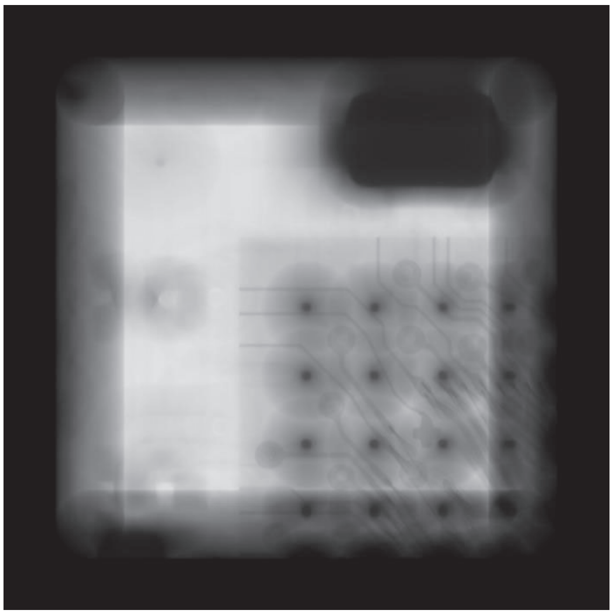


Fig. 28 The above is the image we reconstruct with *z*=120.

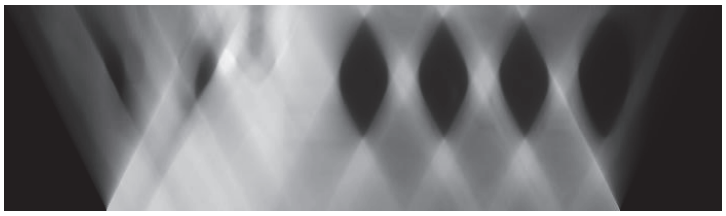


Fig. 29 The above is the image we reconstruct with *x*=350. We can see that the shape of solid ball in the image we reconstructed is not ball but rhombus.

5. REFERENCES

[1] Chen, H. Y. (2017). 用X光影像三維重建 [master's thesis, National Taiwan University]. Airiti Library. https://doi.org/10.6342/NTU201700908
[2] Y. B. Zhang 3D Reconstruction of Solder Balls [https://www.csie.ntu.edu.tw/~fuh/personal/3D%20Reconstruction%20of%20Solder%20Balls.pdf]
[3] MathWorks, "Matlab," http://www.mathworks.com, 2016.