3D Reconstruction with X-Ray Images

Presenter: 方郁婷

E-mail: s830s200s@gmail.com

Phone: 0933826120

OUTLINE

- Purpose
- Reconstruction
- ART (Algebraic Reconstruction Technique)
    - Additive
    - Multiplicative
    - Compare
    - How To Improve?

PURPOSE

- Implement the 3-dimensional reconstruction with X-ray images.
  - Why?

    There are several kinds of defects in PCB.

    Many defects cannot be seen by human eyes.

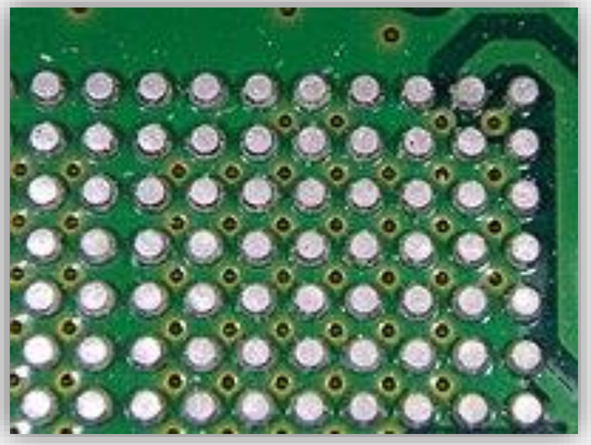
  - Method for inspecting PCB
    - Automated Optic Inspection (AOI)
    - Solder Paste Inspection (SPI)
    - **Automated X-ray Inspection (AXI)**
      - Algebraic Reconstruction Technique (ART)
      - Simultaneous Algebraic Reconstruction Technique (SART)
      - Filtered Back Projection (FBP)

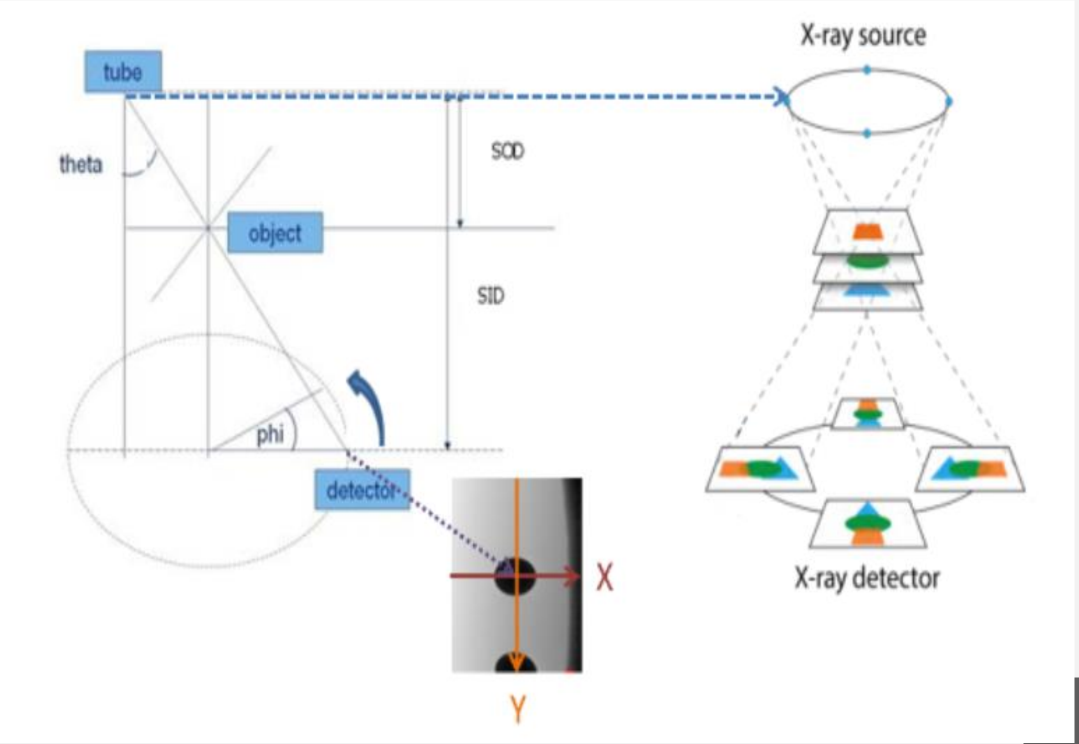
PCB: Printed Circuit Board

SOD: Source to Object Distance
SID: Source to Image Distance

RECONSTRUCTION

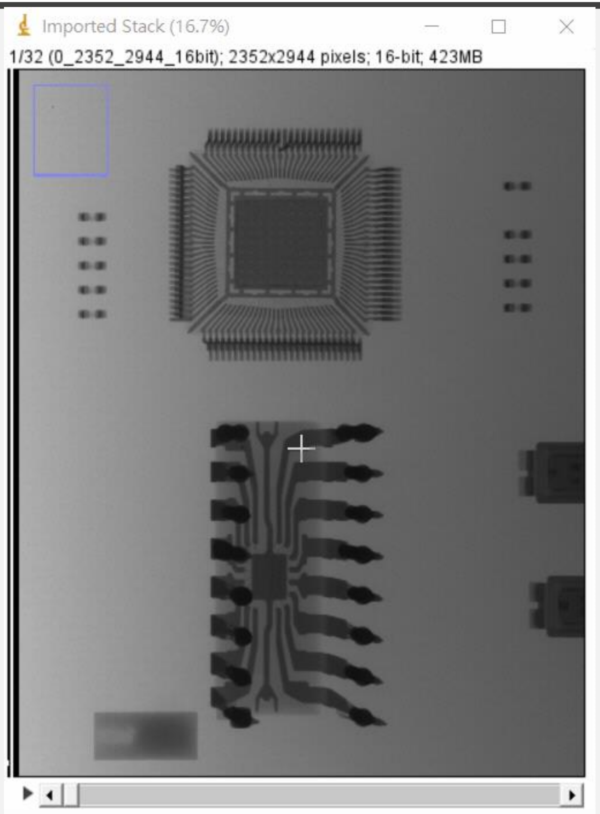
- Reconstruction of 3D solder ball



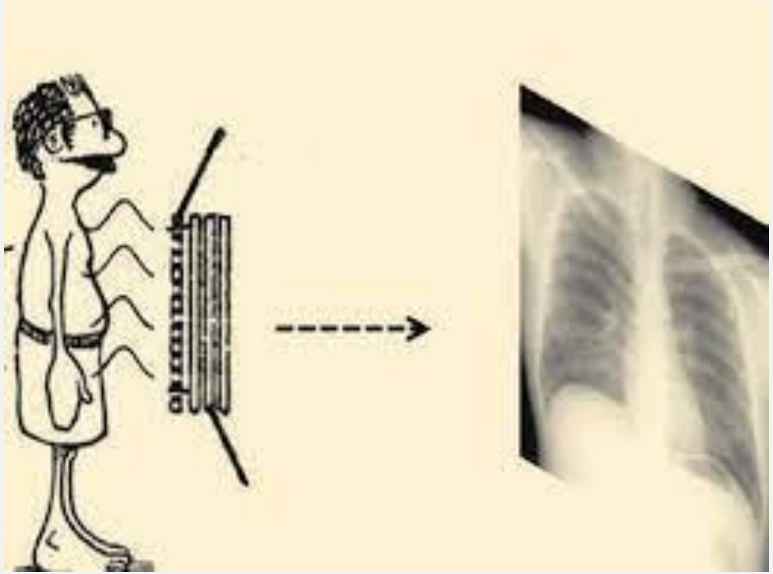


RECONSTRUCTION

- 32 Projections



X-RAY



X-ray is 3D to 2D

We want to convert these 2D images into 3D images

How? => ART

ART (ALGEBRAIC RECONSTRUCTION TECHNIQUE)

- A classic iterative reconstruction method.
- Additive:

$$f_{ij}^{q+1} = f_{ij}^{q} + \frac{g_j - \sum_{i=1}^{N} f_{ij}^{q}}{N}$$

q: iteration

g_j: The measured data for a projection

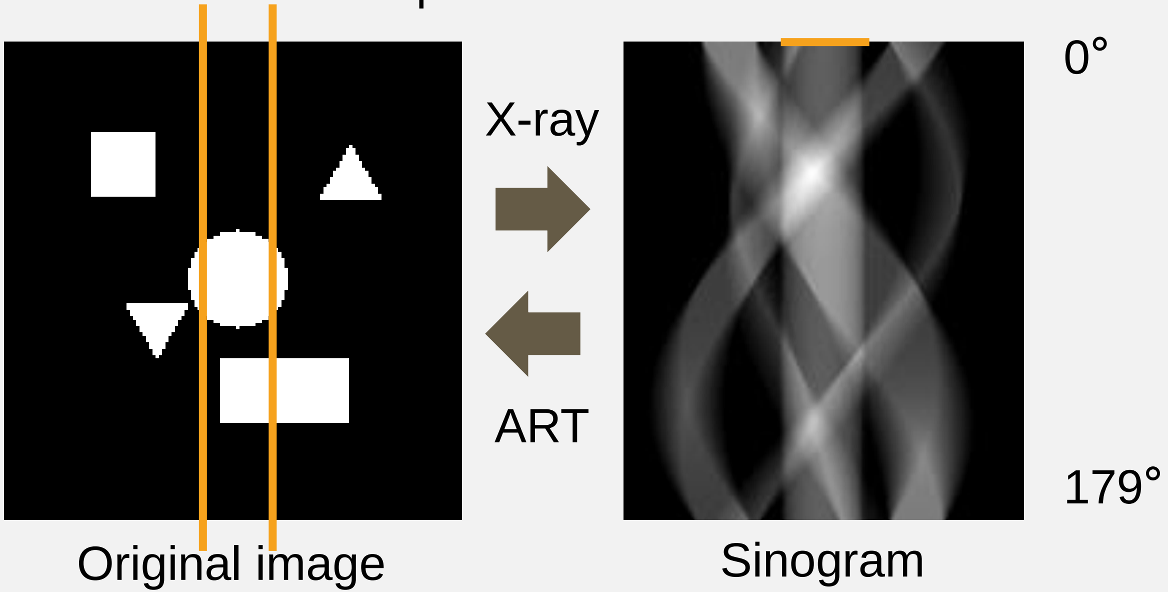
$\sum_{i=1}^{N} f_{ij}^{q}$: The sum of the reconstructed elements along the ray

N: The reconstruction elements

f_{ij}: An element along the jth line forming the projection ray g_j

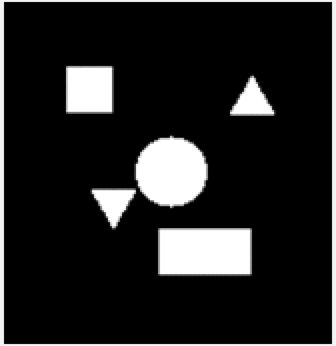
ADDITIVE ART

Example: convert 1D to 2D

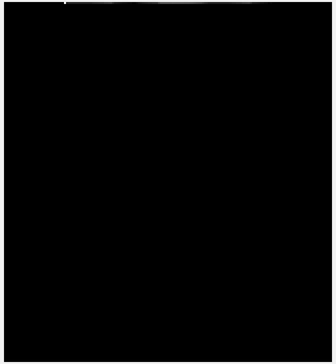


ART: Algebraic Reconstruction Technique
GIF: Graphics Interchange Format

ADDITIVE ART GIF



Original image

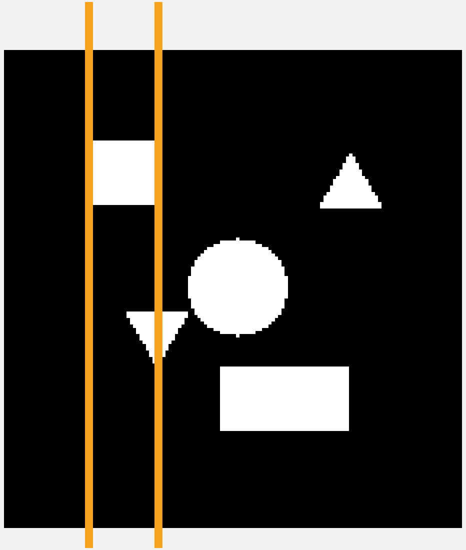


0°

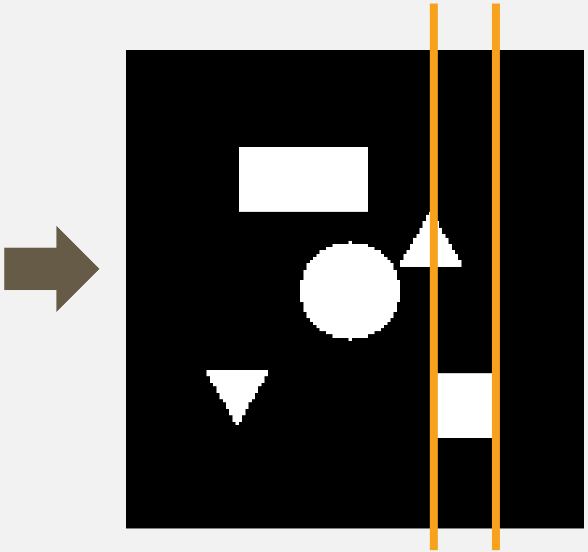
179°

Sinogram

WHY ONLY 179 DEGREE

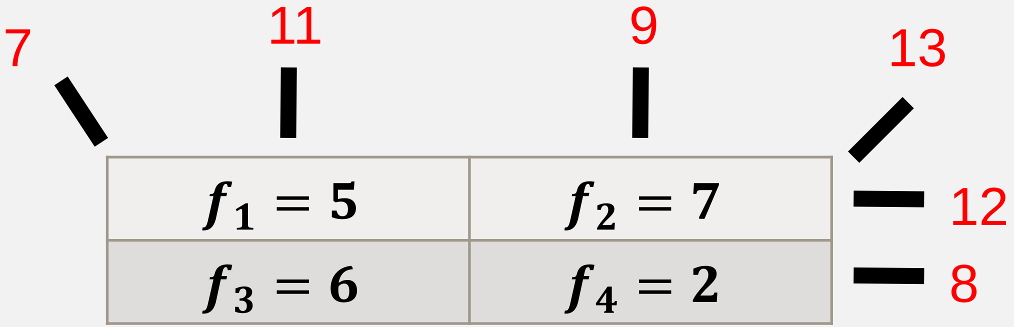


Original image



A picture turned 180 degrees

ADDITIVE ART



$$
\begin{aligned}
f_1 + f_2 &= 12 \\
f_1 + f_3 &= 11 \\
f_1 + f_4 &= 7 \\
f_2 + f_3 &= 13 \\
f_2 + f_4 &= 9 \\
f_3 + f_4 &= 8
\end{aligned}
$$

Add all directions
Just like the above picture rotation
and column addition

ADDITIVE ART

- Vertical

5	7
6	2

11 9

0 0

0	0
0	0

$$f_1 = 0 + \frac{\color{red}{11} - 0}{2} = 5.5$$

$$f_2 = 0 + \frac{\color{red}{9} - 0}{2} = 4.5$$

$$f_3 = 0 + \frac{\color{red}{11} - 0}{2} = 5.5$$

$$f_4 = 0 + \frac{\color{red}{9} - 0}{2} = 4.5$$

ADDITIVE ART

- Horizontal

5	7
6	2

5.5	4.5	—	10	12
5.5	4.5	—	10	8

$$f_1 = 5.5 + \frac{12 - 10}{2} = 6.5$$

$$f_2 = 4.5 + \frac{12 - 10}{2} = 5.5$$

$$f_3 = 5.5 + \frac{8 - 10}{2} = 4.5$$

$$f_4 = 4.5 + \frac{8 - 10}{2} = 3.5$$

ADDITIVE ART

- Diagonal

7
10

13
10

6.5	**5.5**
4.5	**3.5**

5	**7**
6	**2**

$$f_1 = 6.5 + \frac{7 - 10}{2} = 5$$

$$f_2 = 5.5 + \frac{13 - 10}{2} = 7$$

$$f_3 = 4.5 + \frac{13 - 10}{2} = 6$$

$$f_4 = 3.5 + \frac{7 - 10}{2} = 2$$

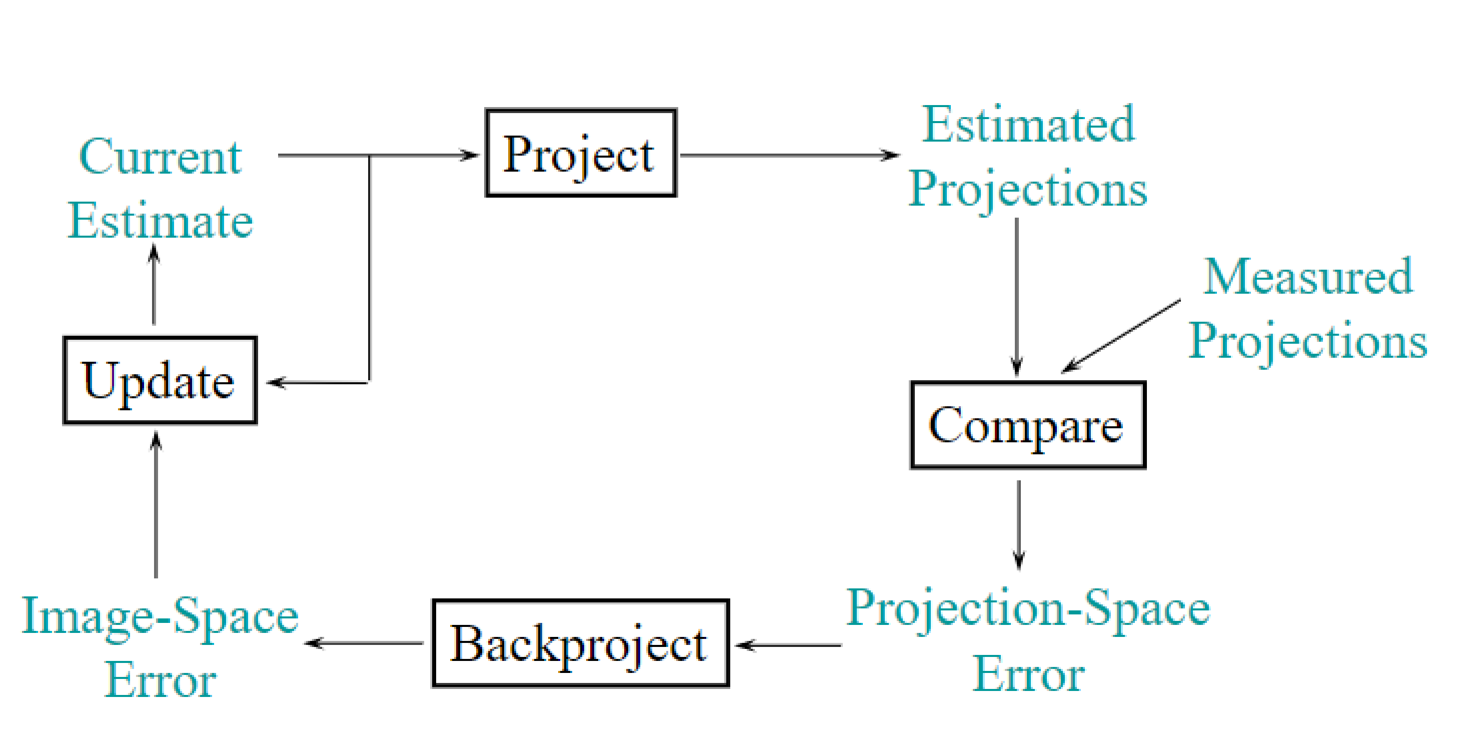
ADDITIVE ART

- Final

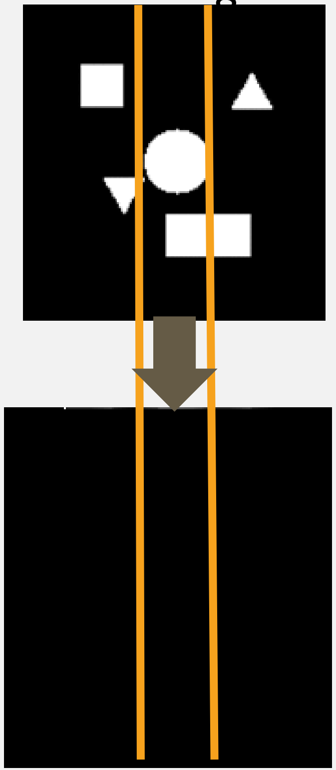
5	7
6	2

5	7
6	2

ITERATIVE RECONSTRUCTION



CODE

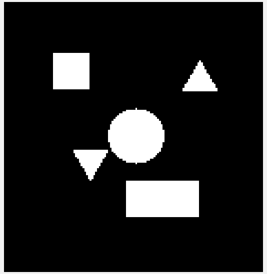



```
for (int angle = 0; angle < 180; angle++) {
    //旋轉圖片(0 ~ 179度)
    rotatedImg = rotate(oriImg, angle);

    float originalColSum;
    //遍歷每個col
    for (int col = 0; col < rotatedImg.cols; col++) {
        //計算每個colSum
        originalColSum = 0;
        for (int row = 0; row < rotatedImg.rows; row++) {
            originalColSum += rotatedImg.at<uchar>(row, col);
        }
        //算完後要與row平均(防止溢位)
        colAddImg.at<float>(angle, col) = originalColSum / oriRows;
    }
}
```

CODE

colAddImg



results

```cpp
Mat results(oriRows, colAddImg.cols, CV_32FC1, Scalar(0));

double anglePerPhoto = 11.25;
int totIter = 100;
for (int iter = 0; iter < totIter; iter++) {
    for (double angle = 0; angle < 180; angle += anglePerPhoto) {
        for (int col = 0; col < results.cols; col++) {
            float colSum = colAddImg.at<float>(angle, col) * oriRows;
            float newColSum = 0;
            for (int row = 0; row < results.rows; row++) {
                newColSum += results.at<float>(row, col);
            }
            for (int row = 0; row < results.rows; row++) {
                results.at<float>(row, col) += ((colSum - newColSum) / oriRows);
            }
        }
        results = rotate(results, anglePerPhoto);
    }
    results = rotate(results, 180);
}
```

CODE

```
Mat results(oriRows, colAddImg.cols, CV_32FC1, Scalar(0));

double anglePerPhoto = 11.25;
int totIter = 100;
for (int iter = 0; iter < totIter; iter++) {
    for (double angle = 0; angle < 180; angle += anglePerPhoto) {
        for (int col = 0; col < results.cols; col++) {
            float colSum = colAddImg.at<float>(angle, col);
            float newColSum = 0;
            for (int row = 0; row < results.rows; row++) {
                newColSum += results.at<float>(row, col);
            }
            for (int row = 0; row < results.rows; row++) {
                results.at<float>(row, col) += ((colSum - newColSum) / oriRows);
            }
        }
        results = rotate(results, anglePerPhoto);
    }
    results = rotate(results, 180);
}
```

p_j

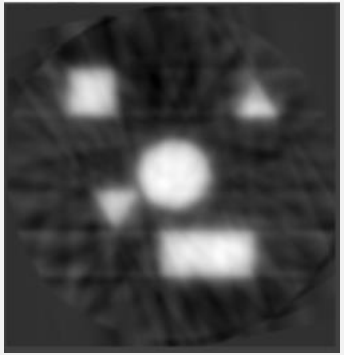
$\sum_{i=1}^{N} f_{ij}^{q}$

f_{ij}^{q} p_j $\sum_{i=1}^{N} f_{ij}^{q}$ N

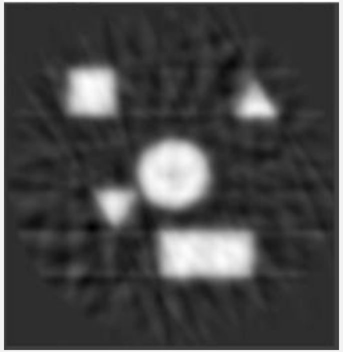
$$f_{ij}^{q+1} = f_{ij}^{q} + \frac{p_j - \sum_{i=1}^{N} f_{ij}^{q}}{N}$$

ADDITIVE ART

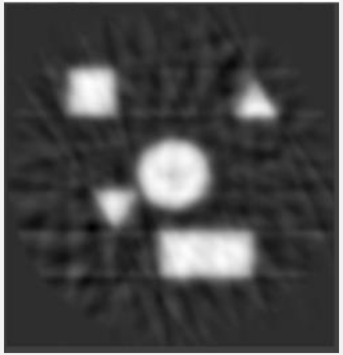
- 16 Projections



1 Iteration



100 Iterations

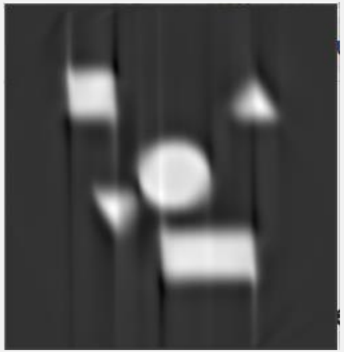


500 Iterations

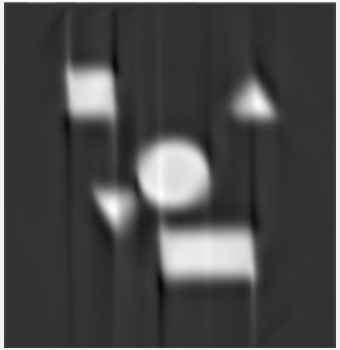
ADDITIVE ART

- 180 Projections

1 Iteration

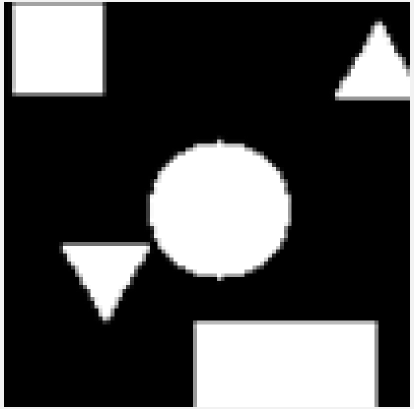


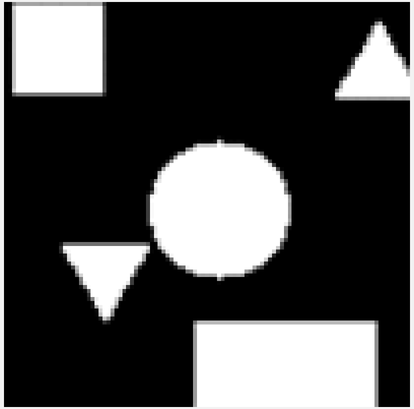
100 Iterations

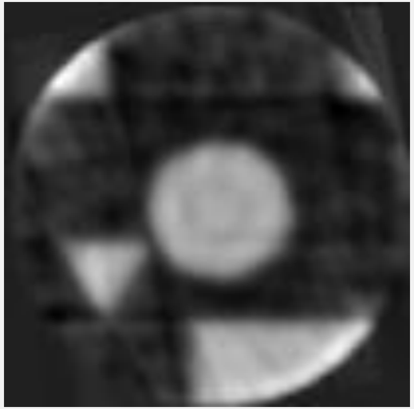


500 Iterations

CLIP







Graphs that are out of range will be clipped

MULTIPLICATIVE ART

$$f_{ij}^{q+1} = \frac{g_j}{\sum_{i=1}^{N} f_{ij}^{q}} f_{ij}^{q}$$

q: iteration

g_j: The measured data for a projection

$\sum_{i=1}^{N} f_{ij}^{q}$: The sum of the reconstructed elements along the ray

f_{ij}: An element along the jth line forming the projection ray g_j

MULTIPLICATIVE ART

- Vertical

$\frac{11}{2}$ $\frac{9}{2}$

1	1
1	1

5	7
6	2

$$f_1 = \frac{11}{2} * 1 = 5.5$$

$$f_2 = \frac{9}{2} * 1 = 4.5$$

$$f_3 = \frac{11}{2} * 1 = 5.5$$

$$f_4 = \frac{9}{2} * 1 = 4.5$$

MULTIPLICATIVE ART

- Horizontal

5	7
6	2

5.5	4.5	10	12
5.5	4.5	10	8

$$f_1 = \frac{12}{10} * 5.5 = 6.6$$

$$f_2 = \frac{12}{10} * 4.5 = 5.4$$

$$f_3 = \frac{8}{10} * 5.5 = 4.4$$

$$f_4 = \frac{8}{10} * 4.5 = 3.6$$

MULTIPLICATIVE ART

- Diagonal

7

10.2

13

9.8

6.6	**5.4**
4.4	**3.6**

5	**7**
6	**2**

$$\boldsymbol{f_1 = \frac{7}{10.2} * 6.6 = 4.53}$$

$$\boldsymbol{f_2 = \frac{13}{9.8} * 5.4 = 7.16}$$

$$\boldsymbol{f_3 = \frac{13}{9.8} * 4.4 = 5.84}$$

$$\boldsymbol{f_4 = \frac{7}{10.2} * 3.6 = 2.47}$$

MULTIPLICATIVE ART

- Final

4.53	**7.16**
5.84	**2.47**

5	**7**
6	**2**

CODE

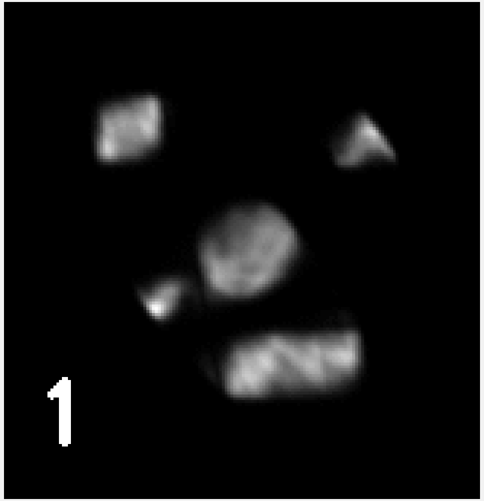
```
//AART
results.at<float>(row, col) += ((colSum - newColSum) / results.rows);
```

```
//MART
if (newColSum == 0) newColSum = 1; //防止除以0
results.at<float>(row, col) *= (colSum / newColSum);
```

$$f_{ij}^{q+1} = \frac{g_j}{\sum_{i=1}^{N} f_{ij}^{q}} f_{ij}^{q}$$

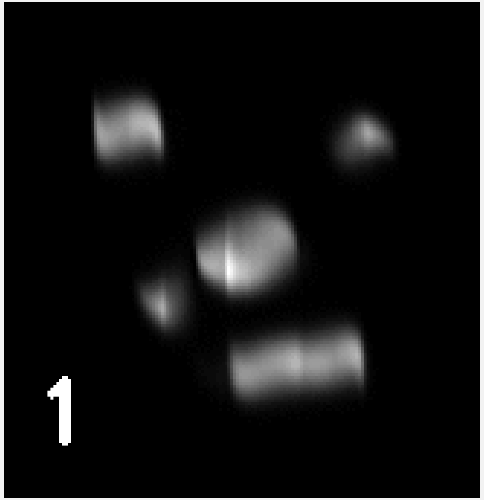
MULTIPLICATIVE ART

- 16 Projections



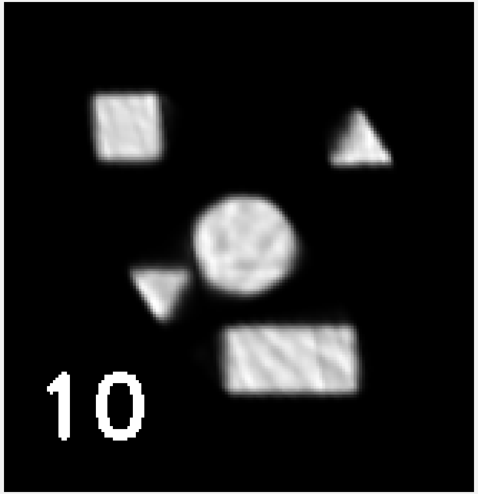
MULTIPLICATIVE ART

- 180 Projections

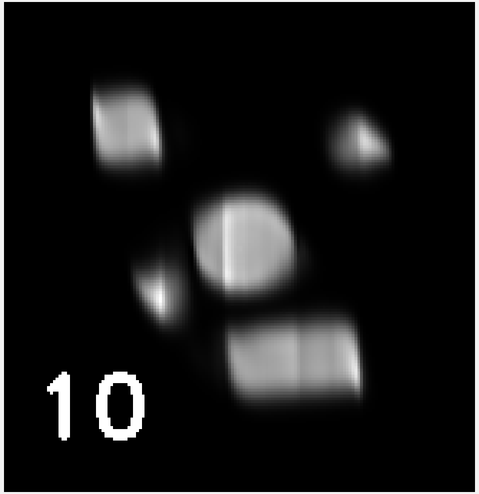


MULTIPLICATIVE ART

- 16 Projections

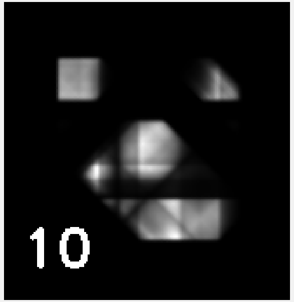


- 180 Projections

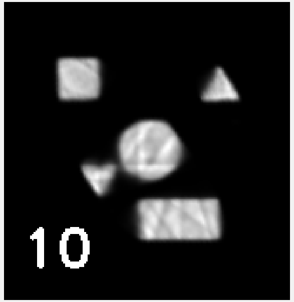


COMPARE PROJECTIONS

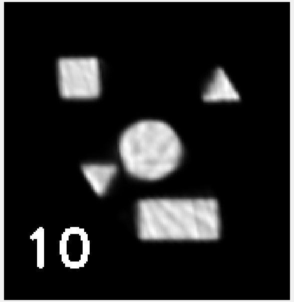
- 4



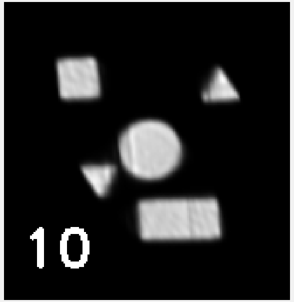
- 10



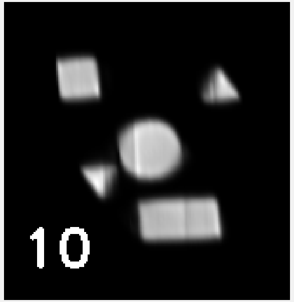
- 16



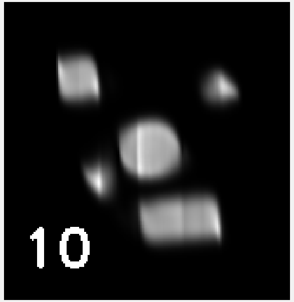
- 30



- 60

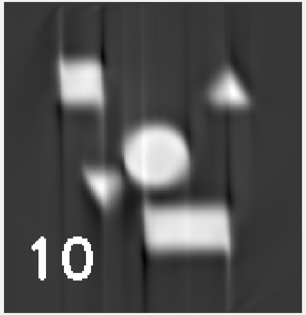


- 180

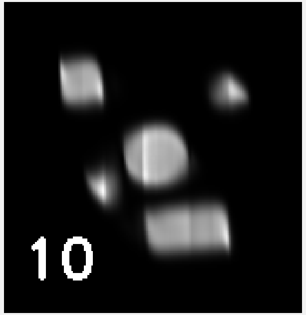


ADDITIVE VS MULTIPLICATIVE

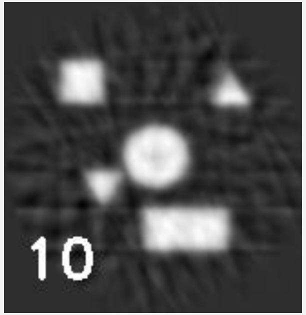
- Multiplicative ART has **better image quality**.
- Given less projections, Additive ART gives a more complete picture, so does Multiplicative ART.



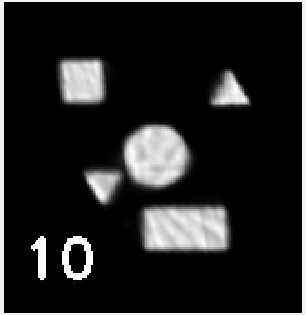
Additive
180 Projections
10 Iterations



Multiplicative
180 Projections
10 Iterations



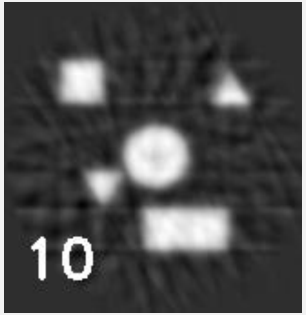
Additive
16 Projections
10 Iterations



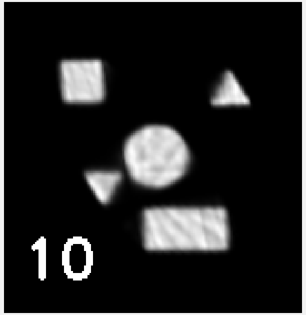
Multiplicative
16 Projections
10 Iterations

ADDITIVE VS MULTIPLICATIVE

More smooth



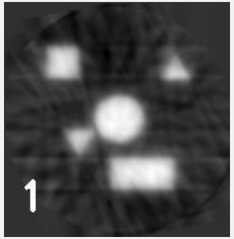
More clear

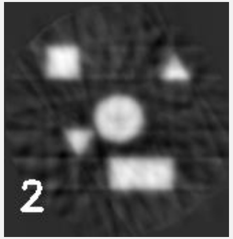


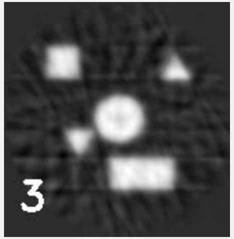
Can we combine the two feature?

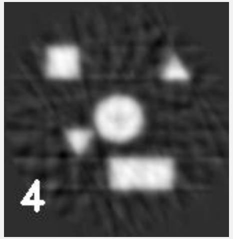
COMBINE THE TWO METHOD

Additive

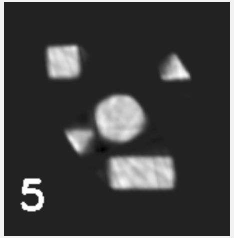


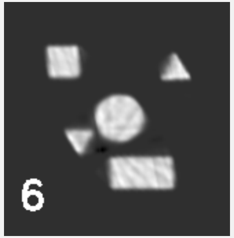


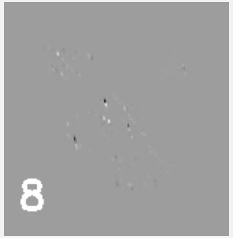




Multiplicative

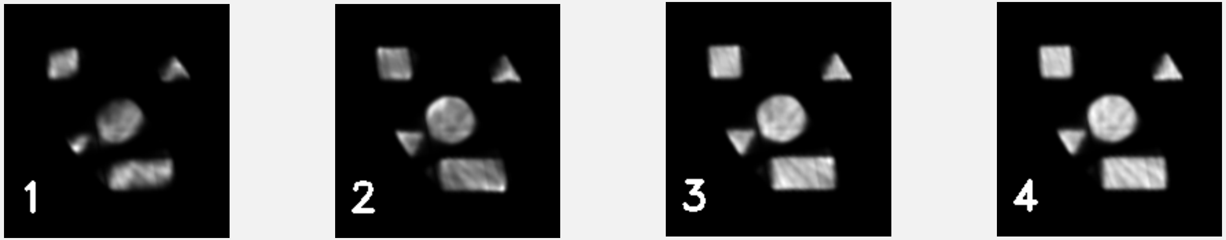




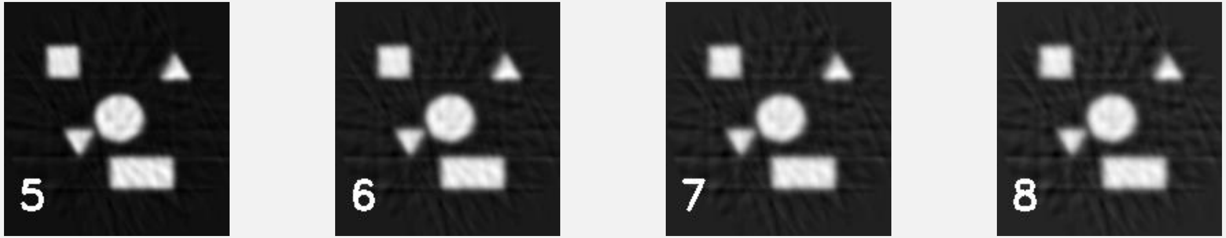


COMBINE THE TWO METHOD

Multiplicative



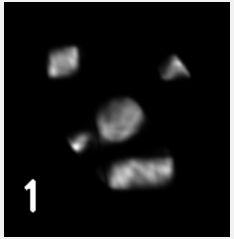
Additive

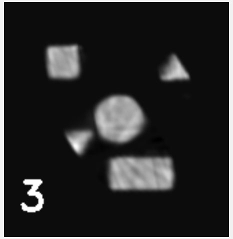


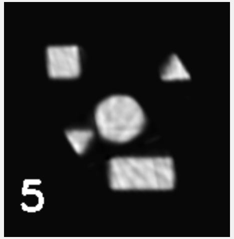
COMBINE THE TWO METHOD

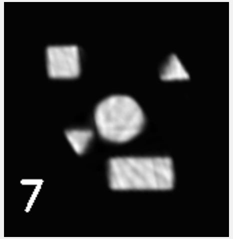
- **Switch Method**

Multiplicative

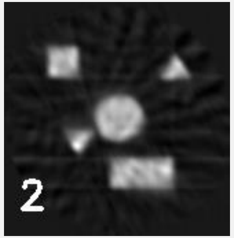


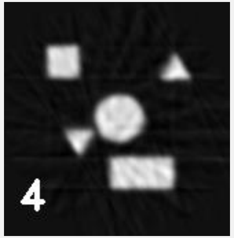


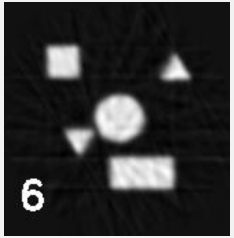


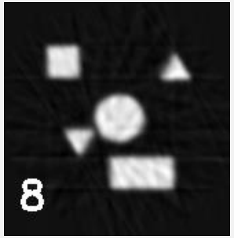


Additive









COMBINE THE TWO METHOD

- What is the difference between the two methods

Additive	Multiplicative
Numbers can be negative	Numbers are always greater than or equal to 0

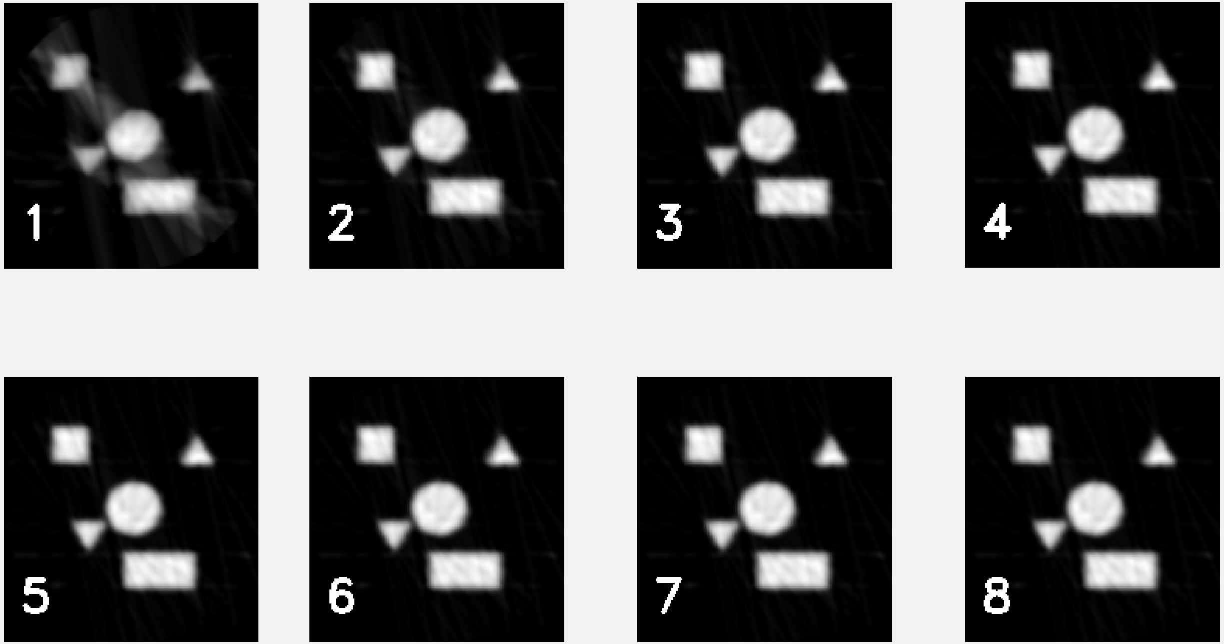
CODE

```
//AART
results.at<float>(row, col) += ((colSum - newColSum) / results.rows);
```

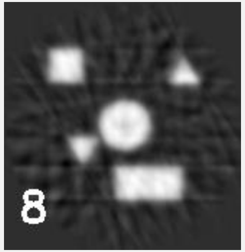
```
//AART
results.at<float>(row, col) += ((colSum - newColSum) / results.rows);
if (results.at<float>(row, col) < 0) results.at<float>(row, col) = 0;
```

COMBINE THE TWO METHOD

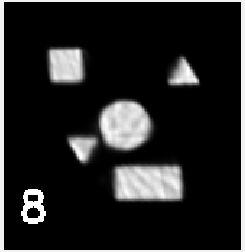
Additive
(non-negative number)



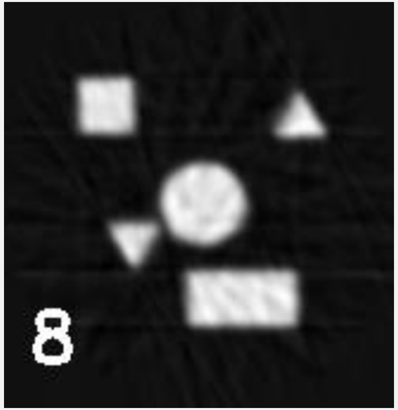
COMBINE



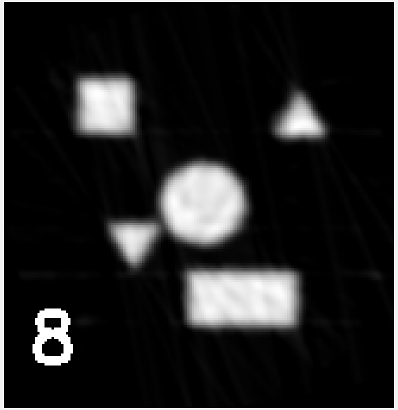
+



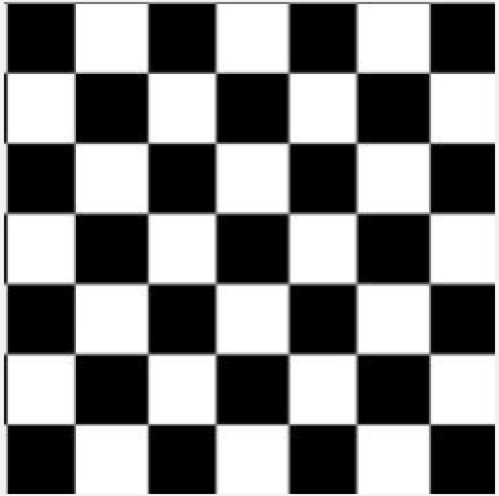
Switch Method



Additive
(non-negative number)

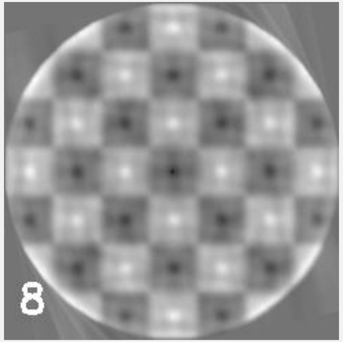


ANOTHER GRAPH

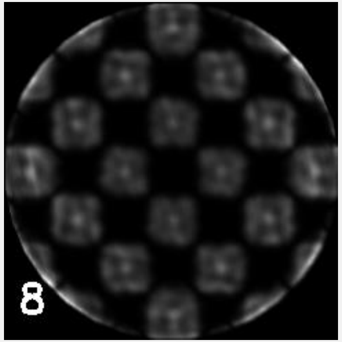


COMBINE

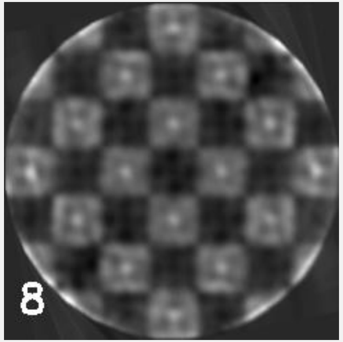
Additive



Multiplicative



Switch Method



Additive
(non-negative number)