MIC-GPU: High-Performance Computing for Medical Imaging on Programmable Graphics Hardware (GPUs)

Parallelism in Medical Imaging

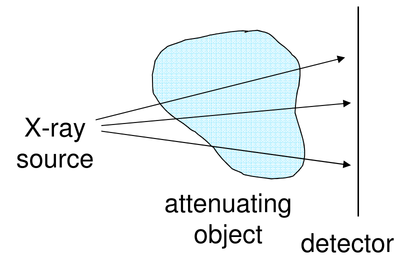
Klaus Mueller, Ziyi Zheng, Eric Papenhausen

Stony Brook University

Computer Science

Stony Brook, NY

Transmission CT: Data Generation



CT Reconstruction

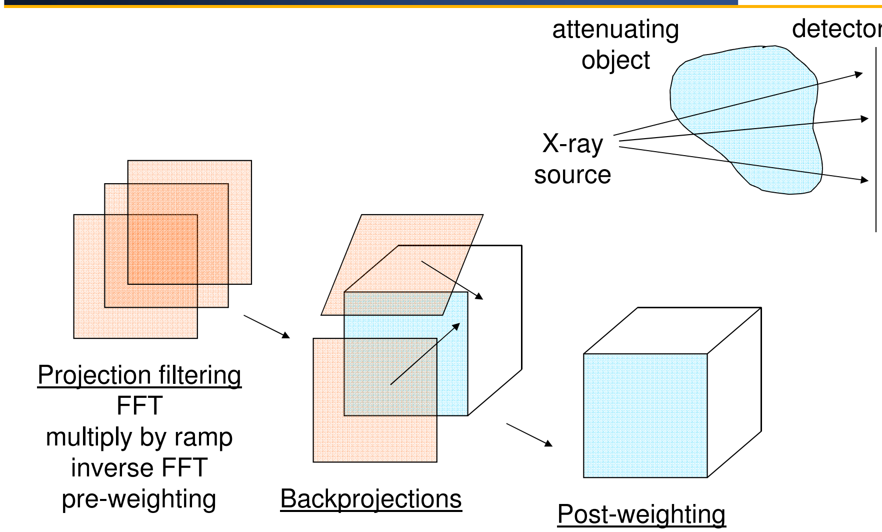
High-dose CT reconstruction usually uses FDK algorithm

- backprojection of filtered views

Low-dose CT reconstruction pipeline typically uses iterative 3D reconstruction with regularization

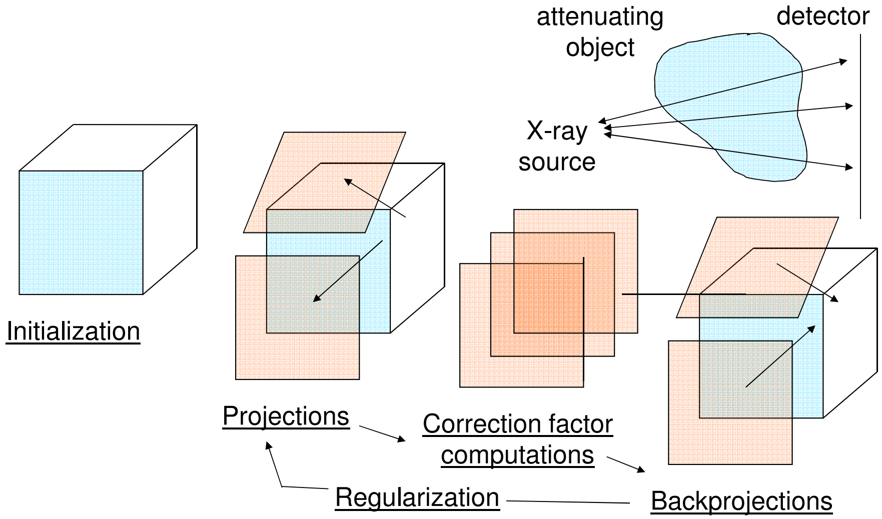
- projection of volume into set's views
- correction factor computation
- backprojection of correction factors (views)
- regularization

Filtered Backprojection Reconstruction



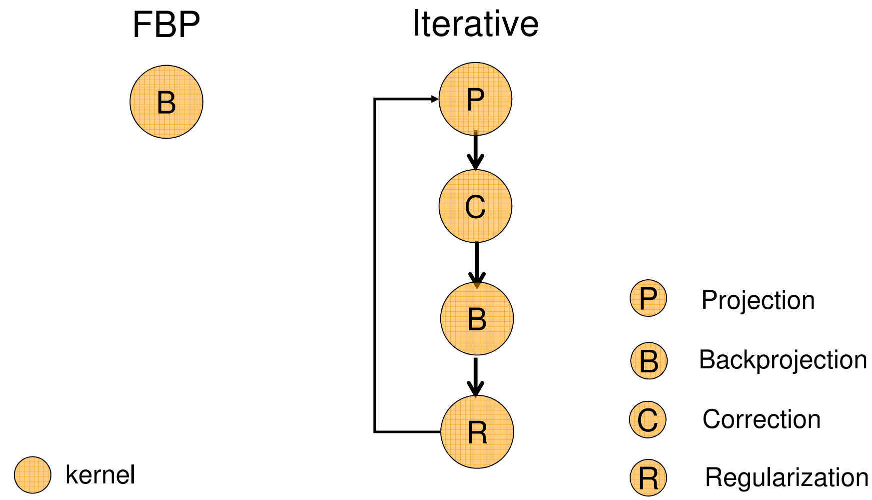
Iterative Reconstruction

SPIE Medical Imaging



Kernel-Centric Decomposition

SPIE Medical Imaging



Kernel-Centric Decomposition

SPIE Medical Imaging

We can consider each of these steps to be a SIMT kernel

Iterative 3D reconstruction with regularization:

- backprojection of volume into set's views → *projection kernel*
- correction factor computation → *correction factor kernel*
- backprojection of correction factors → *backprojection kernel*
- regularization → *regularization kernel*

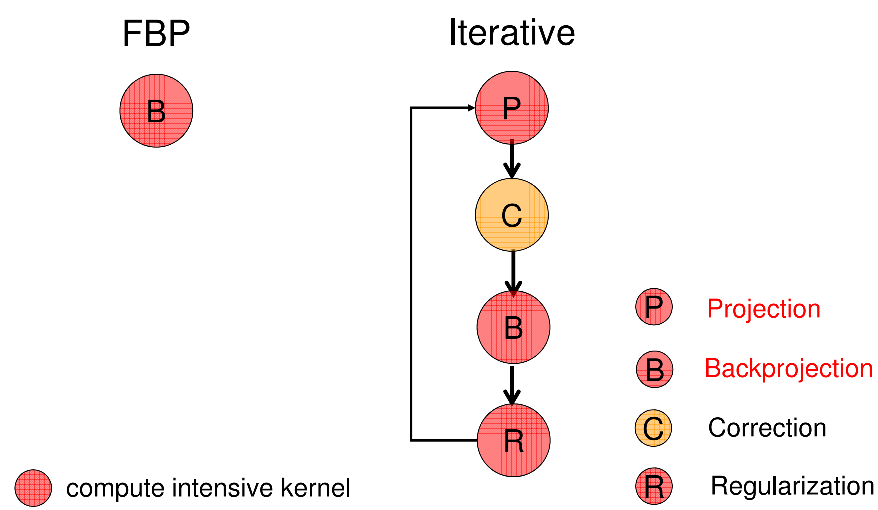
projector with interpolation

vector operations

image processing filters

Kernel-Centric Decomposition

SPIE Medical Imaging



Kernel Scheduling

SPIE Medical Imaging

SIMT can only execute one kernel at a time

- this prohibits kernel overlap, even if mathematically correct
- we may merge kernels if targets are identical
  → this favors load balancing and the reduction of passes

First decompose the reconstruction pipeline into components

- develop an optimized kernel for each component
- overlap (=hide) the loading of data (if needed) with execution of a prior kernel (or within kernel)
- optimize what platform to run the computations (CPU, GPU), but then consider transfer of data

Terminology

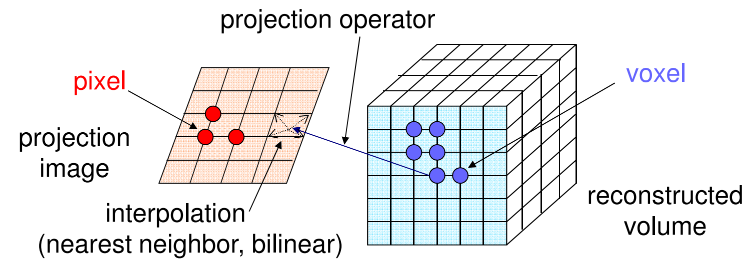
SPIE Medical Imaging

We shall discuss all material in terms of 3D reconstruction

- the reduction to 2D slice reconstruction is straightforward

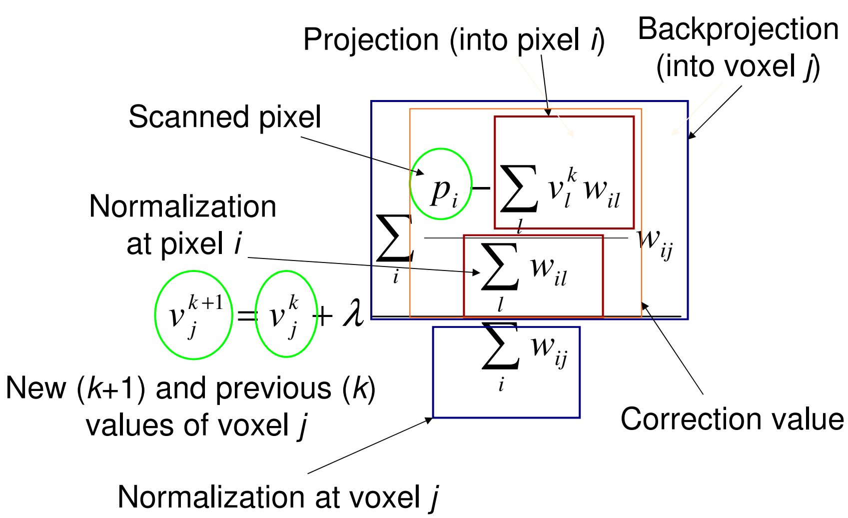
Pixels: the basis elements (point samples) of the projection image (the photon measurements)

Voxels: the basis elements (point samples) of the reconstruction volume (the attenuation densities or the tracer photon emissions)



Iterative CT Example: SART/SIRT

SPIE Medical Imaging



$$v_j^{k+1} = v_j^k + \lambda \frac{\sum_i \frac{p_i - \sum_l v_l^k w_{il}}{\sum_l w_{il}} w_{ij}}{\sum_i w_{ij}}$$

Kernel-Centric Decomposition

SPIE Medical Imaging

$$P:\ p_i = \sum_{j=0}^{N^3-1}\left(v_j \cdot w_{ij}\right) \qquad B:\ v_j = \sum_{i=0}^{M_\varphi-1}\left(p_i \cdot w_{ij}\right)$$

FBP

$$v_j = \sum_{p_i \in P_{set}} p_i w_{ij\_fdk} = \sum_{p_i \in P_{set}} B \cdot S$$

S: scanner projections

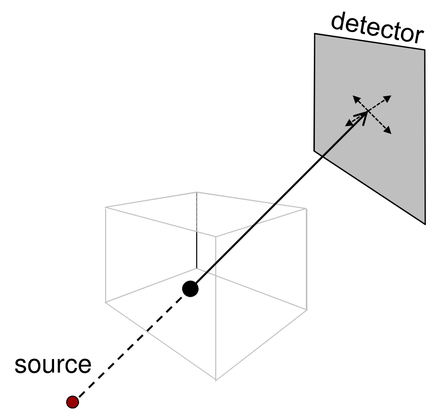
I: identity projection/volume

SART

$$v_j = v_j + \frac{\sum_{p_i \in P\varphi} \left(\frac{\lambda\left(p_i - \sum_{l=0}^{N^3-1} v_l \cdot w_{il}\right)}{\sum_{l=0}^{N^3-1} w_{il}} \right) w_{ij}}{\sum_{p_i \in P_\varphi} w_{ij}} = v_j + \frac{B\left(\lambda \frac{S - P(V)}{P(I)}\right)}{B(I)}$$

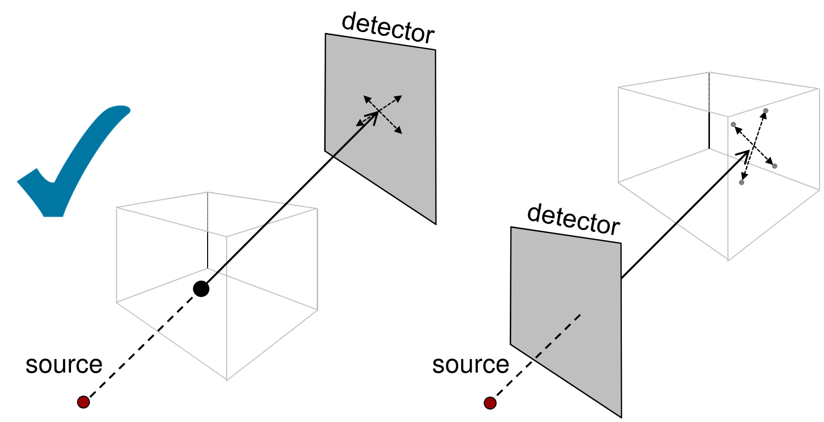
Backprojection: Options

- voxel-driven: sample in projection space
- one write per thread



Backprojection: Options

- voxel-driven: sample in projection space
- one write per thread
- pixel-driven, sample in volume space
- multiple writes per thread (scatter)

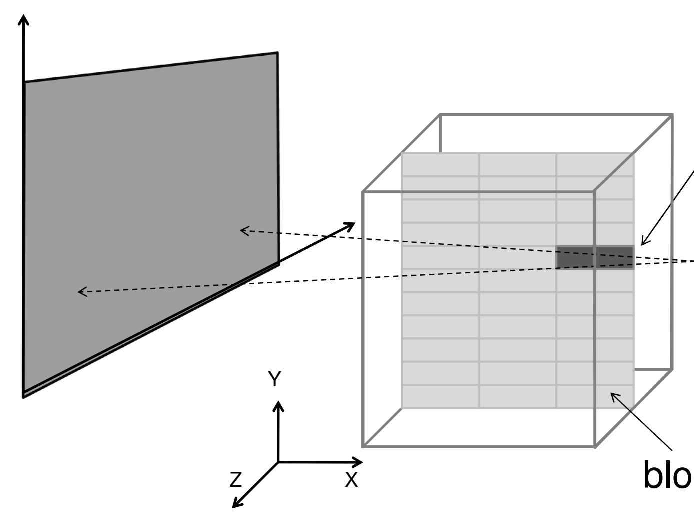


CUDA Memory – Backprojection

	Global Memory	Texture Memory
Access	Read/Write	Read only
Cached	No	Yes
Subject to coalescing	Yes	No
Interpolation	No support	Hardwired
Dimension	arbitrary	1D, 2D, 3D (supported after CUDA 2.0)

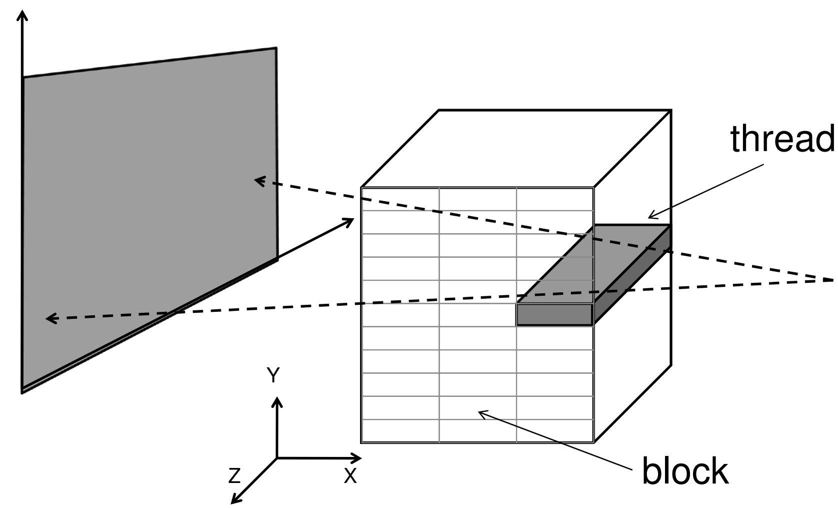
volume projections

CUDA Configuration: 2D



CUDA Configuration: 3D

SPIE Medical Imaging



Transformation Matrix

SPIE Medical Imaging

A 3x4 matrix M transforms 3D voxel coordinates to 2D pixel coordinates on the detector

Perform perspective divide if necessary (cone-beam)

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \end{bmatrix} \begin{bmatrix} x_v \\ y_v \\ z_v \\ 1 \end{bmatrix} = \begin{bmatrix} x_h \\ y_h \\ w_h \end{bmatrix} \qquad P_\varphi(u,v) = \left(\frac{x_h}{w_h}, \frac{y_h}{w_h}\right)$$

CUDA Implementation

SPIE Medical Imaging

[Host]:

for all projections P_i, trigger kernel on device

[Device]: per thread

$$M^{3\times4} \cdot \begin{bmatrix} x_v \\ y_v \\ z_v \\ 1 \end{bmatrix} = \begin{bmatrix} x_h \\ y_h \\ w_h \end{bmatrix}$$

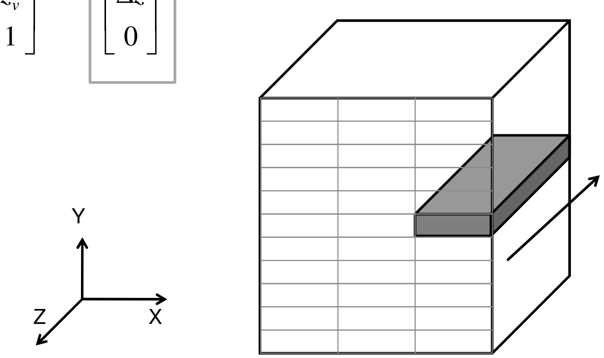
loop through each voxel in the thread

- obtain voxel coordinates in volume space
- compute projected coordinates on the detector using a 3x4 transformation matrix M
- perform perspective-divide if needed $P_\varphi(u,v) = \left(\frac{x_h}{w_h}, \frac{y_h}{w_h}\right)$
- depth weighting if needed
- interpolate pixel values on the detector (bilinear)
- accumulate sampled values on voxel

Incremental Computation

SPIE Medical Imaging

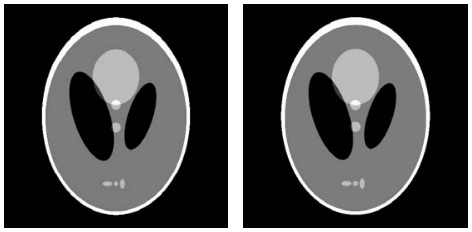
$$M \cdot \begin{bmatrix} x_v \\ y_v \\ z_v + \Delta z \\ 1 \end{bmatrix} = M \cdot \begin{bmatrix} x_v \\ y_v \\ z_v \\ 1 \end{bmatrix} + M \cdot \boxed{\begin{bmatrix} 0 \\ 0 \\ \Delta z \\ 0 \end{bmatrix}}$$



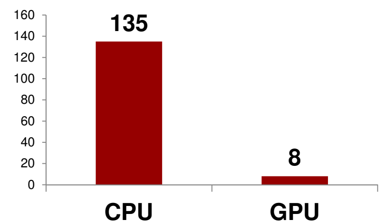
Example: Feldkamp Cone-Beam Reconstruction

SPIE Medical Imaging

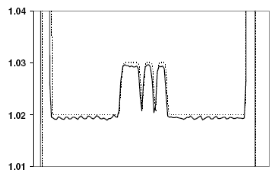
360 projections (1024^2, general position), 512^3 volume



CPU GPU



performance in seconds

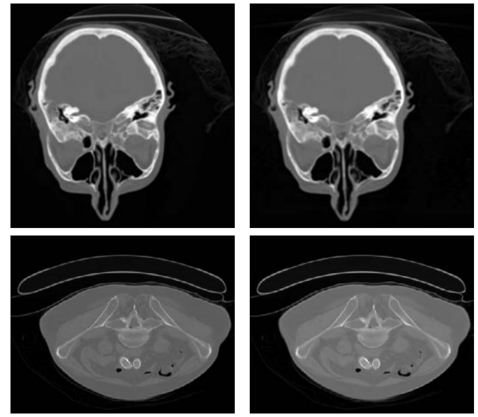


tumor profiles

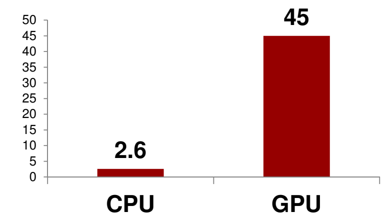
Expressed in Projections/Sec.

SPIE Medical Imaging

360 projections, 512^3 volume



Original GPU-recon

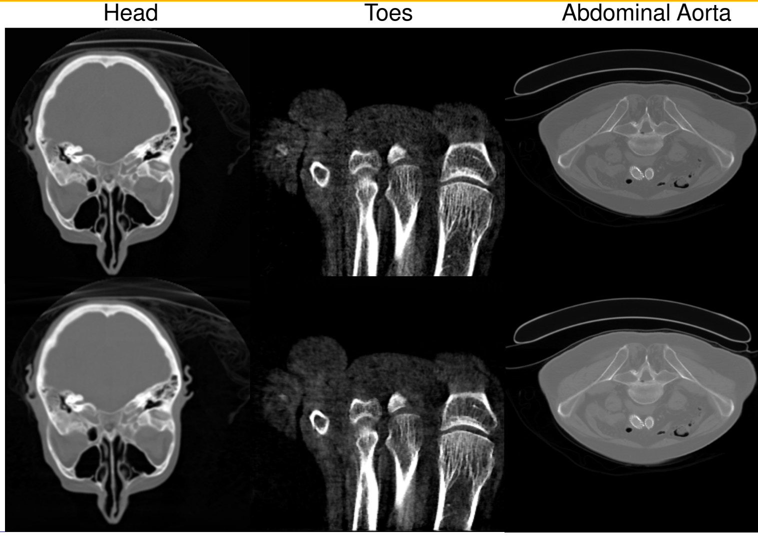


performance in projections/s

FDK: Medical Datasets

SPIE Medical Imaging

Head Toes Abdominal Aorta



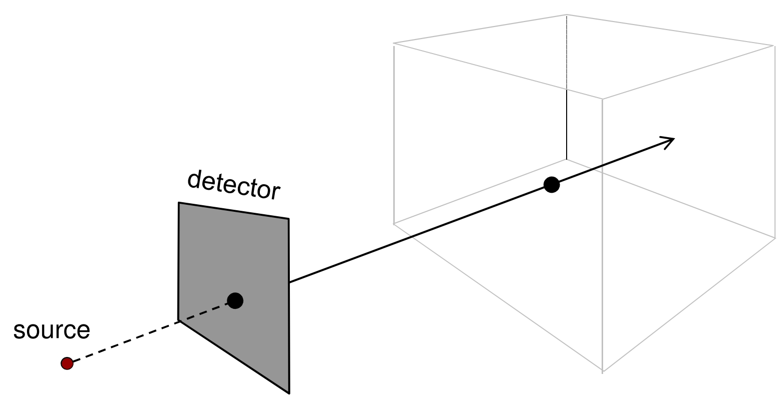
Original

Reconstructed

Forward Projection

SPIE Medical Imaging

Sample in volume space (pixel-driven / ray-driven)



CUDA Memory – Forward Projection

SPIE Medical Imaging

	Global Memory	Texture Memory
Access	Read/Write	Read only
Cached	No	Yes
Subject to coalescing	Yes	No
Interpolation	No support	Hardwired
Dimension	arbitrary	1D, 2D, 3D (supported after CUDA 2.0)

projections volume

Forward Projection: Memory

SPIE Medical Imaging

Ray-driven: sampling in volume space (trilinear interpolation)

Volume can be represented as either

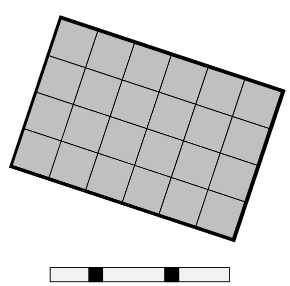
- a single 3D texture (supported after CUDA 2.0)
- stacks of 2D textures
  - A 3rd interpolation between adjacent 2D slices

Projection Algorithm

SPIE Medical Imaging

Raycasting methods [Krueger'03]

- [Host]:
  - generate volume bounding box (aligned with axis X/Y/Z)
  - generate threads for each pixel (ray), trigger kernel on device

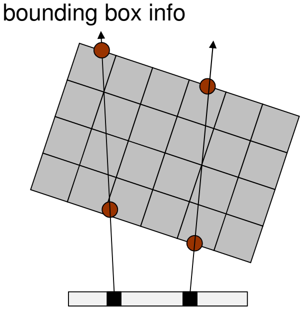


Projection Algorithm

SPIE Medical Imaging

Raycasting methods [Krueger'03]

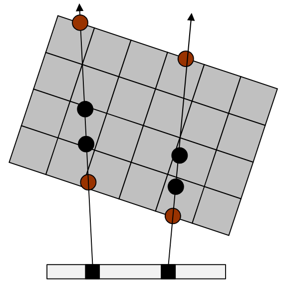
- [Host]:
  - generate volume bounding box (aligned with axis X/Y/Z)
  - generate threads for each pixel (ray), trigger kernel on device
- [Device]: in each thread
  - obtain ray entry & exit points using volume bounding box info
  - get ray directions using entry & exit points



Projection Algorithm

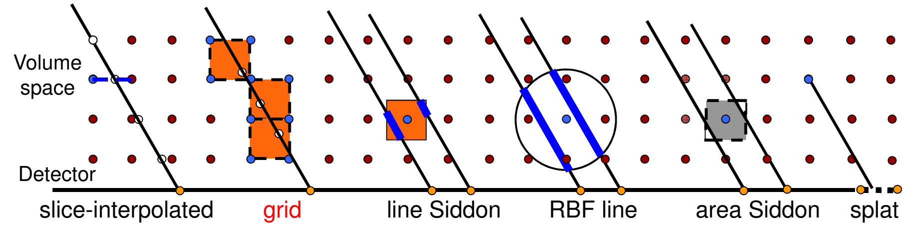
Raycasting methods [Krueger'03]

- [Host]:
  - generate volume bounding box (aligned with axis X/Y/Z)
  - generate threads for each pixel (ray), trigger kernel on device
- [Device]: in each thread
  - obtain ray entry & exit points using volume bounding box info
  - get ray directions using entry & exit points
  - cast rays, inside the loop:
    - sample in volume space
    - accumulate values
    - step forward equidistantly



Projection Accuracy

Investigated various schemes in terms of accuracy:



It was shown that the convenient grid-interpolated (trilinear) scheme is qualitatively competitive to the more involved ones listed here.

- see Xu / Mueller, "A comparative study of popular interpolation and integration methods for use in computed tomography," *IEEE 2006 International Symposium on Biomedical Imaging (ISBI '06)*

Example: Iterative Algorithms

Kernel selection depends on algorithms

Projection/Backprojection

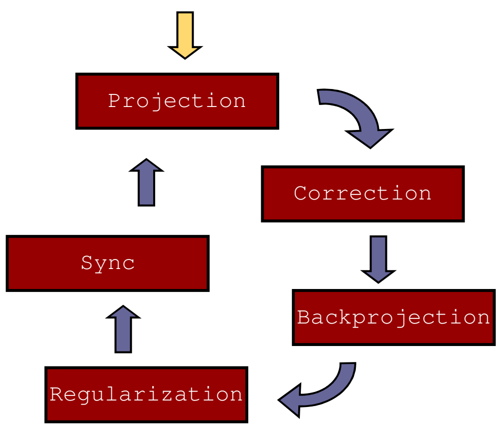
Correction

- pixel-wise operation
- subtraction

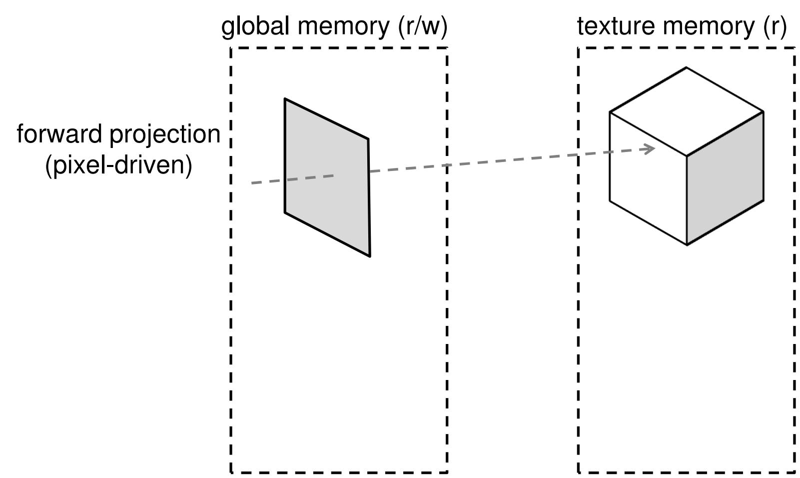
Regularization

- TVM or
- bilateral filter or
- non-local mean filter

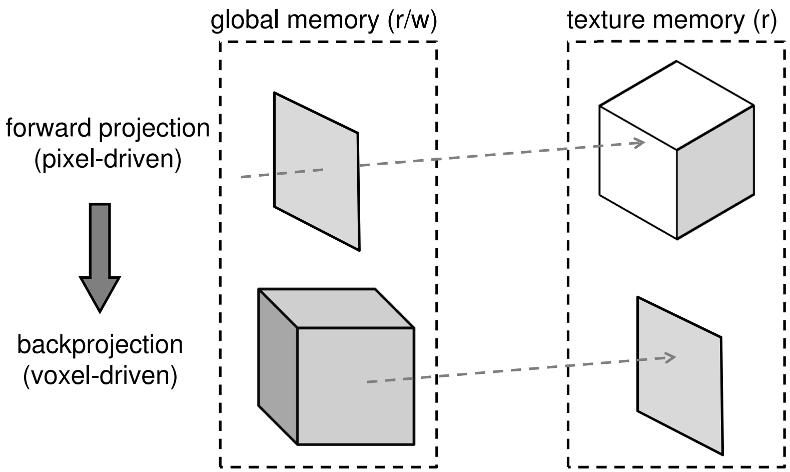
Sync



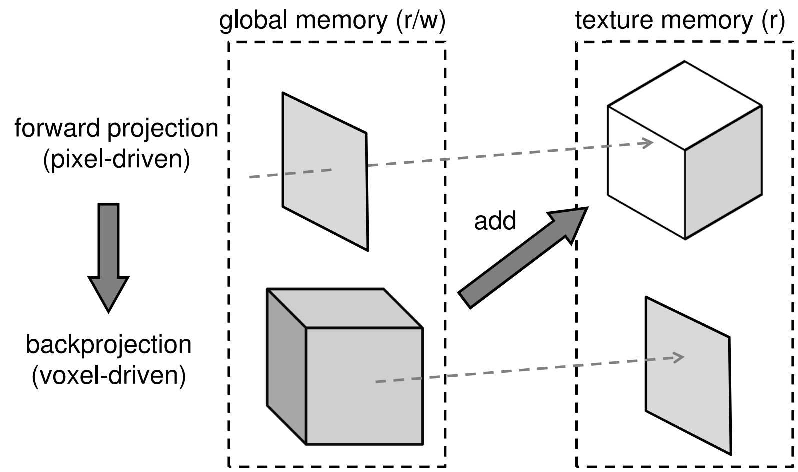
Sync



Sync



Sync



Regularization

Overall goal: make the reconstruction conform to expectations

- reconstruction is not noisy
- reconstruction has sharp edges

Various techniques

- Total Variation Minimization (TVM)
- bilateral filter (BLF)
- non-local means filter (NLM)

TVM

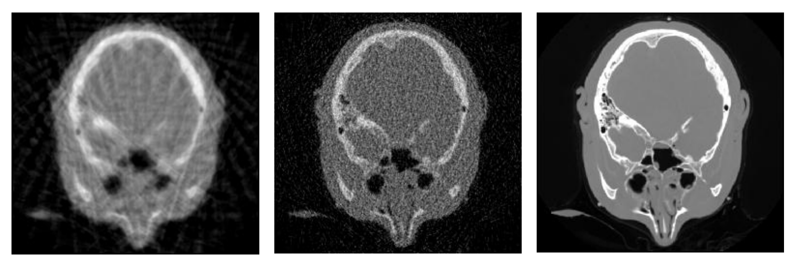
- motivated by compressive sensing (sparseness) theory

BLF, NLM

- popular in image processing and computer vision

Motivation

Want to remove low-dose CT artifacts:

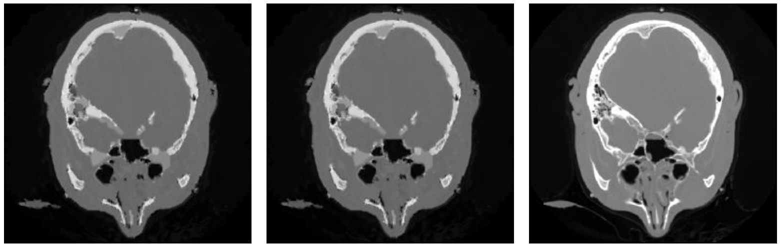


20 projections SNR=10 ↑

CT with low dose data high-dose data CT

Motivation

What we want to achieve – ideally:



20 projections SNR=10

CT + regularization

high-dose CT

Total Variation Minimization (TVM)

Goal is to minimize the overall energy:

$$E_{TV} = \int_{\Omega} |\nabla I| + \frac{1}{2}\lambda (I - I_0)^2 \, dxdy$$

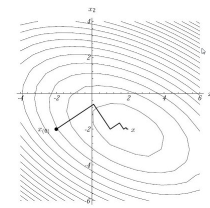
variation fidelity

Minimize using the steepest descent method

- for each voxel v_i do iteratively:

$$v_i^{k+1} = v_i^k - \beta \cdot \left(div\left(\frac{\nabla v_i^k}{|\nabla v_i^k|} \right) + \lambda (v_i^k - v_i^0) \right)$$

original voxel value



Relaxation Parameters (TVM)

Gradient step size β:

- << 1, usually 0.2

Fidelity term λ:

- initially set to 0
- next iterations:

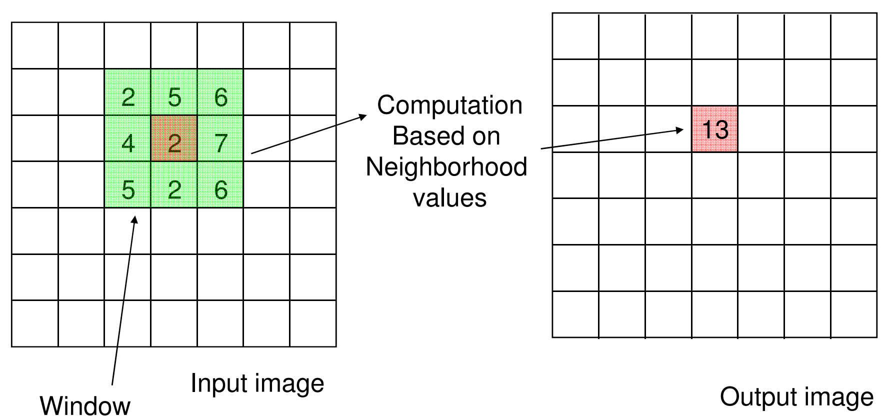
$$\lambda = \frac{1}{\sigma^2 |\Omega|} \int_{\Omega} div\left(\frac{\nabla I}{|\nabla I|}\right)(I - I_0) \, dxdy$$

- assuming:

$$\min_I \int_{\Omega} |\nabla I| \, dxdy \quad \text{subject to} \quad \frac{1}{|\Omega|} \int_{\Omega} (I - I_0)^2 \, dxdy = \sigma^2$$

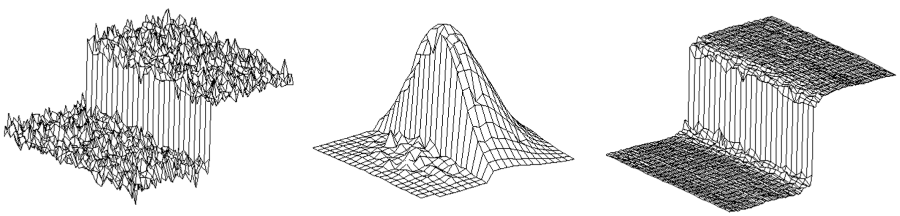
Non-linear Neighborhood Filters

- Generalization of discrete convolution



Bilateral Filter (BLF)

- Edge-preserving non-linear filter:



original edge bilateral filter smoothed edge

Bilateral Filter (BLF)

- Edge-preserving non-linear filter:

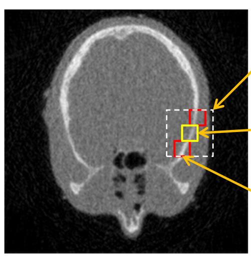
spatial closeness → $c(\xi - x)$

value closeness (similarity) → $s(f(\xi) - f(x))$

$$u(x) = \frac{\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\xi)c(\xi - x)s(f(\xi) - f(x))d\xi}{\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} c(\xi - x)s(f(\xi) - f(x))d\xi}$$

Non-Local Means Filter

Replaces a pixel at *x* with the mean of the pixels *y* with similar Gaussian-weighted neighborhood:



(search) window *W*

patch with updated pixel *x*

Gaussian-weighted neighborhood patches with pixels *y* (only highly-weighted shown)

Non-Local Means Filter

Replaces a pixel at *x* with the mean of the pixels *y* with similar Gaussian-weighted neighborhood:

$$NLM(x) = \frac{\sum_{y\in W} e^{-\frac{\sum_{t\in N} G_a(t)\left|img(x+t)-img(y+t)\right|^2}{h^2}} img(y)}{\sum_{y\in W} e^{-\frac{\sum_{t\in N} G_a(t)\left|img(x+t)-img(y+t)\right|^2}{h^2}}}$$

x, y, t: spatial variables

W: window centered at *x*

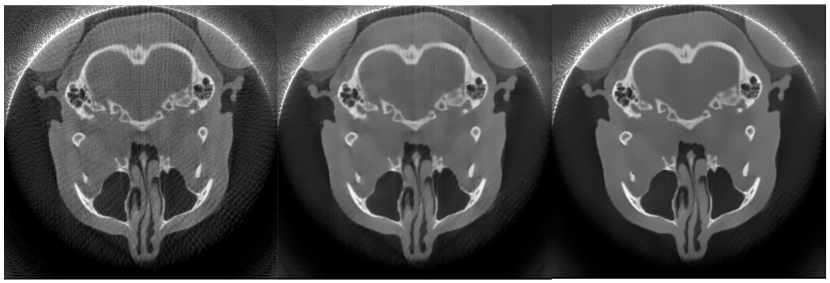
N: neighborhood centered at *x, y*

G_a: Gaussian kernel

h: filtering weight controls the influence of dissimilar pixels

NLM vs. TVM: Quality

NLM is as good (often better) than TVM



input TVM, λ=40 NLM, h=15

NLM vs. TVM: Speed

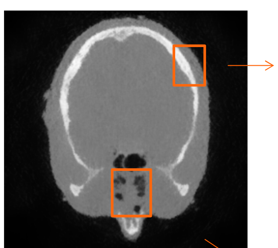
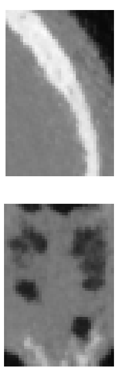
NLM is typically faster than TVM because it is non-iterative

- all parameters were manually set to yield similar visual quality
- CUDA GPU implementations (NVIDIA GTX 480)
- in seconds:

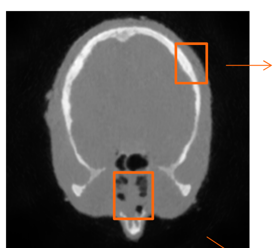
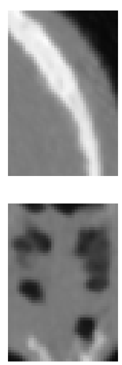
Image size	TV	NLM
256^2	57	12
512^2	80	42

Bilateral vs. NLM

Faster than NLM, but quality is lower

Bilateral
$\sigma_x = \sigma_y = 30$
$\sigma_r = 19$ $r_w = 8$

NLM
$h = 17$
$r_p = 5$ $r_w = 8$

Course Schedule

1:30 – 1:45: Introduction (Klaus)

1:45 – 2:00: Parallel programming primer (Klaus)

2:00 – 2:15: GPU hardware (Ziyi)

2:15 – 3:00: CUDA API, threads (Ziyi)

Coffee Break

3:30 – 4:00: CUDA memory optimization (Eric)

4:00 – 4:15: CUDA programming environment (Ziyi)

4:15 – 4:45: Parallelism in CT reconstruction (Klaus)

4:45 – 5:25: CT reconstruction examples (Eric)

5:25 – 5:30: Closing remarks (Klaus)