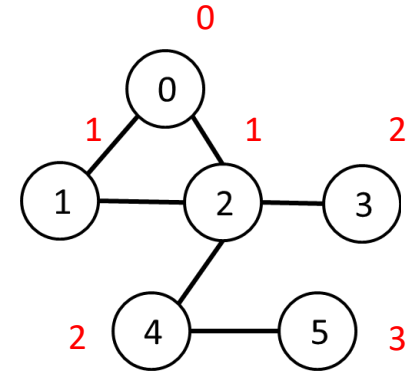


# CS 179: GPU Programming

Lecture 11 / Homework 4

# Breadth-First Search

- Given source vertex S:
  - Find min. #edges to reach every vertex from S
  - (Assume source is vertex 0)



- Sequential pseudocode:

```
let Q be a queue
Q.enqueue(source)
label source as discovered
source.value <- 0

while Q is not empty
  v ← Q.dequeue()
  for all edges from v to w in G.adjacentEdges(v):
    if w is not labeled as discovered
      Q.enqueue(w)
      label w as discovered, w.value <- v.value + 1
```

# Alternate BFS algorithm

- New sequential pseudocode:

Input:  $V_a$ ,  $E_a$ , source (graph in “compact adjacency list” format)

Create frontier (F), visited array (X), cost array (C)

F  $\leftarrow$  (all false)

X  $\leftarrow$  (all false)

C  $\leftarrow$  (all infinity)

F[source]  $\leftarrow$  true

C[source]  $\leftarrow$  0

while F is not all false:

Parallelizable!

for  $0 \leq i < |V_a|$ :

if F[i] is true:

F[i]  $\leftarrow$  false

X[i]  $\leftarrow$  true

for  $E_a[V_a[i]] \leq j < E_a[V_a[i+1]]$ :

if X[j] is false:

C[j]  $\leftarrow$  C[i] + 1

F[j]  $\leftarrow$  true

# GPU-accelerated BFS

- CPU-side pseudocode:

```
Input: Va, Ea, source      (graph in "compact adjacency list" format)
Create frontier (F), visited array (X), cost array (C)
F <- (all false)
X <- (all false)
C <- (all infinity)
```

```
F[source] <- true
C[source] <- 0
while F is not all false:
    call GPU kernel( F, X, C, Va, Ea )
```

Can represent boolean values as integers

- GPU-side kernel pseudocode:

```
if F[threadId] is true:

    F[threadId] <- false
    X[threadId] <- true

    for Ea[Va[threadId]] ≤ j < Ea[Va[threadId + 1]]:
        if X[j] is false:
            C[j] <- C[threadId] + 1
            F[j] <- true
```

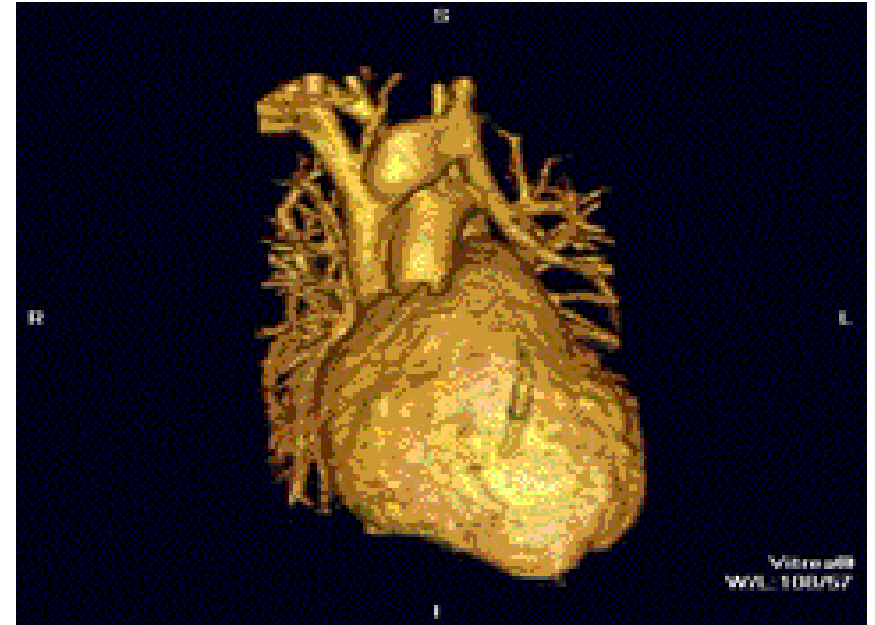
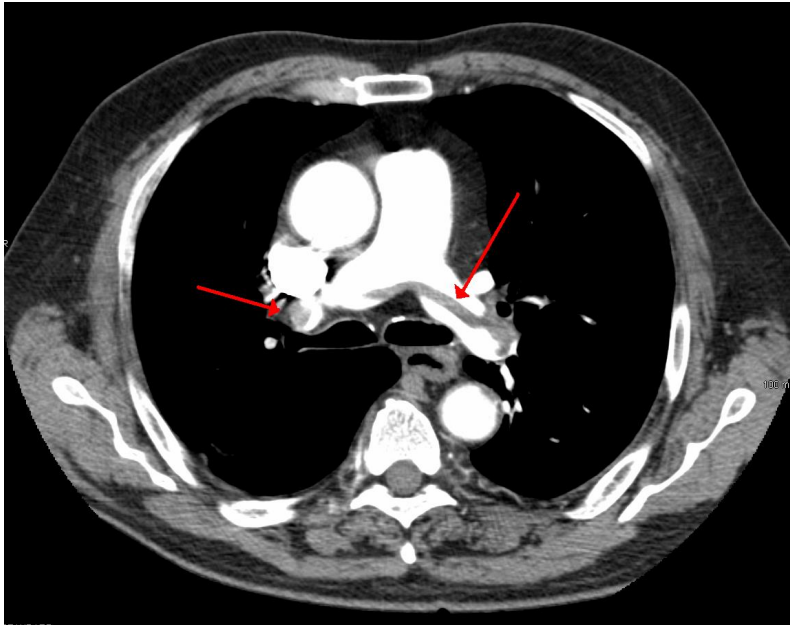
# Texture Memory (and co-stars)

- Another type of memory system, featuring:
  - Spatially-cached read-only access
  - Avoid coalescing worries
  - Interpolation
  - (Other) fixed-function capabilities
  - Graphics interoperability

# X-ray CT Reconstruction

# Medical Imaging

- See inside people!
  - Critically important in medicine today

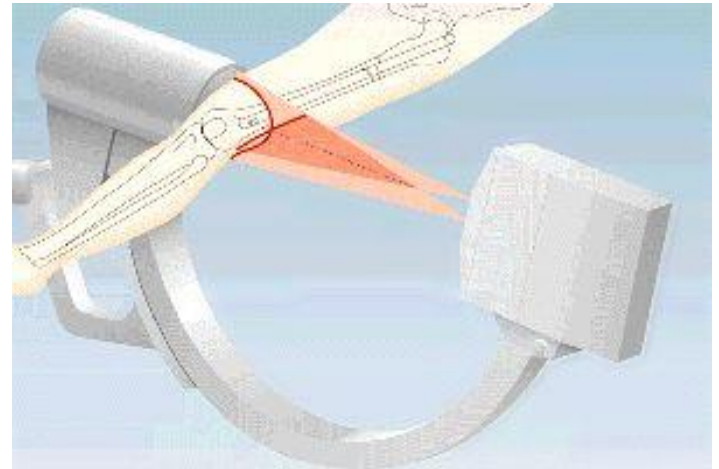


"SaddlePE" by James Heilman, MD - Own work. Licensed under CC BY-SA 3.0 via Wikimedia Commons - <http://commons.wikimedia.org/wiki/File:SaddlePE.PNG#/media/File:SaddlePE.PNG>

"PAPVR". Licensed under CC BY 3.0 via Wikipedia - <http://en.wikipedia.org/wiki/File:PAPVR.gif#/media/File:PAPVR.gif>

# X-ray imaging (Radiography)

- “Algorithm”:
  - Generate electromagnetic radiation
  - Measure radiation at the “camera”
- Certain tissues are more “opaque” to X-rays
- Like photography!



"Coude fp" by MB - Collection personnelle. Licensed under CC BY-SA 2.5 via Wikimedia Commons - [http://commons.wikimedia.org/wiki/File:Coude\\_fp.PNG#/media/File:Coude\\_fp.PNG](http://commons.wikimedia.org/wiki/File:Coude_fp.PNG#/media/File:Coude_fp.PNG)

<http://www.imaginis.com/xray/how-does-x-ray-imaginig-work>

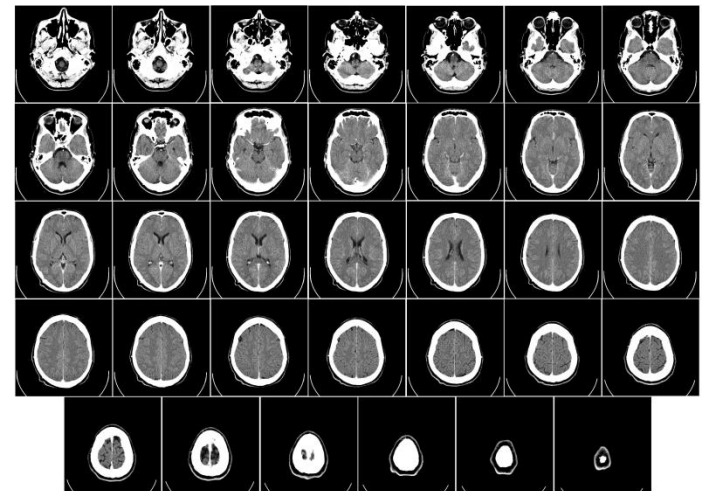


# Radiography limitations

- Generates 2D image of 3D body
- What if we want a “slice” of 3D body?
  - Goal: 3D reconstruction!  
(from multiple slices)



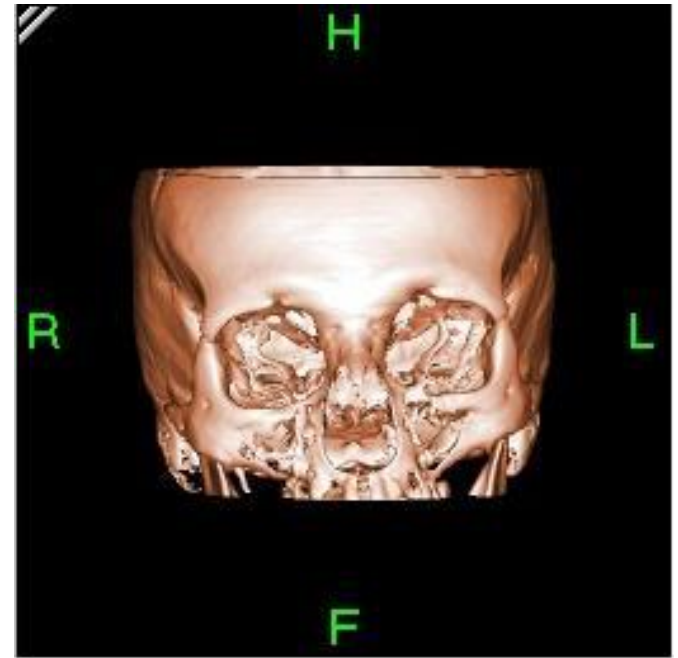
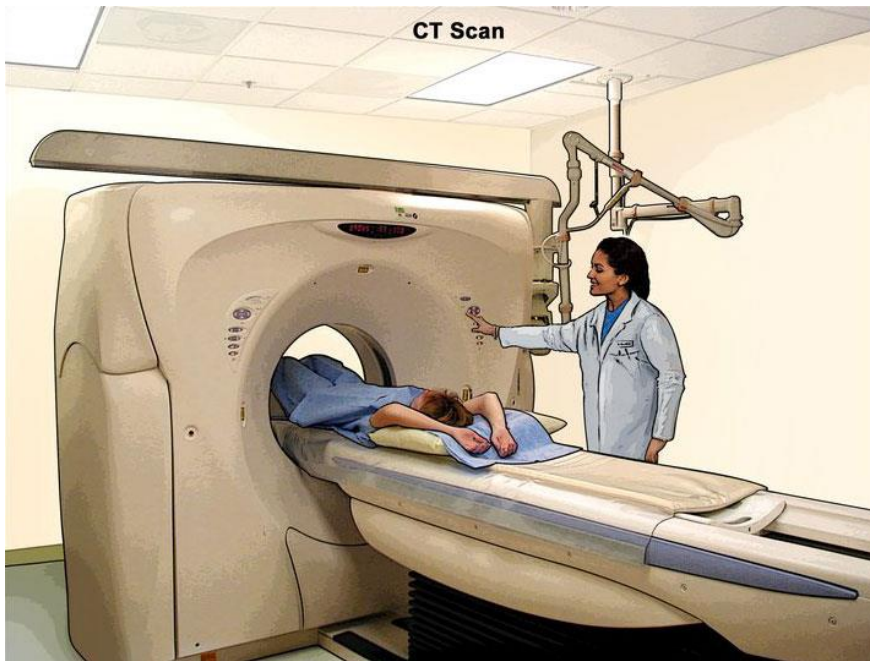
vs.



"Coude fp" by MB - Collection personnelle. Licensed under CC BY-SA 2.5 via Wikimedia Commons - [http://commons.wikimedia.org/wiki/File:Coude\\_fp.PNG#/media/File:Coude\\_fp.PNG](http://commons.wikimedia.org/wiki/File:Coude_fp.PNG#/media/File:Coude_fp.PNG)

"Computed tomography of human brain - large" by Department of Radiology, Uppsala University Hospital. Uploaded by Mikael Häggström. - Radiology, Uppsala University Hospital. Brain supplied by Mikael Häggström. It was taken Mars 23, 2007. Licensed under CC0 via Wikimedia Commons - [http://commons.wikimedia.org/wiki/File:Computed\\_tomography\\_of\\_human\\_brain\\_-\\_large.png#/media/File:Computed\\_tomography\\_of\\_human\\_brain\\_-\\_large.png](http://commons.wikimedia.org/wiki/File:Computed_tomography_of_human_brain_-_large.png#/media/File:Computed_tomography_of_human_brain_-_large.png)

# X-ray Computed Tomography (CT)

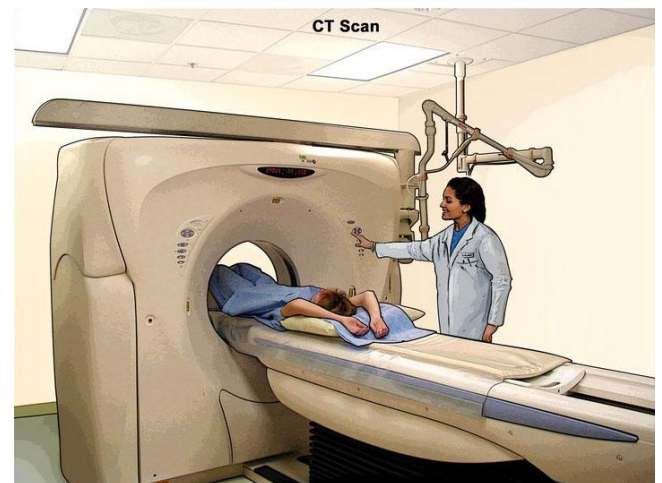
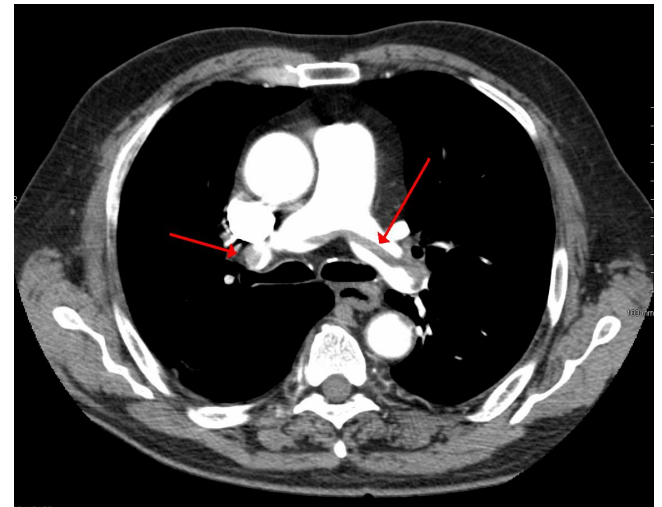


<http://www.cancer.gov/>

"Bonereconstruction" by Original uploader was Zgyorfi at en.wikipedia - Transferred from en.wikipedia; transferred to Commons by User:Common Good using CommonsHelper.. Licensed under CC BY-SA 3.0 via Wikimedia Commons - <http://commons.wikimedia.org/wiki/File:Bonereconstruction.jpg#/media/File:Bonereconstruction.jpg>

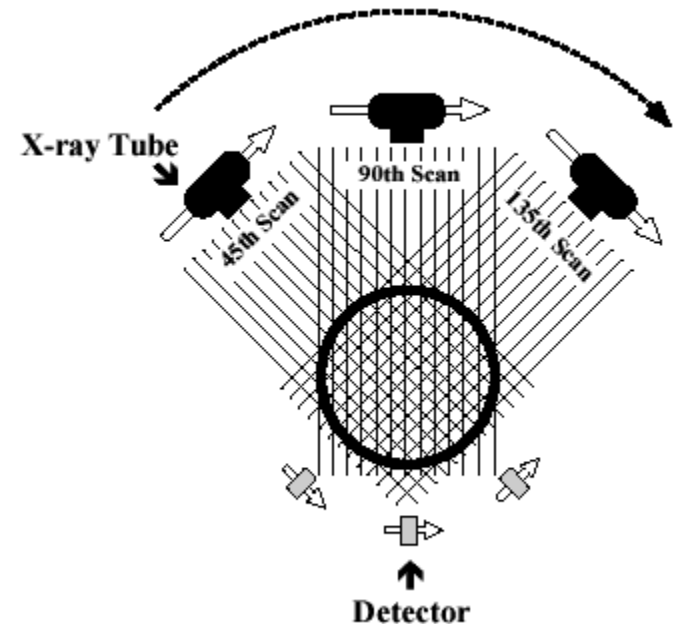
# X-ray Computed Tomography (CT)

- Generate 2D “slice” using 3D imaging
  - New imaging possibilities!
- Reconstruction less straightforward



# X-ray Computed Tomography (CT)

- “Algorithm” (per-slice):
  - Take \*lots\* of pictures at different angles
    - Each “picture” is a 1-D line
  - Reconstruct the many 1-D pictures into a 2-D image
- Harder, more computationally intensive!
  - 3D reconstruction requires multiple slices



# Mathematical Details

- X-ray CT (per-slice) performs a 2D *X-ray transform* (eq. to 2D *Radon transform*):
  - Suppose body density represented by  $f(\vec{x})$  within 2D slice,  $\vec{x} = (x, y)$
  - Assume linear attenuation of radiation
  - For each line  $L$  of radiation measured by detector:

$$I_{detect} = I_{emit} \int_L f = I_{emit} \int_{\mathbb{R}} f(\vec{x}_0 + t\vec{\theta}_L) dt$$

- $\vec{\theta}_L$ : a unit vector in direction of  $L$

# Mathematical Details

$$I_{detect} = I_{emit} \int_L f = I_{emit} \int_{\mathbb{R}} f(\vec{x}_0 + t\vec{\theta}_L) dt$$

- Defined as Lebesgue integral – non-oriented
  - Opposite radiation direction should have same attenuation!
  - Re-define as:

$$I_{detect} = I_{emit} \int_{-\infty}^{\infty} f(\vec{x}_0 + t\vec{\theta}_L) |dt|$$

# Mathematical Details

- For each line  $L$  of radiation measured by detector:

$$I_{detect} = I_{emit} \int_L f = I_{emit} \int_{-\infty}^{\infty} f(\vec{x}_0 + t\vec{\theta}_L) |dt|$$

- Define general X-ray transform (for all lines  $L$  in  $\mathbb{R}^2$ ):

$$(Rf)(L) = \int_{-\infty}^{\infty} f(\vec{x}_0 + t\vec{\theta}_L) |dt|$$

- Fractional values of attenuation
- $\vec{x}_0$  lies along  $L$

# Mathematical Details

- Define general X-ray transform:

$$(Rf)(L) = \int_{-\infty}^{\infty} f(\vec{x}_0 + t\vec{\theta}_L) |dt|$$

- Parameterize  $\vec{\theta} = (\cos \theta, \sin \theta)$

- Redefine as:

$$(Rf)(\vec{x}_0, \theta) = \int_{-\infty}^{\infty} f(\vec{x}_0 + t\vec{\theta}) |dt|$$

- Define for  $\theta \in [0, 2\pi)$



# Mathematical Details

$$(Rf)(\vec{x}_0, \theta) = \int_{-\infty}^{\infty} f(\vec{x}_0 + t\vec{\theta}) |dt|$$

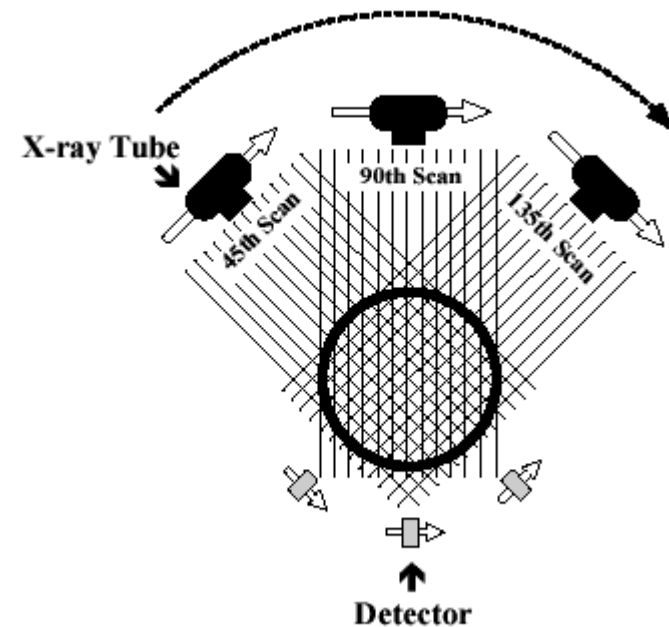
- Important properties:
  - Many  $\vec{x}_0$  are redundant!
  - Symmetry:  $Rf(\vec{x}_0, \theta) = Rf(\vec{x}_0, \theta + \pi)$ 
    - Can define for  $\theta \in [0, \pi)$

# X-ray Computed Tomography (CT)

- Redefined X-ray transform,  $\theta \in [0, \pi)$ :

$$(Rf)(\vec{x}_0, \theta) = \int_{-\infty}^{\infty} f(\vec{x}_0 + t\vec{\theta}) |dt|$$

- In reality:
  - Only defined for some  $\theta$ !



# X-ray CT *Reconstruction*

- Given the results of our scan (the *sinogram*):

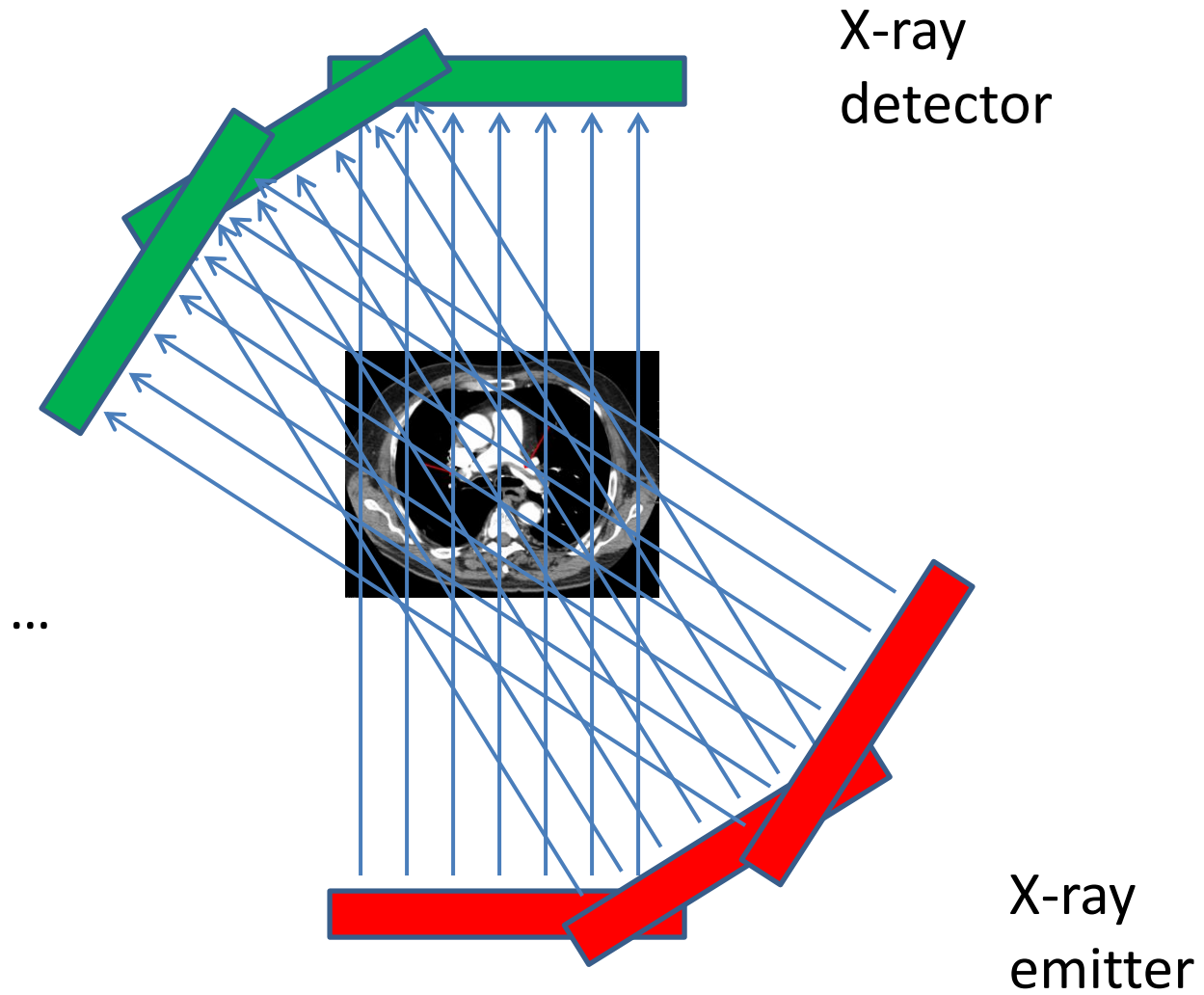
$$(Rf)(\vec{x}_0, \theta) = \int_{-\infty}^{\infty} f(\vec{x}_0 + t\vec{\theta}) |dt|$$

- Obtain the original data: (“*density*” of our body)

$$f(x, y)$$

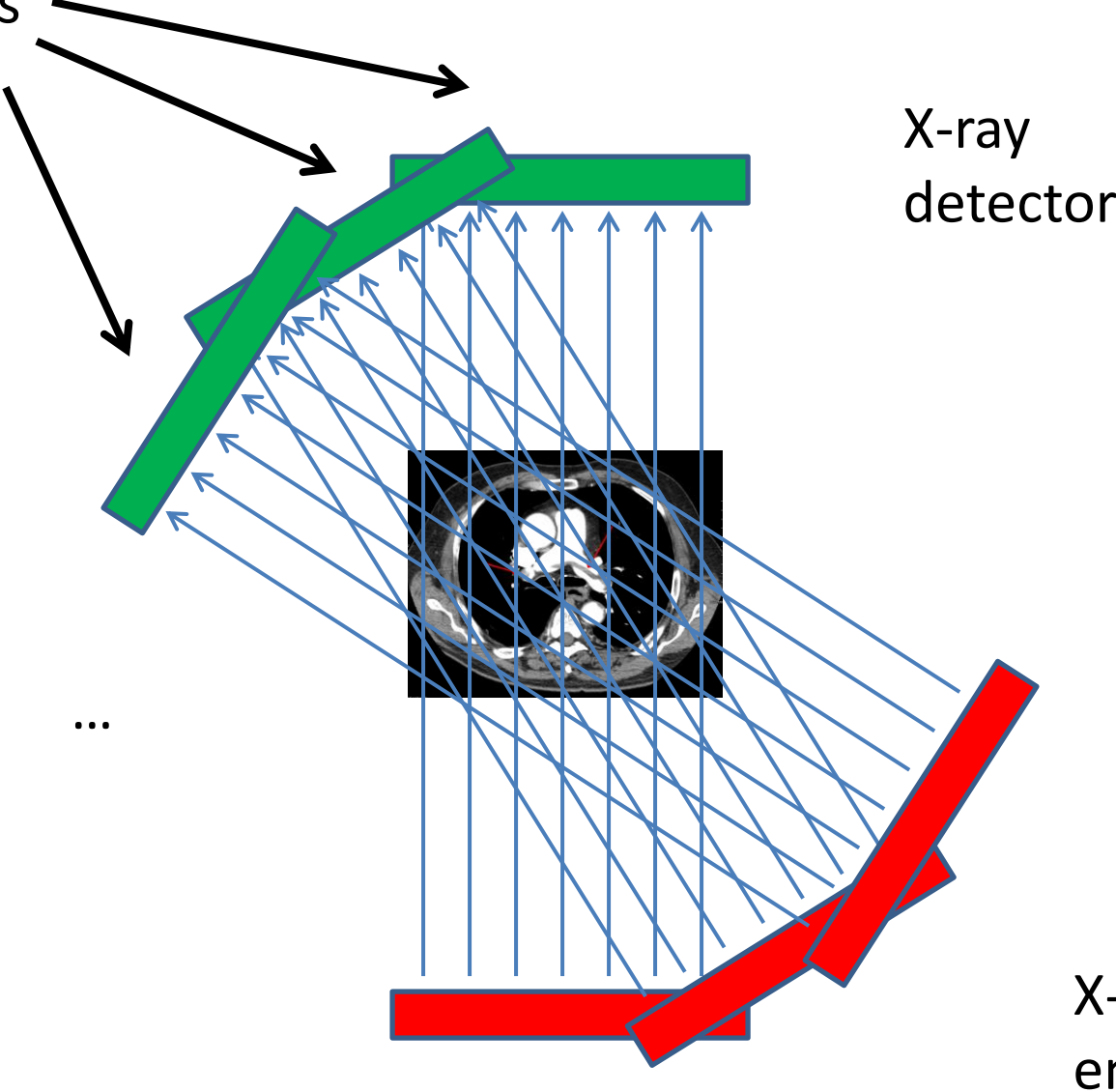
- In reality:
  - This is hard
  - We only scanned at certain (discrete) values of  $\theta$ !
    - Consequence: Perfect reconstruction is impossible!

# Reconstruction



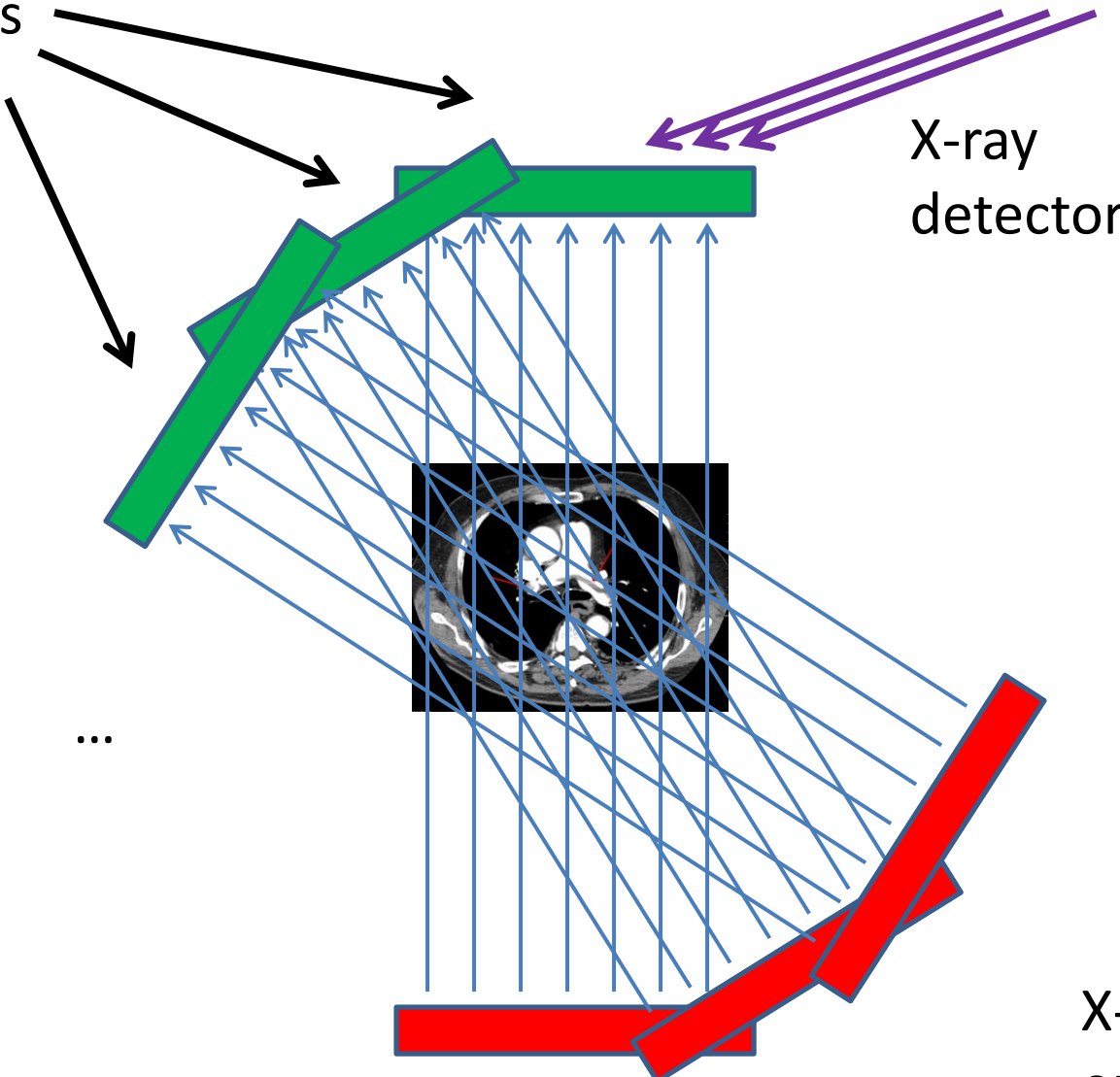
# Reconstruction

Different  $\theta$ 's



# Reconstruction

Different  $\theta$ 's



Each location on detector:  
Corresponds to multiple  $x_0$ 's

X-ray detector

X-ray emitter

# X-ray CT *Reconstruction*

- Given the results of our scan (the *sinogram*):

$$(Rf)(\vec{x}_0, \theta) = \int_{-\infty}^{\infty} f(\vec{x}_0 + t\vec{\theta}) |dt|$$

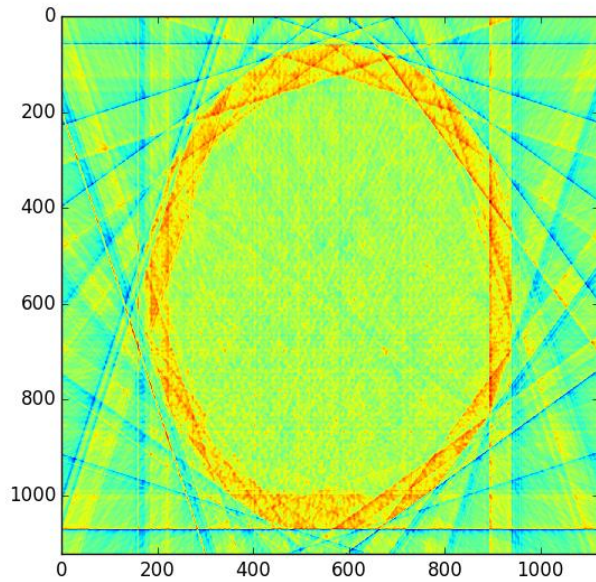
- Obtain the original data: (“*density*” of our body)

$$f(x, y)$$

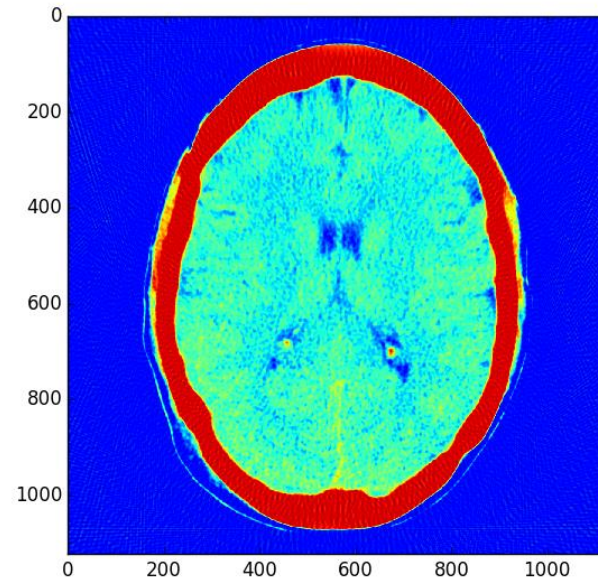
- In reality:
  - This is hard
  - We only scanned at certain (discrete) values of  $\theta$ !
    - Consequence: Perfect reconstruction is impossible!

# Imperfect Reconstruction

10 angles of imaging



200 angles of imaging





# Reconstruction

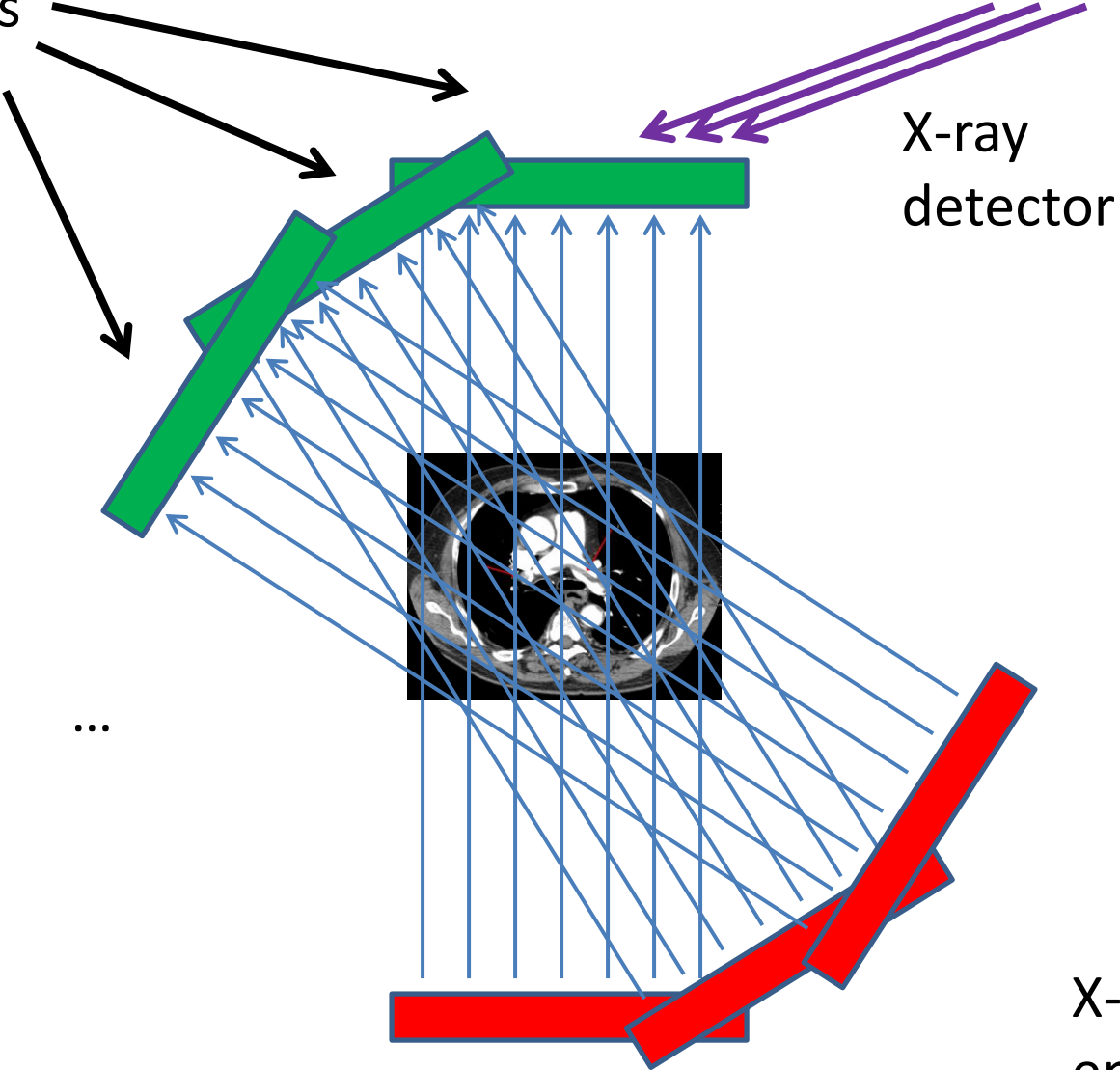
- Simpler algorithm – backprojection
  - Not quite inverse Radon transform!
- Claim: Can reconstruct image as:

$$f_r(\vec{x}) = \sum_{\theta} (Rf)(\vec{x}, \theta) = \sum_{\theta} \int_{-\infty}^{\infty} f(\vec{x} + t\vec{\theta}) |dt|$$

- ( $\theta$ 's where X-rays are taken)
- In other words: To reconstruct point, sum measurement along *every line passing through that point*

# Reconstruction

Different  $\theta$ 's



Each location on detector:  
Corresponds to multiple  $x_0$ 's

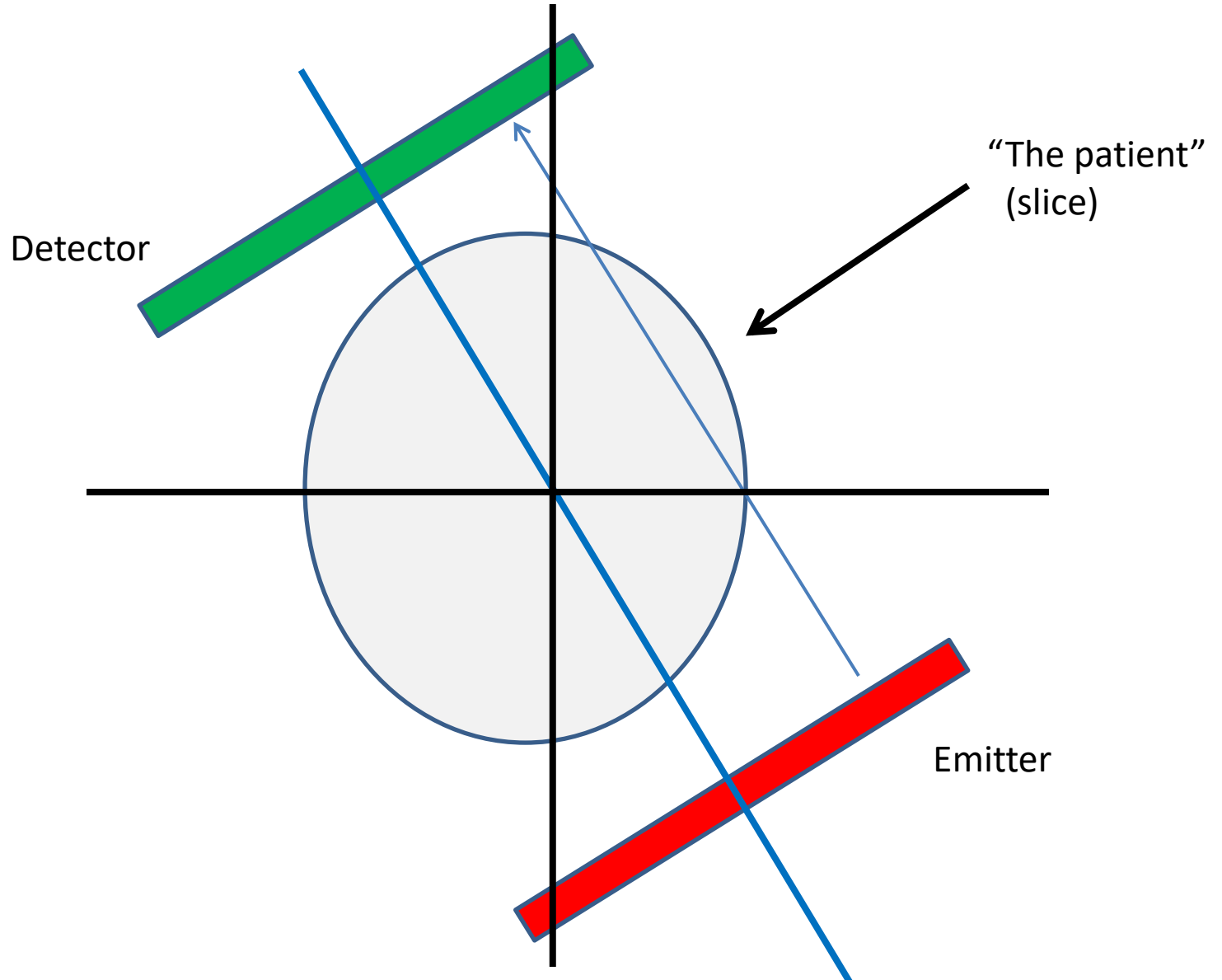
X-ray detector

X-ray emitter

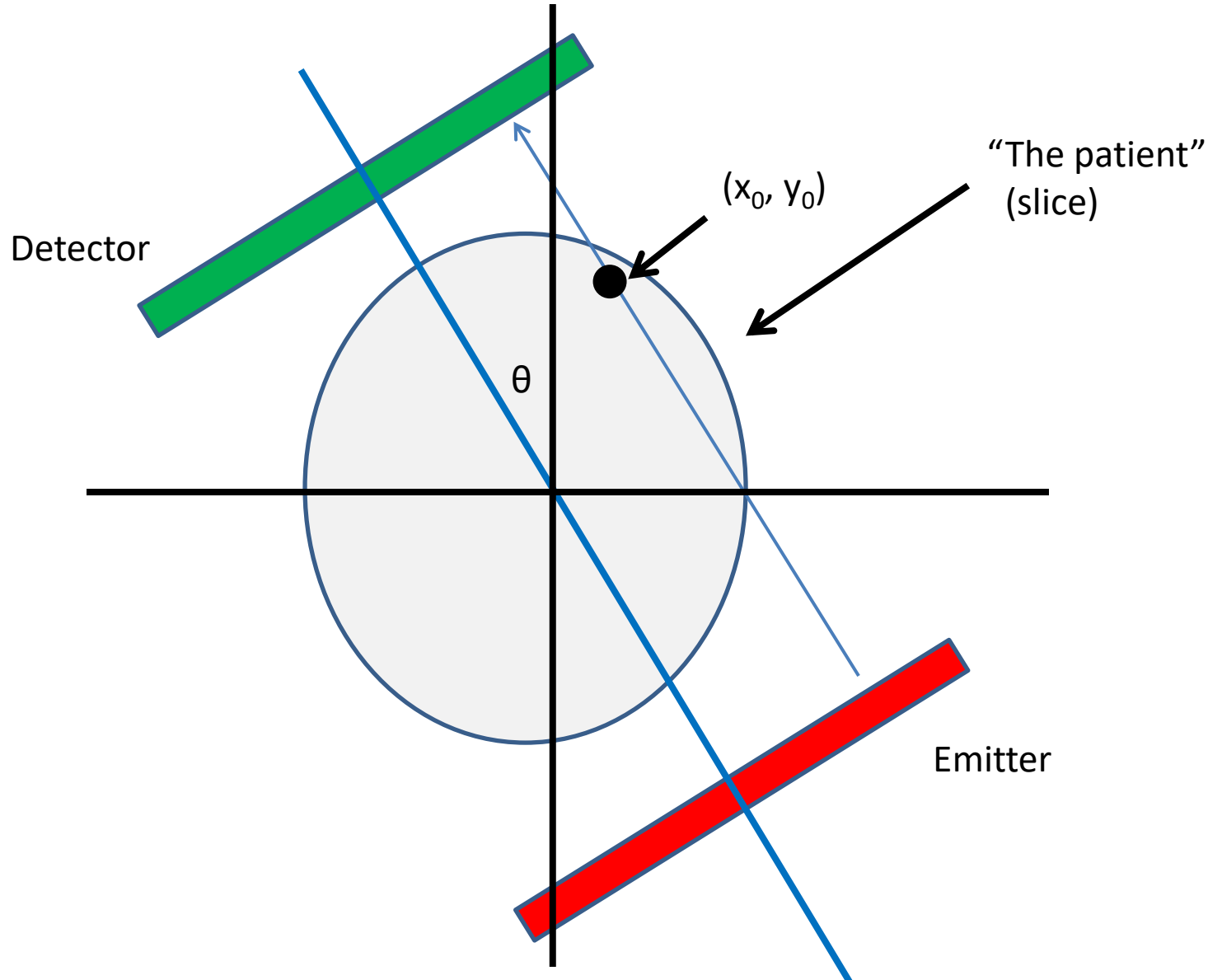
# Geometry Details

- For  $x_0$ , need to find:
  - At each  $\theta$ , which radiation measurement corresponds to the line passing through  $x_0$ ?

# Geometry Details

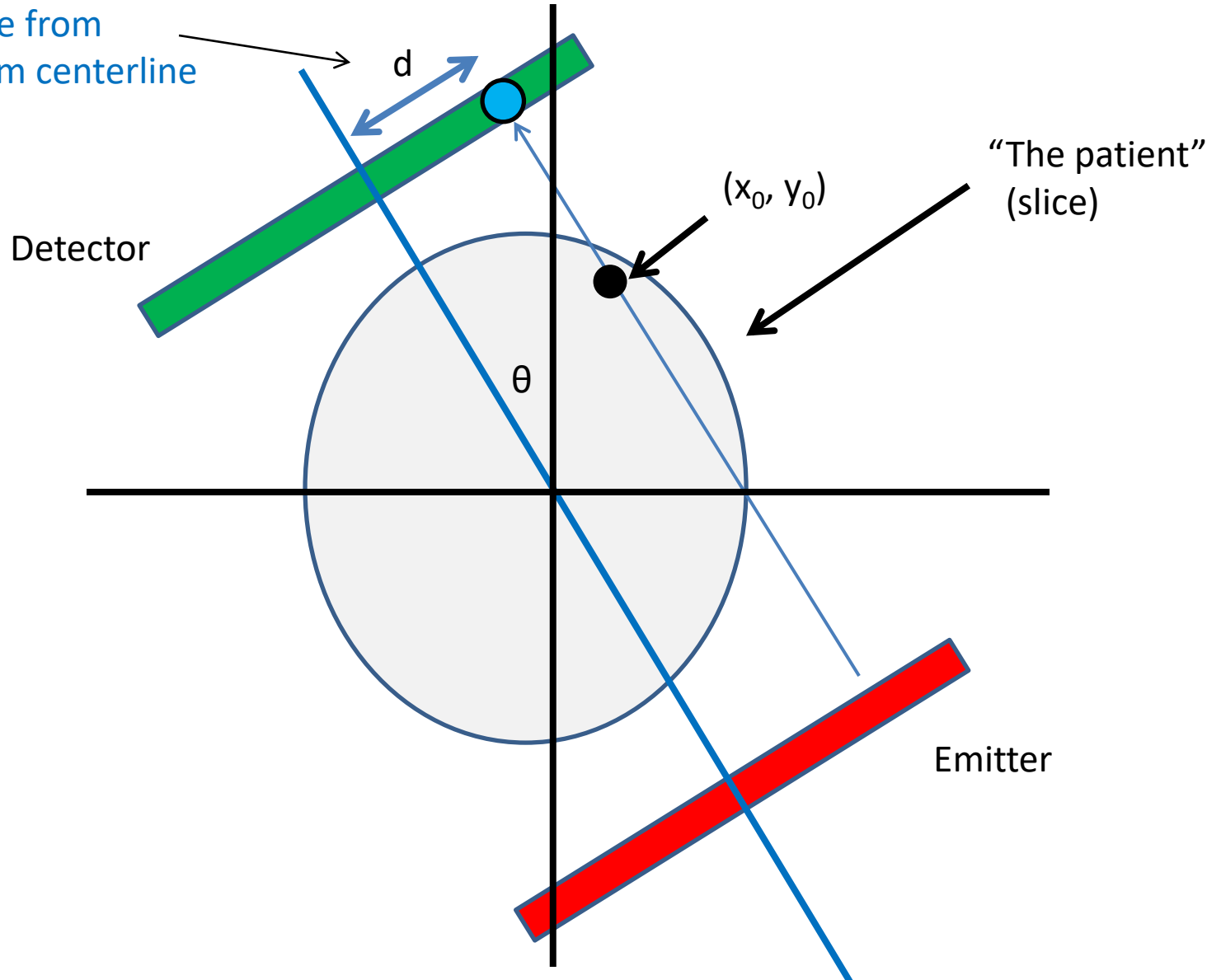


# Geometry Details



# Geometry Details

Distance from  
sinogram centerline



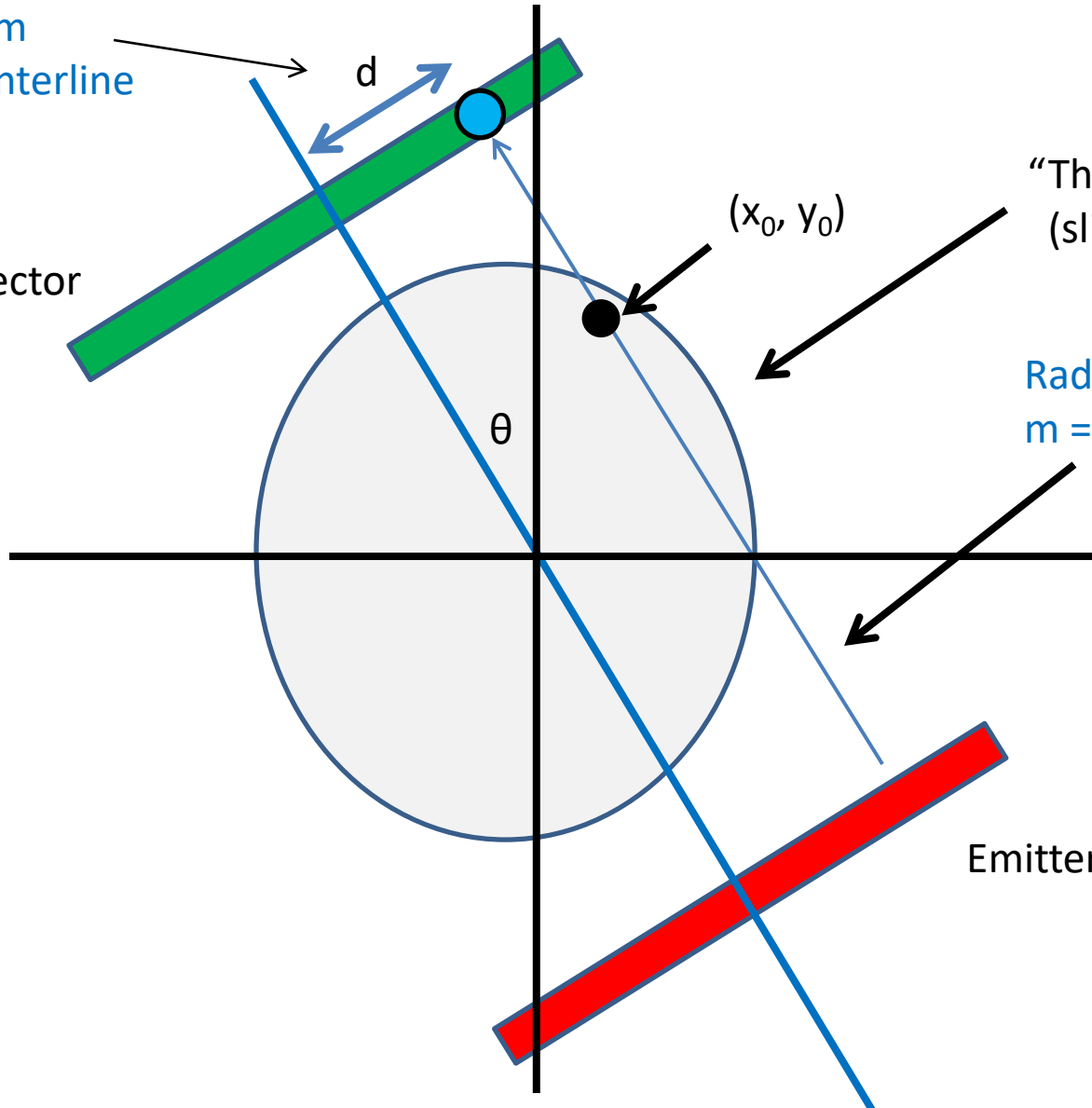
# Geometry Details

Distance from  
sinogram centerline

Detector

“The patient”  
(slice)

Radiation slope:  
 $m = -\cos(\theta)/\sin(\theta)$



$d$

$(x_0, y_0)$

$\theta$

Emitter

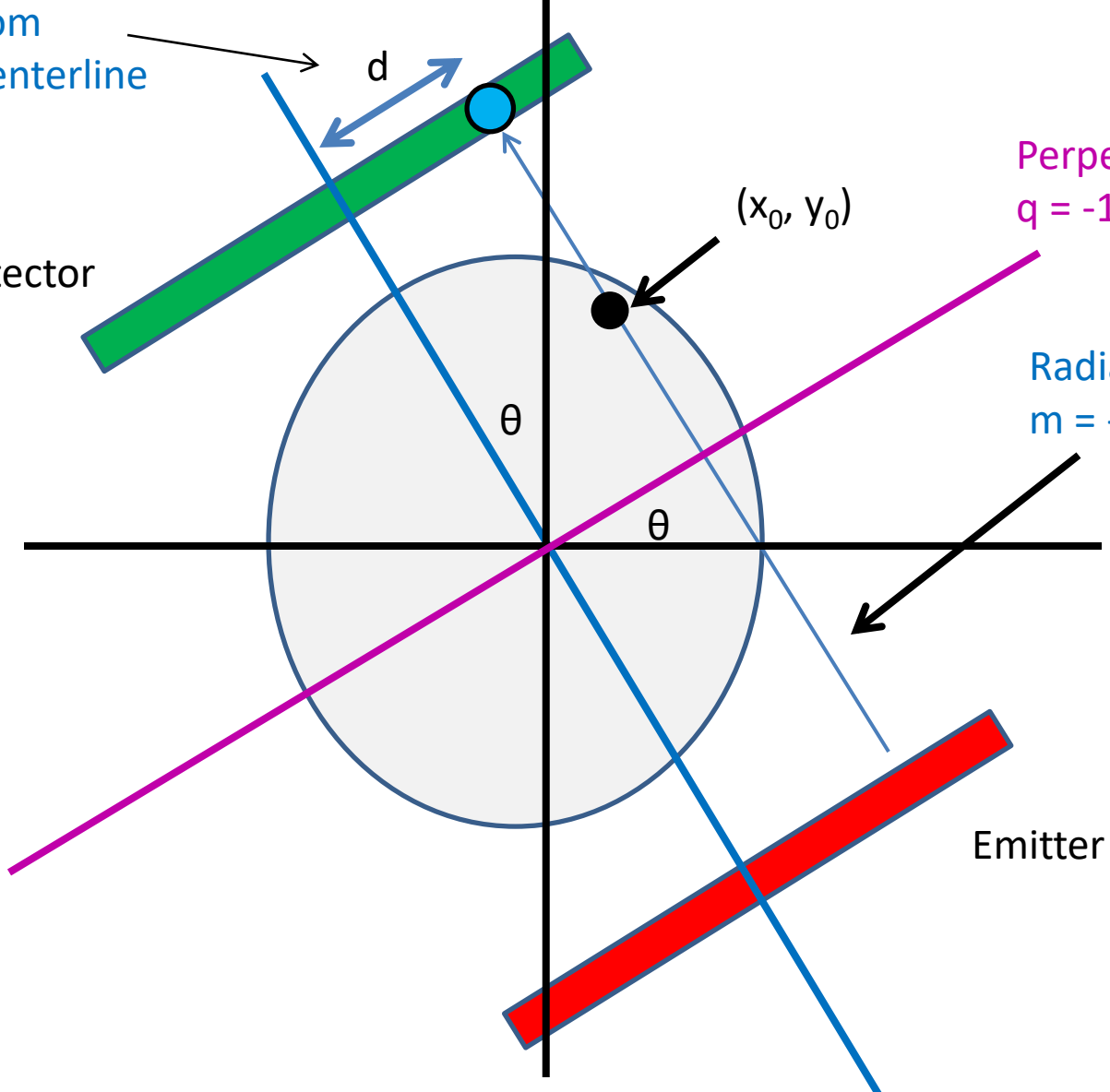
# Geometry Details

Distance from  
sinogram centerline

Detector

Perpendicular slope:  
 $q = -1/m$

Radiation slope:  
 $m = -\cos(\theta)/\sin(\theta)$



$d$

$(x_0, y_0)$

$\theta$

$\theta$

Emitter



# Geometry Details

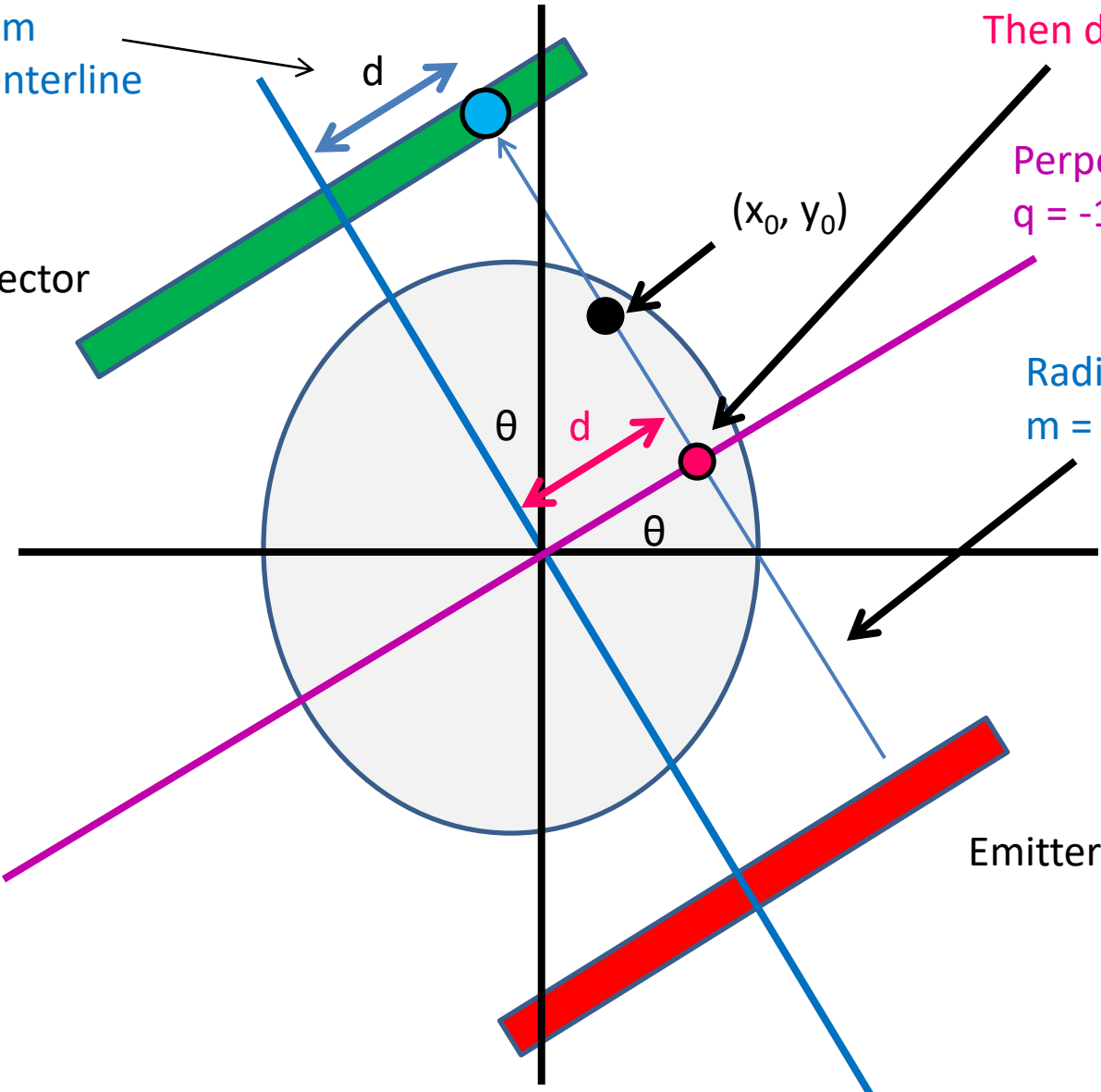
Distance from sinogram centerline

Detector

Find intersection point  $(x_i, y_i)$   
Then  $d^2 = x_i^2 + y_i^2$

Perpendicular slope:  
 $q = -1/m$

Radiation slope:  
 $m = -\cos(\theta)/\sin(\theta)$



Emitter

# Intersection point

- Line 1: (point-slope)

$$(y_i - y_0) = m(x_i - x_0)$$

- Line 2:

$$y_i = qx_i$$

- Combine and solve:

$$x_i = \frac{y_0 - mx_0}{q - m}, y_i = qx_i$$

# Intersection point

- Intersection point:

$$x_i = \frac{y_0 - mx_0}{q - m}, \quad y_i = qx_i$$

- Distance from measurement centerline:

$$d = \sqrt{x_i^2 + y_i^2}$$

# Geometry Details

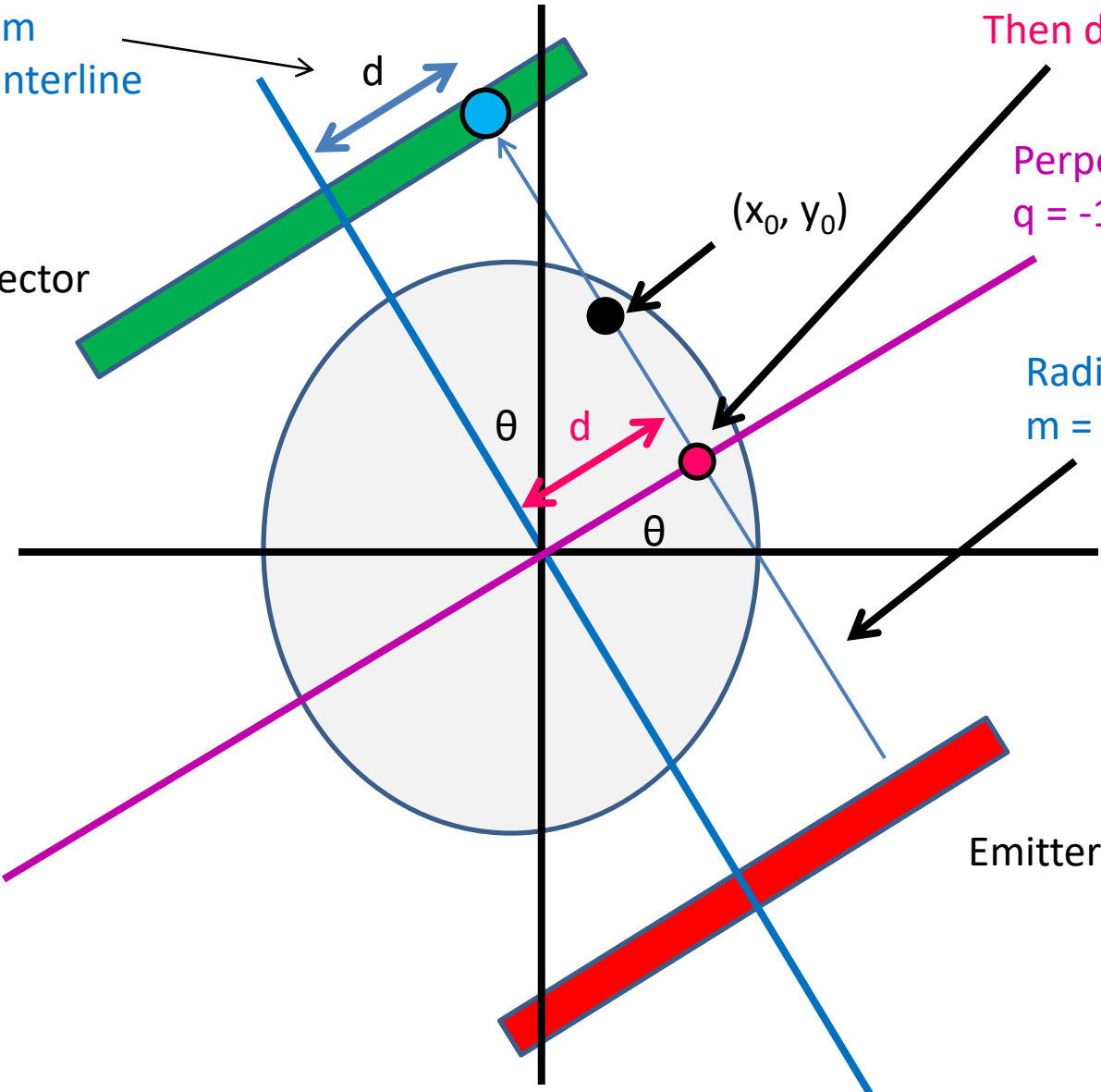
Distance from sinogram centerline

Detector

Find intersection point  $(x_i, y_i)$   
Then  $d^2 = x_i^2 + y_i^2$

Perpendicular slope:  
 $q = -1/m$

Radiation slope:  
 $m = -\cos(\theta)/\sin(\theta)$



Emitter

# Sequential pseudocode

(input: X-ray sinogram):  
(allocate output image)

$$f_r(\vec{x}) = \sum_{\theta} (Rf)(\vec{x}, \theta)$$

```
for all y in image:  
  for all x in image:  
    for all theta in sinogram:
```

```
      calculate m from theta  
      calculate x_i, y_i from -1/m,  
      calculate d from x_i, y_i  
      image[x,y] += sinogram[theta, "distance"]
```

Clarification: Remember not to confuse geometric x,y with pixel x,y!

(0,0) geometrically is the center pixel of the image, and (0,0) in pixel coordinates is the upper left hand corner. Image is indexed row-wise

Clarification:

- d is the distance from the center of the sinogram – remember to center index appropriately
- Use  $-d$  instead of  $d$  as appropriate (when  $-1/m > 0$  and  $x_i < 0$ , or if  $-1/m < 0$  and  $x_i > 0$ )

# Sequential pseudocode

(input: X-ray sinogram):  
(allocate output image)

$$f_r(\vec{x}) = \sum_{\theta} (Rf)(\vec{x}, \theta)$$

```
for all y in image:  
  for all x in image:  
    for all theta in sinogram:  
      calculate m from theta  
      calculate x_i, y_i from -1/m  
      calculate d from x_i, y_i  
      image[x,y] += sinogram[theta, "distance"]
```

Parallelizable!  
Inside loop depends  
only on x, y, theta

# Sequential pseudocode

(input: X-ray sinogram):  
(allocate output image)

$$f_r(\vec{x}) = \sum_{\theta} (Rf)(\vec{x}, \theta)$$

```
for all y in image:  
  for all x in image:  
    for all theta in sinogram:  
      calculate m from theta  
      calculate x_i, y_i from -1/m  
      calculate d from x_i, y_i  
      image[x,y] += sinogram[theta, "distance"]
```

For this assignment, only  
parallelize w/r/to x, y

(provides lots of  
parallelization already,  
other issues)

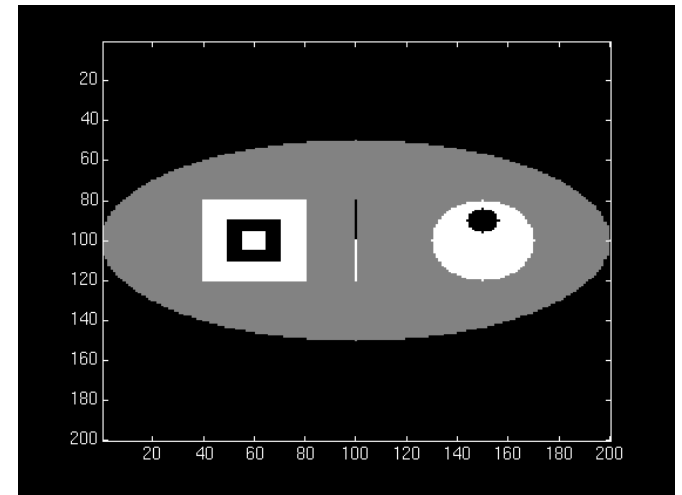
# Cautionary notes

- $y$  in an image is opposite of  $y$  geometrically!
  - (Graphics/computing convention)
- Edge cases (divide-by-0):
  - $\theta = 0$ :
    - $d = x_0$
  - $\theta = \pi/2$ :
    - $d = y_0$

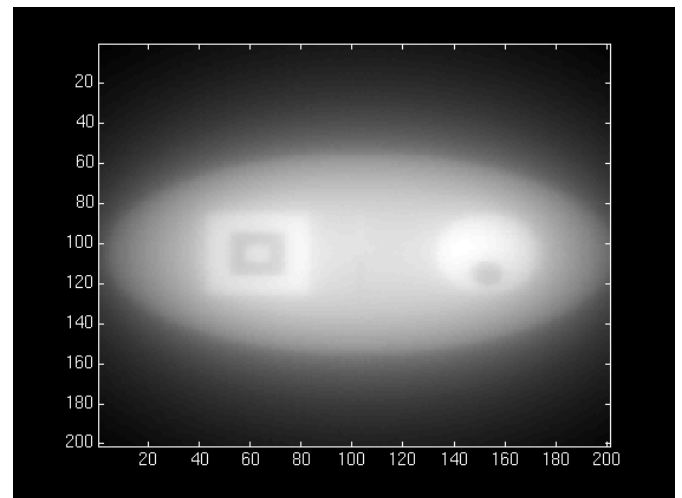


# Almost a good reconstruction!

Original

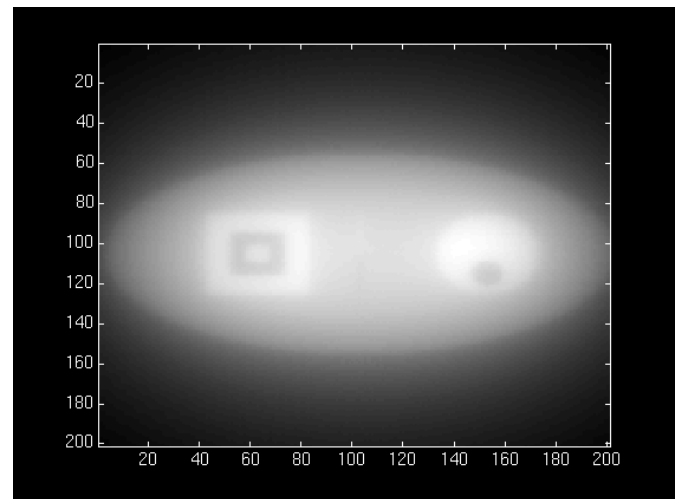
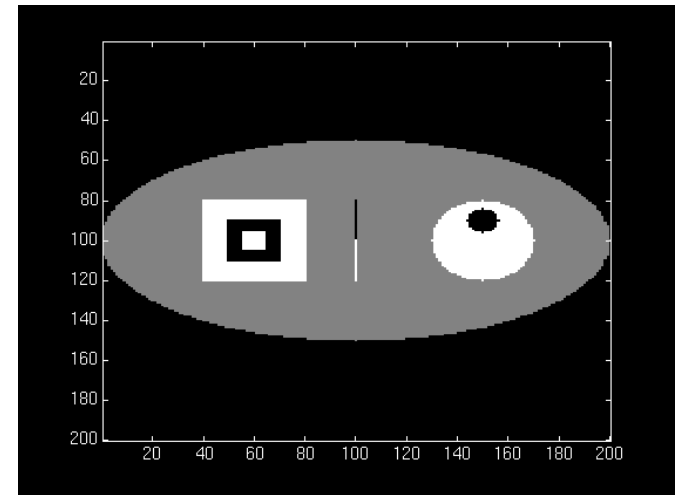


Reconstruction



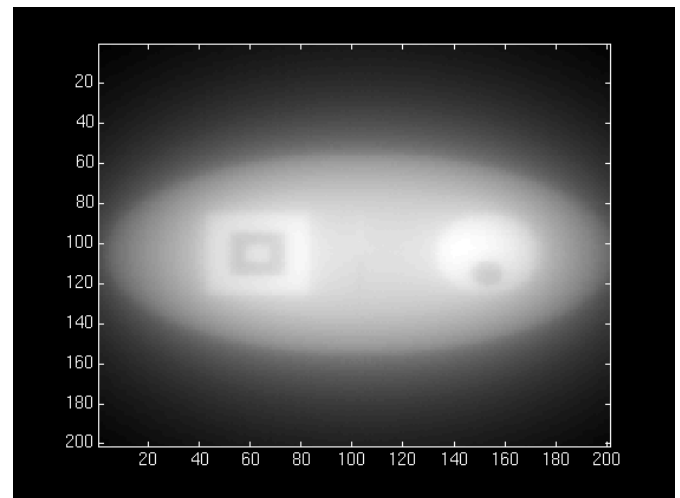
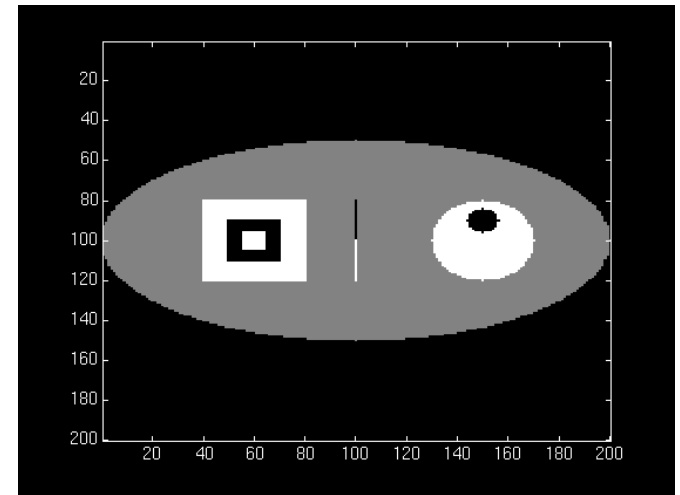
# Almost a good reconstruction!

- “Backprojection blur”
  - Similar to low-pass property of SMA (Week 1)
  - We need an “anti-blur”! (opposite of Homework 1)



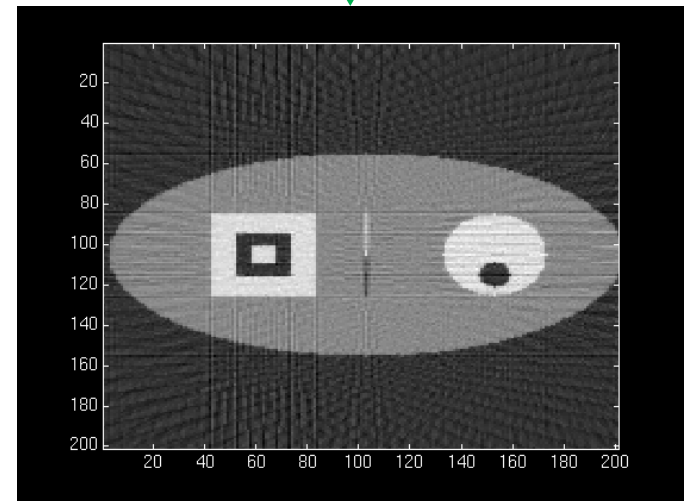
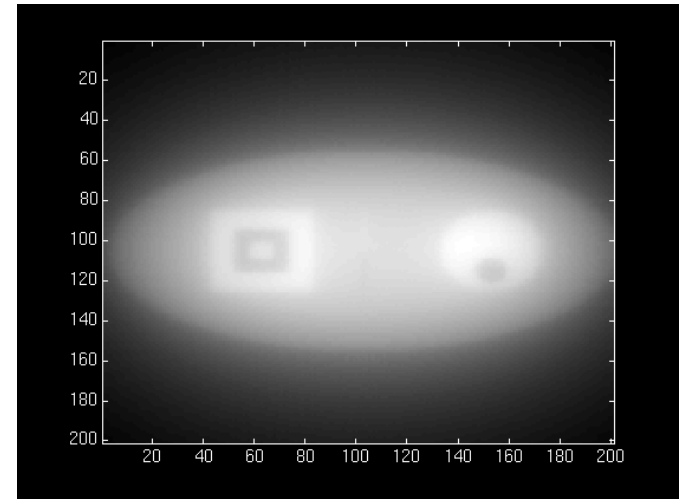
# Almost a good reconstruction!

- Solution:
  - A “high-pass filter”
  - We can get frequency info in parallelizable manner!
    - (FFT, Week 3)



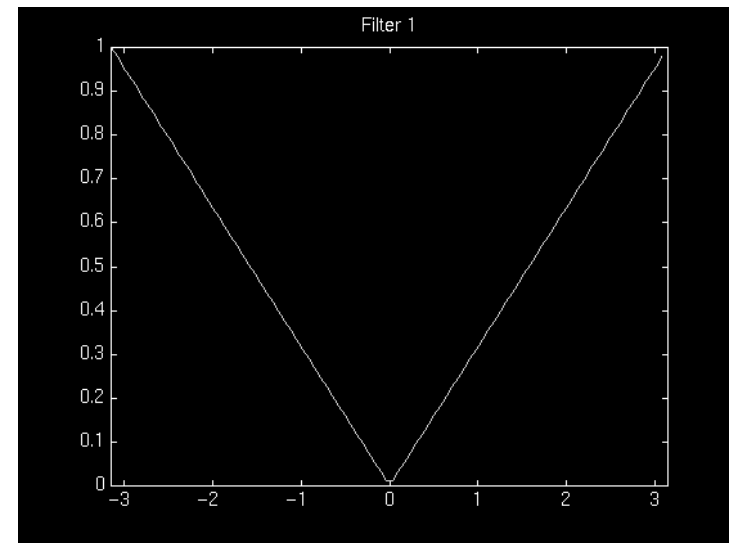
# Almost a good reconstruction!

- Solution:
  - A “high-pass filter”
  - We can get frequency info in parallelizable manner!
    - (FFT, Week 3)



# High-pass filtering

- Instead of filtering on image (2D HPF):
  - Filter on sinogram! (1D HPF)
    - (Equivalent reconstruction by linearity)
  - Use cuFFT batch feature!
- We'll use a “ramp filter”
  - Retained amplitude is linear function of frequency



# Almost a good reconstruction!

- CPU-side:

(input: X-ray sinogram):

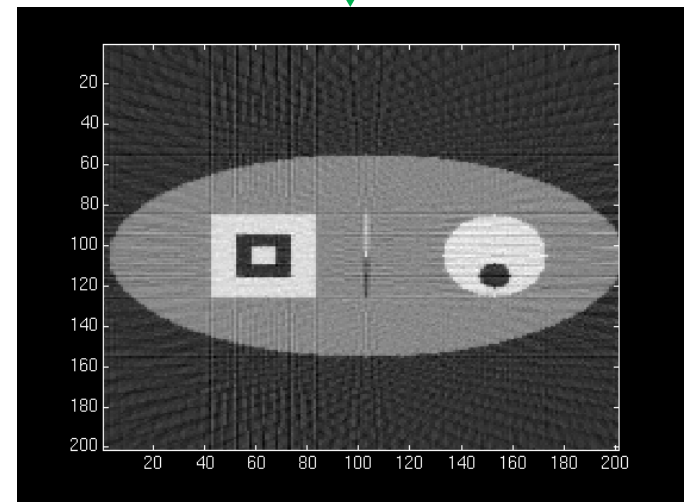
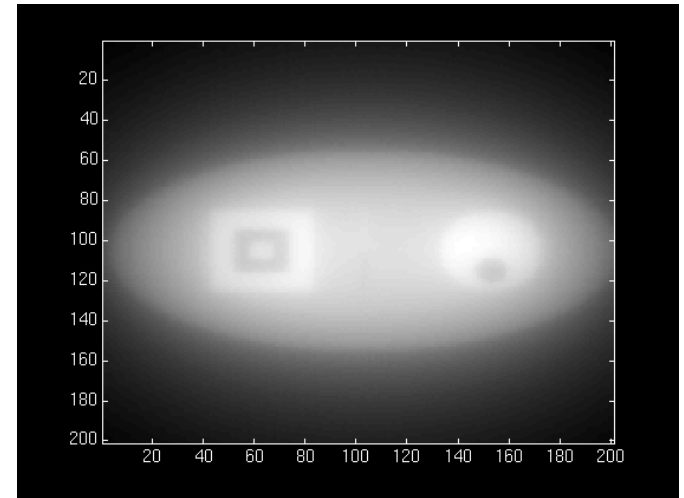
```
calculate FFT on sinogram using cuFFT
call filterKernel on freq-domain data
Calculate IFFT on freq-domain data
-> get new sinogram
```

- GPU-side:

filterKernel:

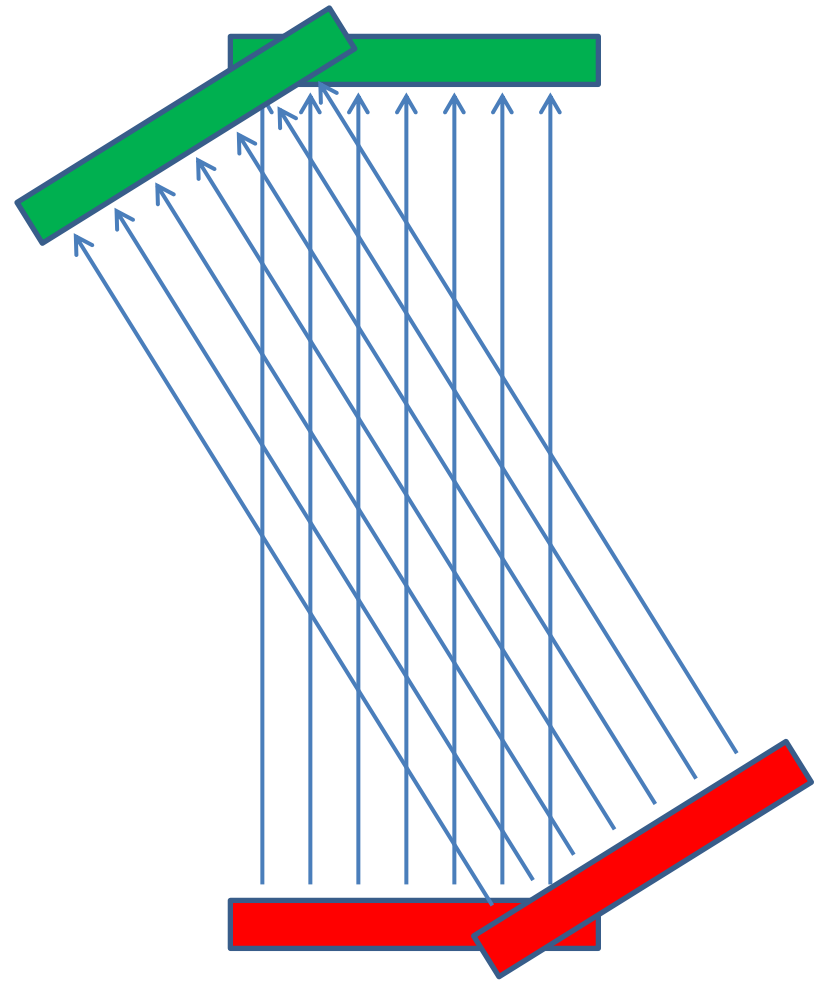
```
Select specific freq-amplitude
based on thread ID
```

```
Get new amplitude from
ramp equation
```



# GPU Hardware

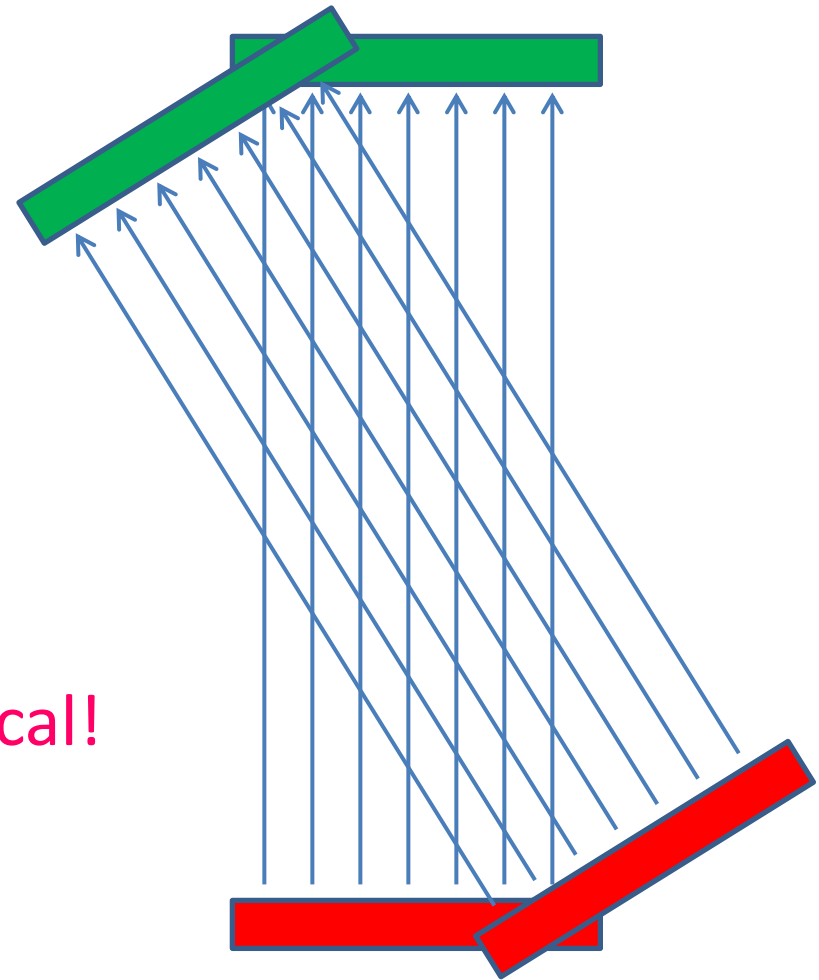
- Non-coalesced access!
  - Sinogram 0, index  $\sim d_0$
  - Sinogram 1, index  $\sim d_1$
  - Sinogram 2, index  $\sim d_2$
  - ...



# GPU Hardware

- Non-coalesced access!
  - Sinogram 0, index  $\sim d_0$
  - Sinogram 1, index  $\sim d_1$
  - Sinogram 2, index  $\sim d_2$
  - ...

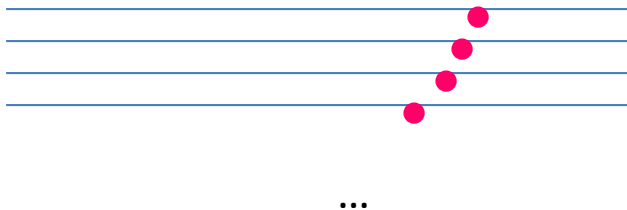
- However:
  - Accesses are 2D spatially local!





# GPU Hardware

- Solution:
  - Cache sinogram in texture memory!
    - Read-only (un-modified once we load it)
    - Ignore coalescing
    - 2D spatial caching!



# Summary/pseudocode

(input: X-ray sinogram)

Filter sinogram (Slide 46)

Set up 2D texture cache on sinogram (Lecture 10):

Copy to CUDA array (2D)

Set addressing mode (clamp)

Set filter mode (linear, but won't matter)

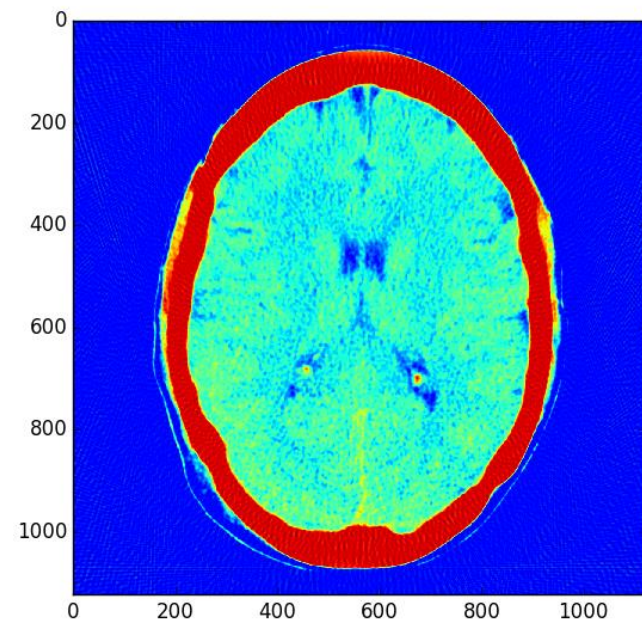
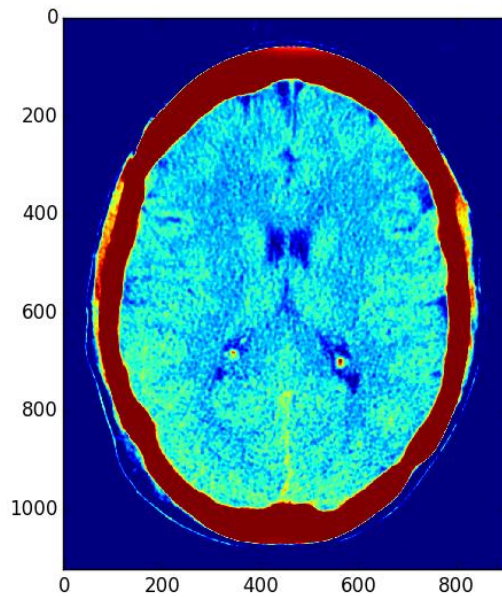
Set no normalization

Bind texture to sinogram

calculate image backprojection (parallelize slide 39)

- **Result: 200-250x speedup! (or more)**

- Result: 200-250x speedup! (or more)



# Admin

- This topic is harder than before!
  - Lots of information
  - I may have missed something
- If there's anything unclear, let us know
  - I can (and likely will) make additional slides/explanatory materials

# Admin

- C/CUDA code should work on all machines
- Pre/post-processing:
  - Python scripts preprocess.py, postprocess.py
    - (To run Python scripts: “python <script>.py”)
  - Either:
    - Use haru
    - Install python, (optionally pip) -> numpy, scipy, matplotlib, scikit-image

# Resources

- Imaging methods:
  - [X-Ray CT in Nuclear Medicine](#)
  - [CT Image Reconstruction \(Peters, at AAPM\)](#)
  - [Elements of Modern Signal Processing \(Candes, at Stanford\)](#)
    - Proof that our algorithm works!