Solder Ball 3D Reconstruction with X-Ray Images Using Filtered Back Projection

Ting-Chen Tsan
Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan 10617
r06922029@ntu.edu.tw

Bing-Jhang Lin
Graduate Institute of Bio-Electronics and Bio-Informatics
National Taiwan University
Taipei, Taiwan 10617
lknight8631@gmail.com

You-Hsien Lee
Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan 10617
r06922085@csie.ntu.edu.tw

Tzu-Chia Tung
Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan 10617
d04944016@csie.ntu.edu.tw

Chiou-Shann Fuh
Graduate Institute of Bio-Electronics and Bio-Informatics
National Taiwan University
Taipei, Taiwan 10617
fuh@csie.ntu.edu.tw

***Abstract*—We aim to reconstruct the 3D model of solder balls on the PCB (Printed Circuit Board) according to many projections of X-ray images of those solder balls. We mainly use a tomographic reconstruction algorithm FBP (Filtered Back Projection) to obtain the cross-section of different levels without breaking the PCB and inspect their inner structure. We also optimize our execution times and compare our results with Volume Graphics.**

Keywords—3D Reconstruction, X-ray inspection, PCB, FBP

I. Introduction

The PCB (Printed Circuit Board) plays an important role in many electronic products. Manufacturers need to ensure the quality for all the finished products. They have to inspect the PCBs if there are defects such as open solder joints, short circuit, solder bridges, component shifted, void, and so on. However, the architecture of PCBs is very precise and many defects cannot be observed by human eyes, so we need to use some technical method such as Automated Optical Inspection (AOI), Solder Paste Inspection (SPI), and Automated X-ray Inspection (AXI). PCBs can be single-sided, double-sided, or multi-layer. For multi-layer PCBs, many overlapped electronic components make the defect inspection more difficult and challenging.

In this paper, we aim to reconstruct the three-dimensional PCB solder balls with the X-ray images to inspect the PCB if there are void in the solder balls or not. The X-ray images are obtained by taking several photographs from different angles. With these X-ray images, we can reconstruct and observe the cross-section of different levels non-invasively without breaking the PCB. Many algorithms are proposed for tomographic reconstruction [8] which is mainly based on the mathematics of the Radon Transform, for example, Algebraic Reconstruction Technique (ART), Simultaneous Algebraic Reconstruction Technique (SART) and Filtered Back Projection (FBP). Unlike ART or SART which are iterative reconstruction algorithms, FBP is a direct method to reconstruct images. The reconstruction algorithm is important because it determines the quality of the images and the effectiveness of the defect inspection. We decide to implement FBP and compare the results with different kinds of filters. On the other hands, we also optimize the process of our program to reduce the execution times and make it more efficient.

This research was supported by the Ministry of Science and Technology of Taiwan, R.O.C., under Grants MOST 104-2221-E-002-133-MY2 and MOST 106-2221-E-002-220, and by Test Research, Jorjin Technologies, III, Egistec, D8AI, and LVI.

II. Methods

A. *X-ray images*

Before reconstruction, first we get the X-ray projection images through the rotational laminography. The structure of the projection system is shown in Fig. 1.

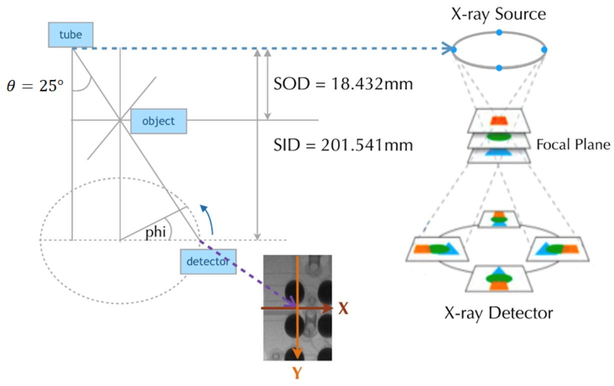

Fig. 1. System setup. Theta is the angle between tube to detector and vertical line. SOD is Source to Object Distance (=18.432mm); SID is Source to Image Distance (=201.541mm); and phi is the angle of rotation.

Through the relative motion between the source and the detector, the system can obtain a projection image including several objects belonging to different layers after rotating phi degrees (phi=360°/#projection images). To avoid overlapping the objects on the projection image, the X-rays from tube to detector are tilted 25 degrees. After the source and detector go through 360 degrees, we can reconstruct the original PCB according to these images and the relative positions of the objects.

B. Reconstruction

Filtered Back Projection is a method to solve the problem of simple back projection shown in Fig. 2 [6]. Compared with correct image, the reconstructed image of simple back projection is blurred and unable to fully present the appearance of the original object.

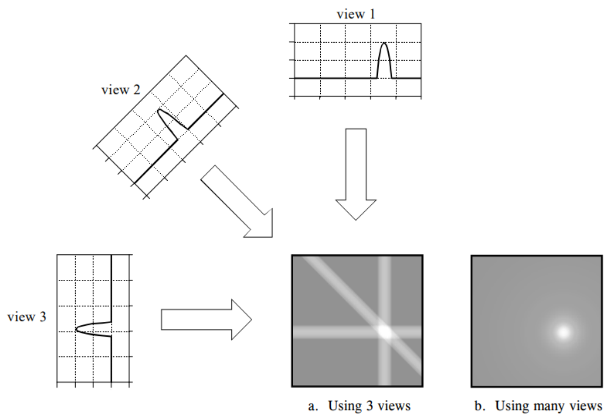

Fig. 2. Simple back projection. Back projection reconstructs an image by taking each view and smearing it along the path it was originally acquired. The resulting image is blurred compared with the correct image.

For FBP, each view is filtered before the back projection to counteract the blurring point spread function shown in Fig. 3. That is, each view is convolved with a filter kernel to create a set of filtered views. These filtered views are back projected to reconstruct the image which is much more exact than the result of simple back projection.

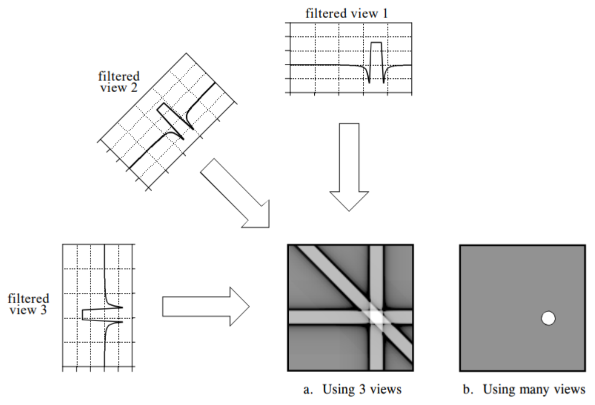

Fig. 3. Filtered back projection. Filtered back projection reconstructs an image by filtering each view before back projection. This removes the blurring seen in simple back projection, and results in an exact reconstruction of the image.

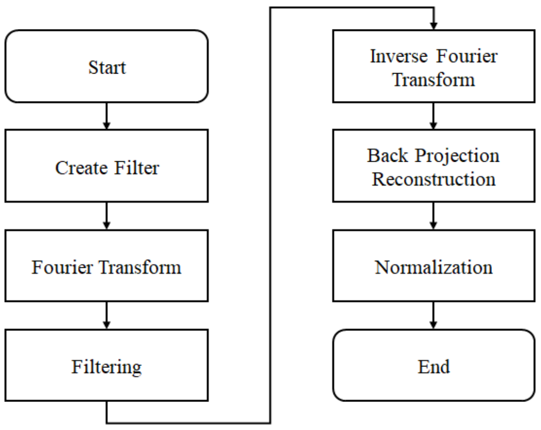

Fig. 4. The flowchart of Filtered Back Projection.

The flowchart of FBP is shown in Fig. 4. FBP combines the Radon Transform with Fourier Slice Theorem. In mathematics, first recall the definition for the 2D inverse Fourier Transform [1], as in (1).

$$f(x,y)=\frac{1}{(2\pi)^2}\iint_{R^2}F(u,v)e^{j(ux+vy)}dxdy \qquad (1)$$

Make a change of variable from rectangular to polar coordinates and replace $F(\varphi,\ \omega)$ with $G(\varphi,\omega)$, we get (2).

$$f(x,y)=\frac{1}{4\pi}\iint_{R^2}G(\varphi,\omega)e^{j\omega(x\sin\varphi+y\cos\varphi)}|w|dxdy \quad (2)$$

where $|w|$ is the determinant of the Jacobian of the change of variable from rectangular to polar coordinates. Notice that we have to multiply our projections by $|w|$ in the Fourier domain. Thus we can see $|w|$ as a filter. Equation (3) is called the filtered back projection at angle φ.

$$Q(\varphi,\omega)=G(\varphi,\omega)|w| \qquad (3)$$

The FBP algorithm is summarized in Algorithm 1 [4].

Algorithm 1 Filtered Back Projection

```
1: Compute a two-dimensional Fourier transform of each projection.
2: Multiply the Fourier transform of each projection by the weighting filter.
3: Compute the inverse two-dimensional Fourier transforms of the filtered projection.
4: Apply back projection to the processed projections in spatial domain to obtain the two-dimensional reconstructed object.
```

C. Filters

For FBP, filter plays a significant role by changing the views in two ways. First, the top of the pulse is made flat, resulting in the final back projection creating a uniform signal level within the circle. Second, negative spikes have been introduced at the sides of the pulse. When back projected, these negative regions counteract the blur near the target object [6].

There are different methods to filter the reconstructed image. We compare with different kinds of filters including Ramp filter, Hann filter, Hamming filter, cosine filter, and Shepp-Logan filter shown in Fig. 5.

1) Ramp filter

We expect that a high-pass filter would eliminate the artifact by amplifying high-frequency components in the back projection. Ramp filter is generally used in practice.

2) Hann filter

The statistical noise in the data manifests in Fourier space as high-frequency components. Thus the process of filtered back projection amplifies noise in the image. In order to reduce this effect, a range of modifications to the ramp filter can be used. The most commonly used of these is the Hann filter [5].

$$w(n) = \frac{1}{2}\left[1 + \cos\left(\frac{2\pi n}{N-1}\right)\right], 0 \le n \le N-1 \tag{4}$$

where N represents the width.

3) Hamming filter

The window with these particular coefficients was proposed by Richard W. Hamming [9]. The window is optimized to minimize the maximum (nearest) side lobe, giving it a height of about one-fifth that of the Hann window.

$$w(n) = 0.54 - 0.46 \times \cos\left(\frac{2\pi n}{N-1}\right), 0 \le n \le N-1 \tag{5}$$

where N represents the width.

4) cosine filter

$$w(n) = \cos\left(\frac{\pi n}{N-1}\right), 0 \le n \le N-1 \tag{6}$$

where N represents the width.

5) Shepp-Logan filter

$$w(n) = \frac{\sin\left(\frac{\pi n}{N-1}\right)}{\frac{\pi n}{N-1}}, 0 \le n \le N-1 \tag{7}$$

where N represents the width.

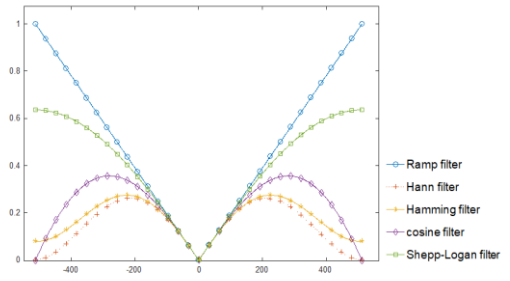

Fig. 5. Comparison with different kinds of filters. The filters include Ramp filter, Hann filter, Hamming filter, cosine filter, and Shepp-Logan filter.

For FBP filter, we generate a modified 2D Shepp-Logan filter where the periphery is steeper to increase the contrast of the reconstructed images. The 2D filters are shown in Fig. 6.

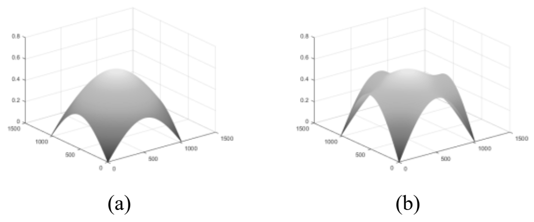

Fig. 6. (a) General 2D Shepp-Logan filter. (b) Modified 2D Shepp-Logan filter.

D. Back Projection

The relative motion between the source and detector generates projections with angle θ (=25°). Thus we generate a reconstruction architecture shown in Fig. 7.

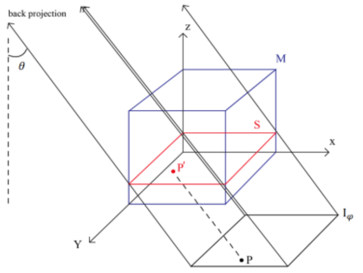

Fig. 7. The reconstuction architecture. Matrix M contains whole PCB; P is a point on projection image I_φ with angle φ; P' is a target point corresponding to P on slice S; and θ is the angle between tube to detector and vertical line.

3D matrix M contains our whole target PCB where I_φ is a projection image with rotated angle φ calculated in (8), where N_p is the number of projection images.

$$\varphi = \frac{360° \times i}{N_p}, \quad 0 < i < N_p \tag{8}$$

X-ray passes some voxels of the 3D matrix and the intensity of these voxels are cumulated into the corresponding pixel in the projection image I. We apply every I to reconstruct the slice S with different heights. The value of each pixel on different slices based on projection rays is calculated in (9).

$$f_{ij} = \sum_{k=1}^{R} \frac{p_k}{n_k},\ i = 1,2,\ldots,N,\ j = 1,2,\ldots,H \qquad (9)$$

f_{ij}: the value of i-th pixel on slice with height j.

N: total number of pixels on each slice.

H: total number of slices.

R: total number of projection rays that pass pixel ij.

p_k: the value of the k-th projection ray.

n_k: number of voxels passed by the k-th projection ray.

Based on Fig. 7, we can further define the coordinate of pixel on the slice with different heights corresponding to the pixel in projection images. According to an idea that the objects on focal plane stay consistent in all projections, we apply focal plane as a reference to derive other coordinates. The coordinate architecture is shown in Fig. 8.

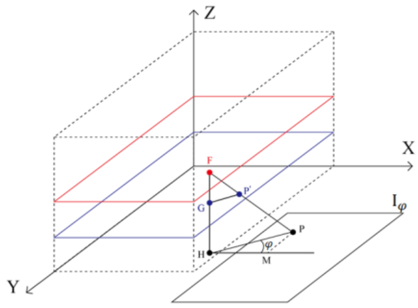

Fig. 8. The coordinate architecture. Find the pixel on $z{=}z_{p'}$ passed by a certain projection ray.

The point F is on the focal plane $z = z_F$, and P is the corresponding point on the projection image with rotated angle φ. The projection ray $\overline{\text{FP}}$ intersects with plane $z = z_{P'}$ at P' and $\overline{\text{FH}}$ is a perpendicular line to plane X-O-Y, which intersects with plane $z = z_{P'}$ at G. Number L_M represents the number of layers of PCB (=200 in our experiments).

The coordinates of P' is our target. Assume the coordinate of F to be

$$F(x_F, y_F, z_F)$$

We can get the coordinates of P, G, and H

$$P(x_F + L_M \times tan\theta \times cos\varphi, y_F - L_M \times tan\theta \times sin\varphi, 0)$$

$$G(x_F, y_F, z_{P'})$$

$$H(x_F, y_F, 0)$$

Triangle FGP' is similar to triangle FHP. Thus,

$$\frac{\overline{\text{GP'}}}{\overline{\overline{\text{HP}}}} = \frac{\overline{\text{FG}}}{\overline{\overline{\text{FH}}}}$$

$$\Rightarrow \begin{cases} \dfrac{x_F - x_{P'}}{x_F - (x_F + L_M \times tan\theta \times cos\varphi)} = \dfrac{z_F - z_{P'}}{z_F} \\ \dfrac{y_F - y_{P'}}{y_F - (y_F - L_M \times tan\theta \times sin\varphi)} = \dfrac{z_F - z_{P'}}{z_F} \end{cases}$$

$$\Rightarrow \begin{cases} x_{P'} = x_F + (1 - \dfrac{z_{P'}}{z_F}) \times L_M \times tan\theta \times cos\varphi \\ y_{P'} = y_F - (1 - \dfrac{z_{P'}}{z_F}) \times L_M \times tan\theta \times sin\varphi \end{cases}$$

Finally, we can get the target point coordinate

$$P'(x_F + (1 - \frac{z_{P'}}{z_F}) \times L_M \times tan\theta \times cos\varphi,$$
$$y_F - (1 - \frac{z_{P'}}{z_F}) \times L_M \times tan\theta \times sin\varphi, z_{P'})$$

E. *Contrast Enhancement*

The X-ray images and the reconstructed images are 16-bit in low contrast. We will convert the 16-bit images to 8-bit images to display on the screen. Rather than converting all the 16-bit intensity, we choose narrower range from half of minimum intensity to 1/128 of maximum intensity in 16-bit image to normalize to 8-bit image. As a result, it can enhance the contrast and preserve some information which may be lost if we use normal converting.

F. *Acceleration*

In addition to focusing on the quality of reconstructed images, we also pursue the efficiency of computing. At first, we consider the architecture of the program to optimize memory access. 2D images, 3D volumes, and other multi-dimensional data frequently require loops that sweep through an array to compute statistics, normalize values, or apply transfer functions [2]. Maintaining a multi-dimensional array within a single linear array is a common performance technique.

The array in original code is directly declared as a three-dimensional array. We declare the array as a one-dimensional array, and the index is still correctly mapped to three-dimensional array. It greatly improves the computational efficiency.

In addition, based on the program including many for-loops, we decide to make them parallel. We choose two tools, Intel® Threading Building Blocks (Intel® TBB) and Open Multi-Processing (OpenMP), and compare their execution times.

TBB is a library that supports scalable parallel programming using standard ISO (International Standards Organization) C++ code [3]. TBB does not require special languages or compilers. TBB is designed to promote scalable data parallel programming. Additionally, TBB fully supports nested parallelism, so users can build larger parallel components from smaller parallel components. To use the library, users specify tasks, not threads, and let the library map tasks onto threads efficiently.

OpenMP is an implementation of multithreading, a method of parallelizing whereby a master thread forks a specified number of slave threads and the system divides a task among them [7]. The threads then run concurrently, with the runtime environment allocating threads to different processors. The section of code that is meant to run in parallel is marked accordingly, with a compiler directive that will cause the threads to form before the section is executed.

With TBB and OpenMP, we not only change the original for-loop to the parallel for-loop, but also need to consider the architecture of the program. Based on characteristic of memory sequential accessing, row-major access is faster than column-major access. This characteristic leads us to access images in row-major parallel.

III. EXPERIMENTAL RESULT

A. Data

In our experiment, our sample contains 16 projection images with resolution 1000×1124 pixels reconstructing object with height 200 pixels. We will show the projection images and reconstructed images of the sample, and compare with the results by Volume Graphics.

B. The Projection Images and Reconstructed Images

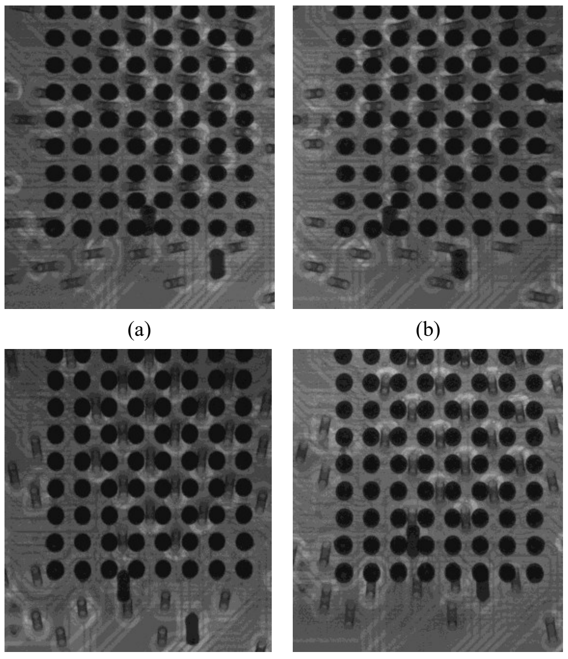
(a) (b) (c) (d)

Fig. 9. The projection images (totally 16 images). (a) The projection image with rotation angle 0°. (b) The projection image with rotation angle 90°. (c) The projection image with rotation angle 180°. (a) The projection image with rotation angle 270°.

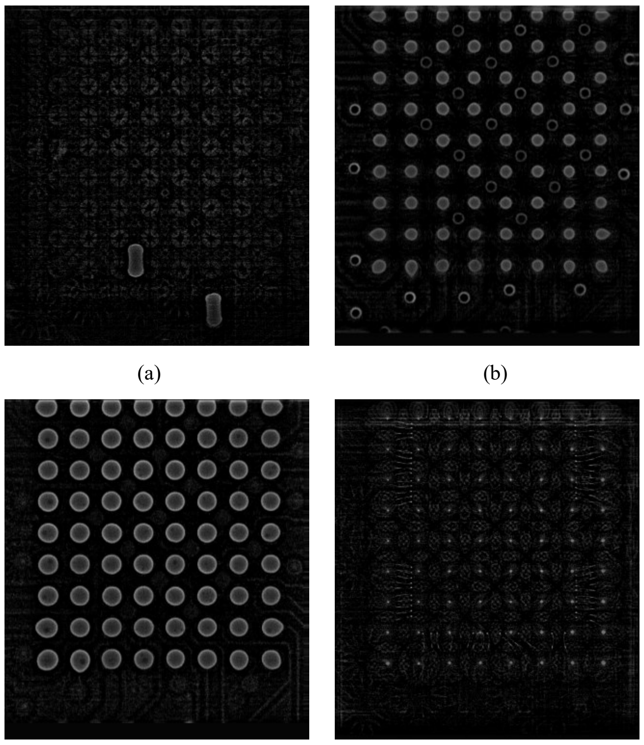
(a) (b) (c) (d)

Fig. 10. Our reconstructed images. (a) The reconstructed image with height 30 pixels. (b) The reconstructed image with height 90 pixels. (c) The reconstructed image with height 100 pixels. (d) The reconstructed image with height 130 pixels.

C. The Comparison of Execution Times

Because acceleration is one of our goals, we will calculate the execution times and observe the changes after memory access optimization and parallel for-loop. Our experimental environment is shown in Table I.

TABLE I. EXPERIMENTAL ENVIRONMENT1

Item	Details
CPU	Intel® Core i5-7500 3.4GHz processor
Memory	8GB
OS	Windows 10
Programming Language	C/C++, OpenCV 3.2.0

The comparison of execution times are shown in Tables II and III. We can get approximately four times acceleration in experimental environment1 after parallel for-loop, where OpenMP is slightly faster than Intel® TBB in performance.

TABLE II. EXECUTION TIMES OF MEMORY ACCESS OPTIMIZATION

Execution time (s)	Memory Access Optimization	
	Before	*After*
Read Projection Images	1.065762	1.409419
Create Filter	2.025230	0.522070
Fourier Transform	19.163595	9.619489
Back Projection	55.174862	46.768969
Normalize and Save Images	5.794322	4.181767
Total	83.223771	62.501714

TABLE III. EXECUTION TIMES OF PARALLEL FOR-LOOP (ACCELERTION OF ABOUT 3.8 TIMES (=62.5/16.3))

Execution time (s)	Parallel For-Loop	
	Intel® TBB	*OpenMP*
Read Projection Images	0.455022	0.417034
Create Filter	0.228068	0.337011
Fourier Transform	2.479723	2.552796
Back Projection	11.982365	11.521719
Normalize and Save Images	1.333518	1.425259
Total	16.478697	16.253818

D. Comparison of FBP with Volume Graphics

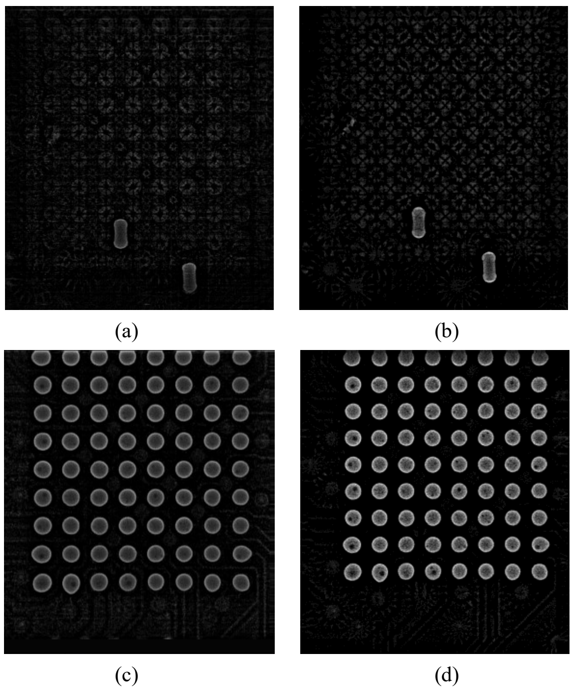
(a) (b) (c) (d)

Fig. 11. The reconstructed images. (a) The reconstructed image with height 30 pixels by our method. (b) The reconstructed image with height 30 pixel by Volume Graphicss. (c) The reconstructed image with height 100 pixels by our method. (d) The reconstructed image with height 100 pixels by Volume Graphics.

TABLE IV. EXPERIMENTAL ENVIRONMENT2

Item	Experimental Environment	
	Our Environment	*Environment of Test Research Incorporation (TRI)*
CPU	Intel® Core™ i7-8700K 3.70GHz processor	Intel® Xeon E5-2687 2.53GHz processor
Memory	32GB	96GB
OS	Windows 10	Windows 7
Programming Language	C/C++, OpenCV 3.2.0	C/C++

TABLE V. EXECUTION TIMES OF OUR RESULTS AND VOLUME GRAPHICS RESULTS

Execution time (s)	Methods	
	Our FBP	*Volume Graphics*
Total	4.0	2.5

IV. CONCLUSION AND FUTURE WORK

We implement Filtered Back Projection (FBP) with modified Shepp-Logan filter to reconstruct the three-dimensional PCB solder balls with two-dimensional projection images. Our reconstructed images have similar quality to Volume Graphics, but the low-contrast problem still exists for the void in solder ball in our results.

We also apply Intel® TBB and OpenMP to accelerate our program. Our hardware is significantly inferior to TRI, thus our execution time is slower than Volume Graphics. In the future, we aim to further accelerate by computing with Graphics Processing Unit (GPU) using Computed Unified Device Architecture (CUDA).

REFERENCES

[1] C. Pan, “Image Projections and the Radon Transform,” https://www.clear.rice.edu/elec431/projects96/DSP/bpanalysis.html, 1996.

[2] D. R. Nadeau, “C/C++ Tip: How to Loop through Multi-Dimensional Arrays Quickly,” http://nadeausoftware.com/articles/2012/06/c_c_tip_how_loop_through_multi_dimensional_arrays_quickly#Benchmarkresultsndashcompiledwithoptimizations, 2012.

[3] Intel, “Introducing the Intel® Threading Building Blocks (Intel® TBB),” https://software.intel.com/en-us/node/506042#introducing_main, 2018.

[4] P. A. Penczek, “Fundamentals of three-dimensional reconstruction from projections,” https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3165033/, 2010.

[5] R. Badawi, “Introduction to PET Physics,” http://depts.washington.edu/nucmed/IRL/pet_intro/toc.html, 1999.

[6] S.W. Smith, “The Scientist & Engineer's Guide to Digital Signal Processing,” http://www.dspguide.com/ch25/5.htm, 1997.

[7] Wikipedia, “OpenMP,” https://en.wikipedia.org/wiki/OpenMP, 2018.

[8] Wikipedia, “Tomographic Reconstruction,” https://en.wikipedia.org/wiki/Tomographic_reconstruction, 2018.

[9] Wikipedia, “Window Function,” https://en.wikipedia.org/wiki/Window_function#Hamming_window, 2018.