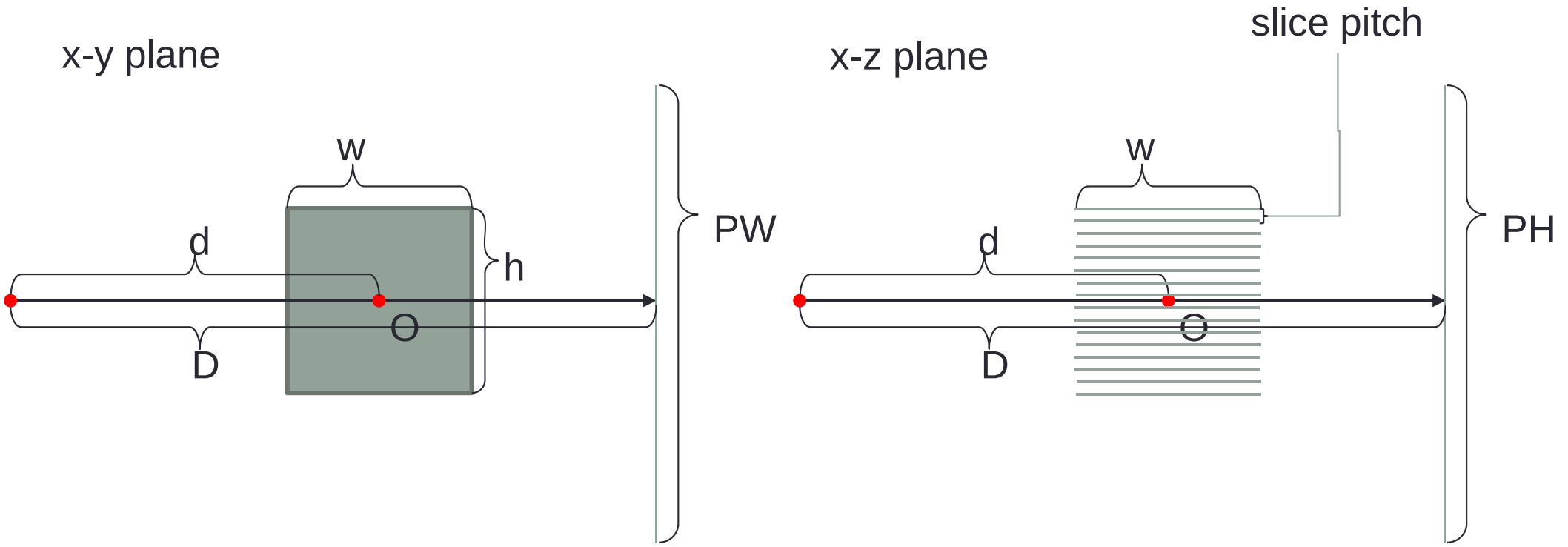
FAST GPU-BASED 3D RECONSTRUCTION

Byeonghun Lee
intellee@cglab.snu.ac.kr

Set geometry parameters

- Distance from source to detector : D
- Distance from source to object : d
- Detector pixel size : DET_X, DET_Y
- Detector pixel pitch : DET_PITCH_X, DET_PITCH_Y
- Detector offset : DET_OFFSET_X, DET_OFFSET_Y
- Cube pixel size : CUBE_X, CUBE_Y, CUBE_Z
- Cube pixel pitch : CUBE_PITCH_X, …
- Cube offset : CUBE_OFFSET_X, …
- Projection count : PROJ_COUNT
- Scan angle : SCAN_ANGLE
- Start angle : START_ANGLE
- Rotate direction

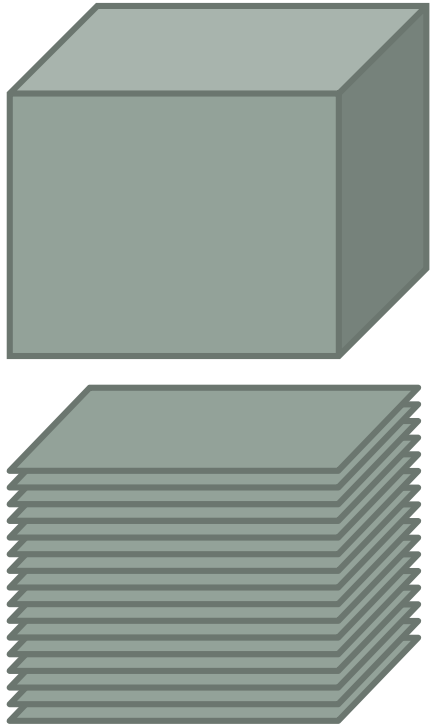
Geometry



w : cube width = (CUBE_X*CUBE_PIXEL_PITCH_X)
h : cube height = (CUBE_Y*CUBE_PIXEL_PITCH_Y)
PW : projection width = (DET_X*DET_PITCH_X)
PH : projection height = (DET_Y*DET_PITCH_Y)
slice pitch = CUBE_PITCH_Z

Initialize

- Allocate the memory space for volume and projection images
  - Volume
    - 3D texture
      - ID3DTexture3D *pVolumeTexture;
      - CreateTexture3D()
    - 2D texture slice stack
      - ID3DTexture2D *pSliceTexture[CUBE_Z];
      - CreateTexture2D()
  - Projection
    - 2D texture size of (DET_X, DET_Y)



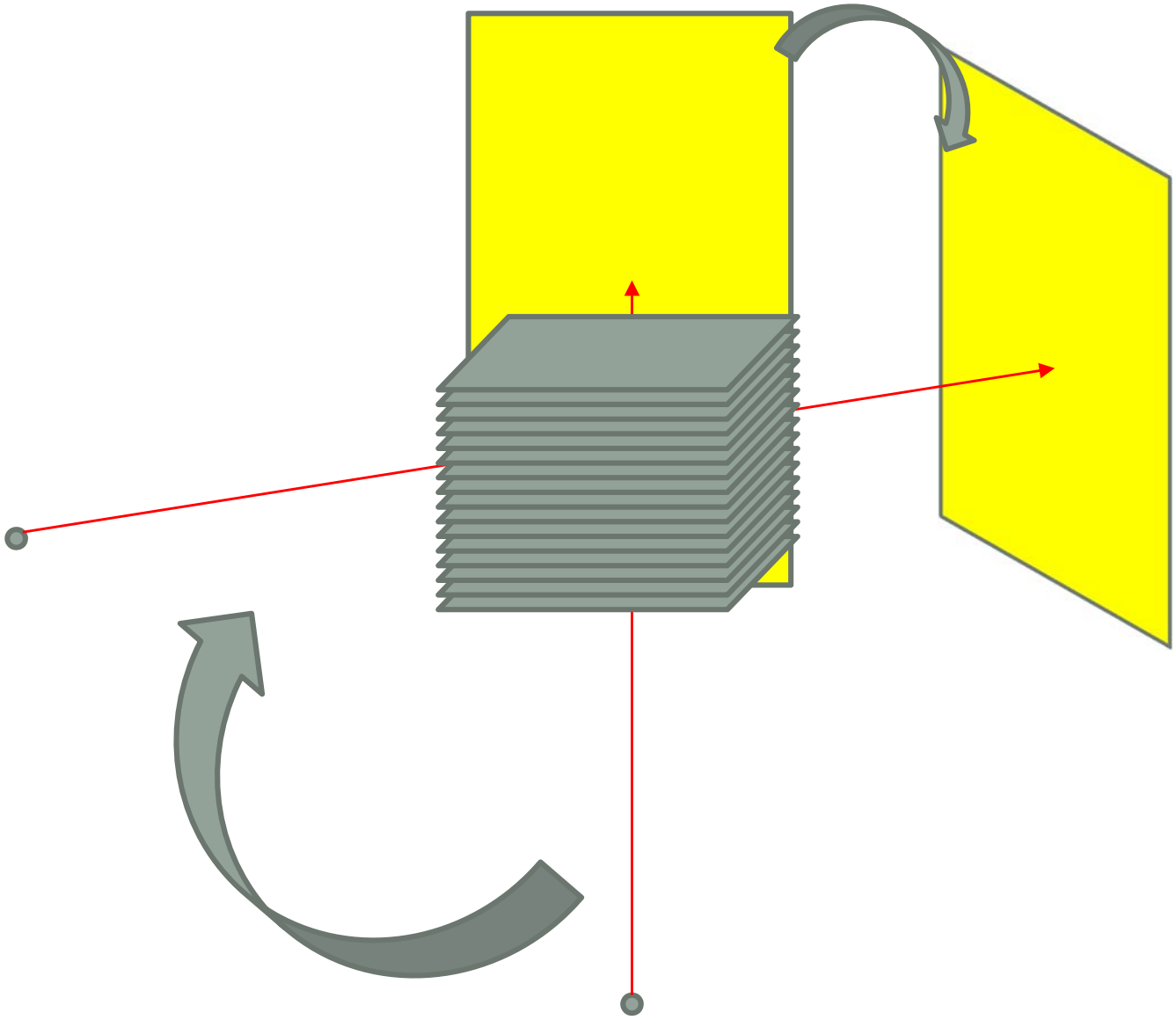
Initialize

- Two viewports
  - Filtering: detector pixel size
  - back-projection: cube pixel size(CUBE_X, CUBE_Y)
- Update geometry constant buffer
  - Invariant values during reconstruction such as size, pitch, …
- Create vertex buffer
  - Position: (-1, 1), (1, 1), (-1, -1), (1, -1)
  - Texcoord: (-w/2, -h/2), (w/2, -h/2), (-w/2, h/2), (w/2, h/2)
    - w : cube width = (CUBE_X*CUBE_PIXEL_PITCH_X)
    - h : cube height = (CUBE_Y*CUBE_PIXEL_PITCH_Y)

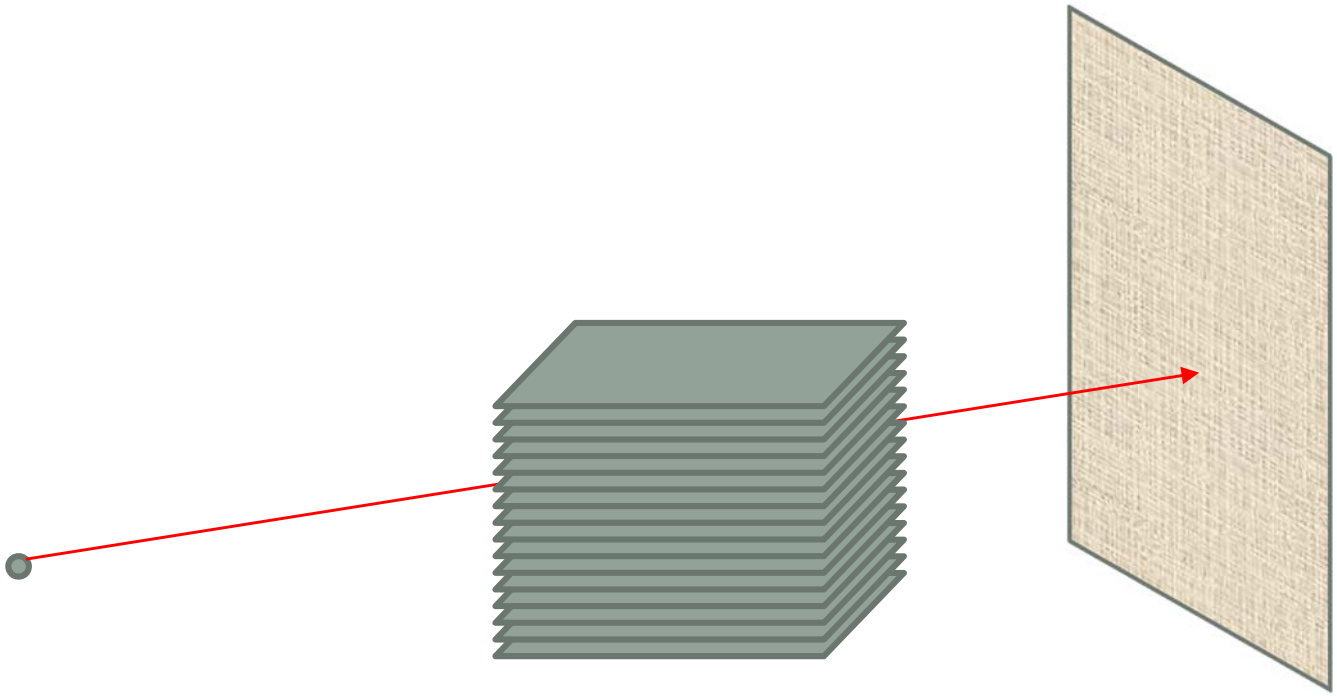
Reconstruction

- For all projection images
  - Update the current angle
    - Use a constant buffer
  - Upload the projection image for the current angle
    - Use Map()
  - Apply filtering (optional)
  - For all volume slices
    - Update the current slice position of z-axis
      - Use a constant buffer or create dedicate vertex buffers for each slice
    - Back-projection
      - Set render target for the current volume slice
      - Draw()

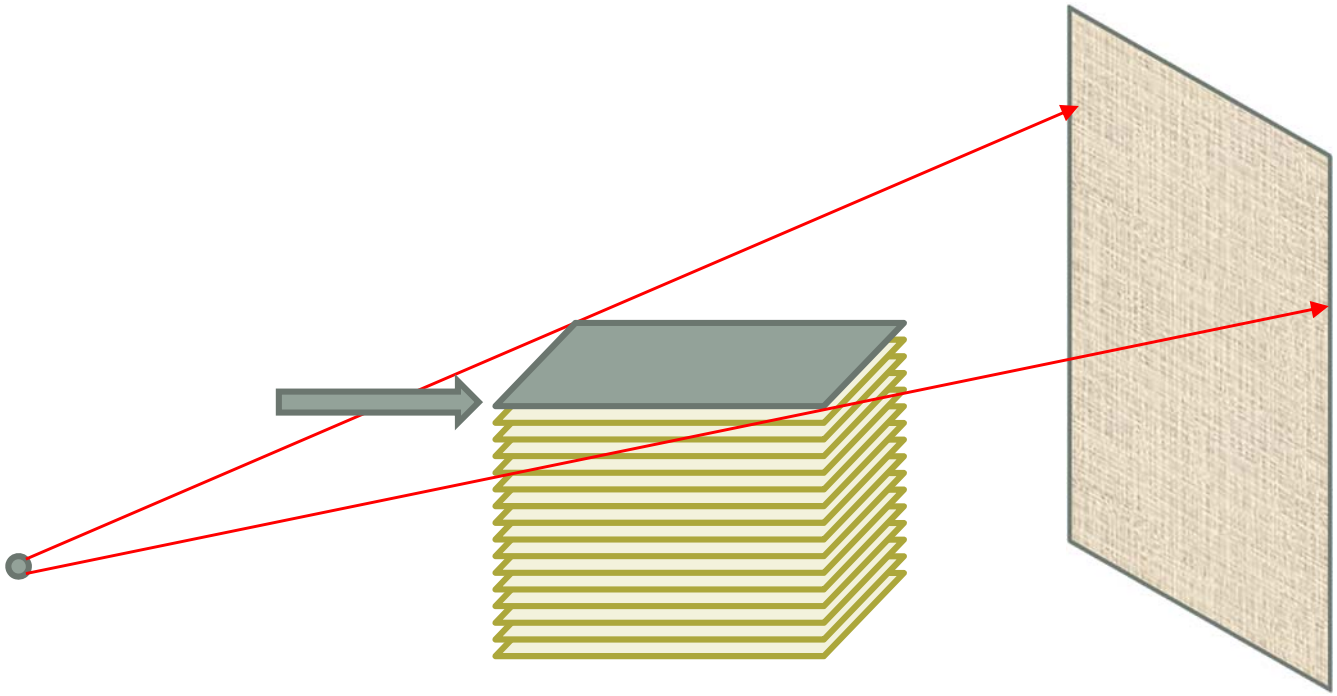
Update current angle



Upload the projection image

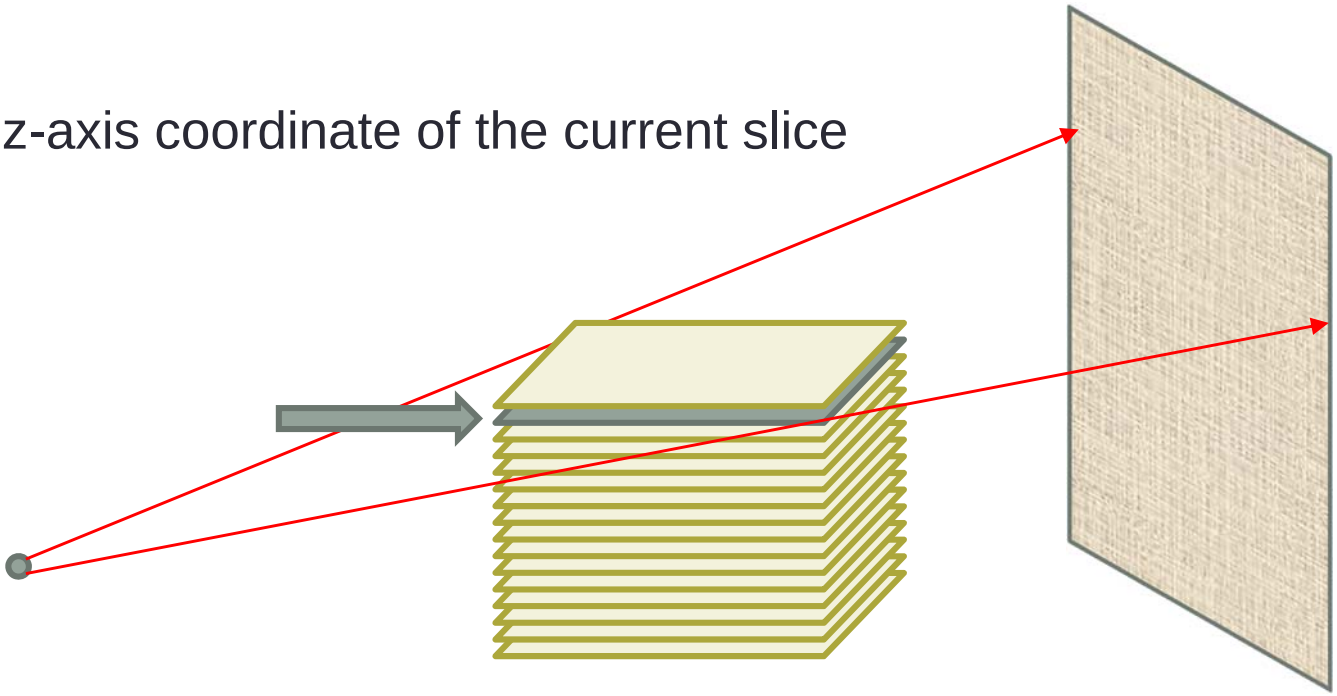


For all volume slices



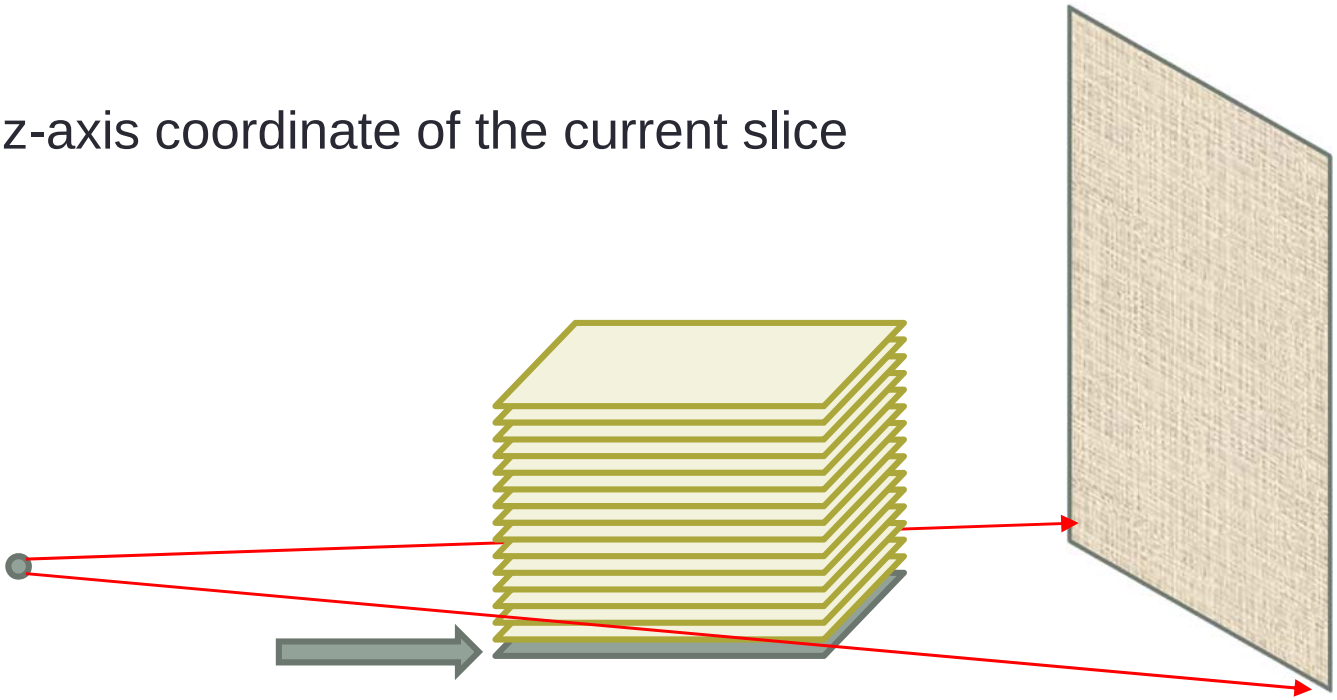
For all volume slices

Update z-axis coordinate of the current slice



For all volume slices

Update z-axis coordinate of the current slice



Reconstruction

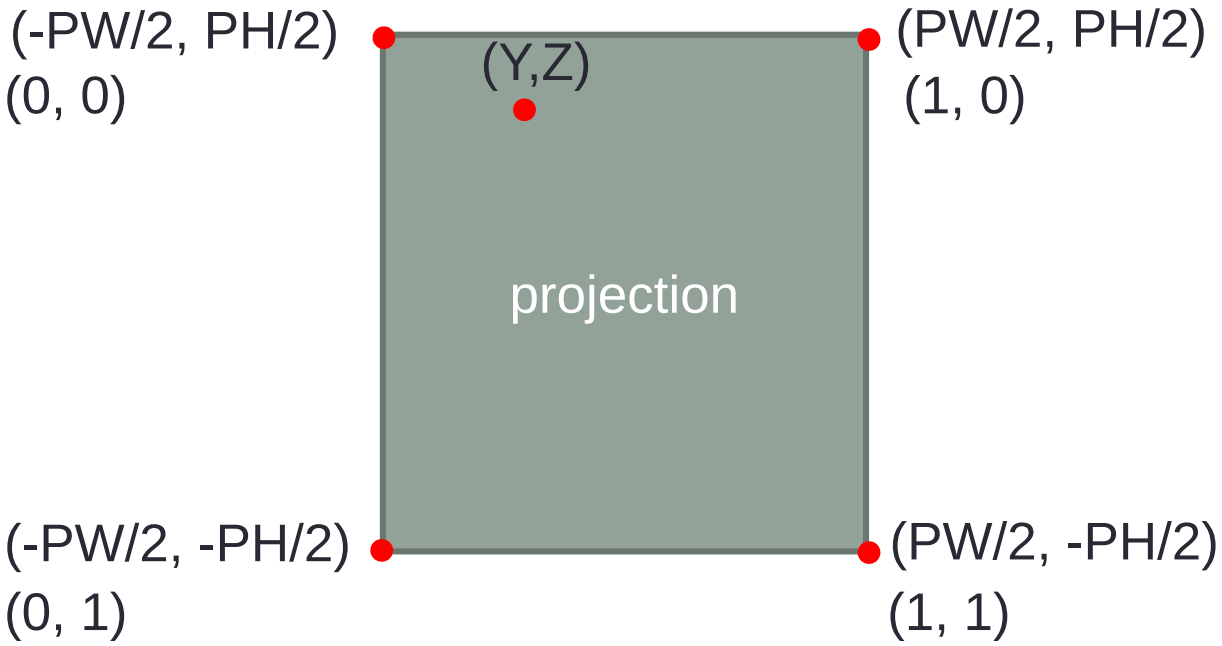
- Read-back from GPU to CPU
  - Create temporary 2D textures for read-back
  - For all volume slices
    - Convert FLOAT32 to SNORM16 using shaders
    - Use CopyResource() and Map()

Basic implementation

- Back-projection
  - CPU
    - Set blend state for BLEND_ADD
    - Calculate the rotation matrix of the current angle
    - Update it via constant buffer
  - GPU
    - Rotate texcoord using a matrix in vertex shader
    - Perform back-projection in pixel shader
      - Convert coordinate
        - mm space <-> pixel space <-> texcoord space

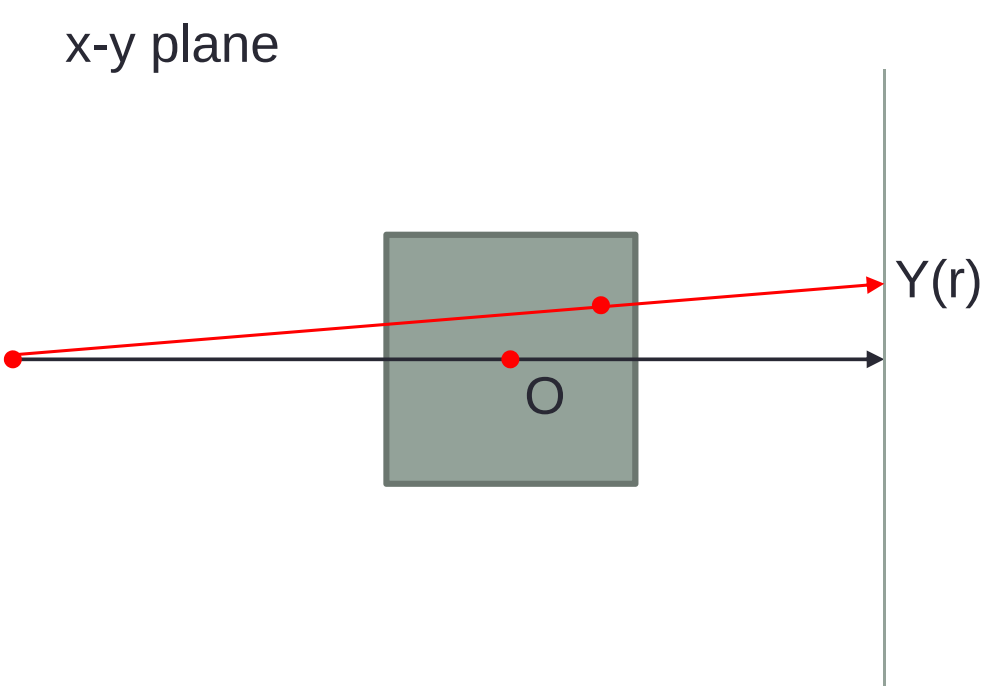
Basic implementation

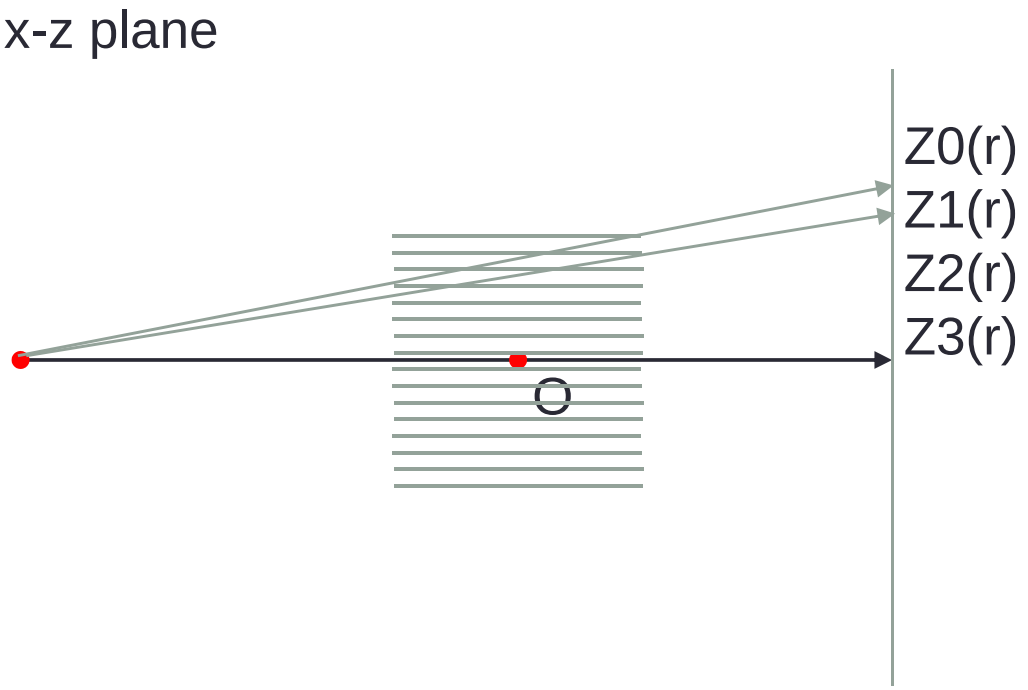
- Convert coordinate
  - Let
    - PW: projection width = (DET_X*DET_PITCH_X)
    - PH: projection height = (DET_X*DET_PITCH_X)
    - (Y, Z): sampled position



Optimization 1

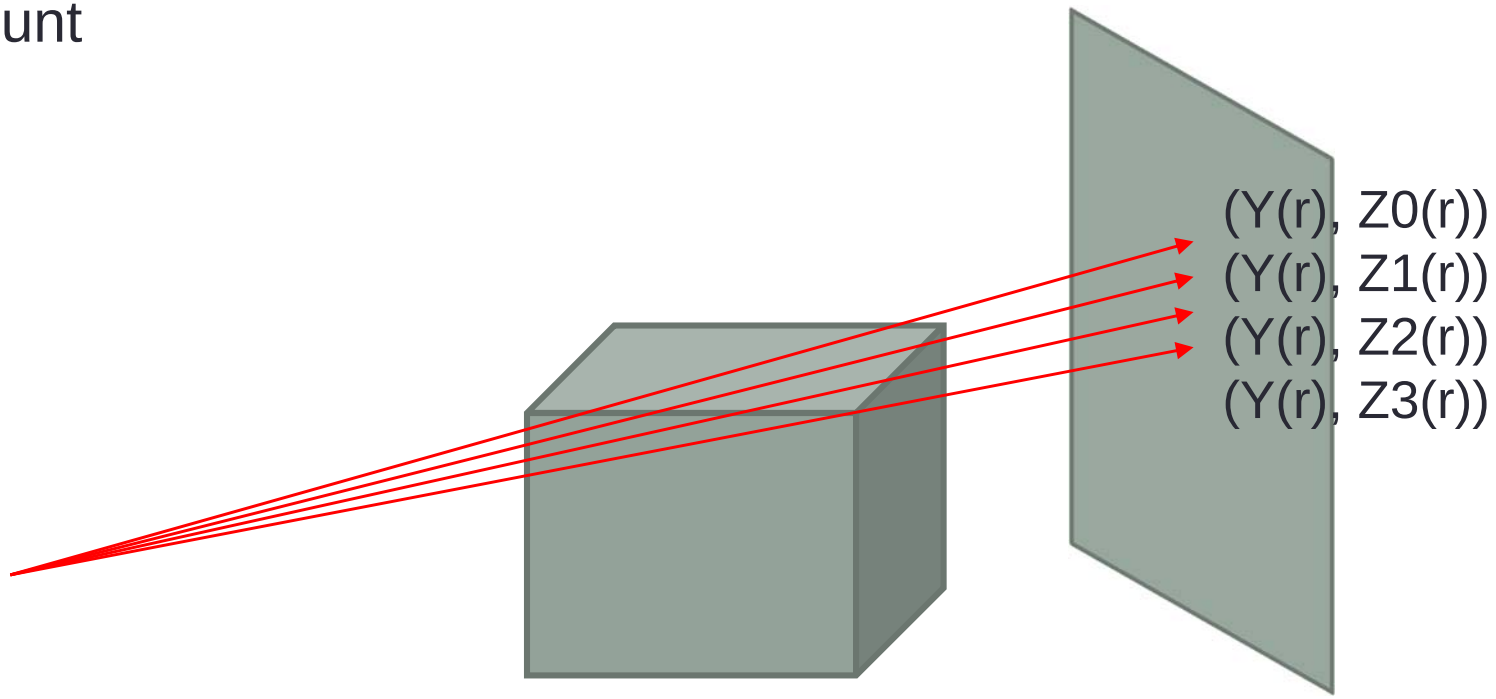
- Use position coherency





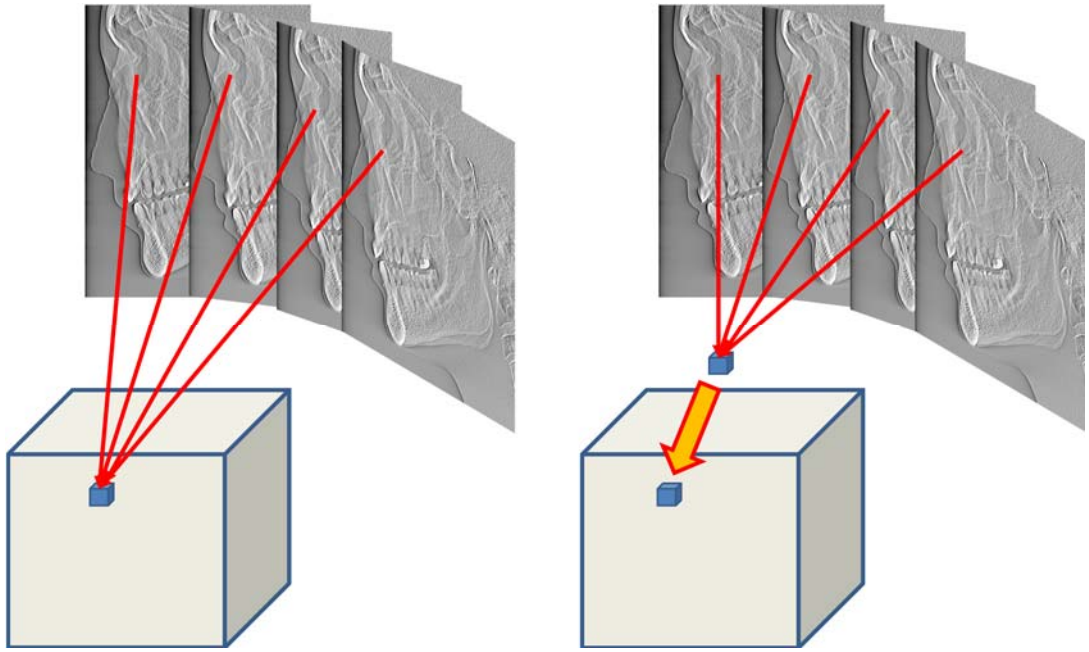
Optimization 1

- Use position coherency
  - Create volume slices with R32G32B32A32
  - Back-project 4 voxels per pixel shader
  - This reduces the operation count as well as global memory access count



Optimization 2

- Use memory hierarchy
  - Register is much faster than global memory
  - Reduce writing to global memory using multiple projections



Term projection

- Implement FDK using D3D11
  - Parameter file
    - Geometry information
    - You should parse from a file
  - Projection
    - Format : SNORM16(filtered), UINT16(unfiltered)
    - 1000x1000, 400 slices
    - File name: 0001.raw, 0002.raw, …
    - Z:\GPU특강 (Samba)
  - Volume
    - Format : FLOAT32(intermediate), SNORM16(final)
    - 512x512x512
    - Save as slices: 0001.slice, 0002.slice, …

File load and save

- File location is provided by parameter files
- You should do everything via parameter files
  - Geometry information and file information
- Hard coding is not allowed!!!

Elapsed time check

- Check time below
  - Initialization
  - Reconstruction
  - Read-back
- Use QueryPerformance
  - If you don’t know, do Googling!!

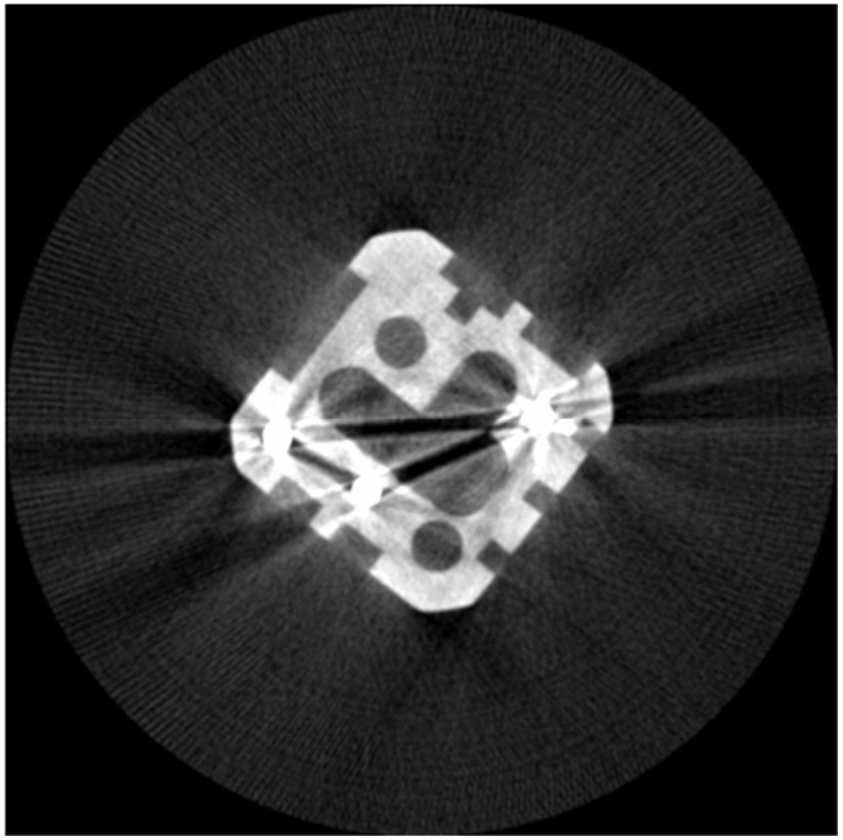
Evaluation

- Score
  - Basic implementation : 50%
  - Optimization 1 : 25%
  - Optimization 2 : 25%
  - If you apply additional optimizations, you will get extra points
  - 10% deduction per memory leakage
- Submit
  - Report
    - Your development environment
      - GPU spec.
    - Performance results
      - Basic implementation
      - Optimization 1
      - Optimization 1&2
      - (optional) additional optimizations
  - Source code
    - 3 options: basic and optimizations
- No delay allowed!!
  - Submit your result as much as you've done

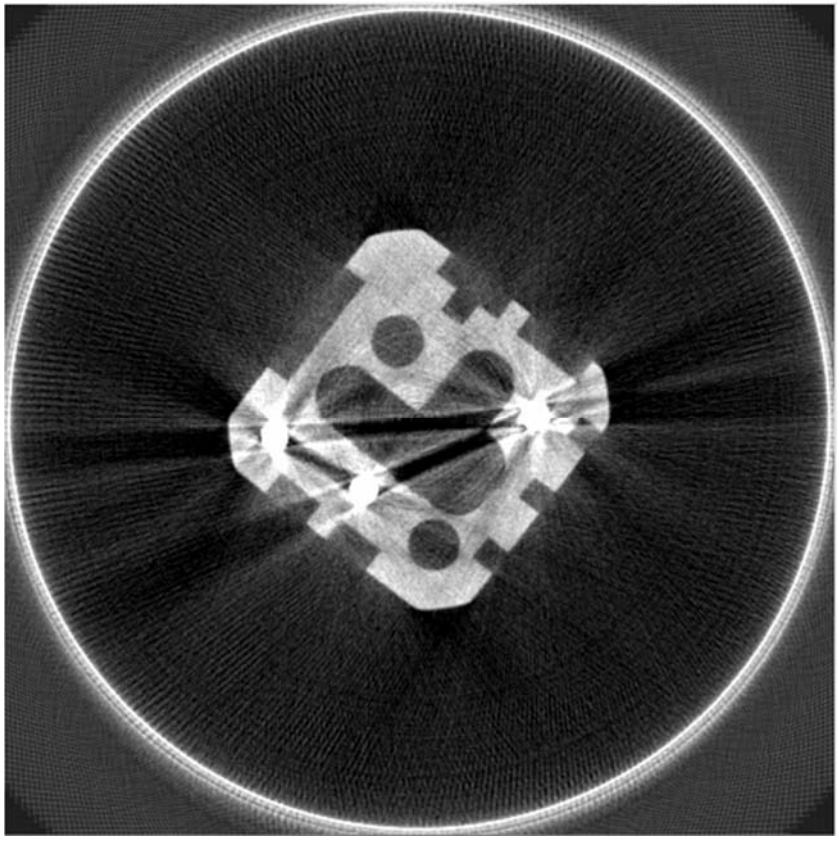
Parameter file example

- 84.5494 // source to object
- 498 // source to detector
- 0 // start angle
- 360 // scan angle
- 1 // rotate direction
- 400 // projection count
- 1000 // detector width (pixel)
- 1000 // detector height (pixel)
- -6 // detector x-offset (pixel)
- 0 // detector y-offset (pixel)
- 0.122 // detector x-pitch (mm)
- 0.122 // detector y-pitch (mm)
- 512 // object width (pixel)
- 512 // object height (pixel)
- 512 // object depth (pixel)
- 0.040155 // object x-pitch (mm)
- 0.040155 // object y-pitch (mm)
- 0.035536 // object z-pitch (mm)
- 0 // object x-offset (pixel)
- 0 // object y-offset (pixel)
- 0 // object z-offset (pixel)
- short // projection format
- short // object format
- null // open
- null // dark
- D:\projection // projection file location
- D:\slice // object file location

Example



CPU



GPU

Thank you