3D Reconstruction with X-Ray Images

Presenter: 方郁婷

E-mail: s830s200s@gmail.com

Phone: 0933826120

OUTLINE

- Purpose
- Reinstruction
- ART (Algebraic Reconstruction Technique)
  - additive

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.

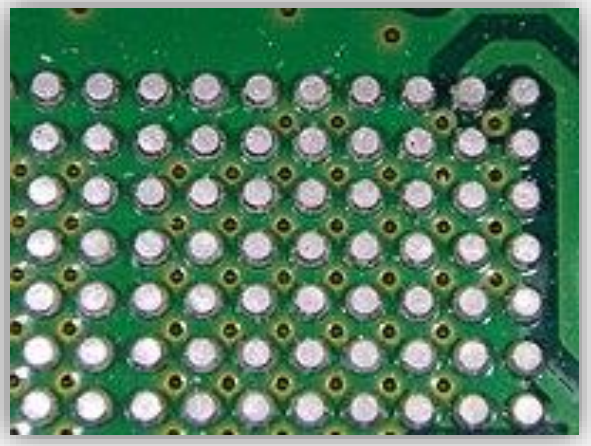
  - Method for inspecting PCB
    - Automated Optic Inspection (AOI)
    - Solder Paste Inspection (SPI)
    - **Automated X-ray Inspection (AXI)**
      - Algebraic Reconstruction Technique (ART)
      - Simultaneous Algebraic Reconstruction Technique (SART)
      - Filtered Back Projection (FPB)

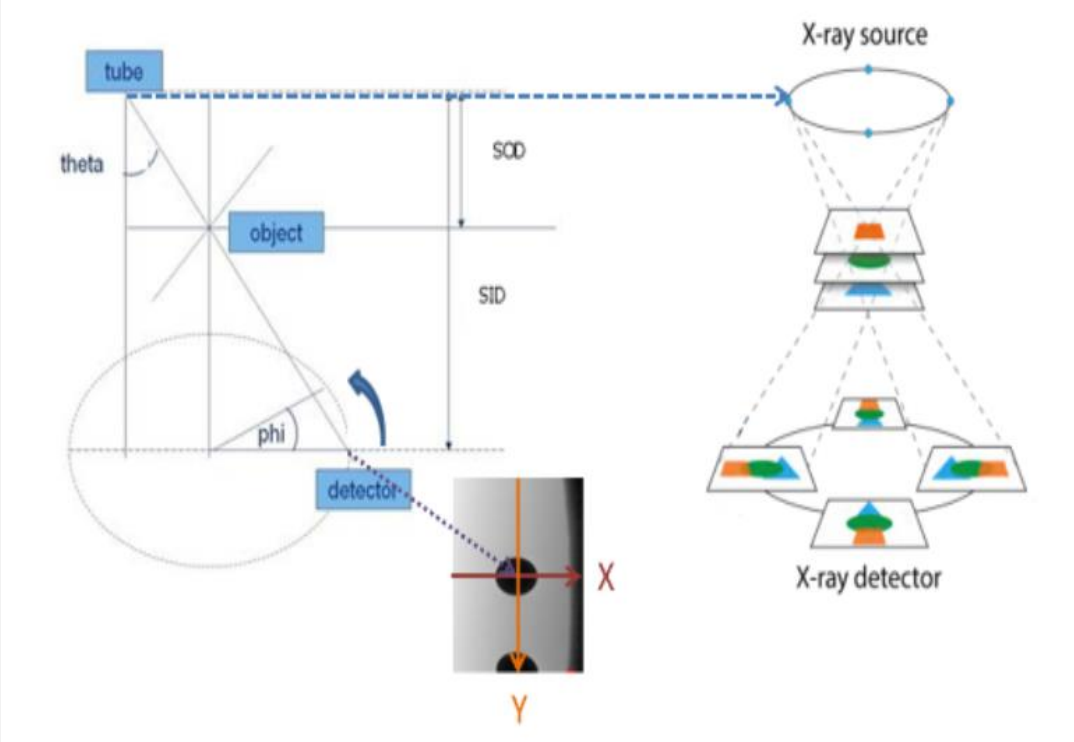
PCB: Printed Circuit Board

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

REINSTRUCTION

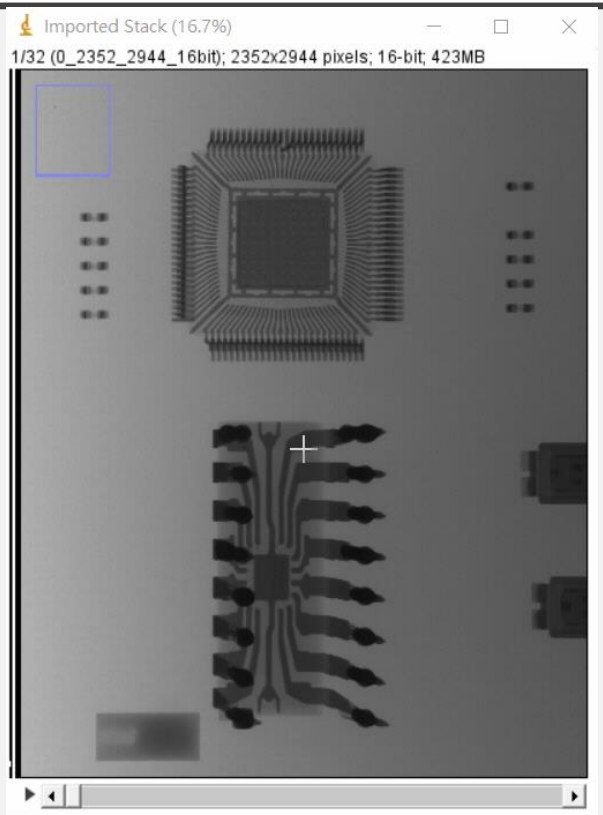
- Reconstruction of 3D solder ball



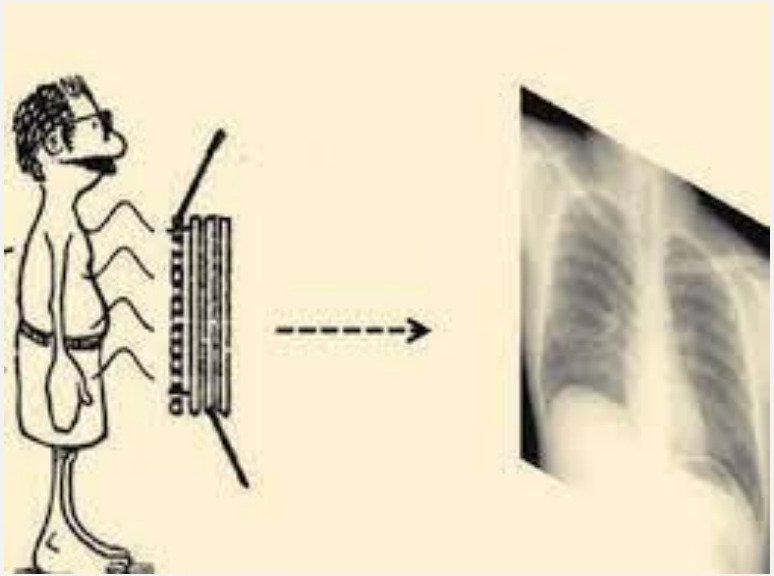


REINSTRUCTION

- 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)

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

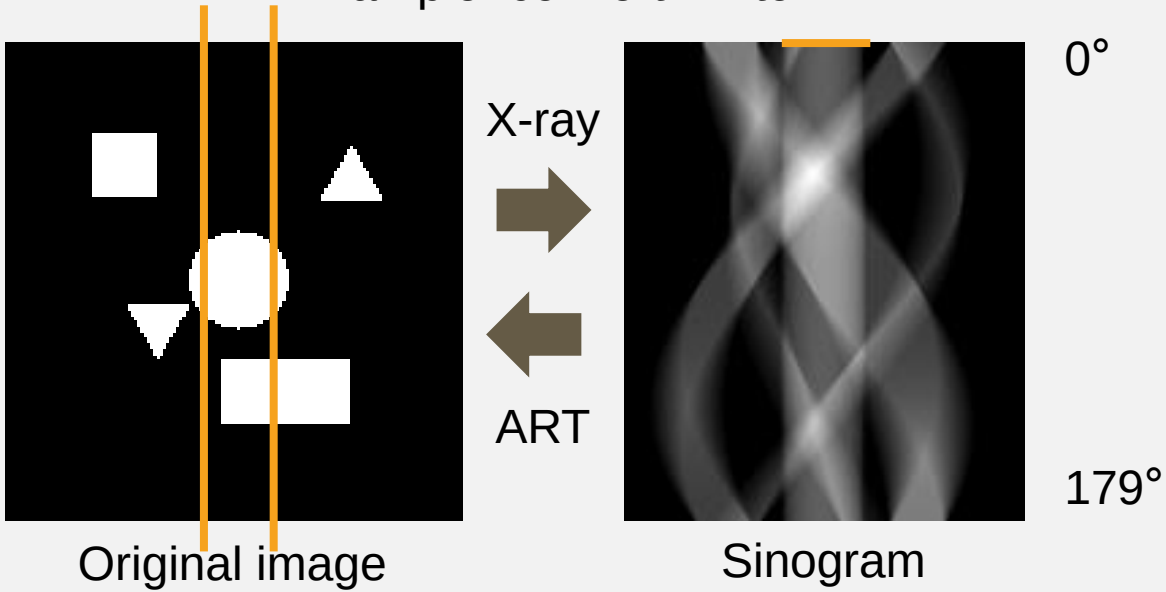
$\sum_{i=1}^{N} f_{ij}^{q}$: The sum of the reconstructed elements along the ray

N: The reconstruction elements

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

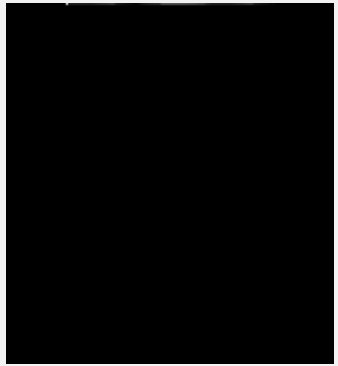
ADDITIVE ART

Example: convert 1D to 2D



ART: Algebraic Reconstruction Technique
GIF: Graphics Interchange Format

ADDITIVE ART GIF



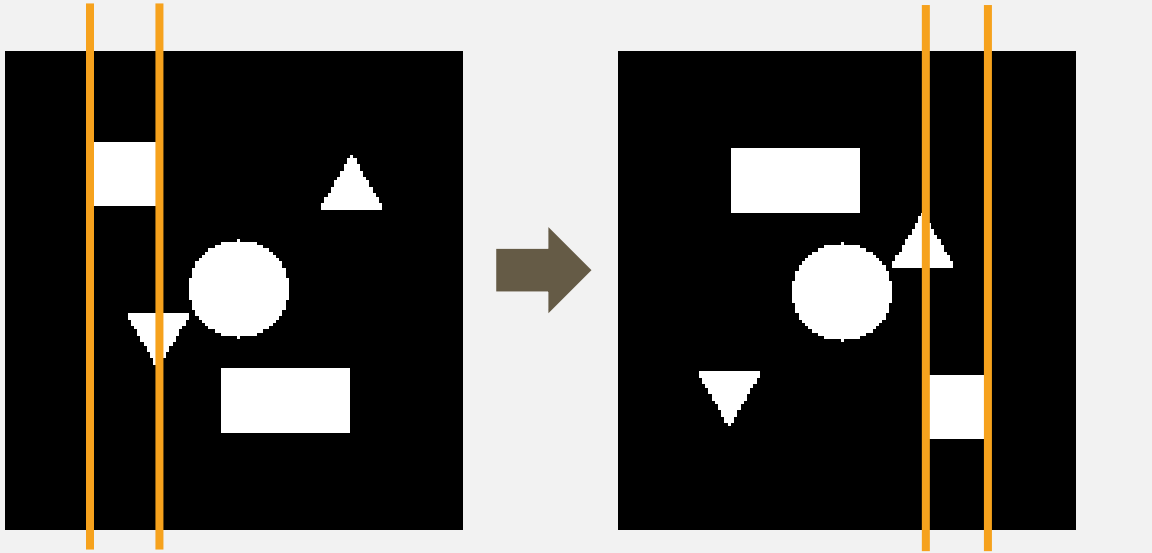
0°

179°

Original image

Sinogram

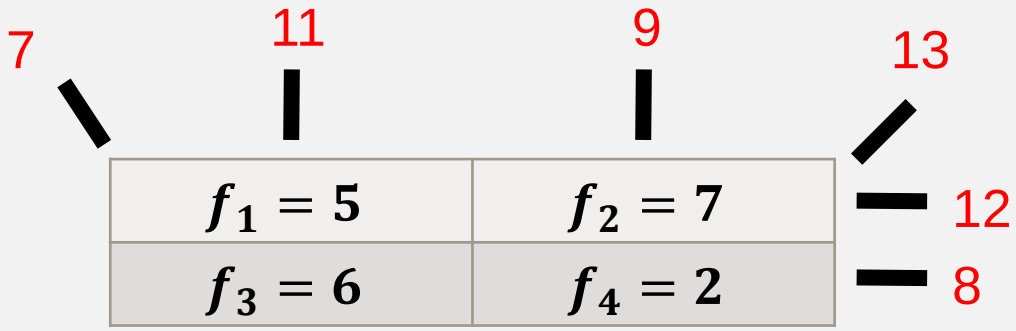
WHY ONLY 179 DEGREE



Original image

A picture turned 180 degrees

ADDITIVE ART



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

$$f_1 + f_2 = 12$$
$$f_1 + f_3 = 11$$
$$f_1 + f_4 = 7$$
$$f_2 + f_3 = 13$$
$$f_2 + f_4 = 9$$
$$f_3 + f_4 = 8$$

ADDITIVE ART

- Vertical

5	7
6	2

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

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

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

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

ADDITIVE ART

- Horizontal

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

5.5	**4.5**	=	10	12
5.5	**4.5**	=	10	8

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

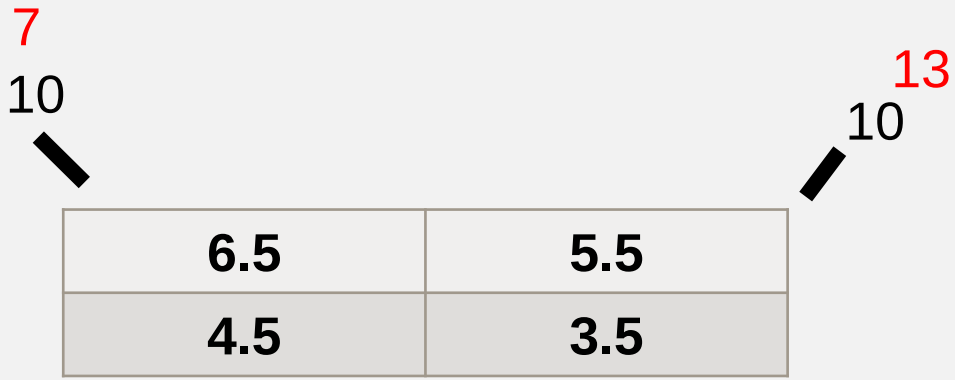
$$f_2 = 4.5 + \frac{12 - 10}{2} = 5.5$$

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

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

ADDITIVE ART

- Diagonal



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

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

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

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

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

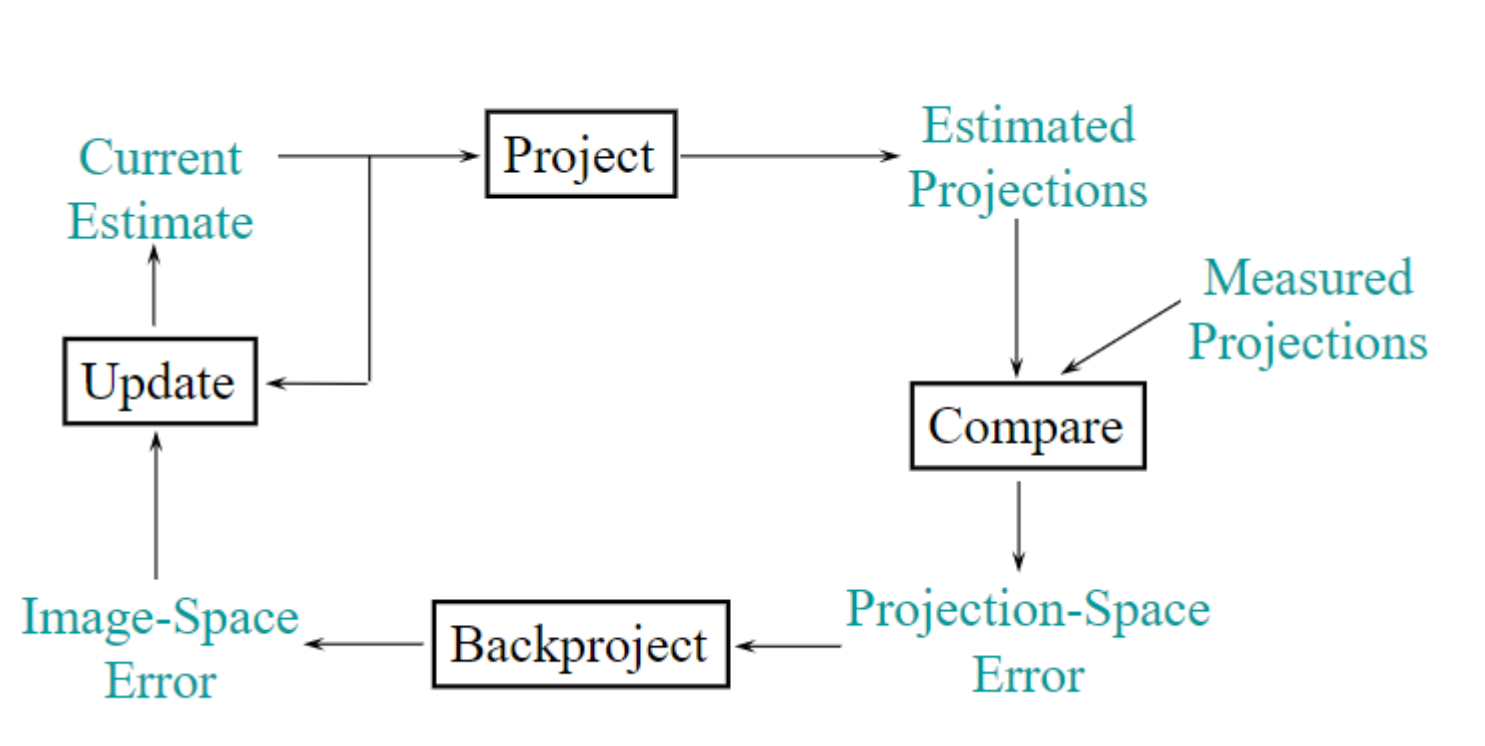
ADDITIVE ART

- Final

5	7
6	2

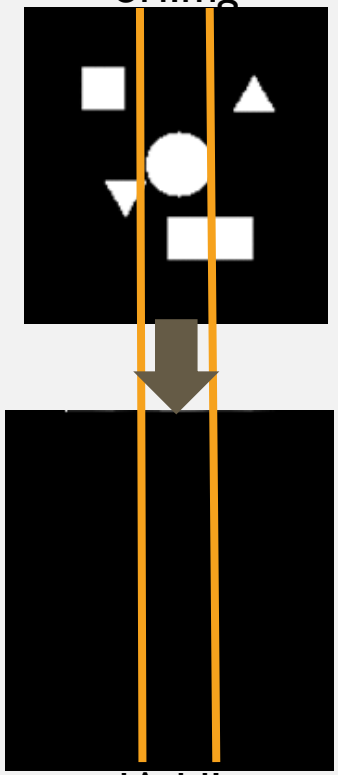
5	7
6	2

ITERATIVE RECONSTRUCTION



CODE

oriImg



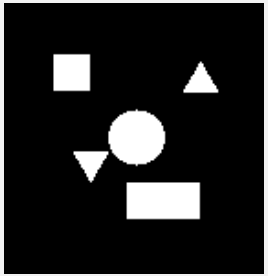
colAddImg

```
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

```
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

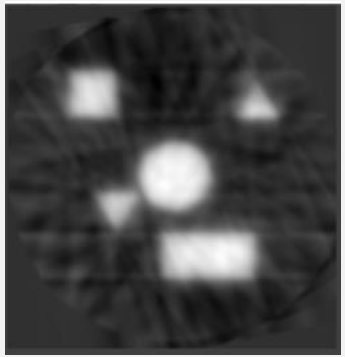
$\sum_{i=1}^{N} f_{ij}^{q}$

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

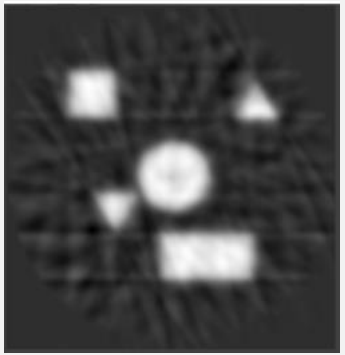
$$f_{ij}^{q+1} = f_{ij}^{q} + \frac{p_j - \sum_{i=1}^{N} f_{ij}^{q}}{N}$$

ADDITIVE ART

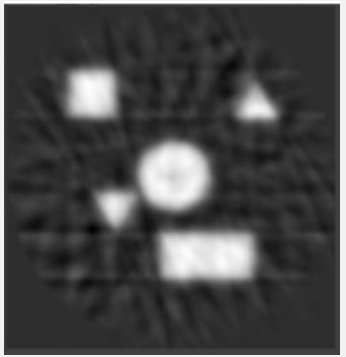
- 32 Projections



1 Iteration



100 Iterations

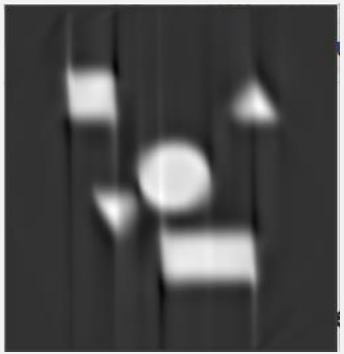


500 Iterations

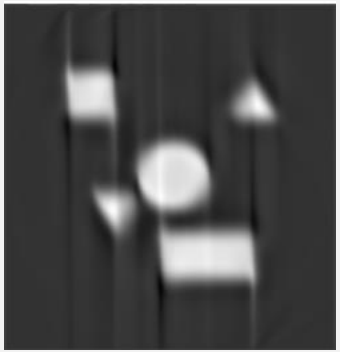
ADDITIVE ART

- 180 Projections

1 Iteration



100 Iterations



500 Iterations

PROGRESS

- Progress this time
  - read paper
  - clarify goals
  - understand how the solution works
  - build python and C++ environments (jupyter and vs2022)
  - program implementation

- Progress next time
  - continue to understand the principles of other methods (Multiplicative ART, SART, FBP)
  - program implementation
  - think about how to improve