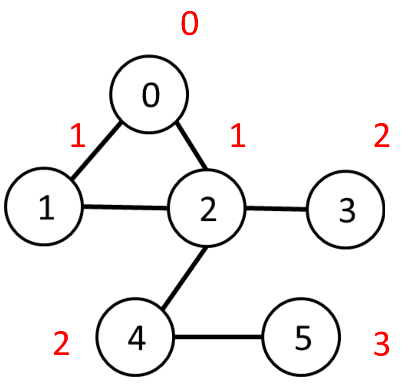
CS 179: GPU Programming

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: 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:

   for 0 ≤ i < |Va|:
      if F[i] is true:

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

         for Ea[Va[i]] ≤ j < Ea[Va[i+1]]:
            if X[j] is false:

               C[j] <- C[i] + 1
               F[j] <- true
```

Parallelizable!

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
```

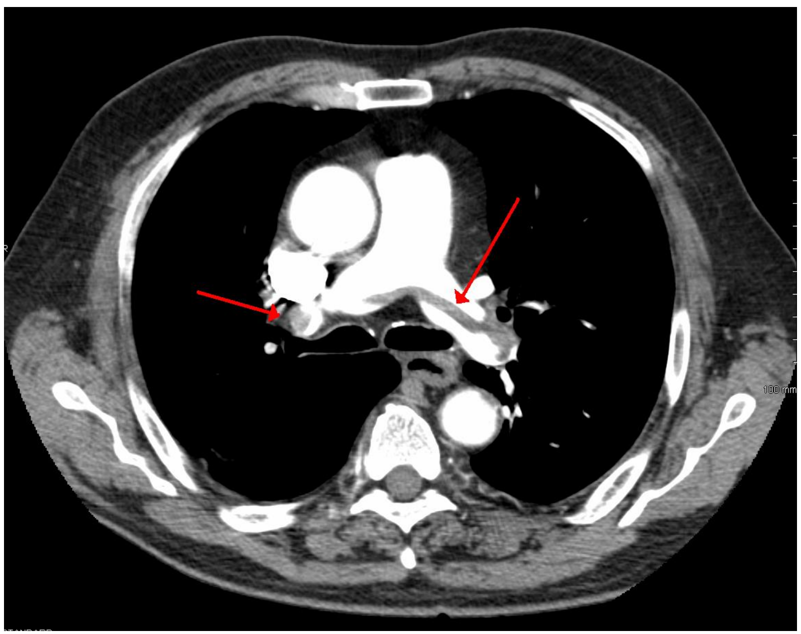
Texture Memory (and co-stars)

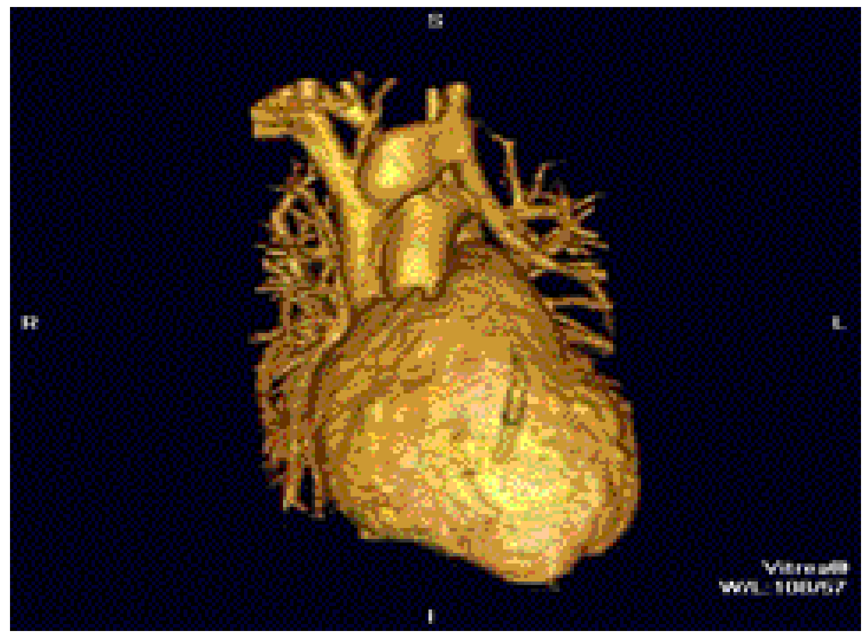
- Another type of memory system, featuring:
  - Spatially-cached read-only access
  - Avoid coalescing worries
  - Interpolation
  - (Other) fixed-function capabilities
  - Graphics interoperability

X-ray CT Reconstruction

Medical Imaging

- See inside people!
  - Critically important in medicine today



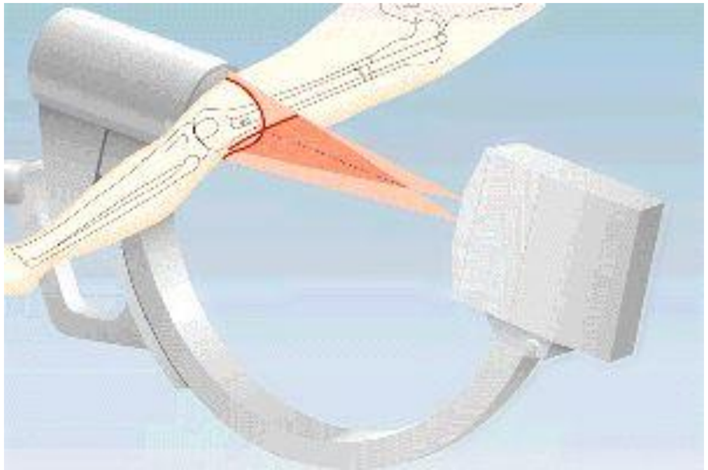


X-ray imaging (Radiography)

- "Algorithm":
  - Generate electromagnetic radiation
  - Measure radiation at the "camera"

- Certain tissues are more "opaque" to X-rays

- Like photography!



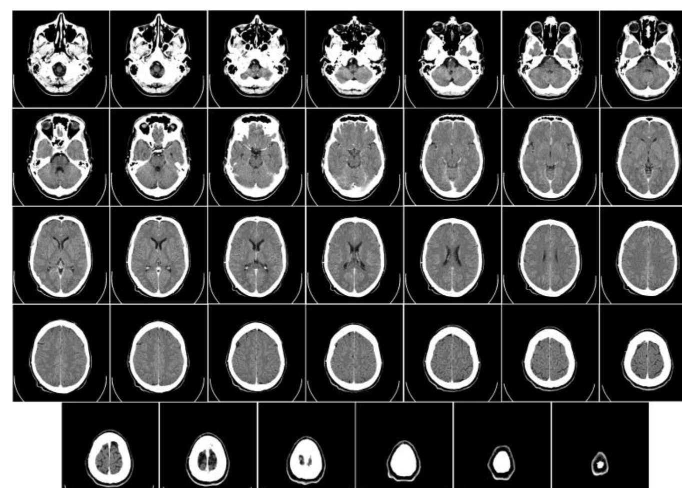
http://www.imaginis.com/xray/how-does-x-ray-imaginig-work

Radiography limitations

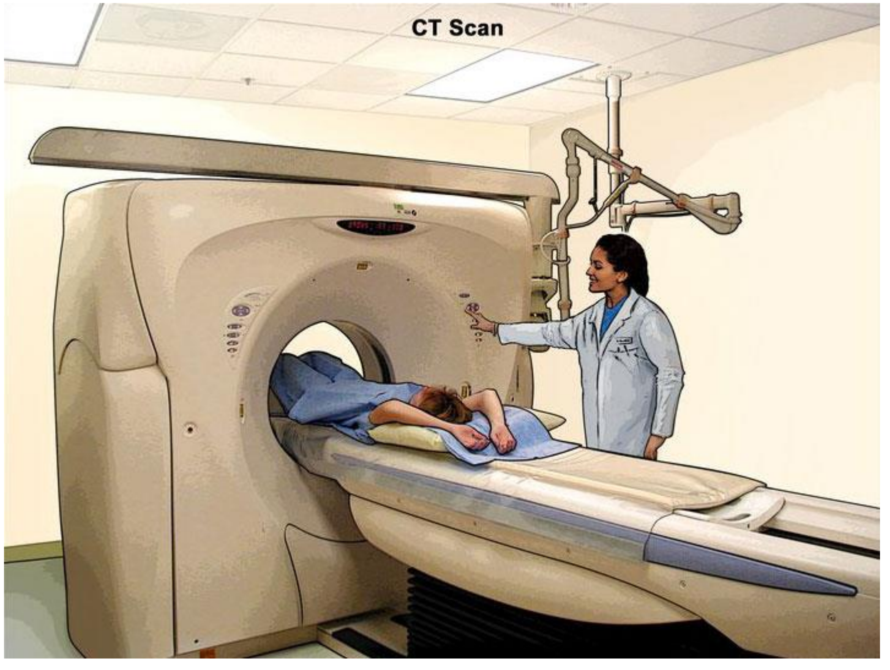
- Generates 2D image of 3D body

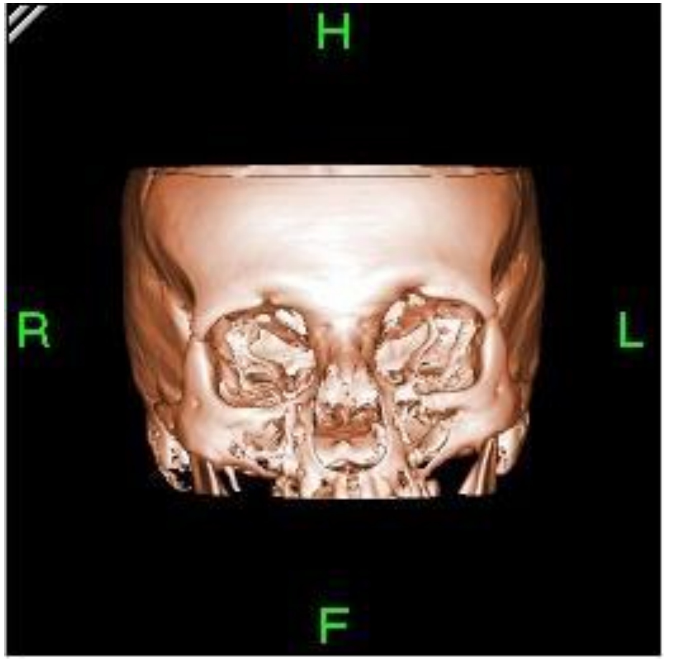
- What if we want a “slice” of 3D body?
  – Goal: 3D reconstruction! (from multiple slices)

vs.



X-ray Computed Tomography (CT)





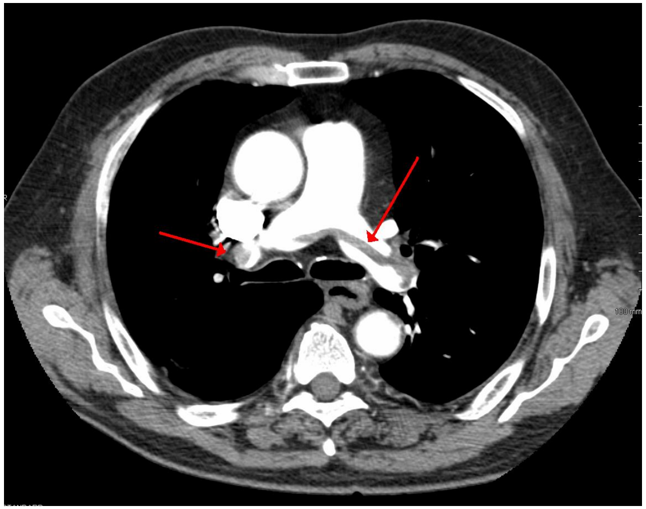
http://www.cancer.gov/

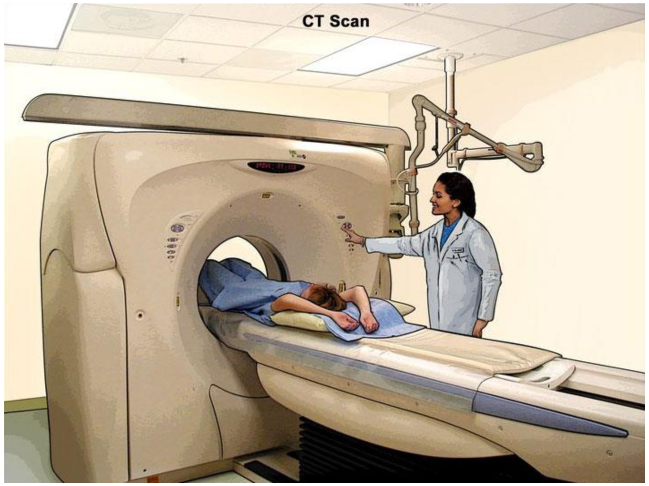
"Bonereconstruction" by Original uploader was Zgyorfi at en.wikipedia - Transferred from en.wikipedia; transferred to Commons by User:Common Good using CommonsHelper.. Licensed under CC BY-SA 3.0 via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:Bonereconstruction.jpg#/media/File:Bonereconstruction.jpg

X-ray Computed Tomography (CT)

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

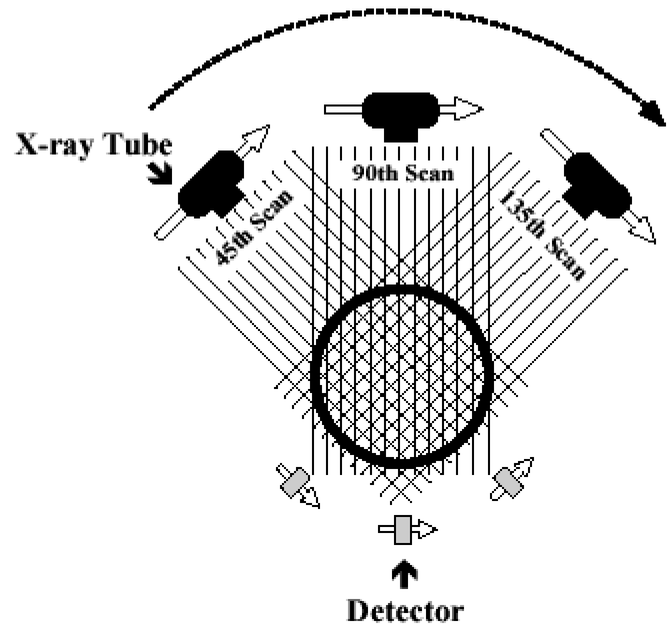
- Reconstruction less straightforward





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

http://www.thefullwiki.org/Basic_Physics_of_Nuclear_Medicine/X-Ray_CT_in_Nuclear_Medicine

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

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

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

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

Mathematical Details

- For each line L of radiation measured by detector:

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

- Define general X-ray transform (for all lines L in R^2):

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

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

Mathematical Details

- Define general X-ray transform:

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

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

- Redefine as:

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

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

Mathematical Details

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

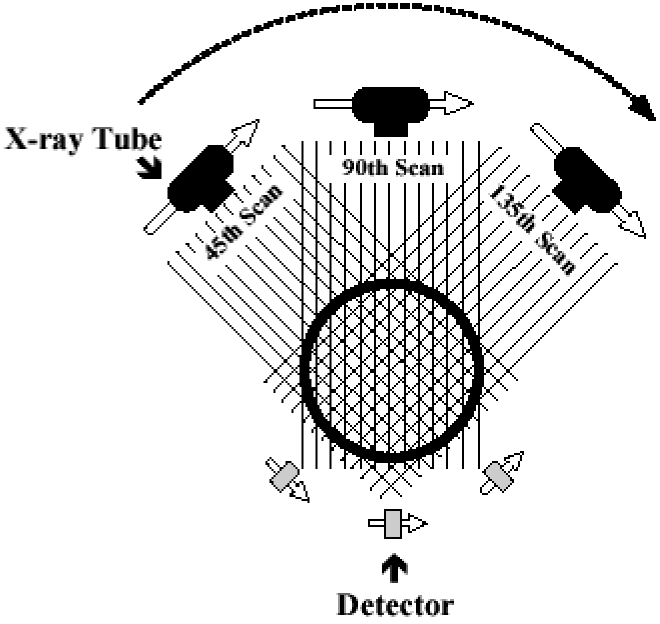
- Important properties:
  - Many $\vec{x}_0$ are redundant!
  - Symmetry: $Rf(\vec{x}_0, \theta) = Rf(\vec{x}_0, \theta + \pi)$
    - Can define for $\theta \in [0, \pi)$

X-ray Computed Tomography (CT)

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

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

- In reality:
  - Only defined for some θ!



X-ray CT *Reconstruction*

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

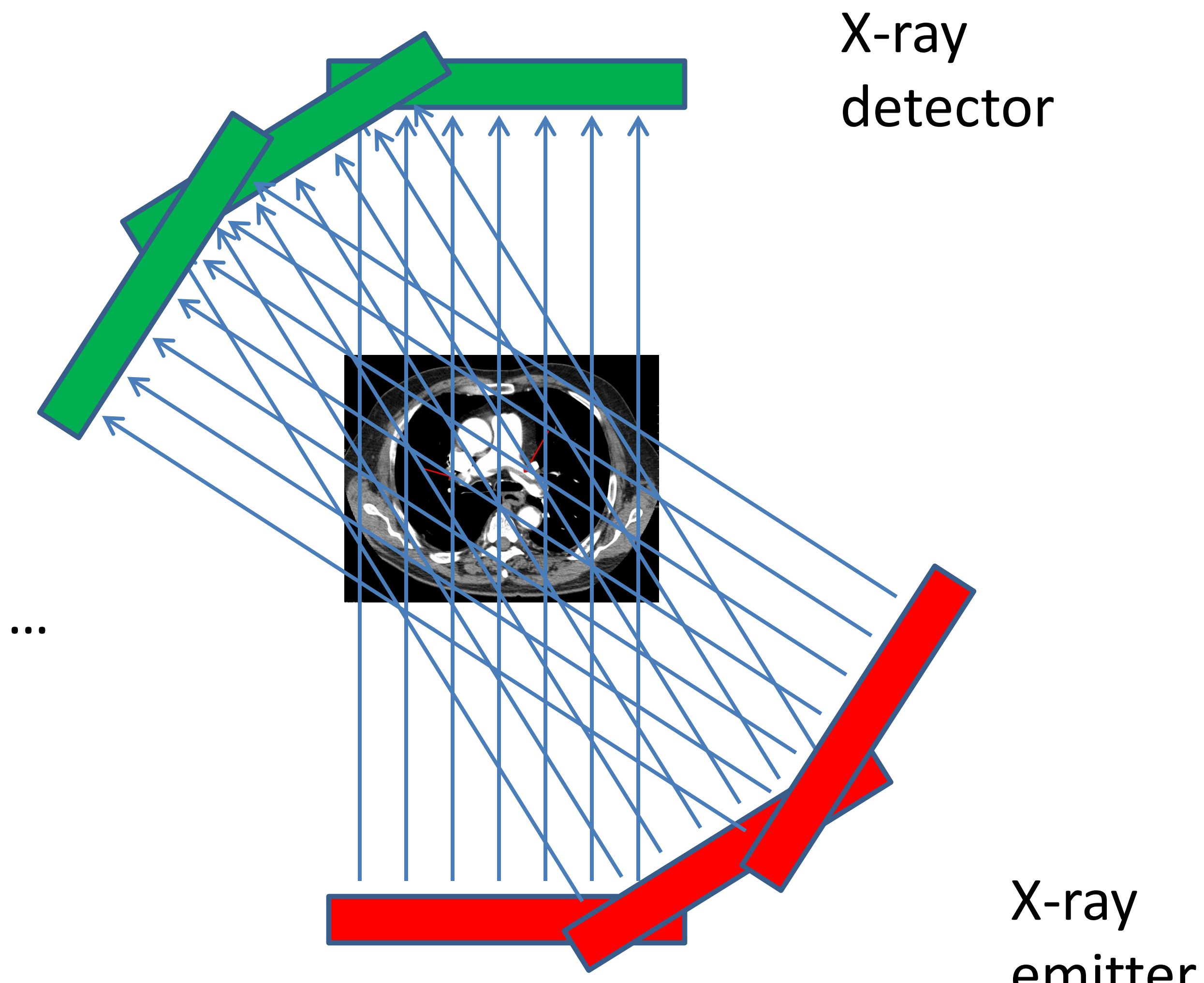
$$(Rf)(\vec{x}_0, \theta) = \int_{-\infty}^{\infty} f\left(\vec{x}_0 + t\vec{\theta}\right) |dt|$$

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

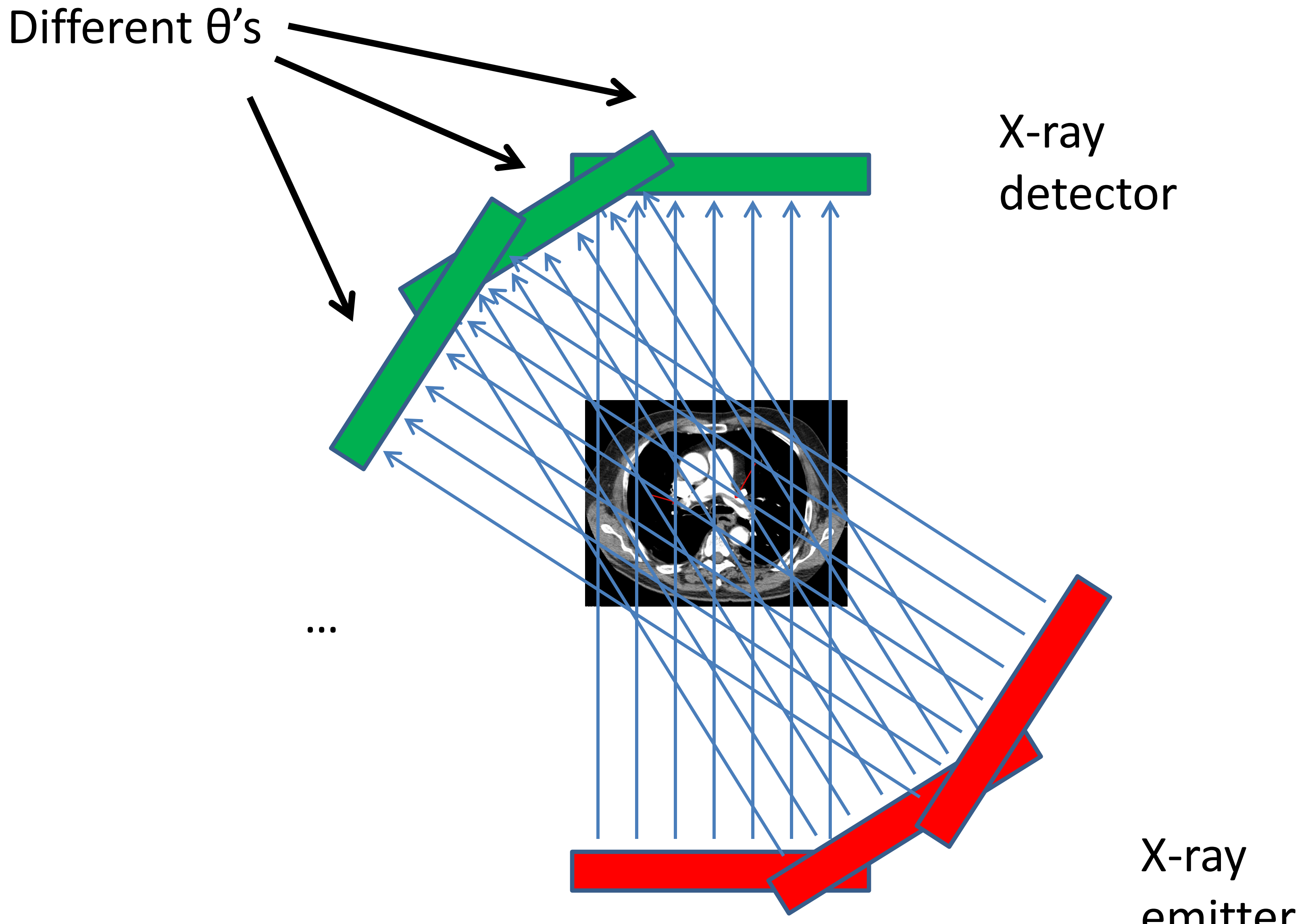
$$f(x, y)$$

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

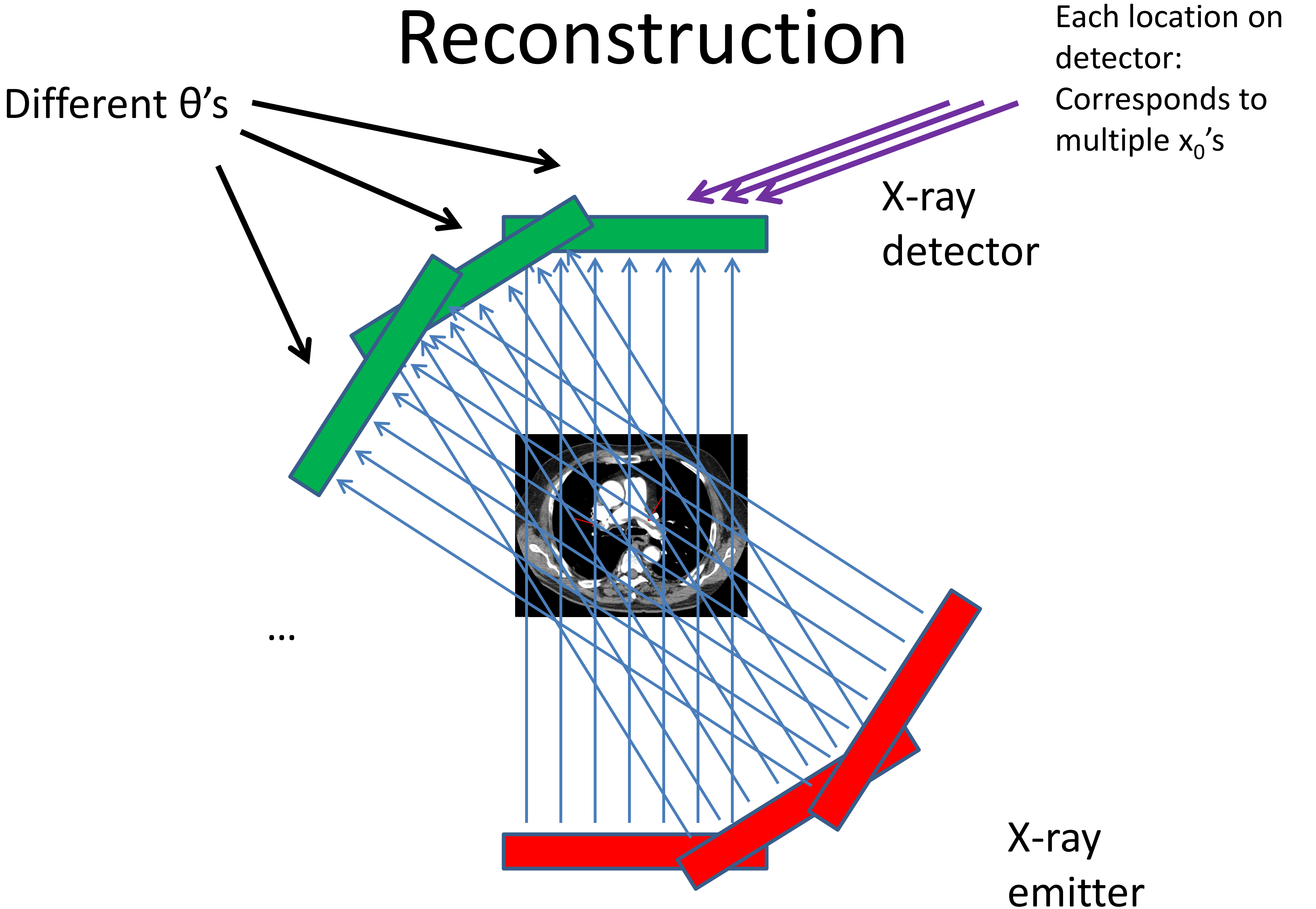
Reconstruction



Reconstruction



Reconstruction



X-ray CT *Reconstruction*

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

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

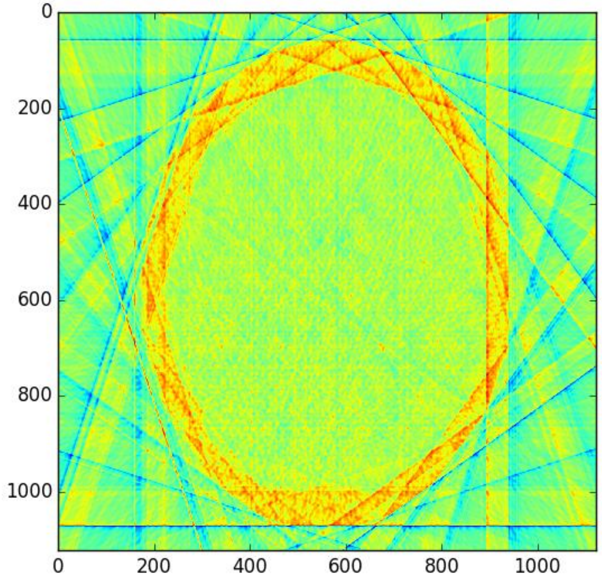
- Obtain the original data: (*"density" of our body*)

$$f(x, y)$$

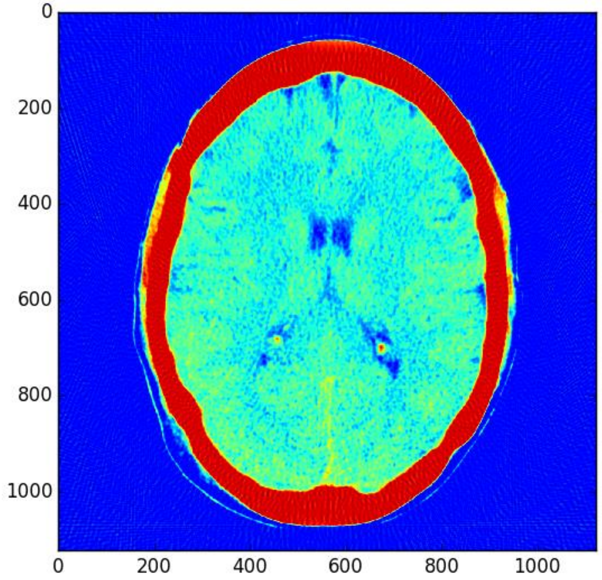
- In reality:
  - This is hard
  - We only scanned at certain (discrete) values of θ!
    - Consequence: Perfect reconstruction is impossible!

Imperfect Reconstruction

10 angles of imaging



200 angles of imaging



Reconstruction

- Simpler algorithm – backprojection
  - Not quite inverse Radon transform!

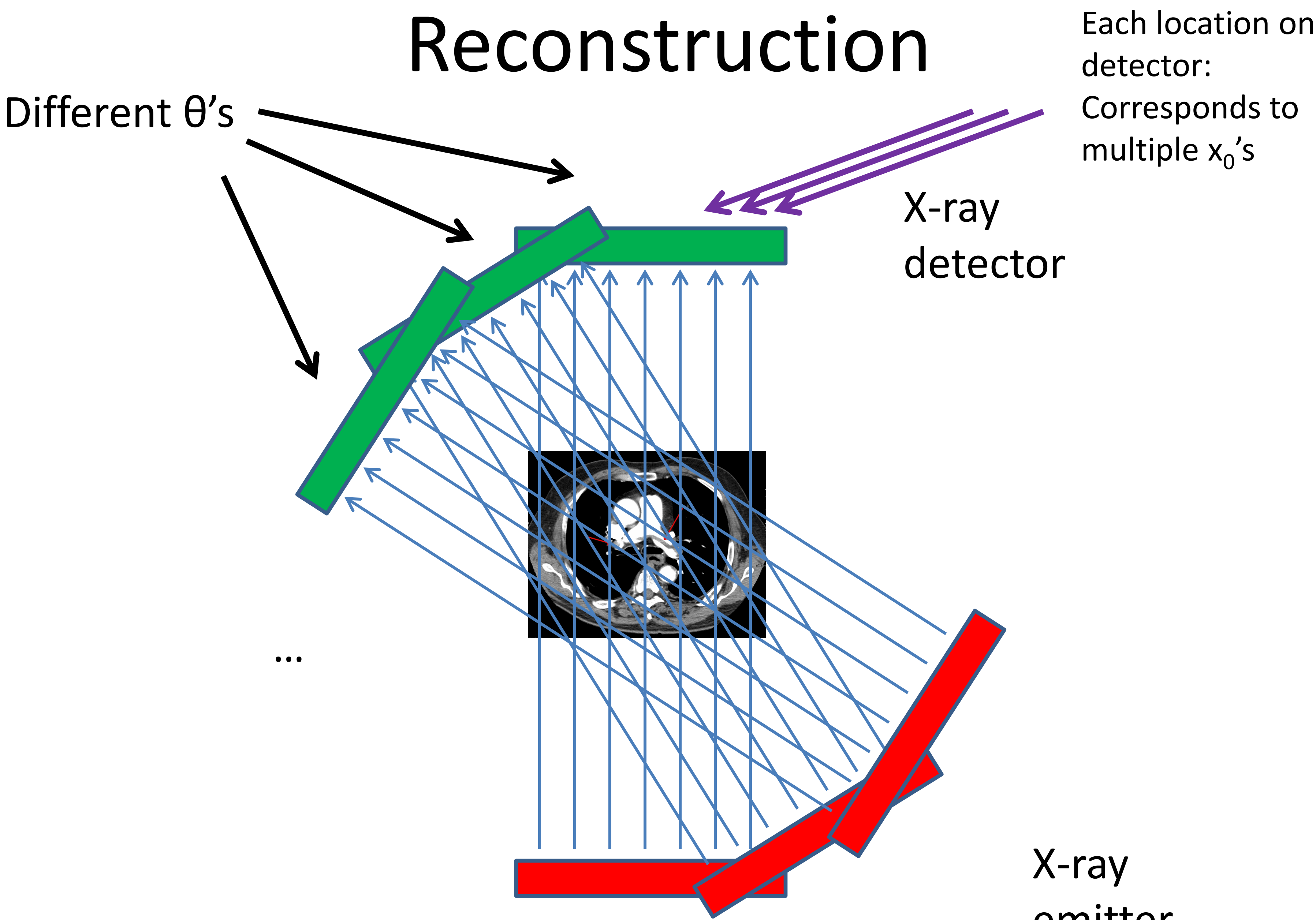
- Claim: Can reconstruct image as:

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

  - (θ's where X-rays are taken)

  - In other words: To reconstruct point, sum measurement along *every line passing through that point*

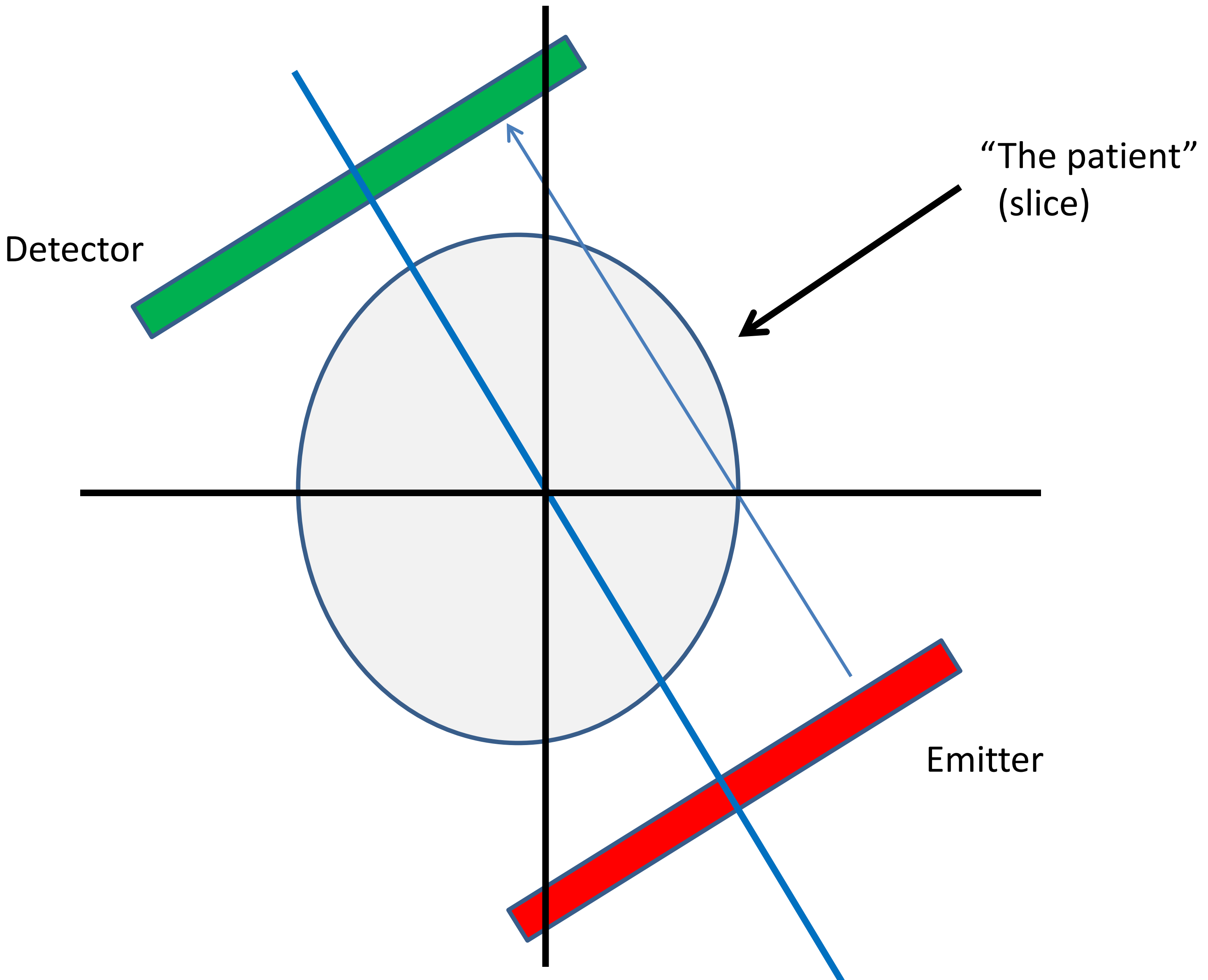
Reconstruction



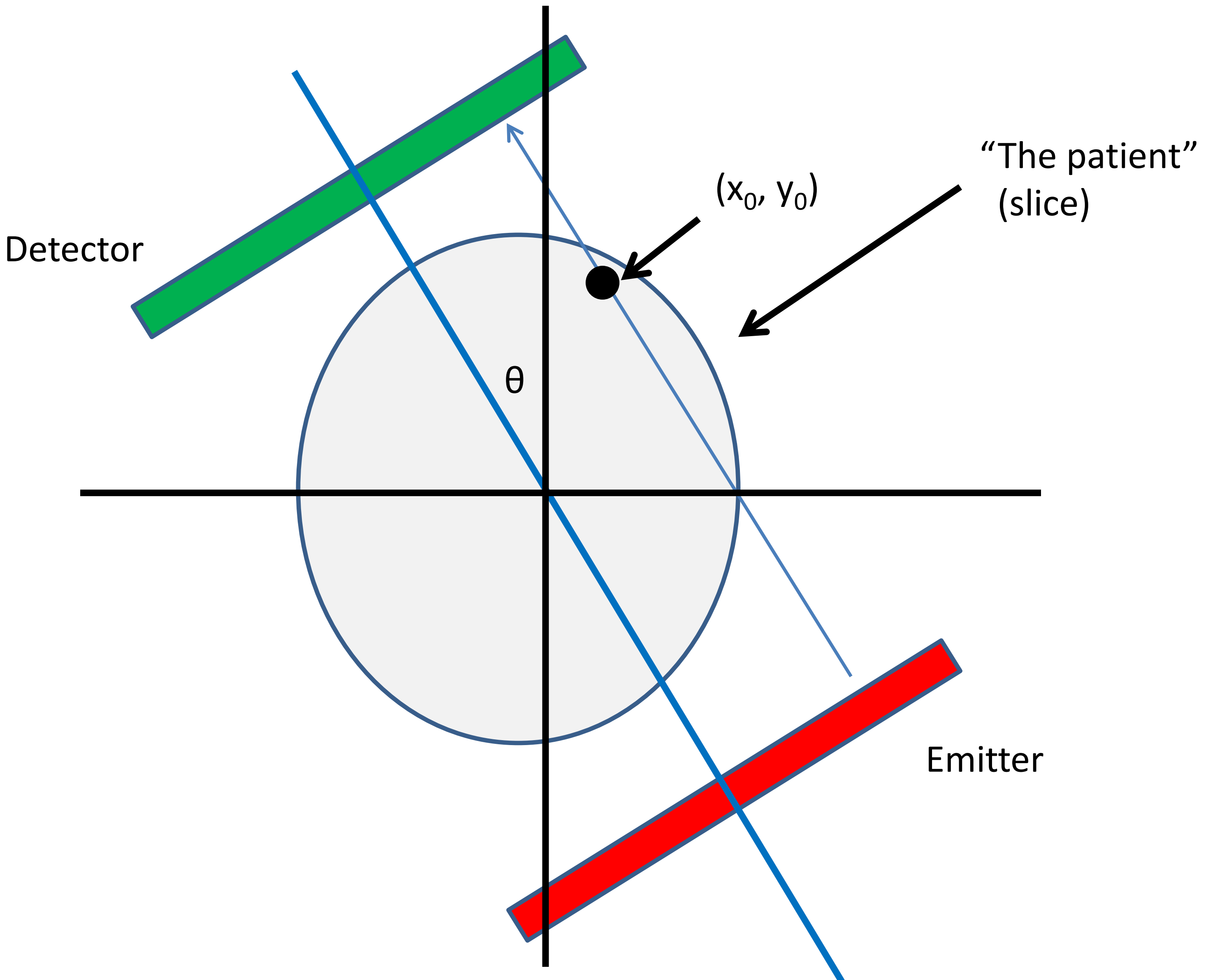
Geometry Details

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

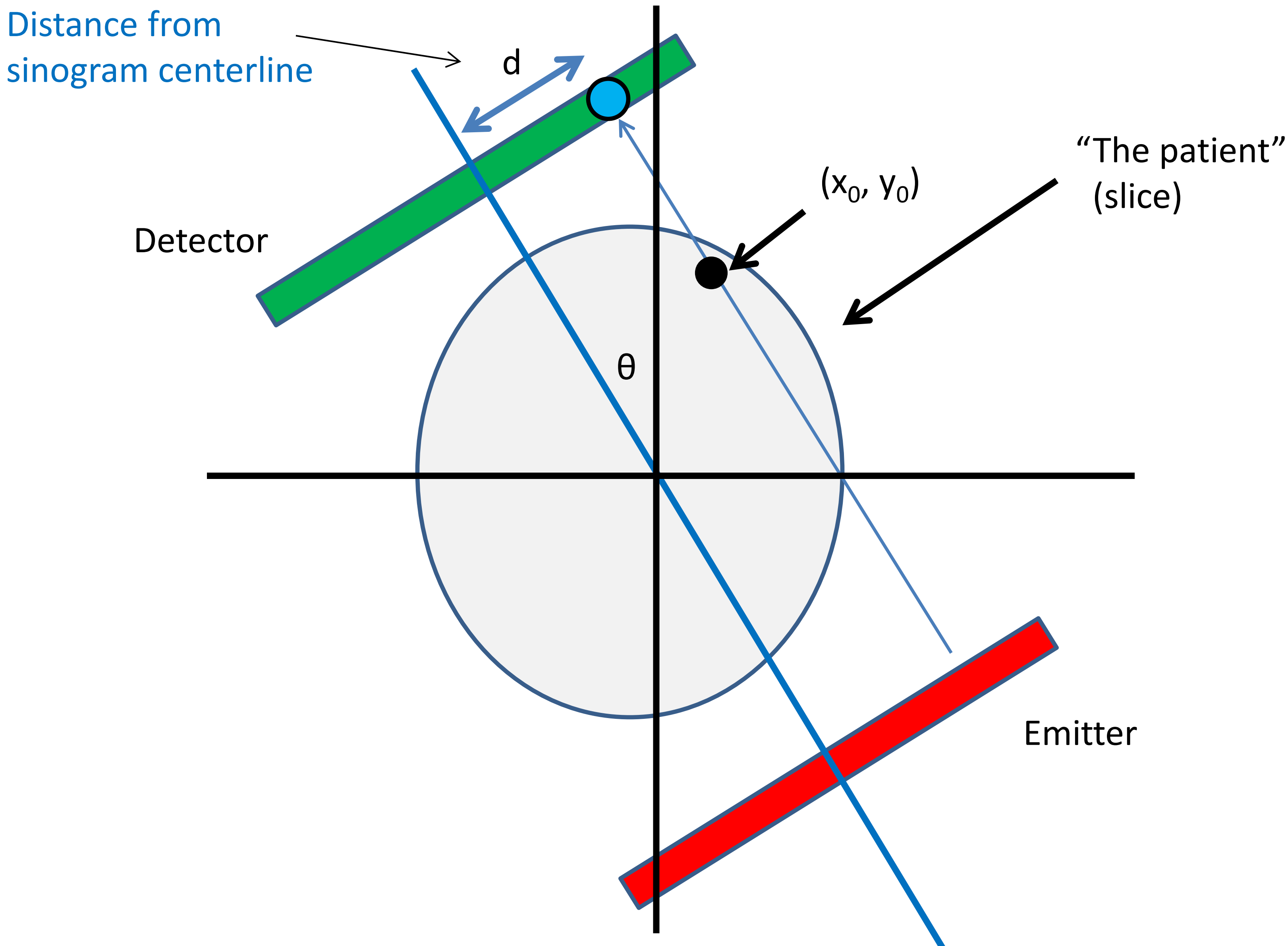
Geometry Details



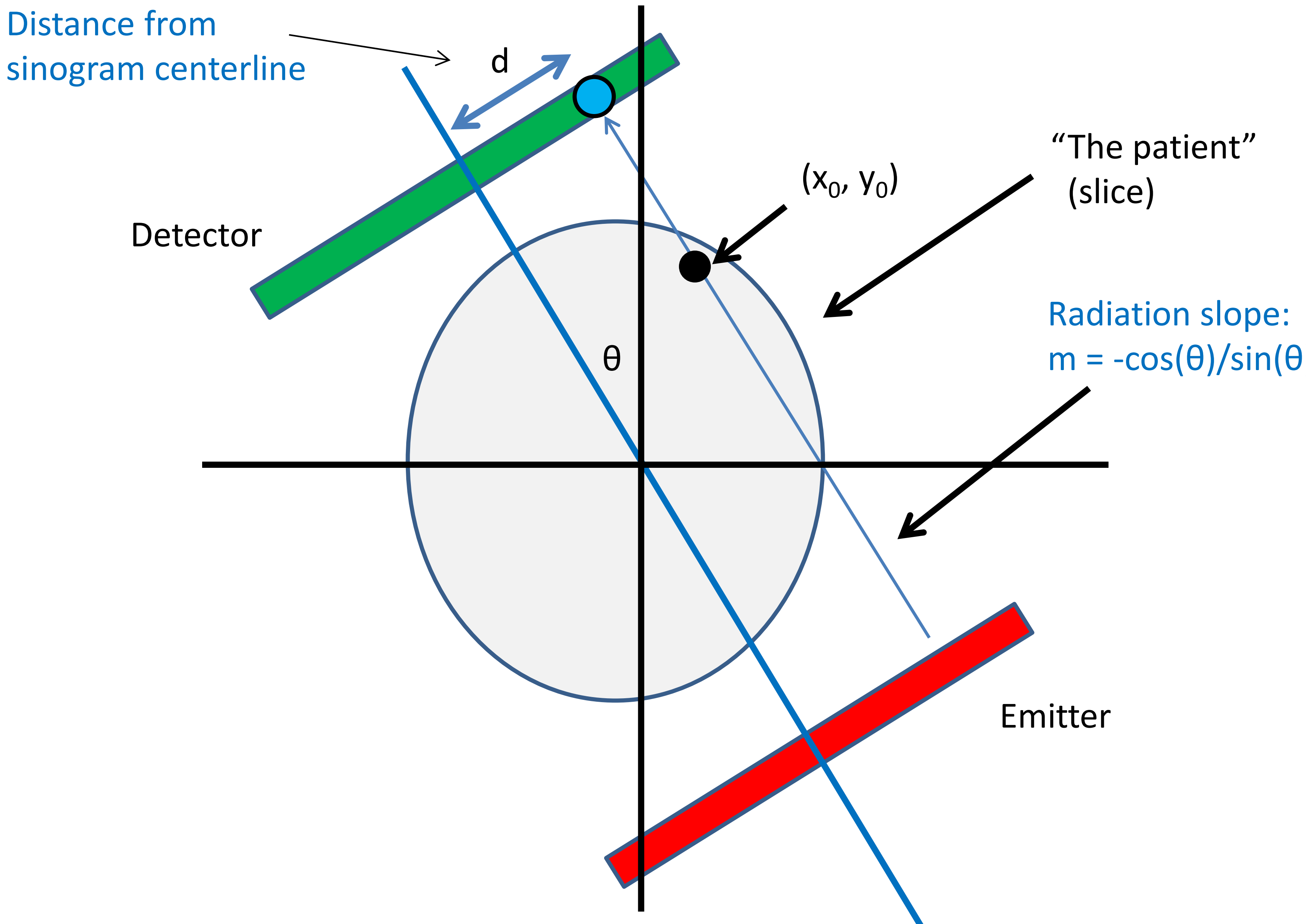
Geometry Details



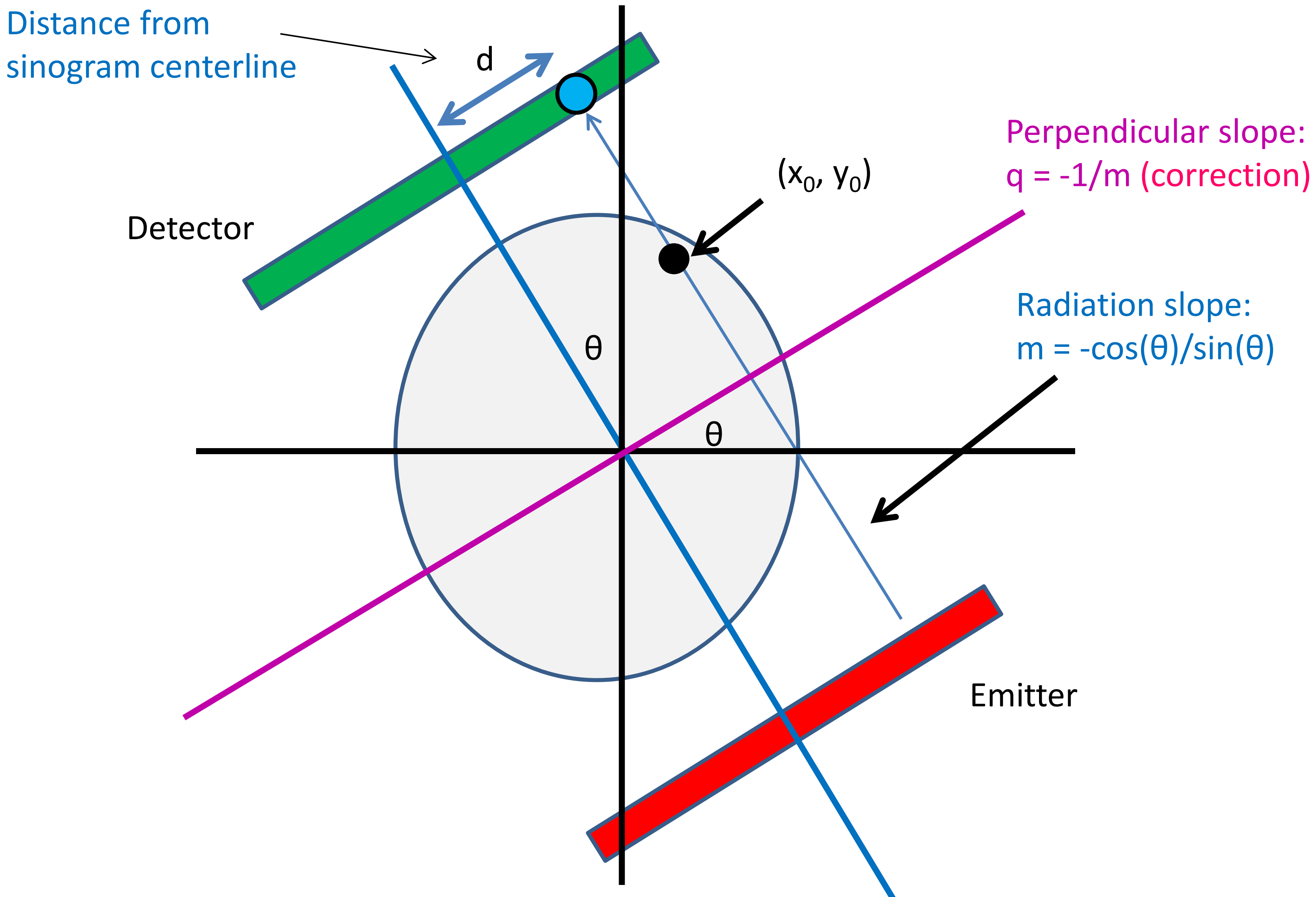
Geometry Details



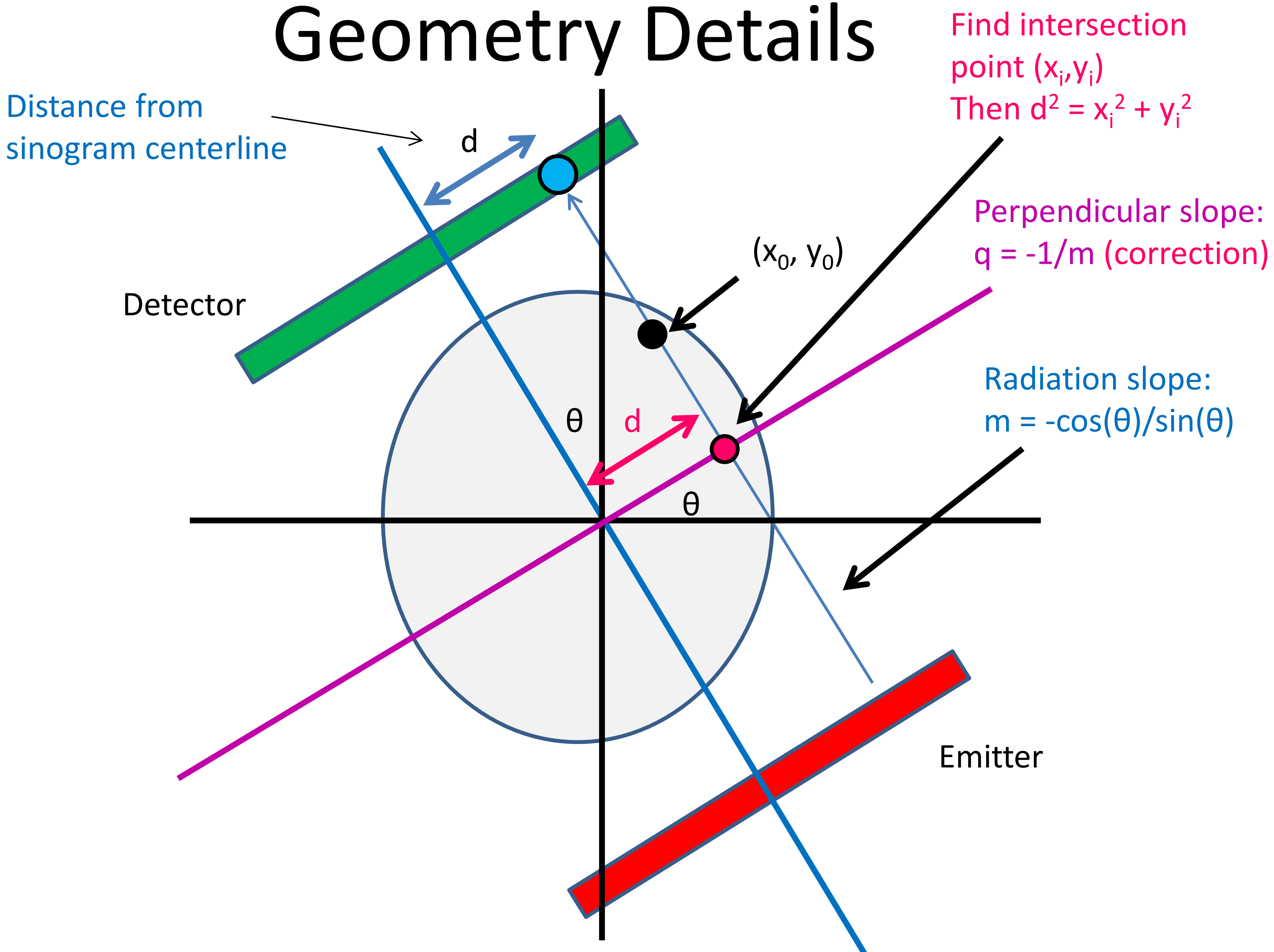
Geometry Details



Geometry Details



Geometry Details



Intersection point

- Line 1: (point-slope)

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

- Line 2:

$$y_i = qx_i$$

Corrections

- Combine and solve:

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

Intersection point

- Intersection point:

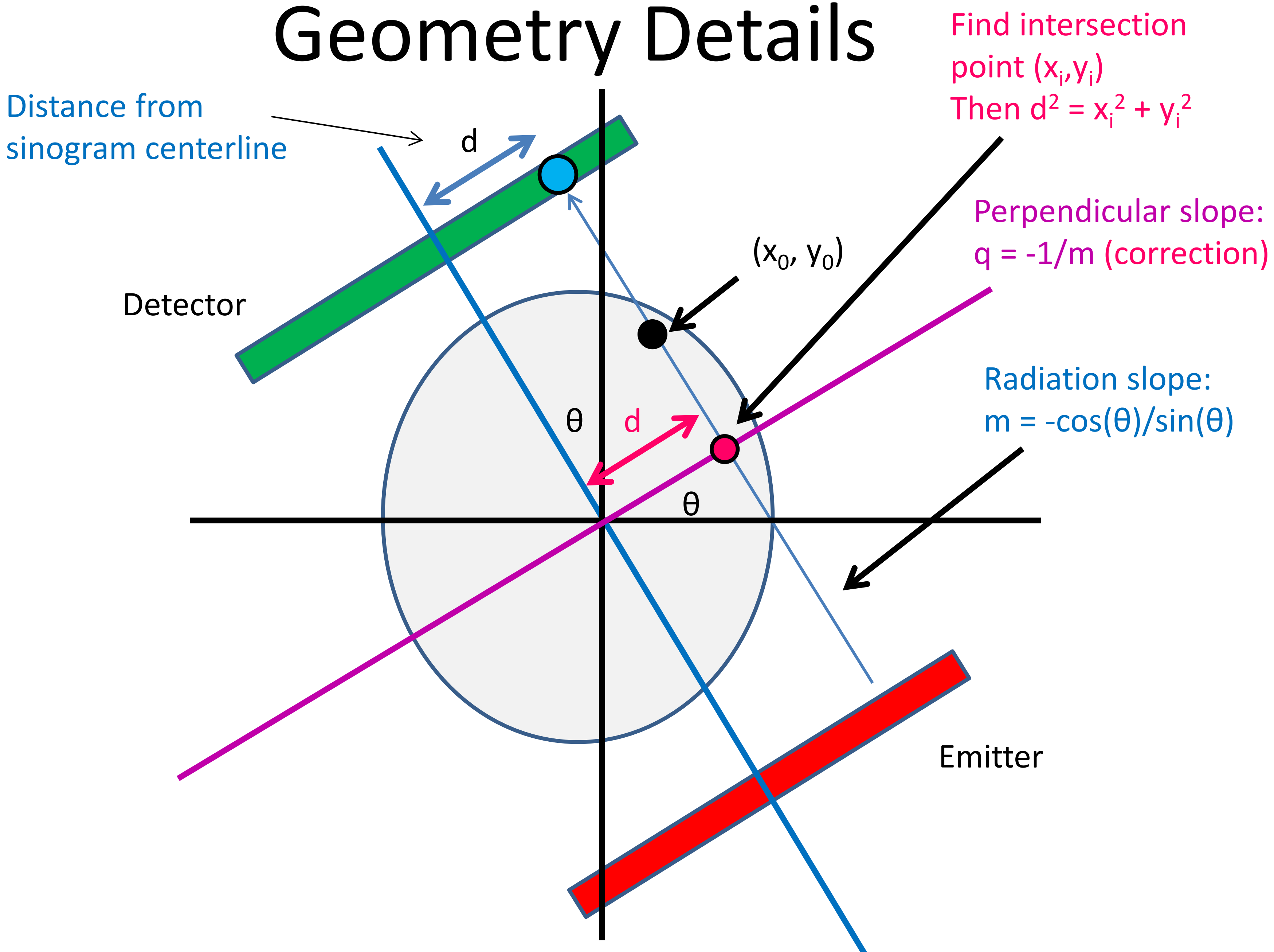
$$x_i = \frac{y_0 - m x_0}{q - m}, \qquad y_i = q x_i$$

Corrections

- Distance from measurement centerline:

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

Geometry Details



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 m, -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

Correction/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)
```

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

Parallelizable!
Inside loop depends only on x, y, 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 m, -1/m
                        calculate d from x_i, y_i
                        image[x,y] += sinogram[theta, "distance"]
```

(corrections/clarification – see slide 37)

Sequential pseudocode

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

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

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

(provides lots of parallelization already, other issues)

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

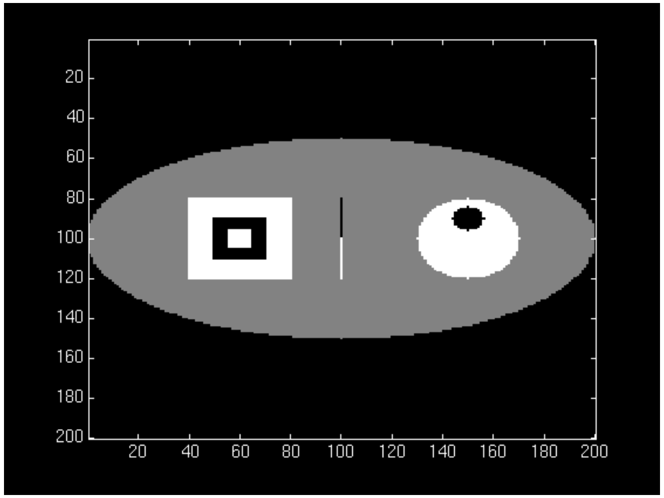
(corrections/clarification – see slide 37)

Cautionary notes

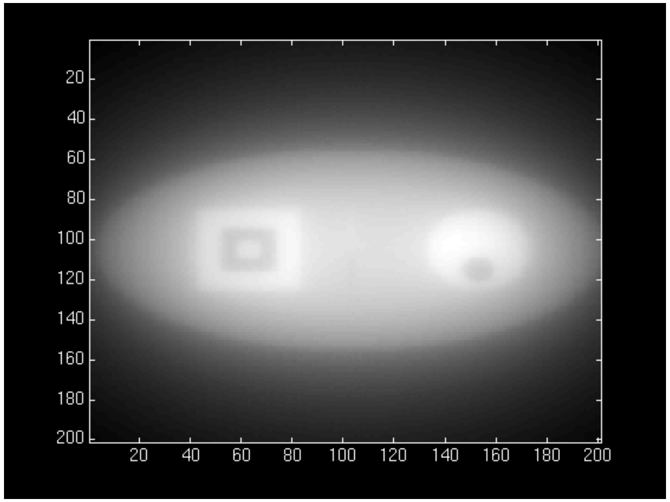
- *y* in an image is opposite of *y* geometrically!
  - (Graphics/computing convention)
- Edge cases (divide-by-0):
  - $\theta = 0$:
    - $d = x_0$
  - $\theta = \pi/2$:
    - $d = y_0$

Almost a good reconstruction!

Original

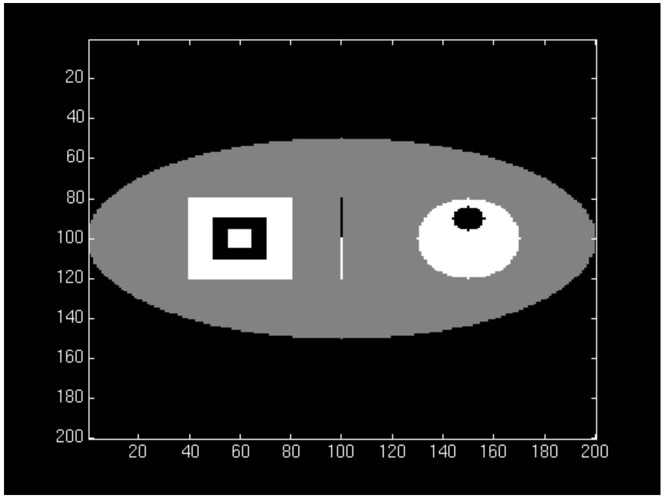


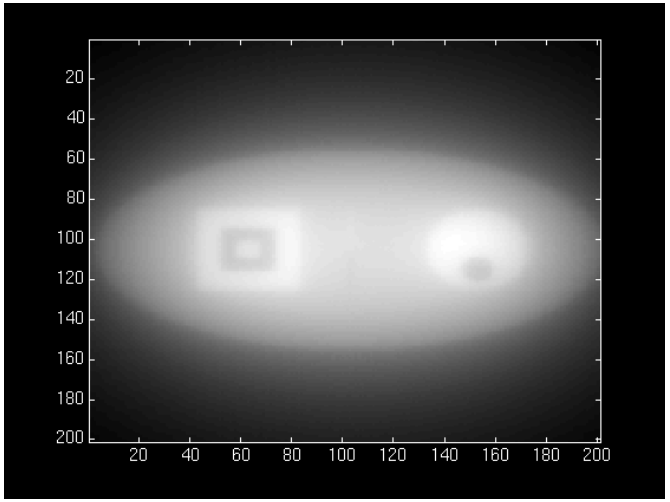
Reconstruction



Almost a good reconstruction!

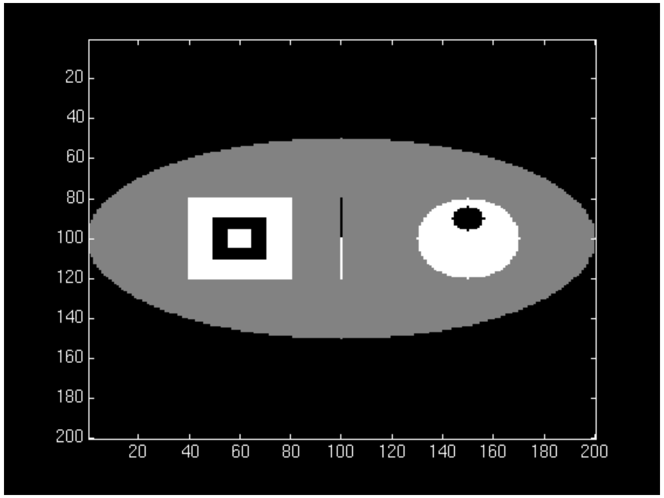
- "Backprojection blur"
  - Similar to low-pass property of SMA (Week 1)
  - We need an "anti-blur"! (opposite of Homework 1)

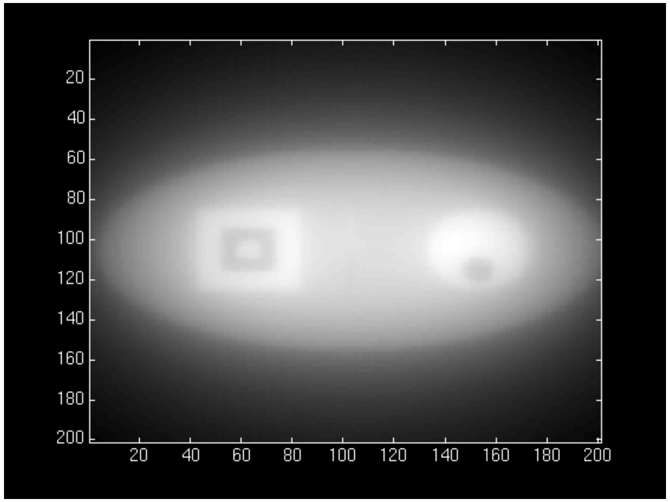




Almost a good reconstruction!

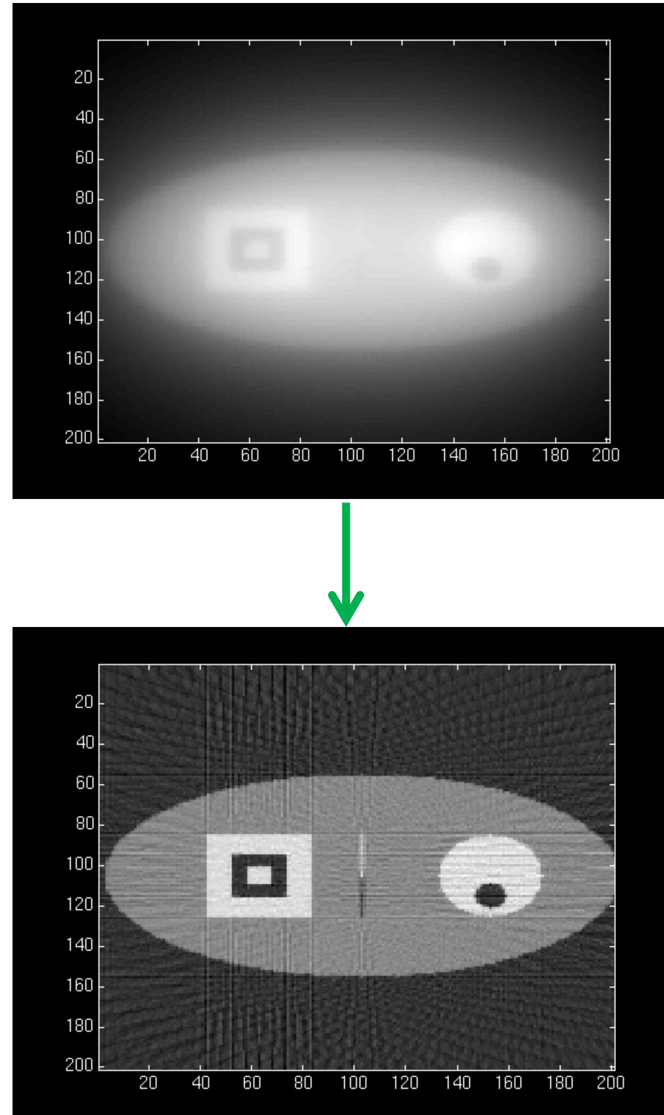
- Solution:
  - A "high-pass filter"
  - We can get frequency info in parallelizable manner!
    - (FFT, Week 3)





Almost a good reconstruction!

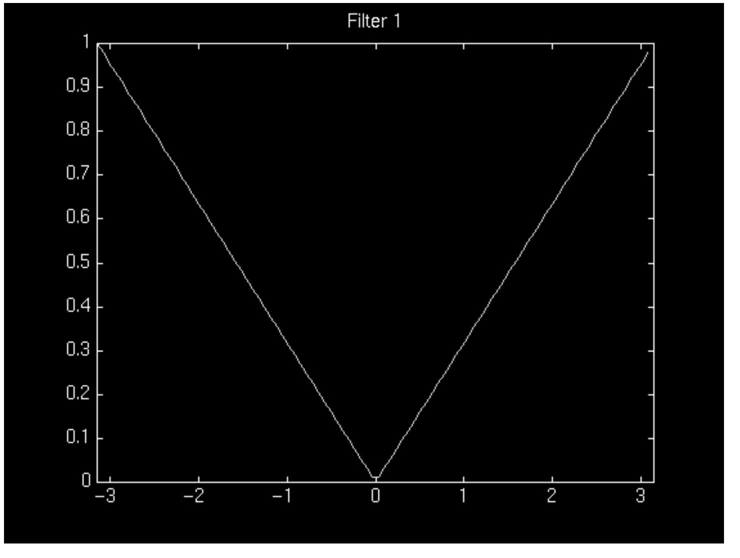
- Solution:
  - A “high-pass filter”
  - We can get frequency info in parallelizable manner!
    - (FFT, Week 3)



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



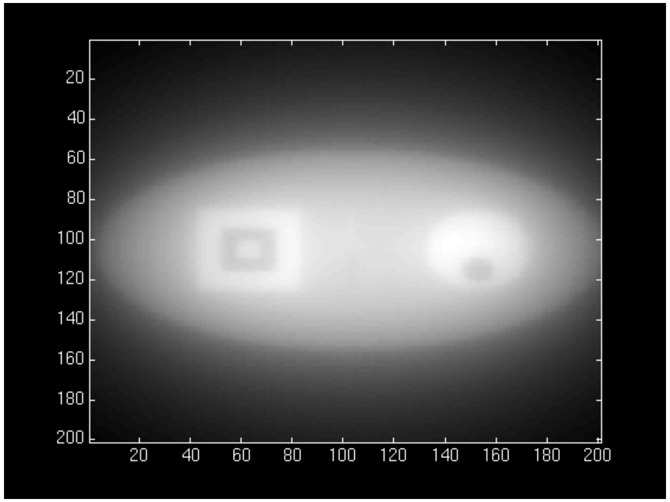
Almost a good reconstruction!

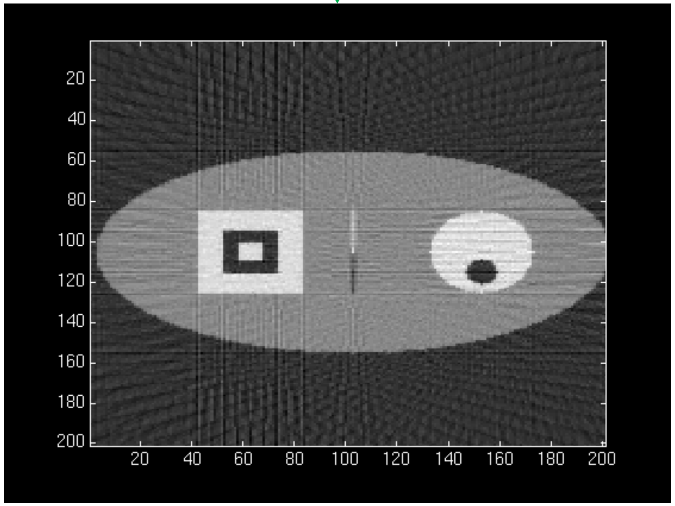
- CPU-side:

```
(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:

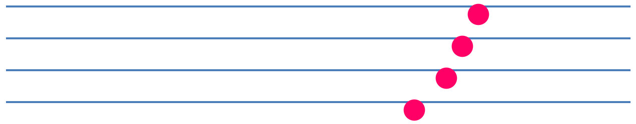
```
filterKernel:
        Select specific freq-amplitude
        based on thread ID

        Get new amplitude from
        ramp equation
```



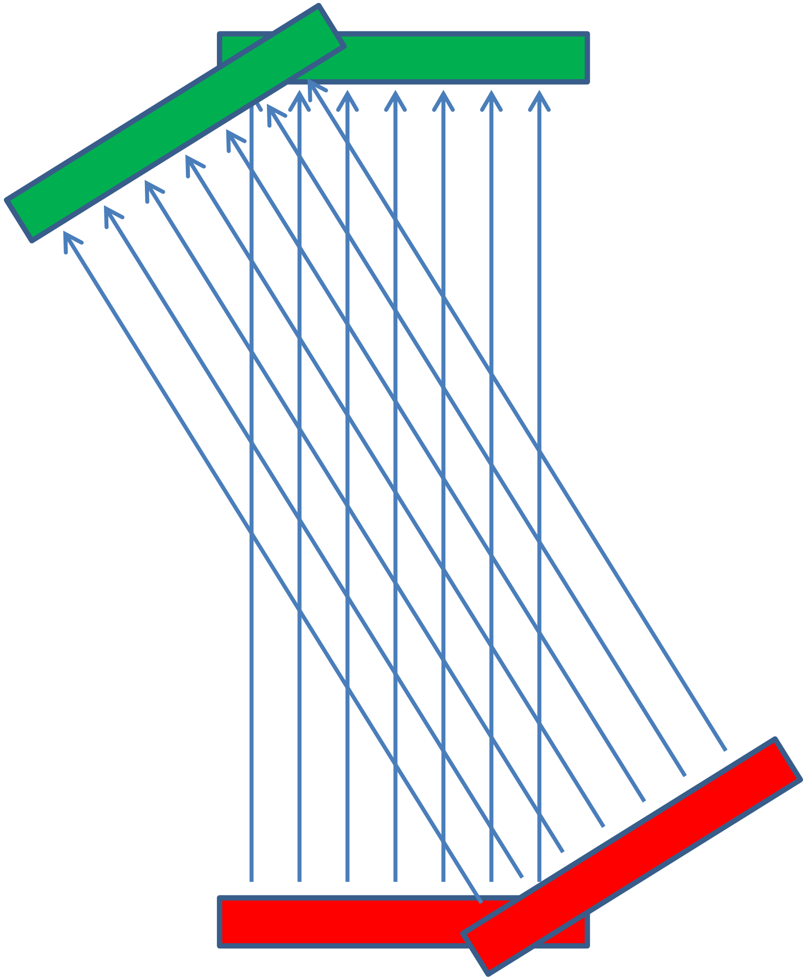


GPU Hardware

- Non-coalesced access!
  - Sinogram 0, index ~d_0
  - Sinogram 1, index ~d_1
  - Sinogram 2, index ~d_2
  - …



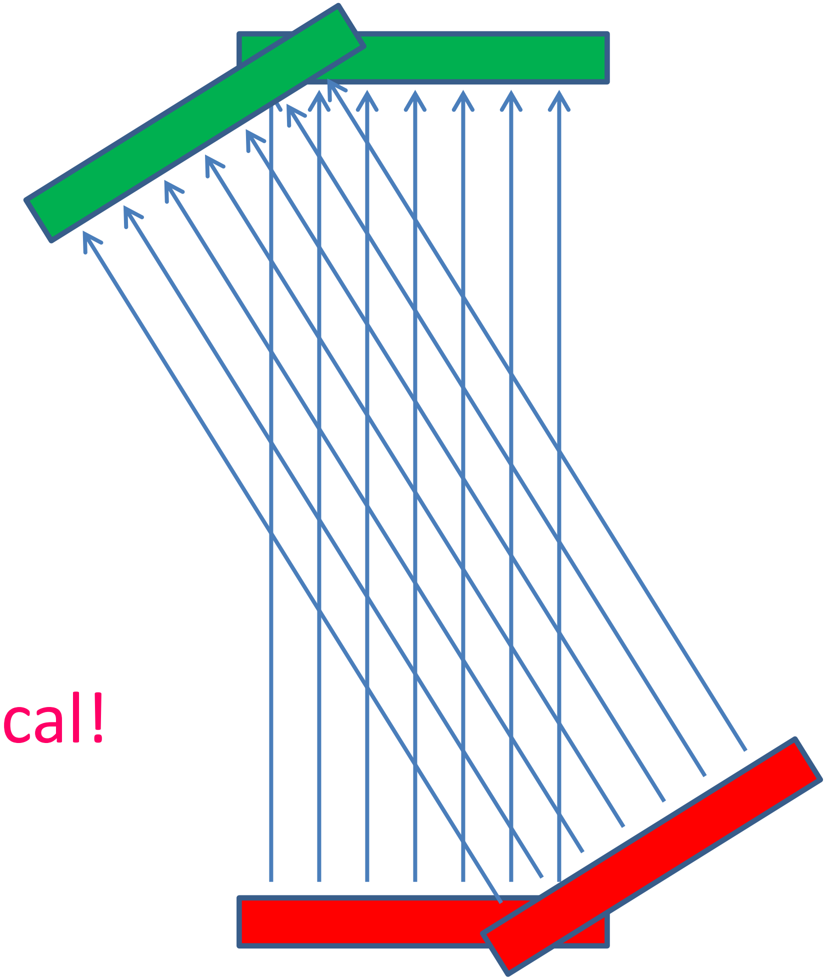
…



GPU Hardware

- Non-coalesced access!
  - Sinogram 0, index ~d_0
  - Sinogram 1, index ~d_1
  - Sinogram 2, index ~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!

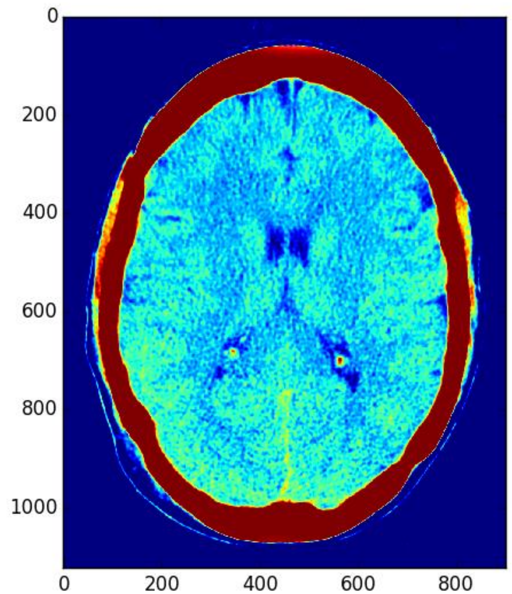
…

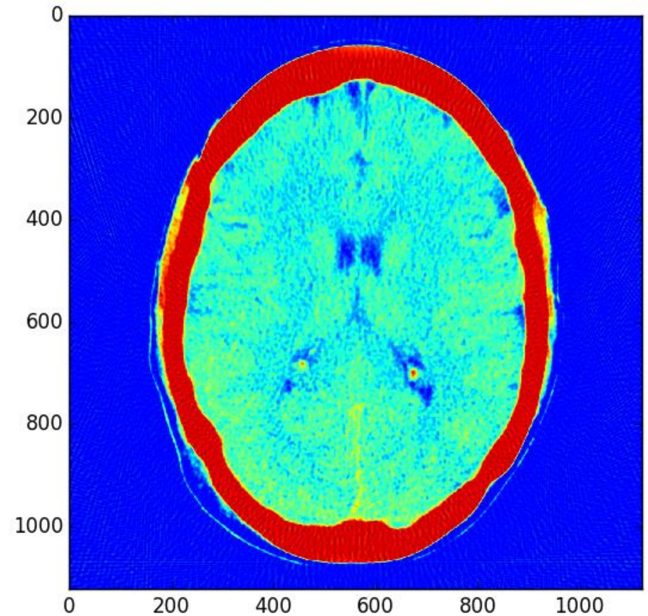
Summary/pseudocode

```
(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!