Parallel XCT Image Reconstruction

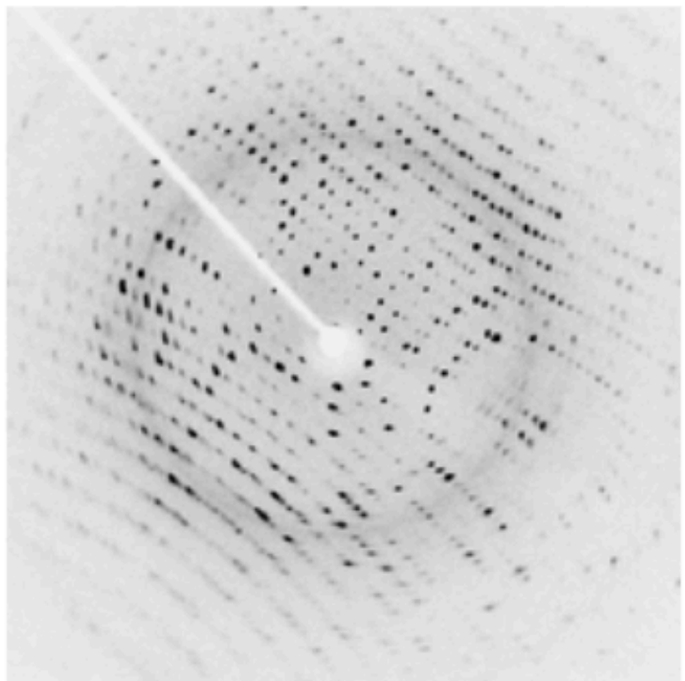
Aditya Maheshwari - COMP 5704

Introduction

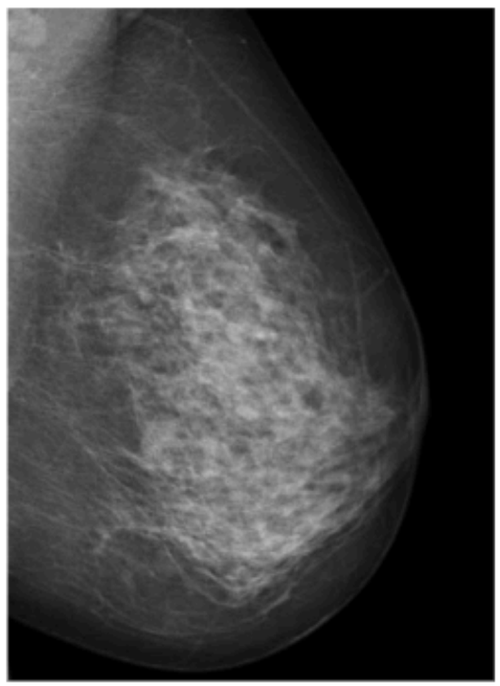
Medical CT

Airport security

X-ray crystallography

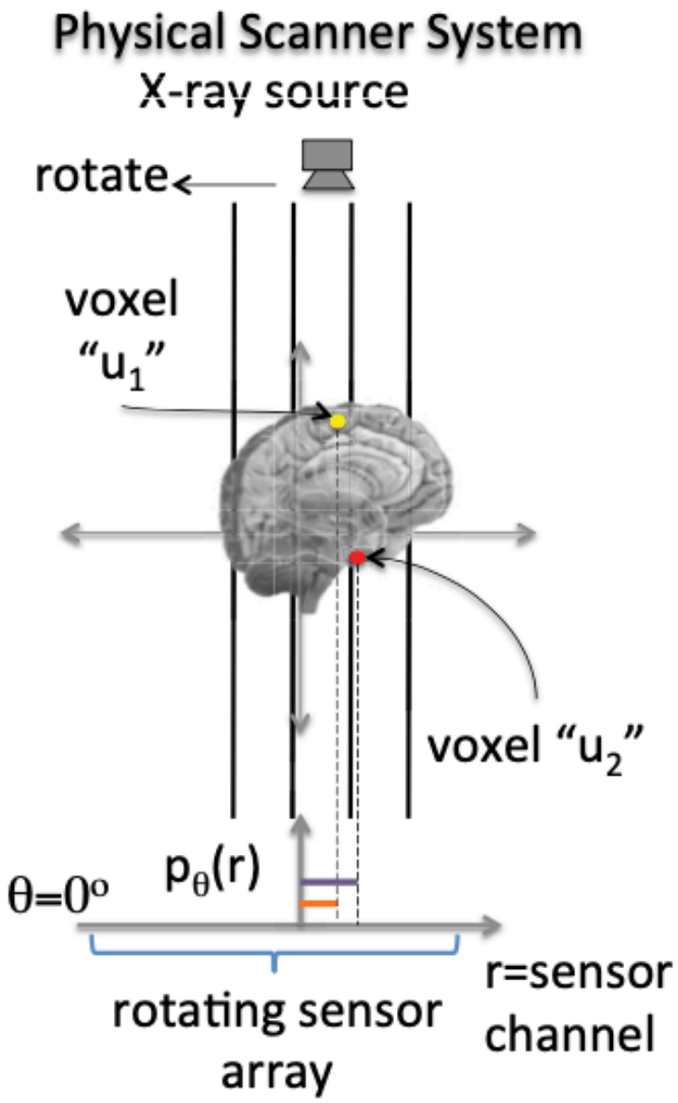


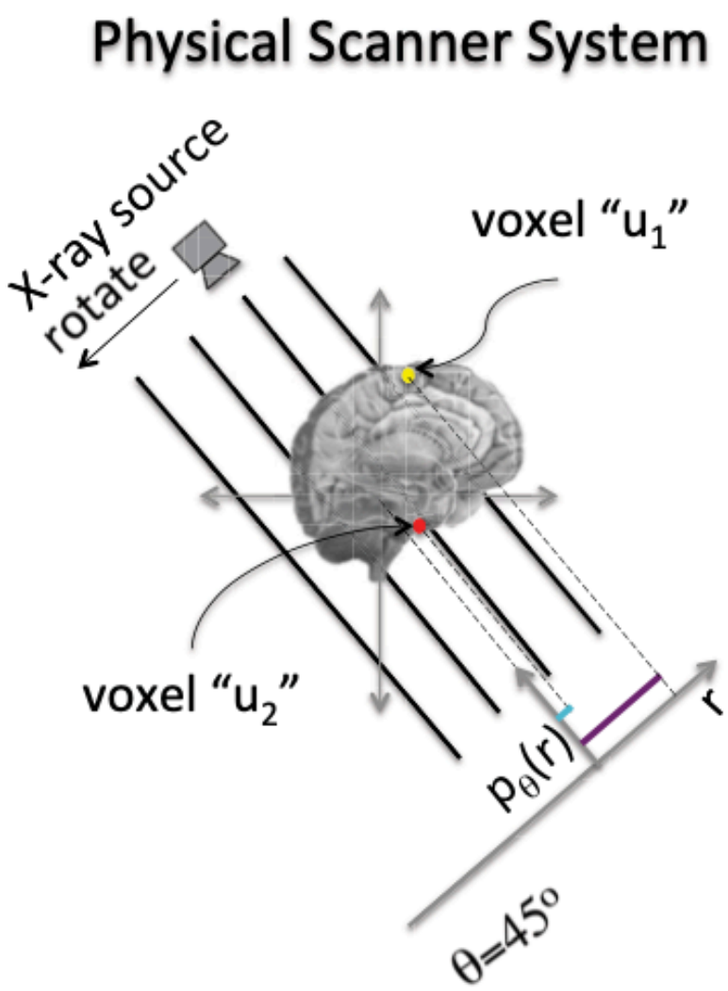
Mammography

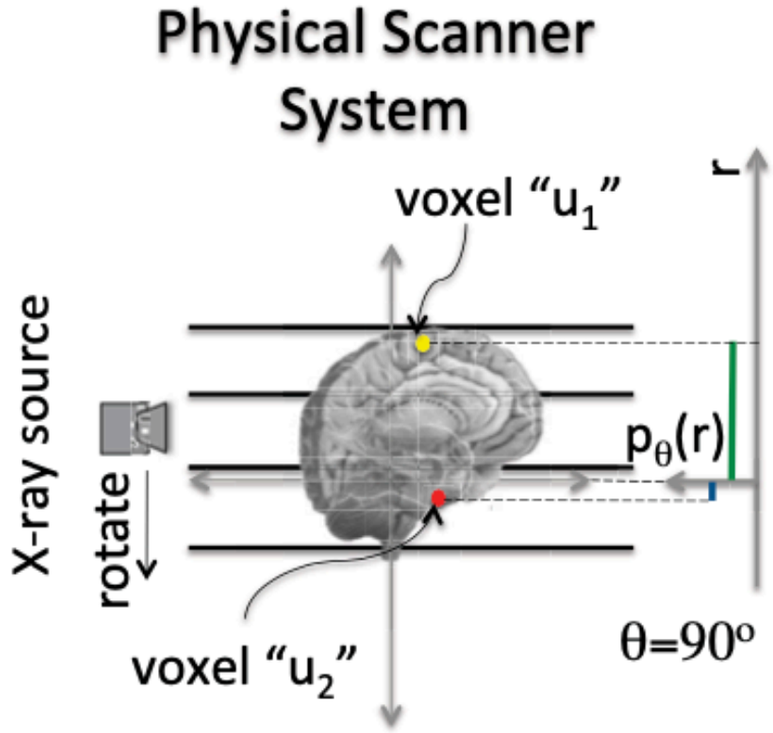


- X-ray Computed Tomography (XCT) is used for non-destructive and contact-free testing in many domains
- By taking many projections from different angles, one can approximate the density of different parts of the inner object
- Potentially more applications coming like testing whether a fruit/vegetable is ripe or raw

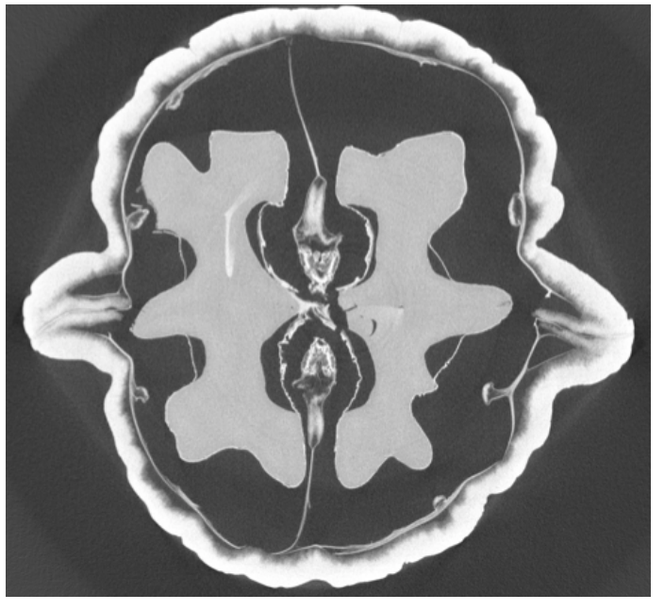
The XCT Process

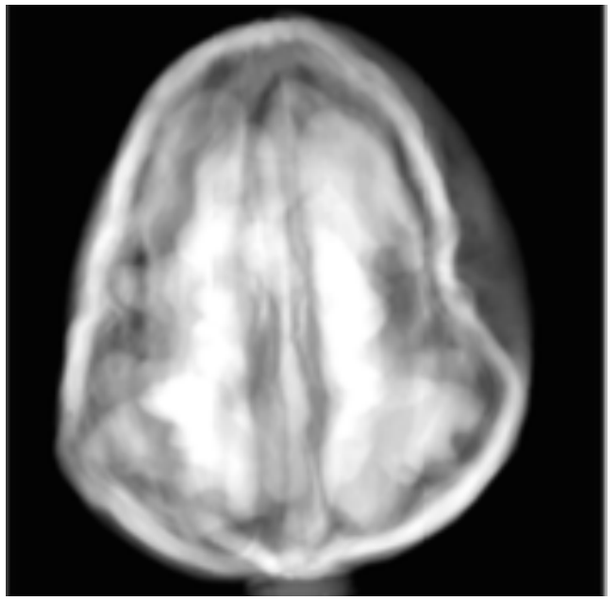


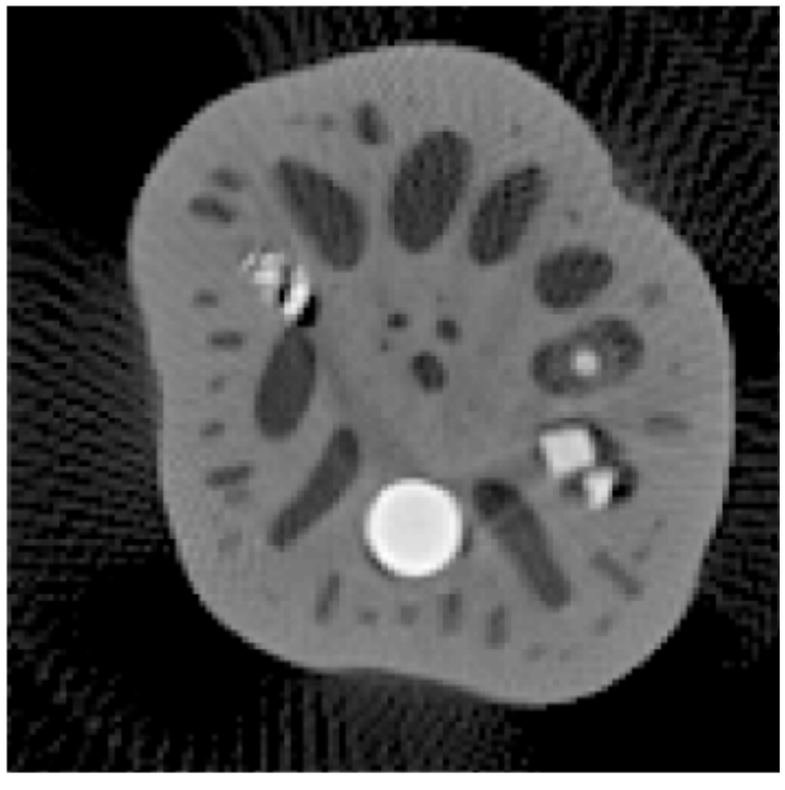


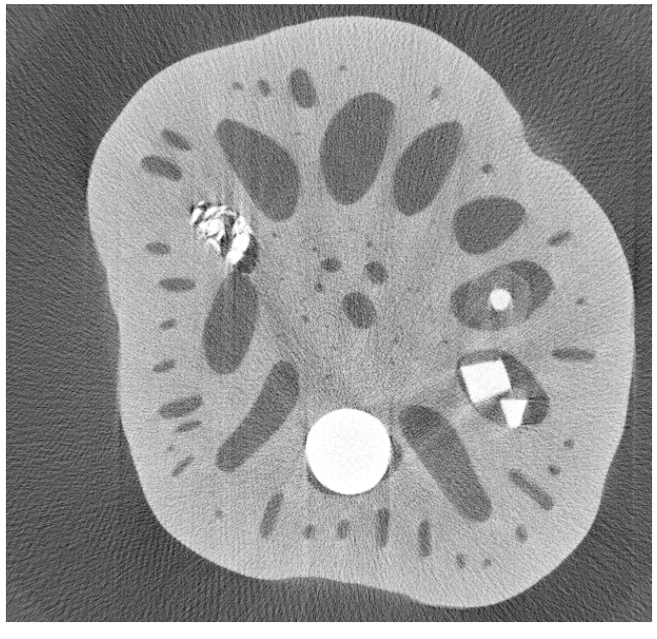


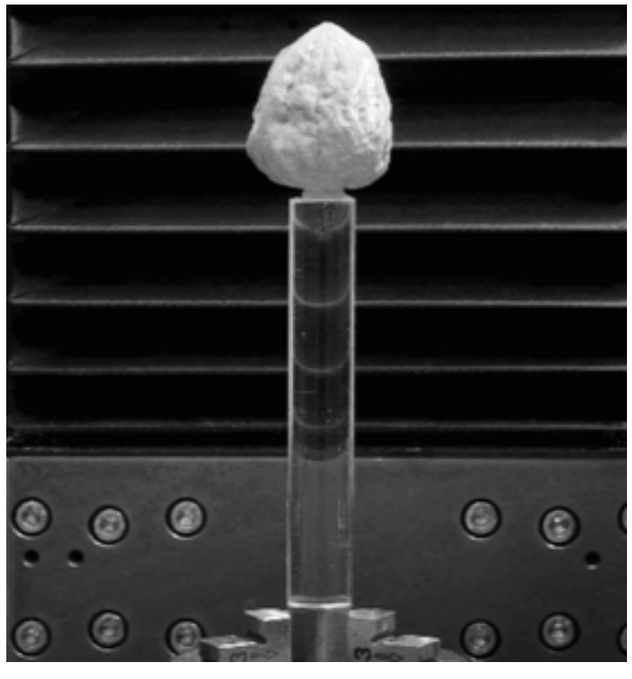
Controlled Data For Project





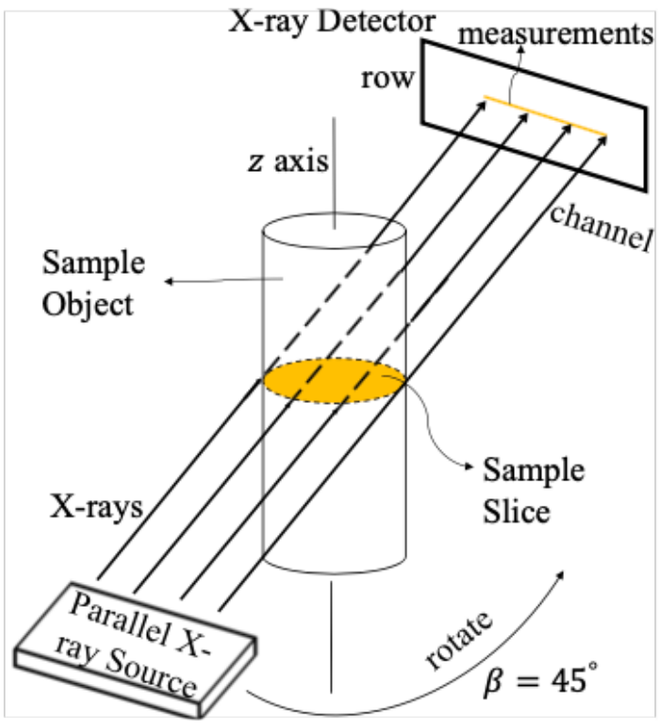


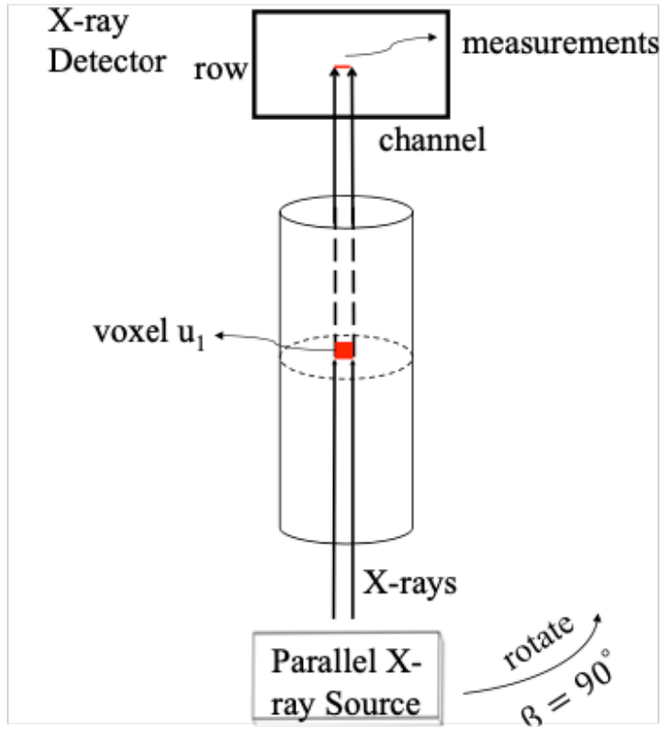


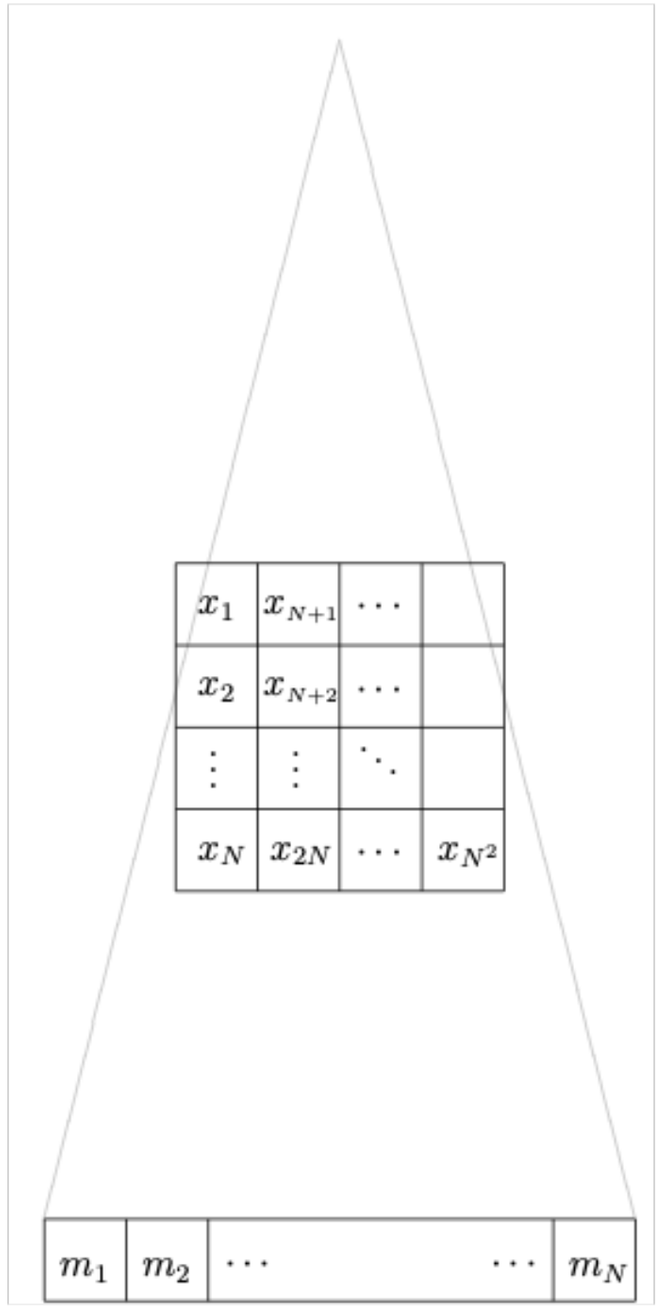


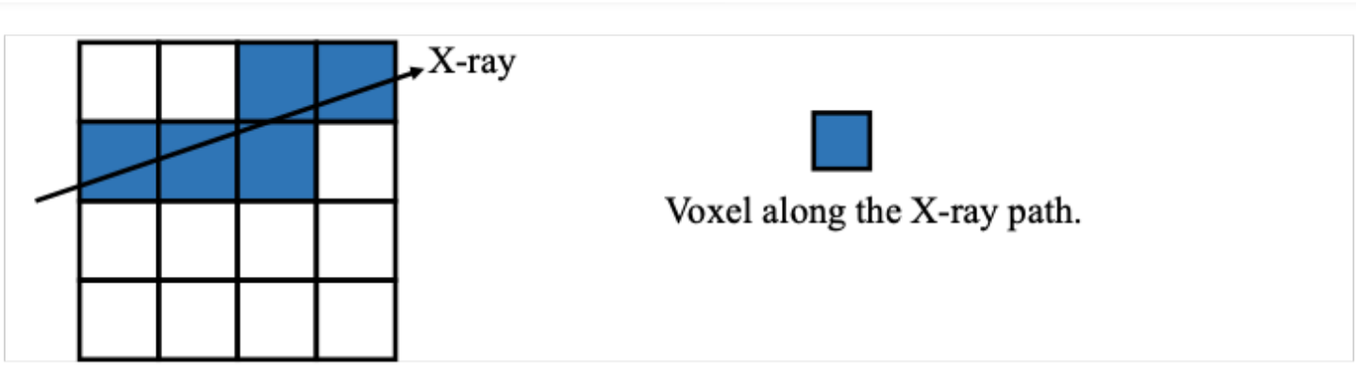
From Finnish Inverse Problems Society: http://fips.fi/dataset.php; they also have carved cheese, and an many emoji's made out of ceramic stones

How Data Is Collected









Size of Data

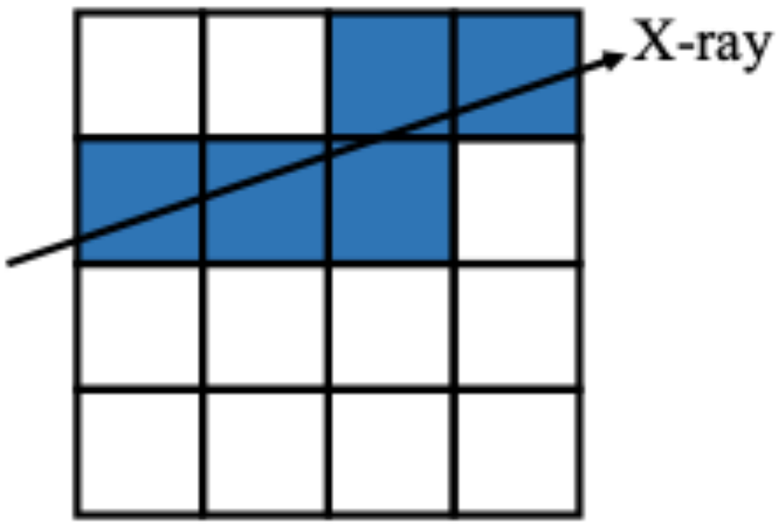
- In a real setting they can take 90 to 360 views (so each view is separated by 4 degrees to 1 degree)

- At each view they can take as many projections as they want (400, 500, etc) usually depending on how big and specific the detector is

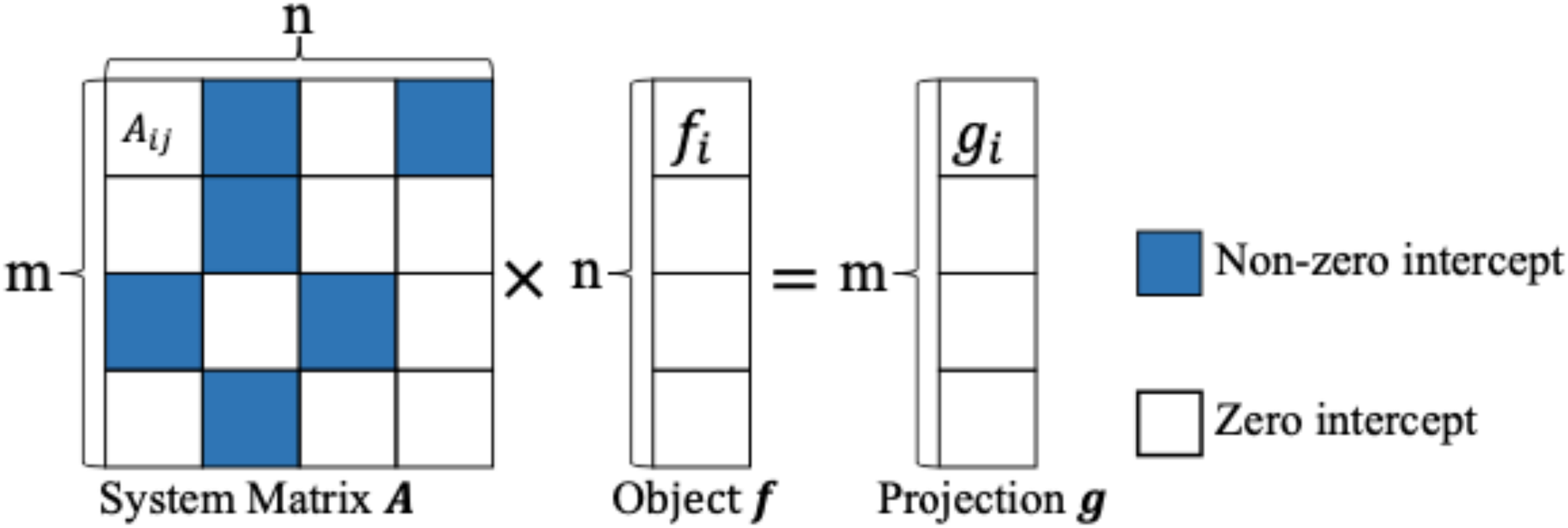
- Of course, if we can reconstruct the object to the desired accuracy with fewer projections, this means the object gets less radiation which is good in medical situations

- The object is divided into voxels (3D pixels)

Computational Problem



Voxel along the X-ray path.



Optimization Problem

- We want to minimize:

$$E(f) = D(g, Af) + \alpha R(f)$$

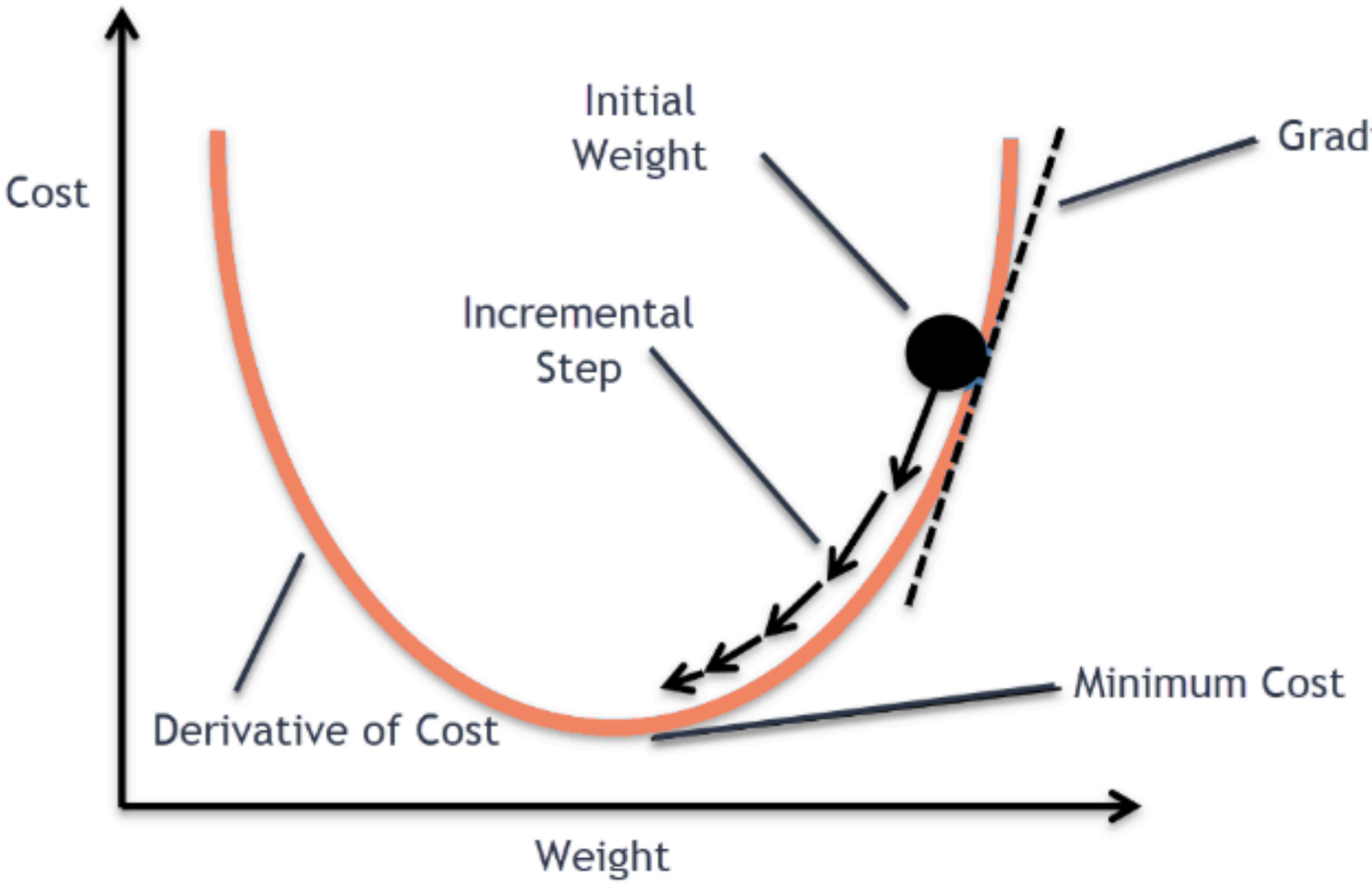
Variable	Definition
D	Distance Measure (MSE)
R	Regularizer (Not an intensive computation)
A	Data from source of x-ray (m * n)
g	Data from sensor (m * 1)
f	Image to reconstruct (n * 1)

Solving for the minimum

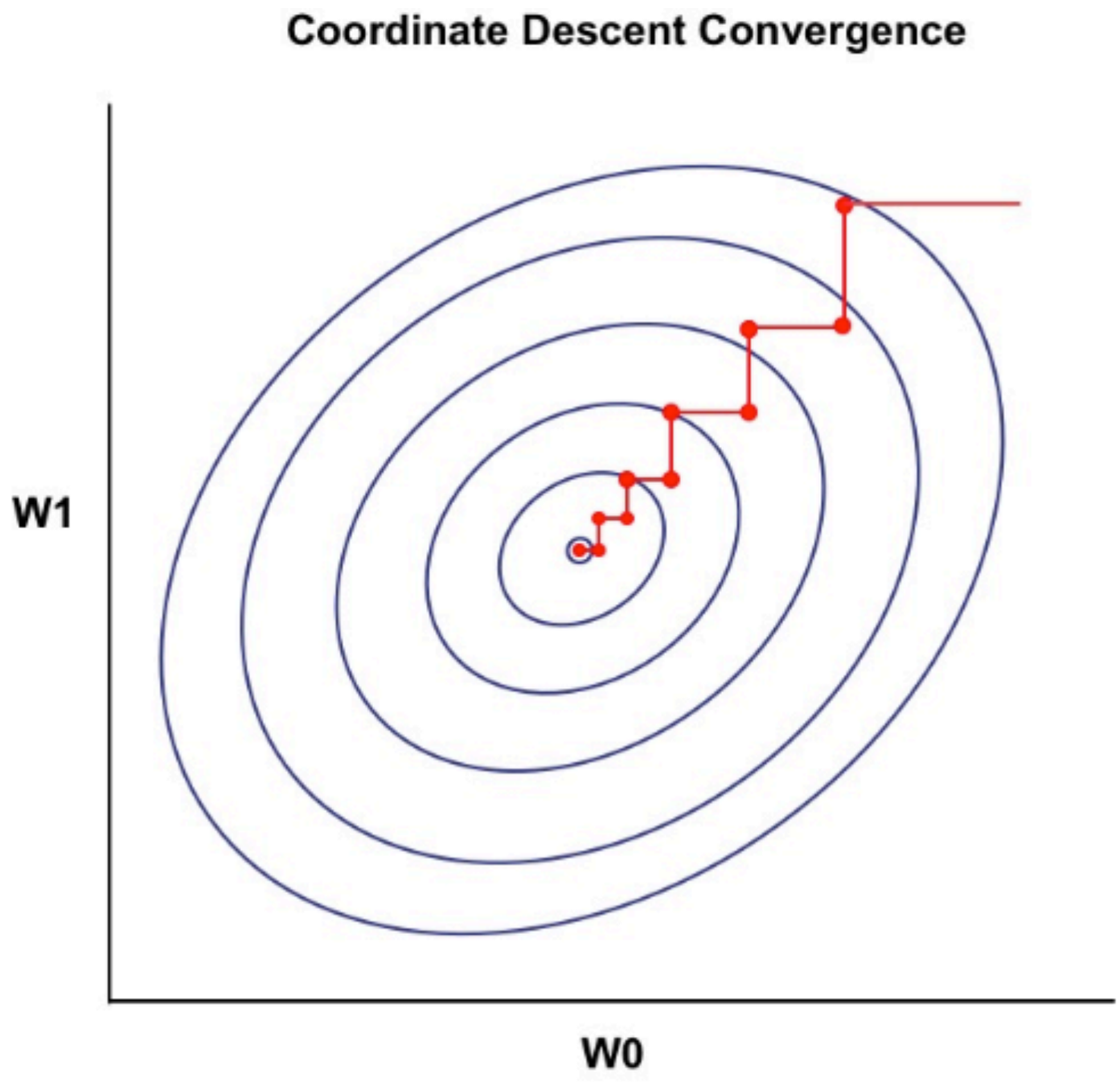
- A is too big and too sparse to approximate an inverse in reasonable time
- 2 approaches to approximate f: Updating the entire volume simultaneously (SGD) and updating it voxel by voxel (ICD).
- In SGD, update reconstructed image “projection by projection” for all voxels at the same time
    - Fast but ~1000s of iterations
- In ICD, we each voxel for all the projections
    - Slow and computationally heavy per iteration but ~15 iterations
- A is sparse, so there are ways to parallelize the solution!

Stochastic Gradient Descent

$$E(f) = D(g, Af) + \alpha R(f)$$

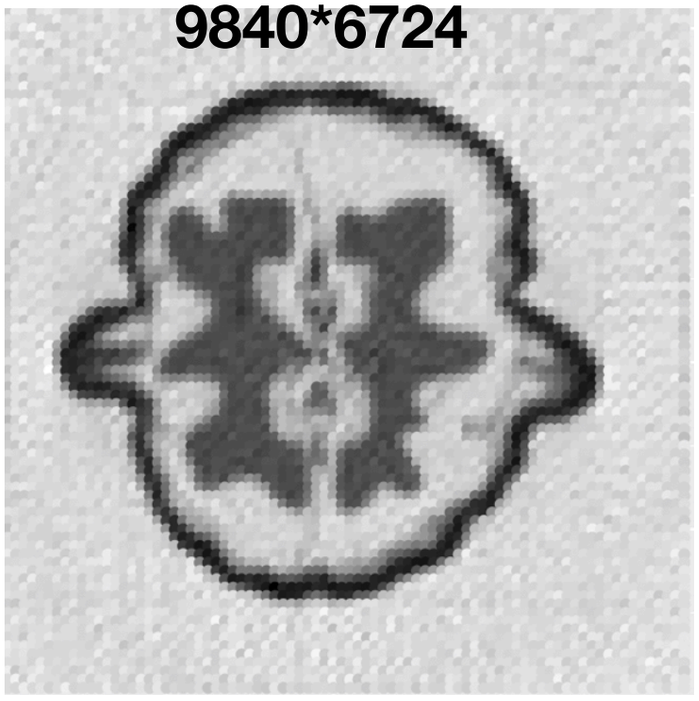


Iterative Coordinate Decsent



Results - Walnut

9840*6724



SGD

82*82

Err: 101.14

35 Seconds

5000 iters

ICD

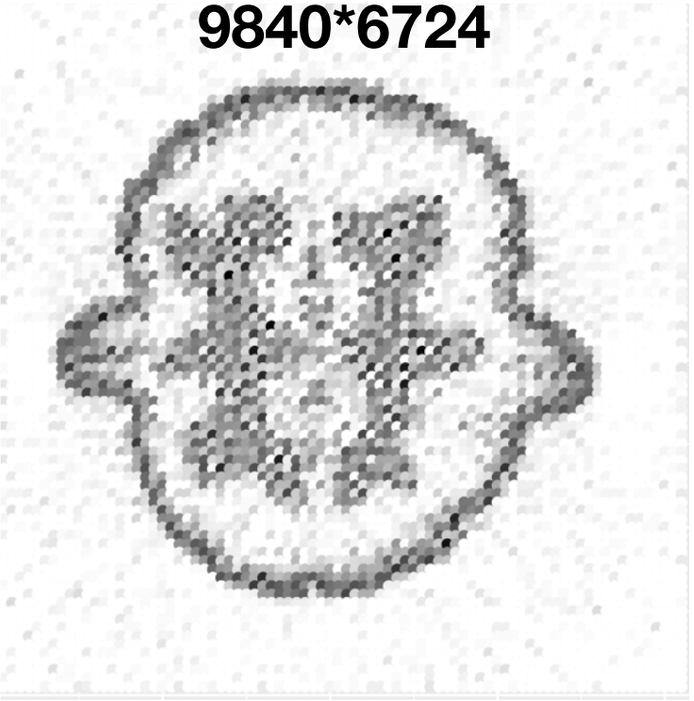
82*82

Err: 127.33

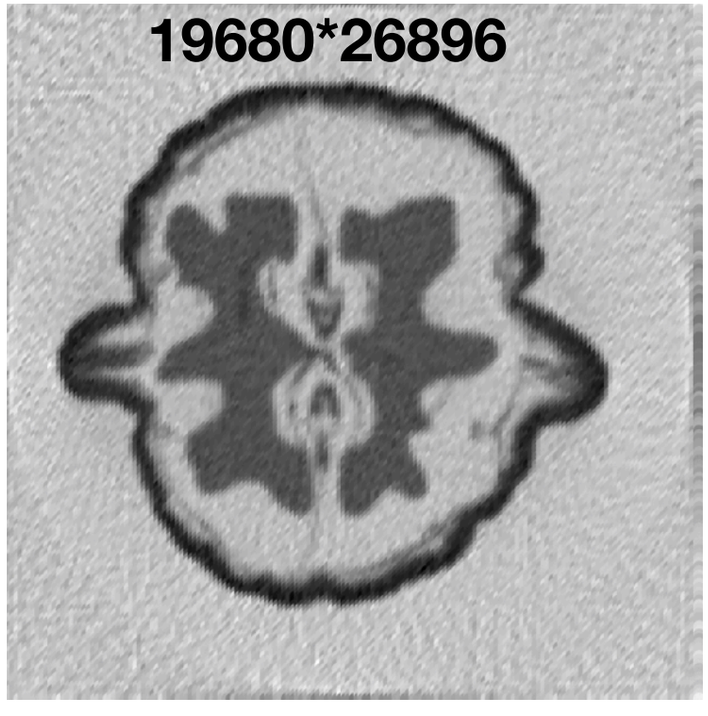
197 Seconds

5 iters

9840*6724



19680*26896



SGD

164*164

Err: 187.58

105 Seconds

5000 iters

ICD

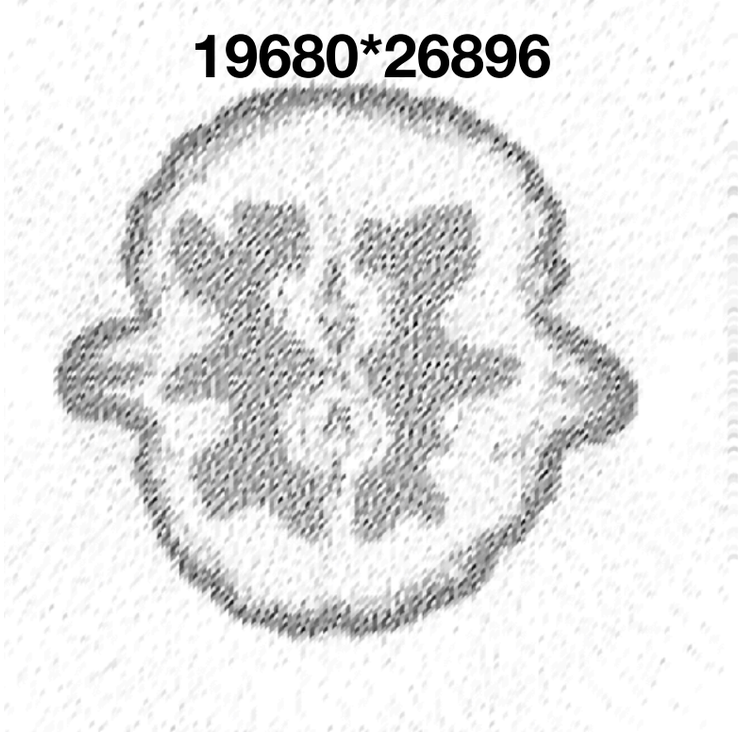
164*164

Err: 218.2693

1538 Seconds

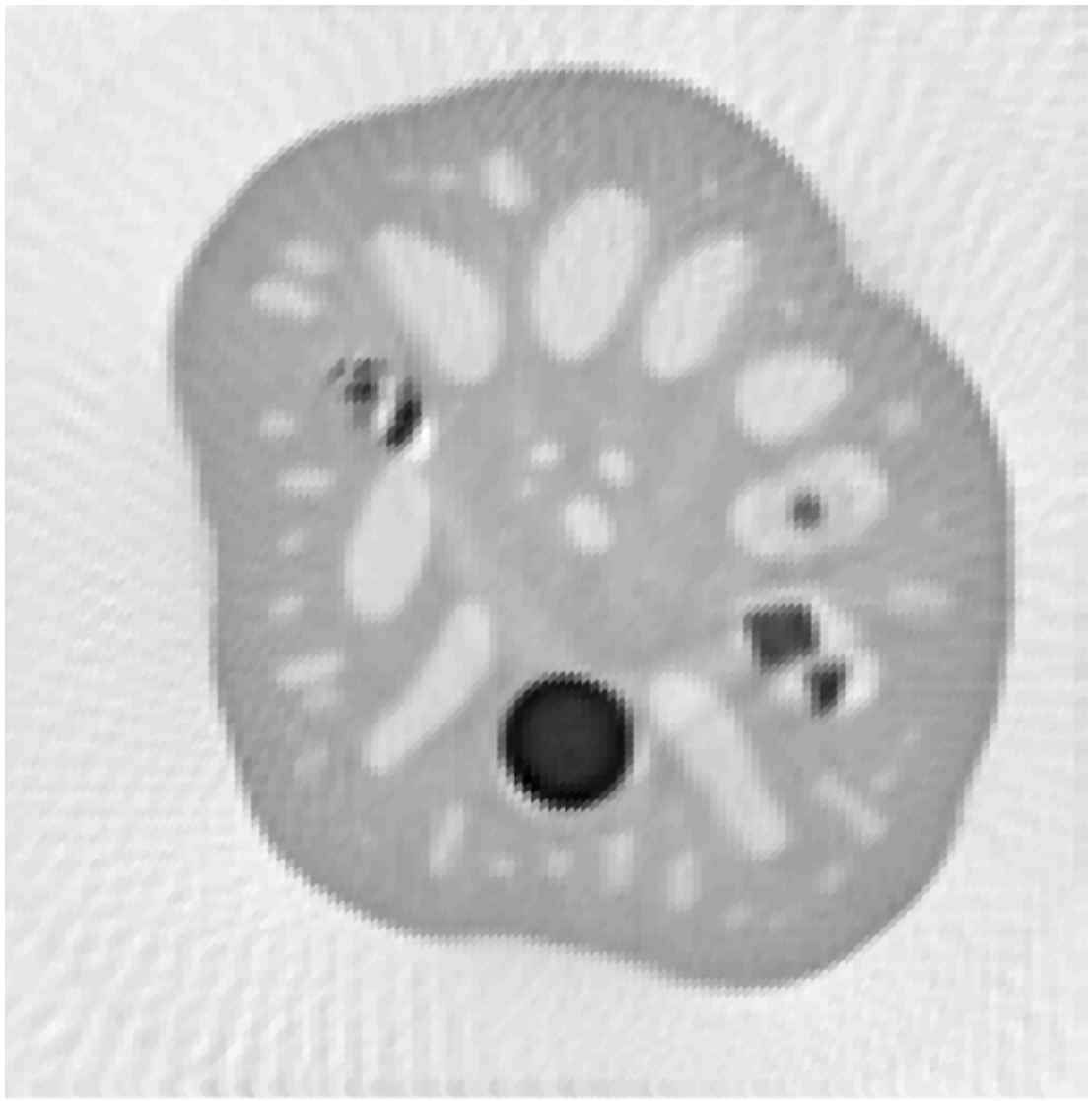
5 iters

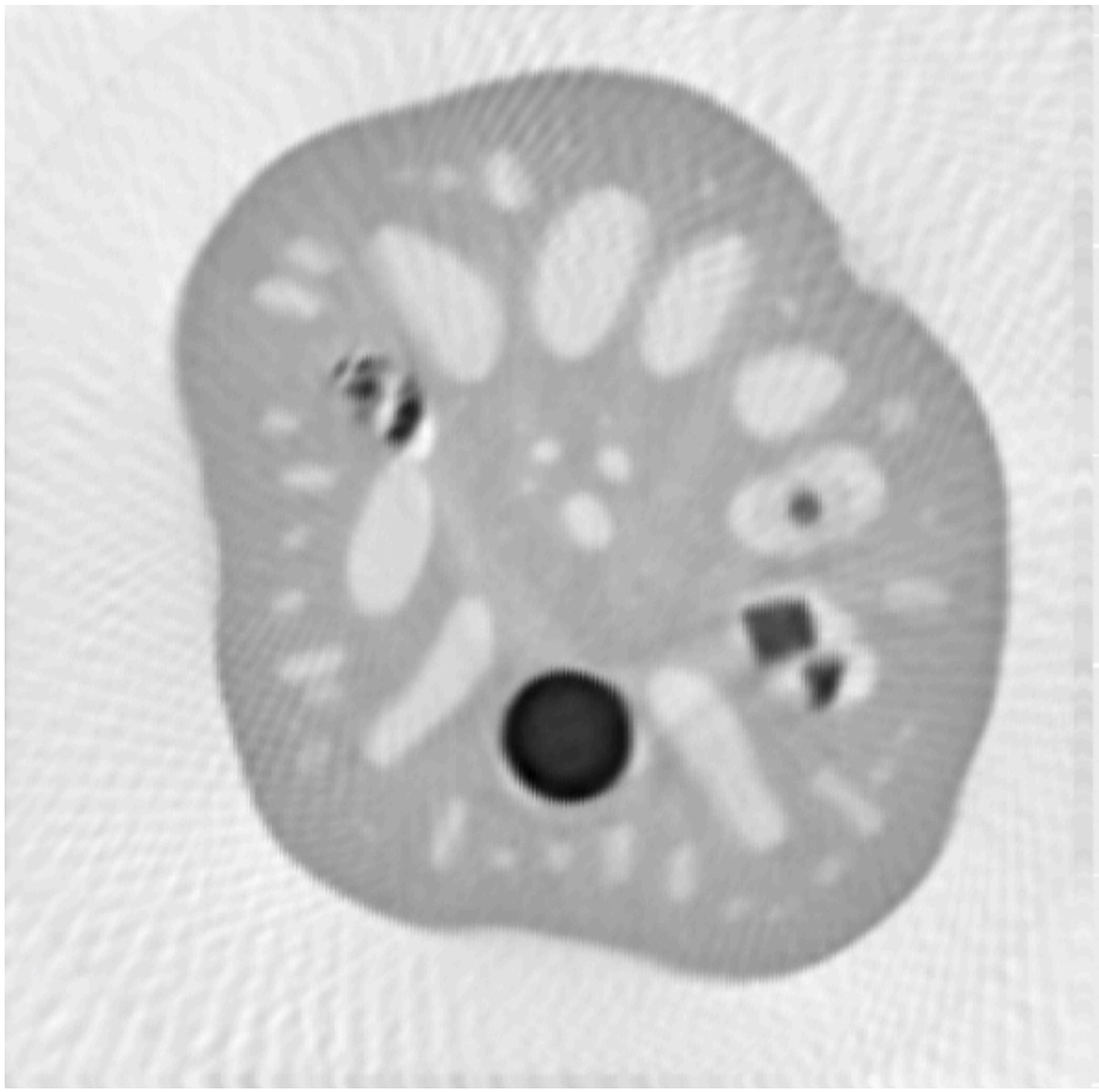
19680*26896



Lotus Root Results

- 128*128 and 256*256 Reconstruction both with 51480 Projections





- Both with SGD and took 4.67 and 18.92 mins to reconstruct (5k iters)

GPUs

- GPUs have many streaming microprocessors connected by a global memory

- Each SM schedules threads in groups of 32 called warps

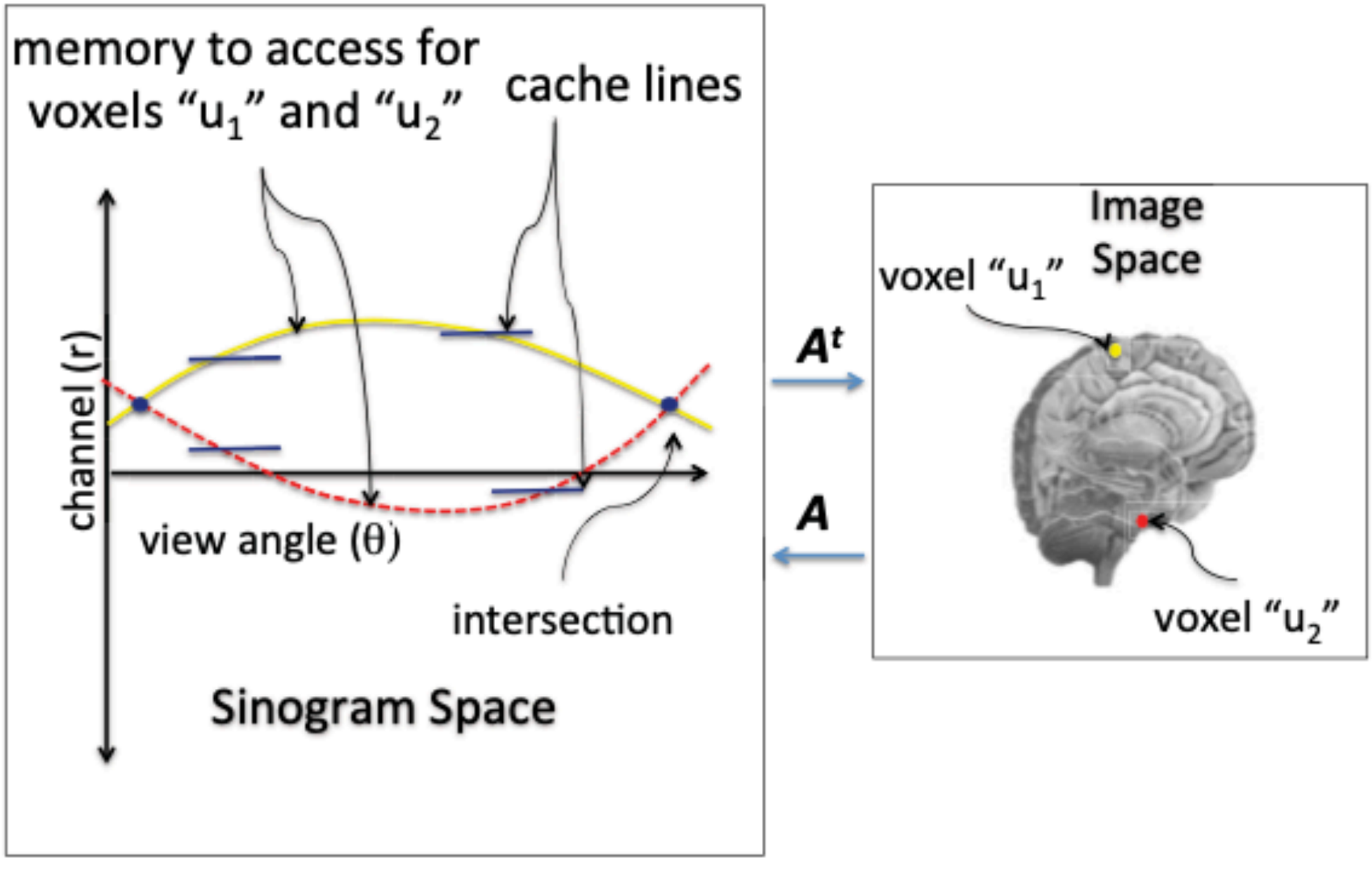
- When a warp tries to write to global memory (the image vector f), it coalesces memory accesses of threads within the warp into one or more transactions

- Two types of irregular accesses result in memory collisions and non-coalesced memory access

Dealing With Irregular Memory Accesses

- Multiple threads in same warp writing to same memory address = memory collision
- Different warps writing to the same memory address = non-coalesced memory access
- Minimizing memory collisions and non-coalesced memory accesses are inverse problems
- In a GPU minimizing memory collisions matters more

Parallelizing SGD for XCT



Parallelizing SGD for XCT

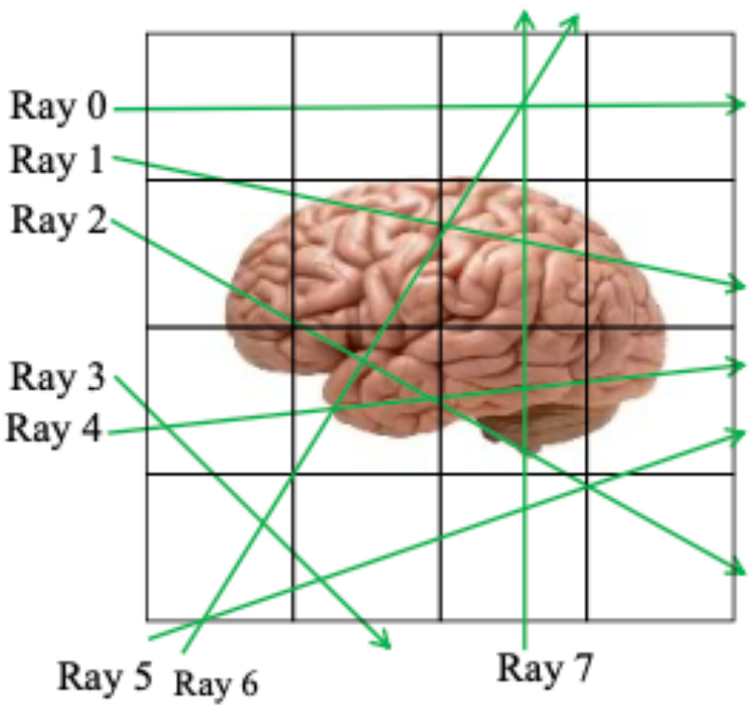
- Avoid memory collision by placing projections that update the same voxels in separate warps

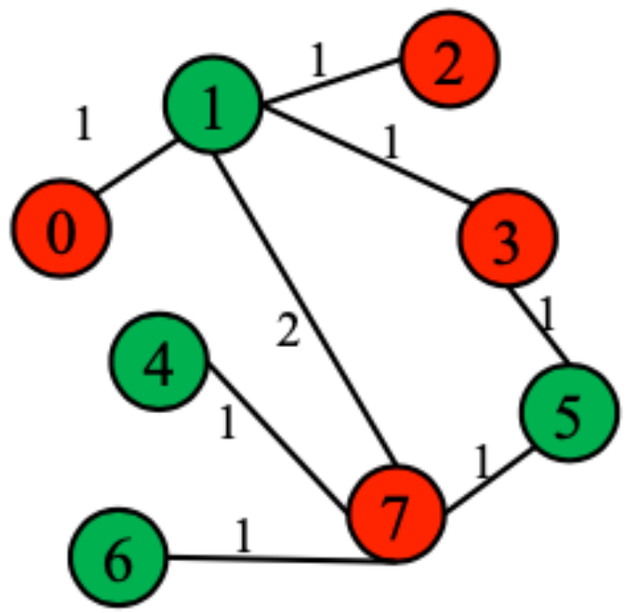
- Can in theory be done with a graph algorithm or a clustering algorithm (where distance between two points is number of same voxels they pass through)

- Realistically easier to group when we know scanner geometry

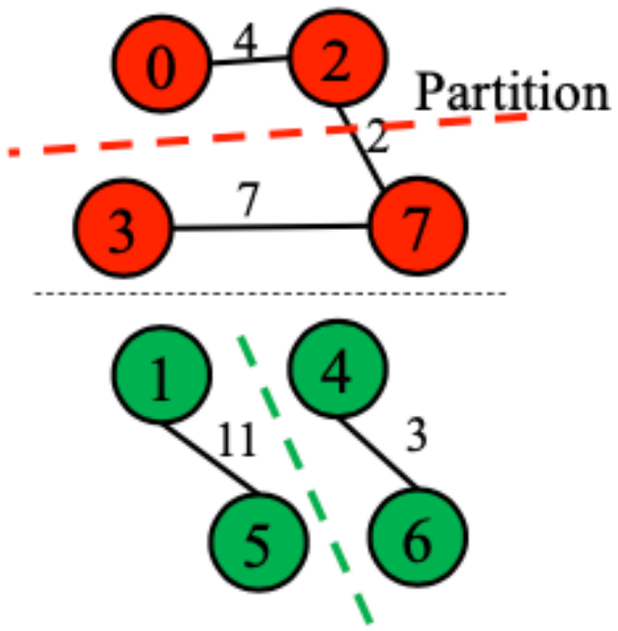
- Can further speed by taking projections that update voxels close in memory together

Grouping Projections





(a) Collision Graph



(b) Coalescing Graph

Speeding Up ICD

- Intra-voxel parallelism
    - Involves computing the first and second derivative which is a dot product of size (views * avg number of projections per voxel per view)
        - For 120 views with 429 projections per view and on average 600 voxels per projection, this is in the tens of thousands
- We can also parallelize groups voxels which do not share too many projections
    - Either with calculations or by prior knowledge, we can figure out which voxels are dissimilar

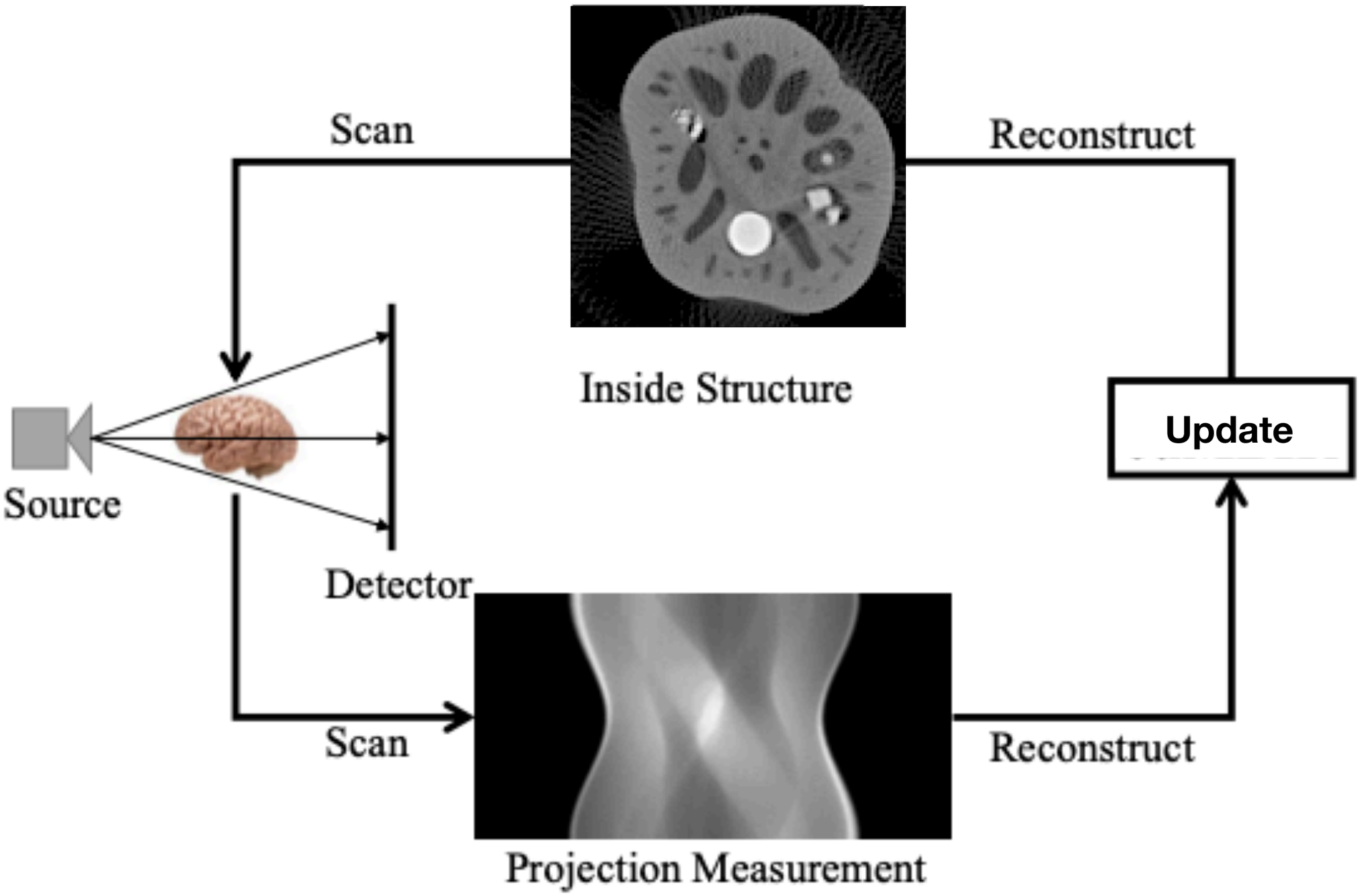
Literature Results

- Parallelizing by grouping voxels or in the SGD case grouping projections can lead to a 1.5x to 2x speedup over sending in single voxels

- This is a bit misleading as partitioning the voxels or projections wrongly can lead to a significant reduction in accuracy and this partitioning takes time

- Provided these are done properly then combined with doing dot products and matrix vector multiplications faster some real results show up

Summary


Scan
Reconstruct
Inside Structure
Update
Source
Detector
Scan
Reconstruct
Projection Measurement

Questions

1. What does the matrix A represent (rows/columns) and what are f and g

2. How do ICD and SGD differ in terms of the look of the actual image they reconstruct

3. Can you think of an efficient way to reconstruct the object in 3D (instead of just stacking slices)?

Extra Things (For Now)

Parallel ICD

1: **for** each voxel u_j in the $voxelSet$ **do in parallel**
2: $\tilde{u_j} \leftarrow u_j$
3: $\theta_1 \leftarrow Compute$ as in Equation (3)
4: $\theta_2 \leftarrow Compute$ as in Equation (4)
5: $u_j \leftarrow \text{argmin}_{r \geq 0}\{\theta_1 r + \frac{\theta_2 (r - \tilde{u_j})^2}{2} + f(r, u_{\partial j})\}$
6: lock
7: $e \leftarrow e + A_{*j}(u_j - \tilde{u_j})$
8: unlock
9: **end for**
10: **if** image not converged **then**
11: Go back to step 1.
12: **end if**

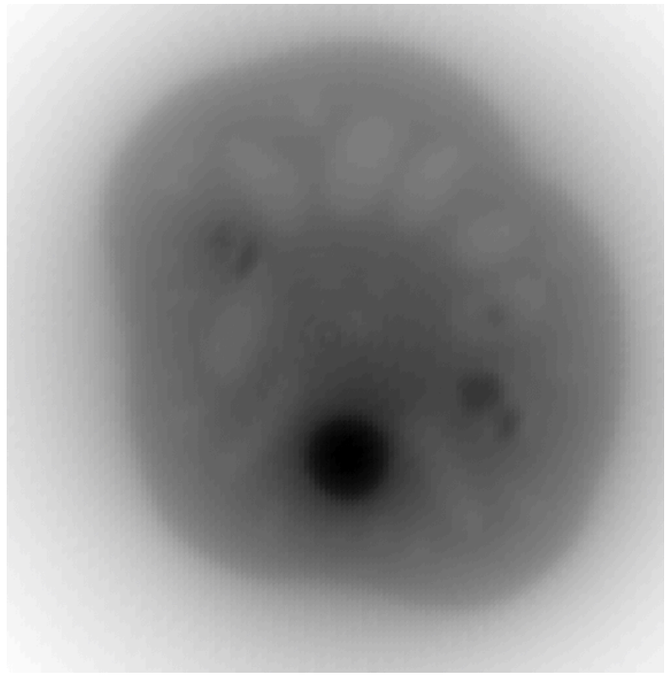
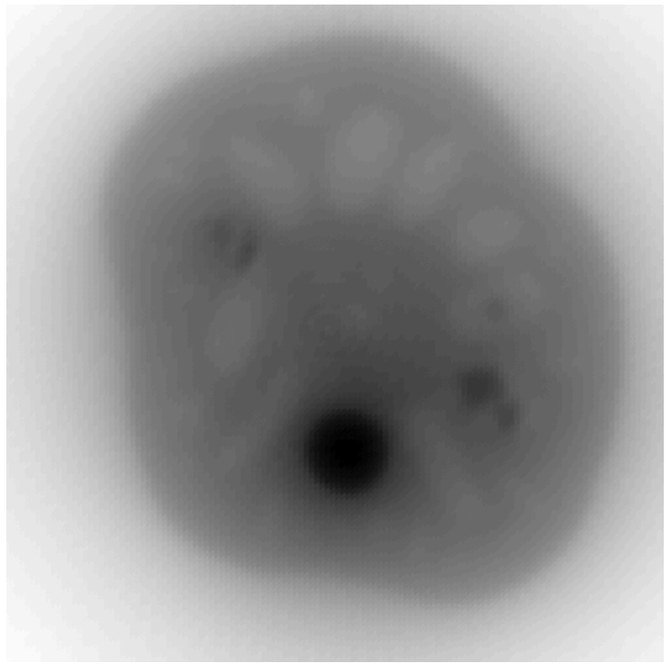
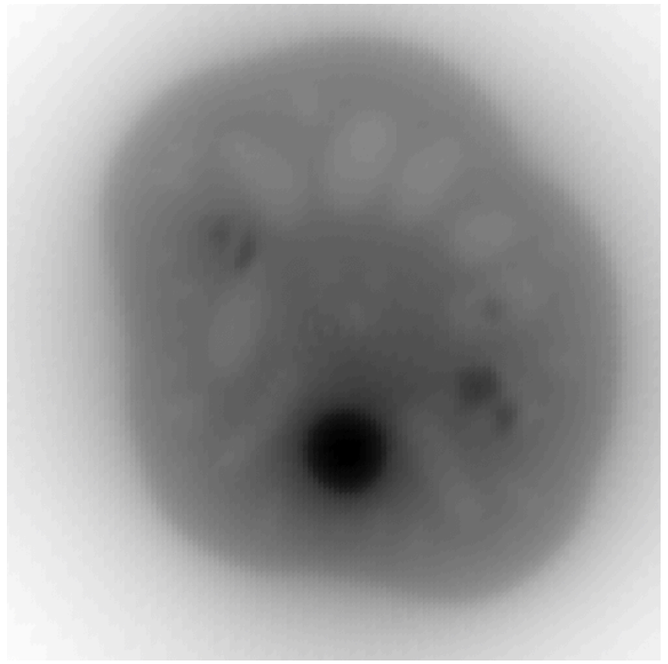
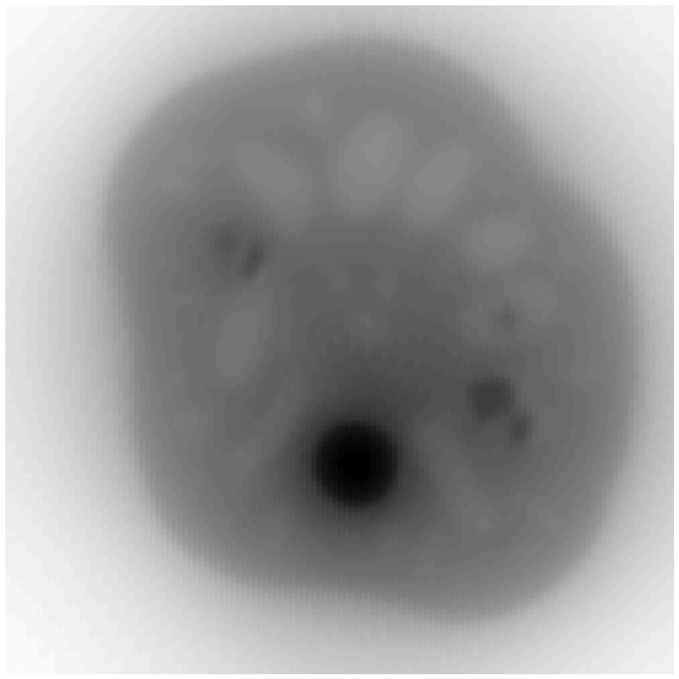
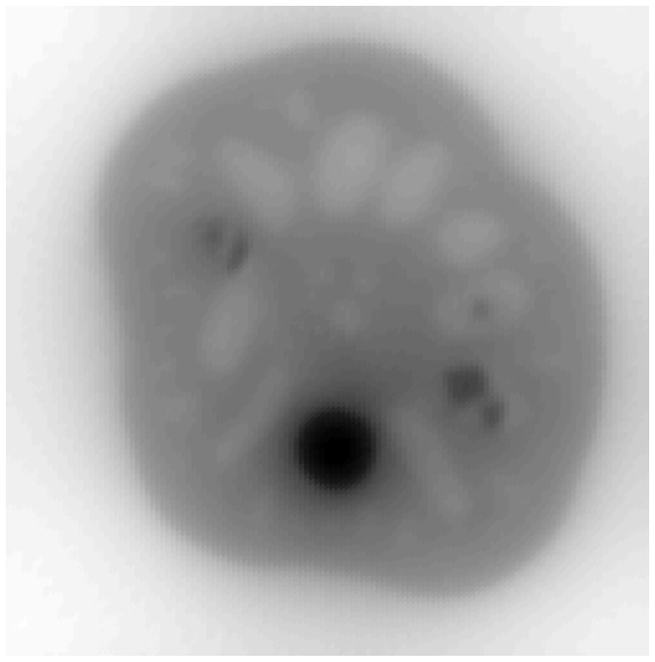
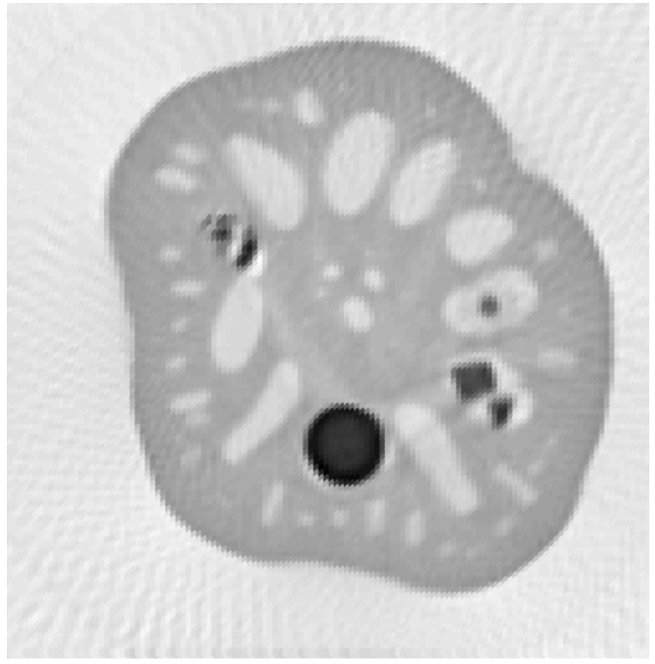
SGD Pseudo

3: **while** Image f has not converged **do**
4: **for each** projection g_i from sinogram g **do**
5: // *Get the voxels this X-ray goes through*
6: Calculate **v** and **w**
7: // *update all voxels in* **v**
8: **for all** voxels $\in$ **v do**
9: $delta \leftarrow \nabla(D(g, Af) + \alpha R(f))$;
10: $f \leftarrow f + \gamma \times delta$
11: **end for**
12: **end for**
13: **end while**

Kernel where one thread corresponds to g_i

Lotus Root after n Iterations

- 5, 10, 15, 20, 50

Easy Solution that Doesn’t Work

- If we could trace each voxel in the detector array, we would be done quite fast! Unfortunately the projection for each voxel can be picked up by a different part of the detector array each time