Automated Segmentation of Medical Imaging Studies Utilizing CUDA

Dr. Supratik Moulik

Cardiovascular Imaging Fellow
Department of Radiology

09/22/10

Penn Medicine

Conflict of Interest

- **Dr. Supratik Moulik does not have a significant financial stake in any company, nor does he receive financial support or grants from any corporate or government entity.**

Background

- Who am I?
  - Radiology Cardiovascular Imaging Fellow
  - Background in Engineering and Physics from Carnegie Mellon
  - Arrested Development (Season 2 episode 8)
    - Gob :“I'm an ideas man Michael. I think I proved that with “&%$* Mountain".
- Who might benefit from this talk?
  - Medical imagers
  - Medical informatics specialists
  - CUDA programmers
- What can be expected?
  - Segmentation issues
  - Existing CPU algorithms
  - CUDA Code walk-through
  - CPU vs GPU performance

Development Algorithm

- **Write segmentation code**
- **Identify where code fails**
- **Predict/detect failure**
- **Build in method for addressing issues**
  - Branch point
  - Feedback Loop
  - Iterative segmentation
- **Accept that I will never write code that can account for all versions of normal/abnormal**

Overview

I. Modalities In Medical Imaging
II. Isotropic Voxels and Volumetric Imaging
III. Utilizing CUDA for Image Analysis
IV. Outstanding Challenges in Medical Imaging
VI. Future of GPU Computing in Informatics

Overview

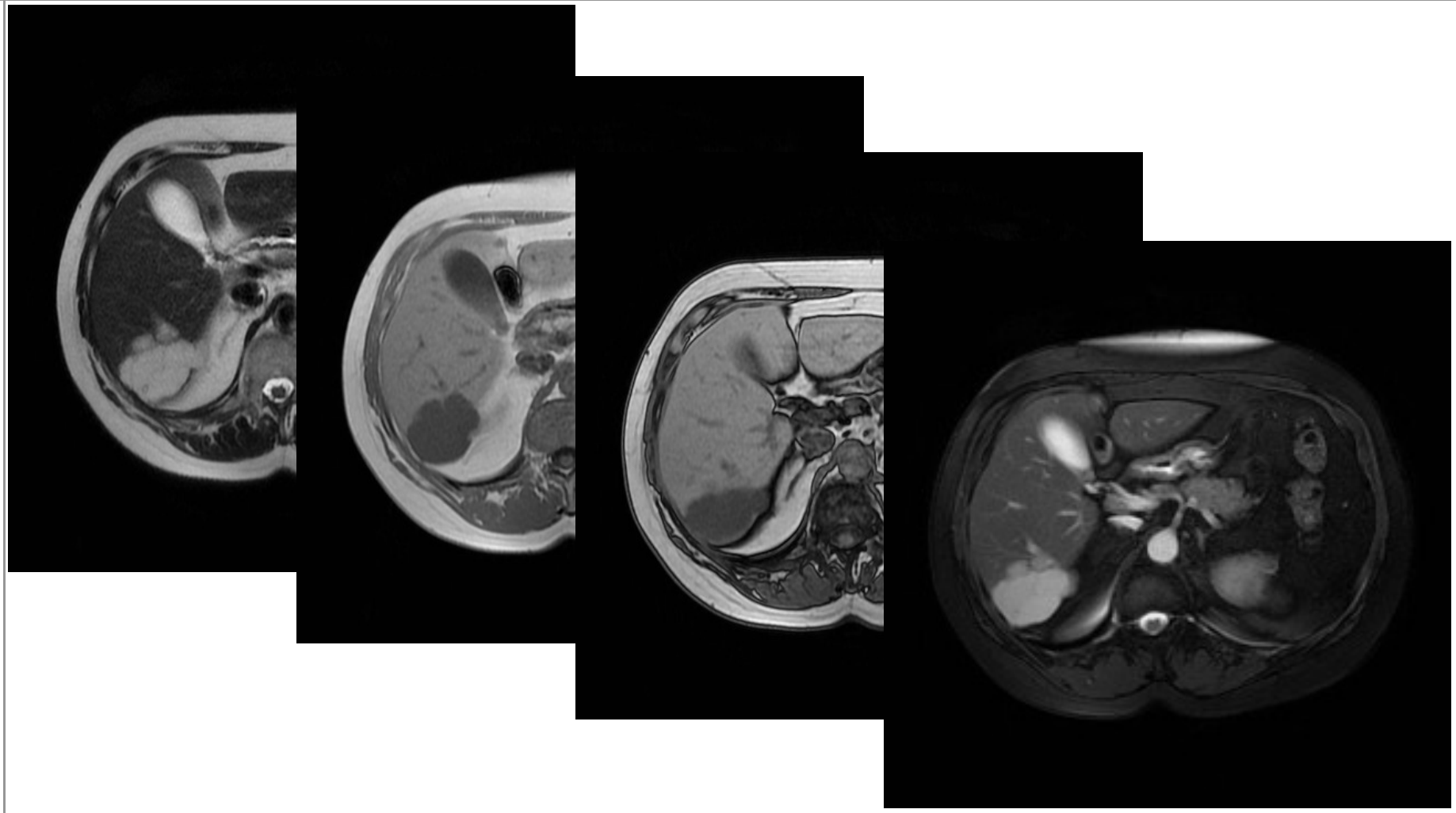
I. **Modalities In Medical Imaging**
   A. CT/MRI
   B. Informatics Basics
II. **Isotropic Voxels and Volumetric Imaging**
III. **Utilizing CUDA for Image Analysis**
IV. **Outstanding Challenges in Medical Imaging**
VI. **Future of GPU Computing in Informatics**

Magnetic Resonance Imaging

- Strong magnetic fields align magnetization.
- RF pulses to encode spatial information
- Read out data and convert to images

- Relatively low spatial resolution
- High contrast resolution
- Numerous artifacts

Magnetic Resonance Imaging

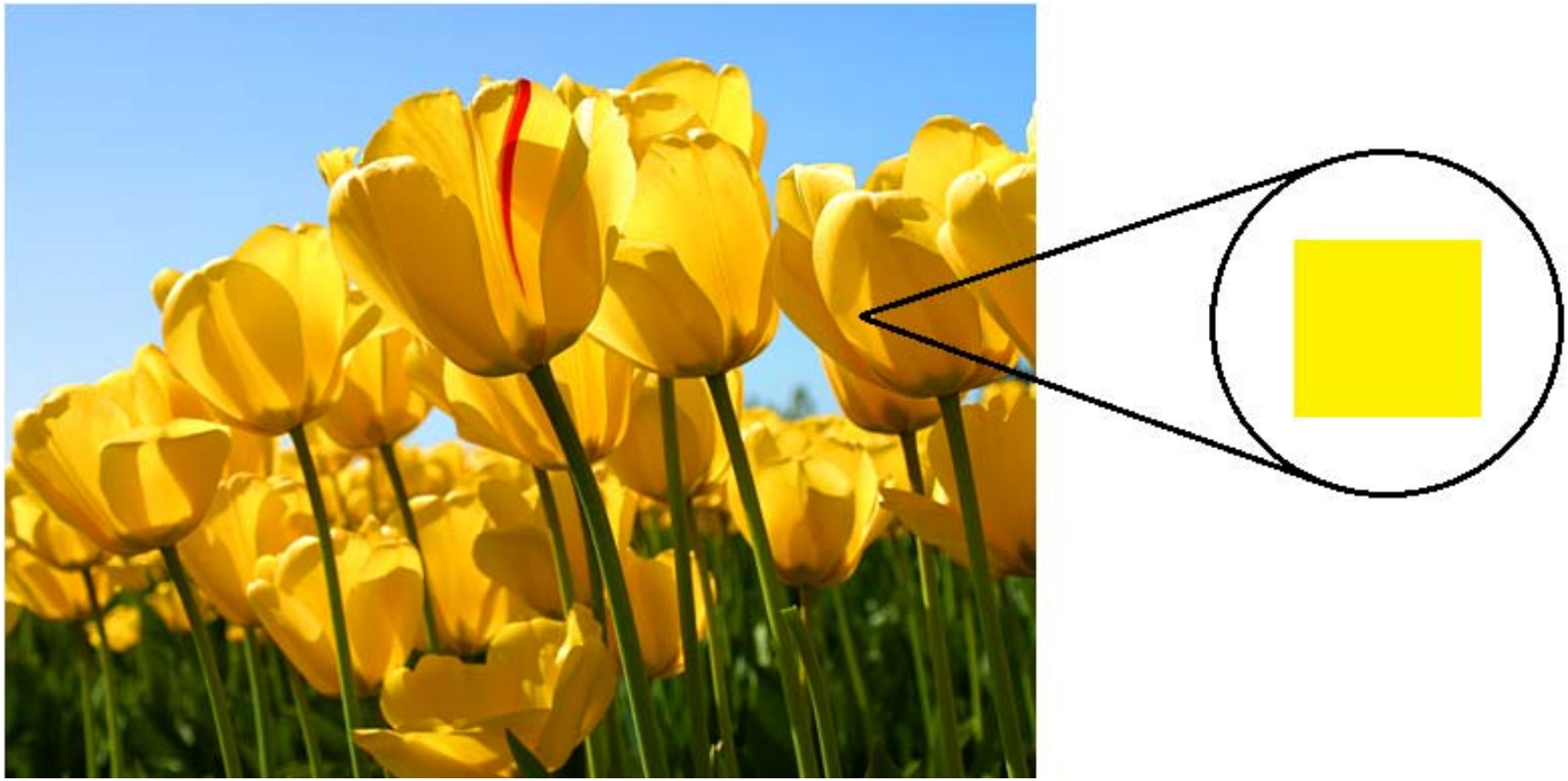


Computed Tomography

- Object irradiated and scatter radiation detected
- Density Profile
  - ρ(x,y,z)
- Volumetric acquisition

Pixel

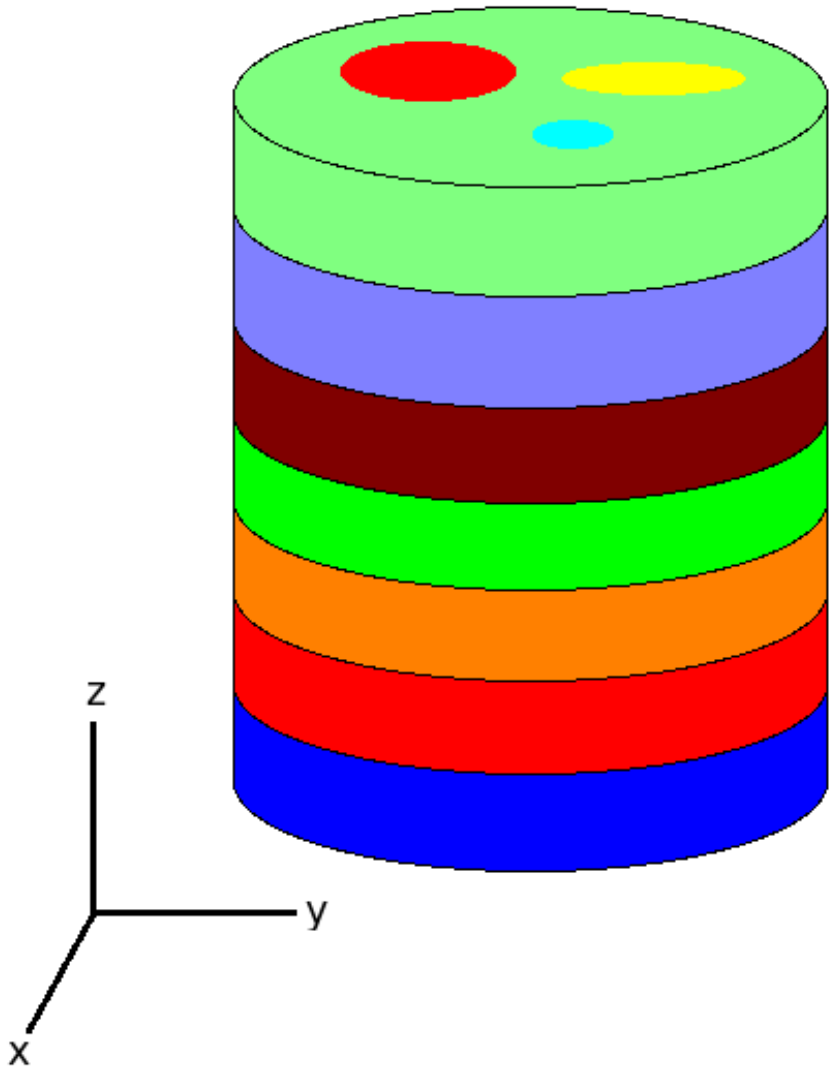
- **Pixel (picture element)**
  - 2D unit
  - smallest component of image.
  - A point/discrete sampling of a scalar field (e.g. texture maps)



Density profile

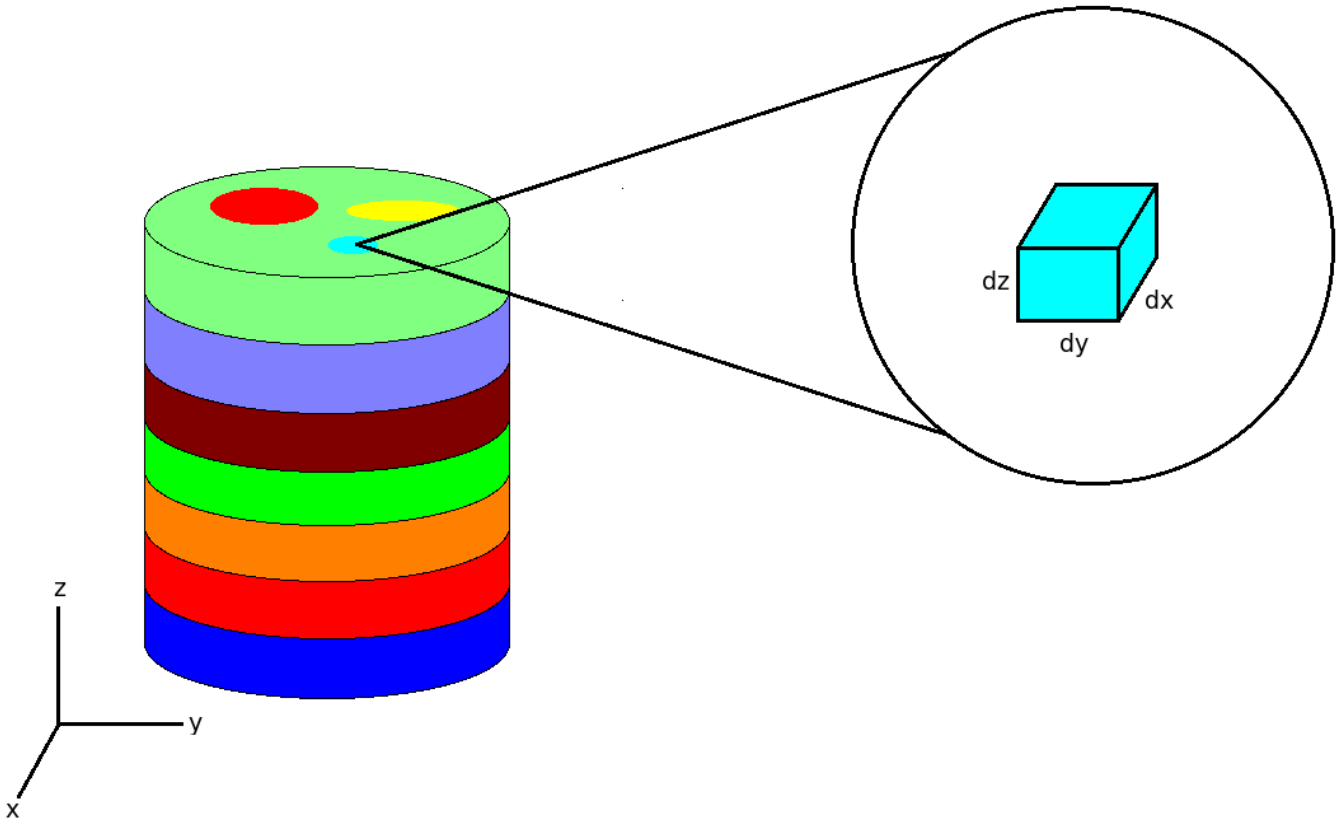
- **Purpose of CT scanning is to determine density profile of object being scanned**

- **ρ(x,y,z)**
  - complex function
  - Spatial and temopral variability

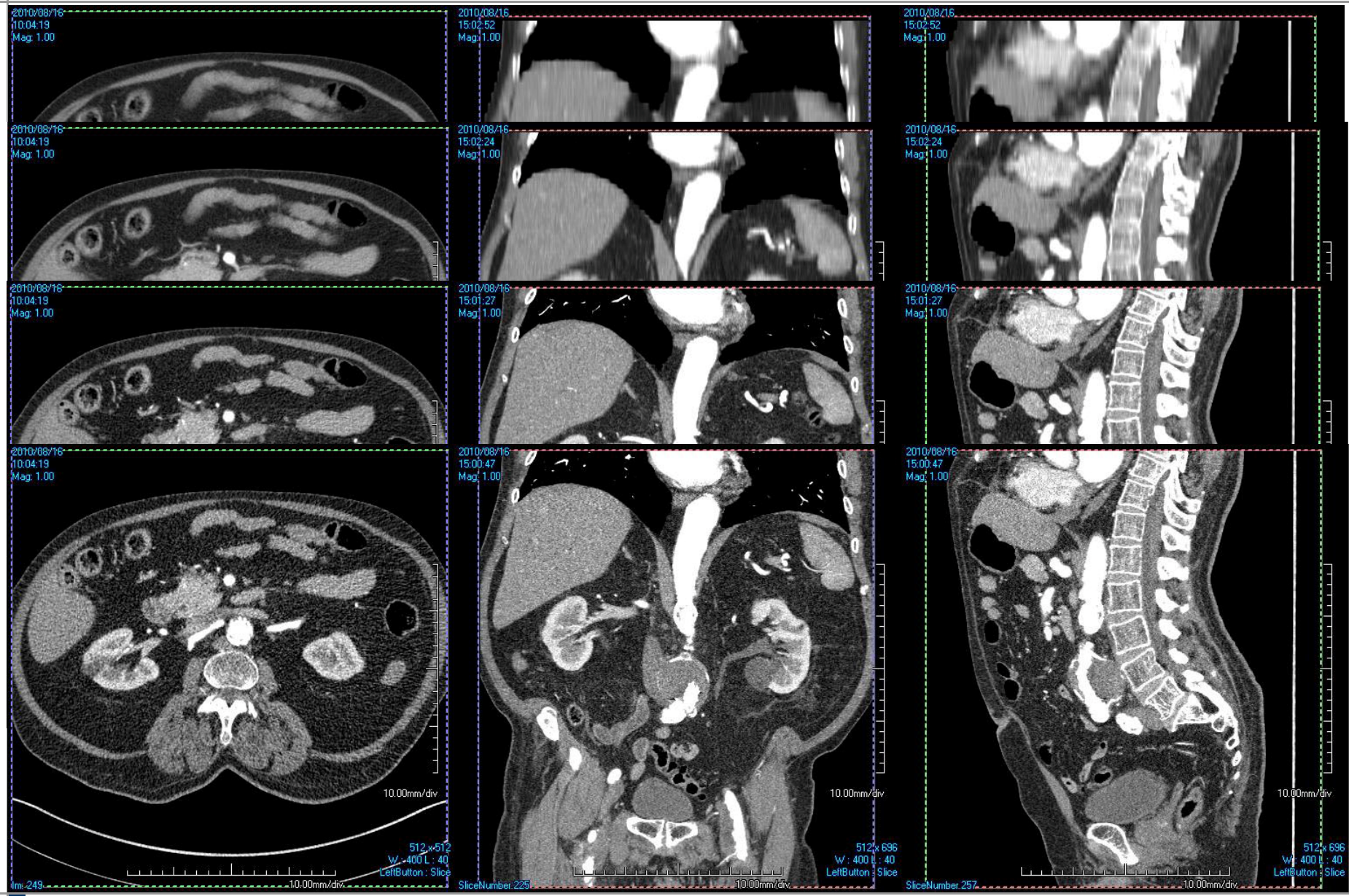


Volumetric Pixels

- **Voxel (volumetric pixel)**
  - xy component (spatial resolution)
  - z-component (function of resolution and reconstruction parameters)
  - Isotropic Voxel (dx == dy == dz)



Z component of Voxels

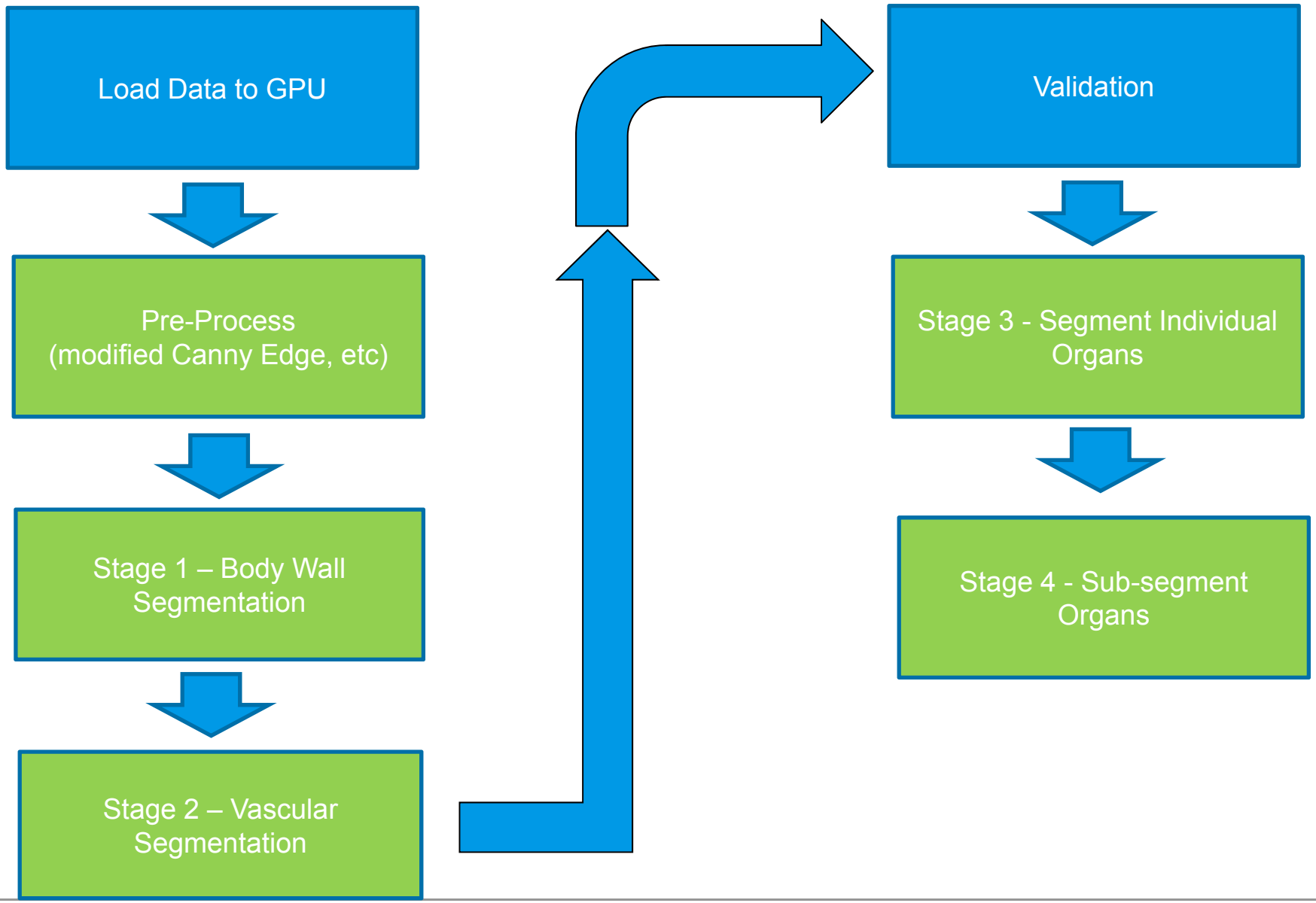


Bear's Home Recording Studio

Overview

I. Modalities In Medical Imaging
II. Isotropic Voxels and Volumetric Imaging
III. *Utilizing CUDA for Image Analysis*
    A. Wireframe Fitting
    B. Vessel Tracking
    C. Rigid Registration
IV. Outstanding Challenges in Medical Imaging
VI. Future of GPU Computing in Informatics

Automated Segmentation on CUDA



Wireframe Fitting

- **Deformable wireframes**
  - State based process in CUDA
  - Each iteration of entire wireframe is one kernel call
- **Allow for constrained motion**
  - Indistinct margins
  - Low count statistics (i.e. grainy images)
- **Conditions for isolating ROI**
  - Adjacent organs with similar density
  - e.g. left kidney / spleen interface

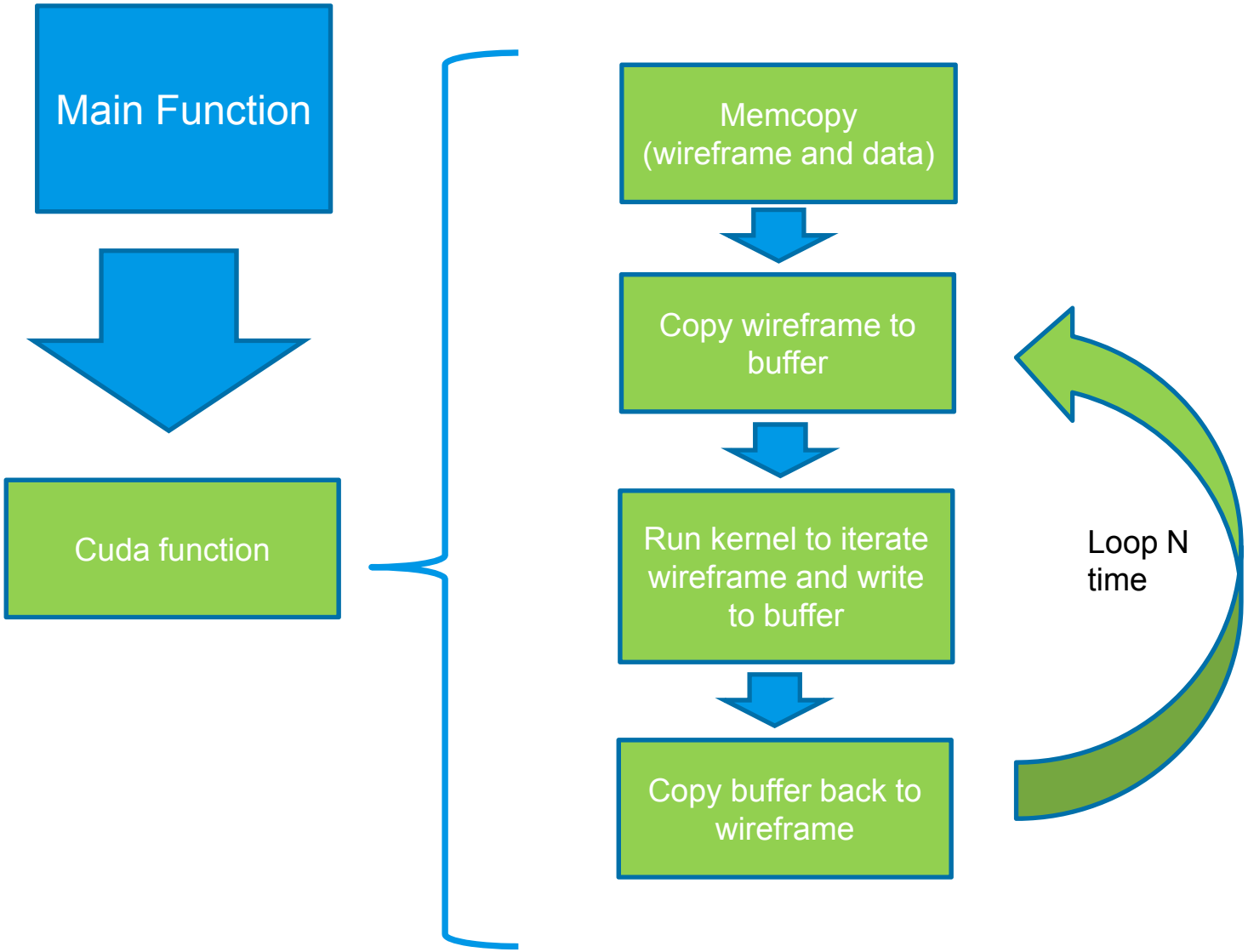
CPU Code

```
for (int i = (y_range_top-top_buffer); i<=(y_range_bottom+bottom_buffer); i++){e_node = Get_Random_abs(1,i,*slice); Encompass_Insert_New_Node(two, e_node);}
two->currnode = two->head; brk=1;

 while(brk)
  {
    while(brk_i)
      {
          if(two->currnode->fixed==0)
          {
              if(two->currnode == two->head){diff.x = 0; diff.y = two->currnode->x - two->currnode->next->x;}
              else if(two->currnode == two->tail){diff.x = two->currnode->x - two->currnode->prev->x; diff.y = 0;}
              else{diff.x = two->currnode->x - two->currnode->prev->x; diff.y = two->currnode->x - two->currnode->next->x;}

              if(diff.x <= 1 && diff.y <= 1)
              {
                  two->currnode->x++;
                  kd++;
                  pos.x = two->currnode->x;
                  pos.y = two->currnode->y;
                  pos.z = *slice;
                  cnode = Get_Random_f3(pos);

                  tpos = pos;
                  while(brk_t)
                  {
                      tpos.x += 1;
                      tnode = Get_Random_f3(tpos);
                      thk = abs(tpos.x - pos.x);
                      if(tnode.pixel < min_pix || tnode.pixel > max_pix){brk_t=0;}
                      else if(tpos.x - pos.x > 20){brk_t=0;}
                  }brk_t=1;


                  if(cnode.pixel > epidermis && cnode.pixel < mx_pix && thk >= bw_thk){two->currnode->fixed = 1;}
              }

          }
          if(two->currnode == two->tail){brk_i=0;two->currnode->next = NULL;}
          else if(two->currnode != two->tail){two->currnode = two->currnode->next;}
      }brk_i=1;
    two->currnode = two->head;
    if(kd==0){brk =0;}
    kd=0;
  }brk=1;
```

Inner Loop (loops over elements of wireframe)

Outer loop (checks for cessation of wireframe motion)

CUDA diagram



CUDA kernel

```
__global__ void VWF_Iterate(short4* DATA, float4* WIREFRAME, float4* tWIREFRAME, float* OD)
{
  int tx = threadIdx.x;
  int bx = blockIdx.x;

  short4 data;
  int thd_index = tx + bx * 512;
  int pt_index;
  float4 wf_point = WIREFRAME[thd_index];
  int var = -1;

  if(wf_point.w ==0 && wf_point.x >0)
    {
      int txm, bxm;
      if(tx ==0){txm = tx +1;} else if(tx == 511){txm = tx -1;} else{txm = tx;}
      if(bx ==0){bxm = bx +1;} else if(bx == 511){bxm = bx -1;} else{bxm = bx;}

      int thd_index_adj1 = txm + 1 + bxm * 512;
      int thd_index_adj2 = txm - 1 + bxm * 512;
      int thd_index_adj3 = txm + (bxm+1) * 512;
      int thd_index_adj4 = txm + (bxm-1) * 512;



      float4 adj1 = WIREFRAME[thd_index_adj1];
      float4 adj2 = WIREFRAME[thd_index_adj2];
      float4 adj3 = WIREFRAME[thd_index_adj3];
      float4 adj4 = WIREFRAME[thd_index_adj4];
      float4 diff;

      diff.x = wf_point.x - adj1.x;
      diff.y = wf_point.x - adj2.x;
      diff.z = wf_point.x - adj3.x;
      diff.w = wf_point.x - adj4.x;
```

Thread Indexing

Wireframe node indexing

Global Memory Access

```
if(diff.x >=var && diff.y >=var && diff.z >=var && diff.w >= var)
  {
      float2 asin, acos;
      int3 posi; float3

      asin.x = __sinf(wf
      asin.y = __sinf(wf
      acos.x = __cosf(wf
      acos.y = __cosf(wf

      wf_point.x -=1;

      posf.x = wf_point.
      posf.y = wf_point.
      posf.z = wf_point.
      posi.x = float2int
      posi.y = float2int
      posi.z = float2int

      posi.x += center_r
      posi.y += center_r
      posi.z += center_r

      if(posi.x < 0){pos
      if(posi.y < 0){pos
      if(posi.z < 0){pos

      pt_index = posi.x
      data = DATA[pt_ind

      if(data.y >=1){wf_
      else if(data.z > 64 && wf_point.x < 32 && data.y !=2){wf_point.w = 1;}

      tWIREFRAME[thd_index] = wf_point;
  }
 }
}
```

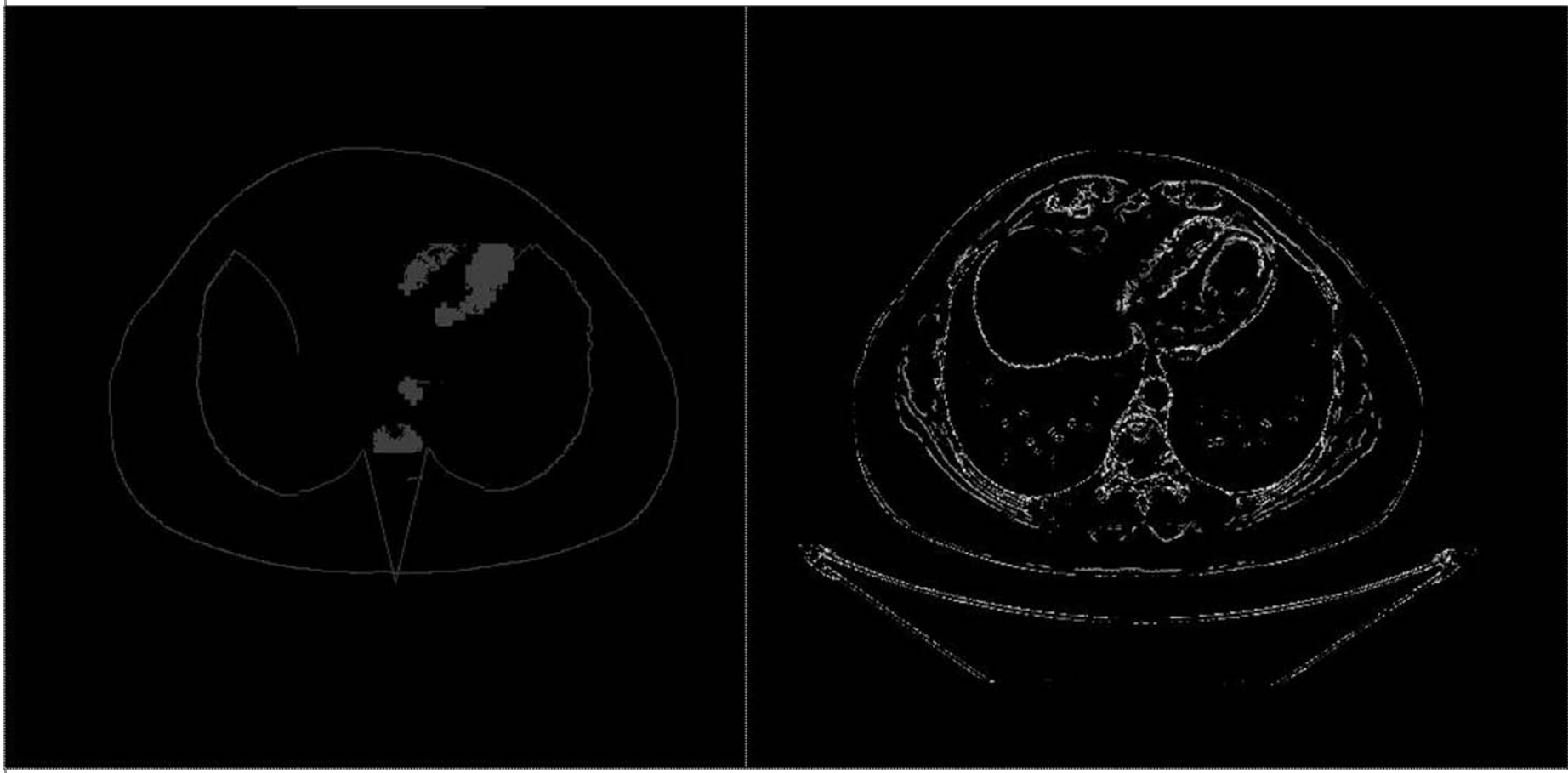
Texture maps in medical imaging are like drinking beer. It is a lot of fun and makes everything more attractive, but it impairs judgment and you don't want to get caught doing it at work.

ates

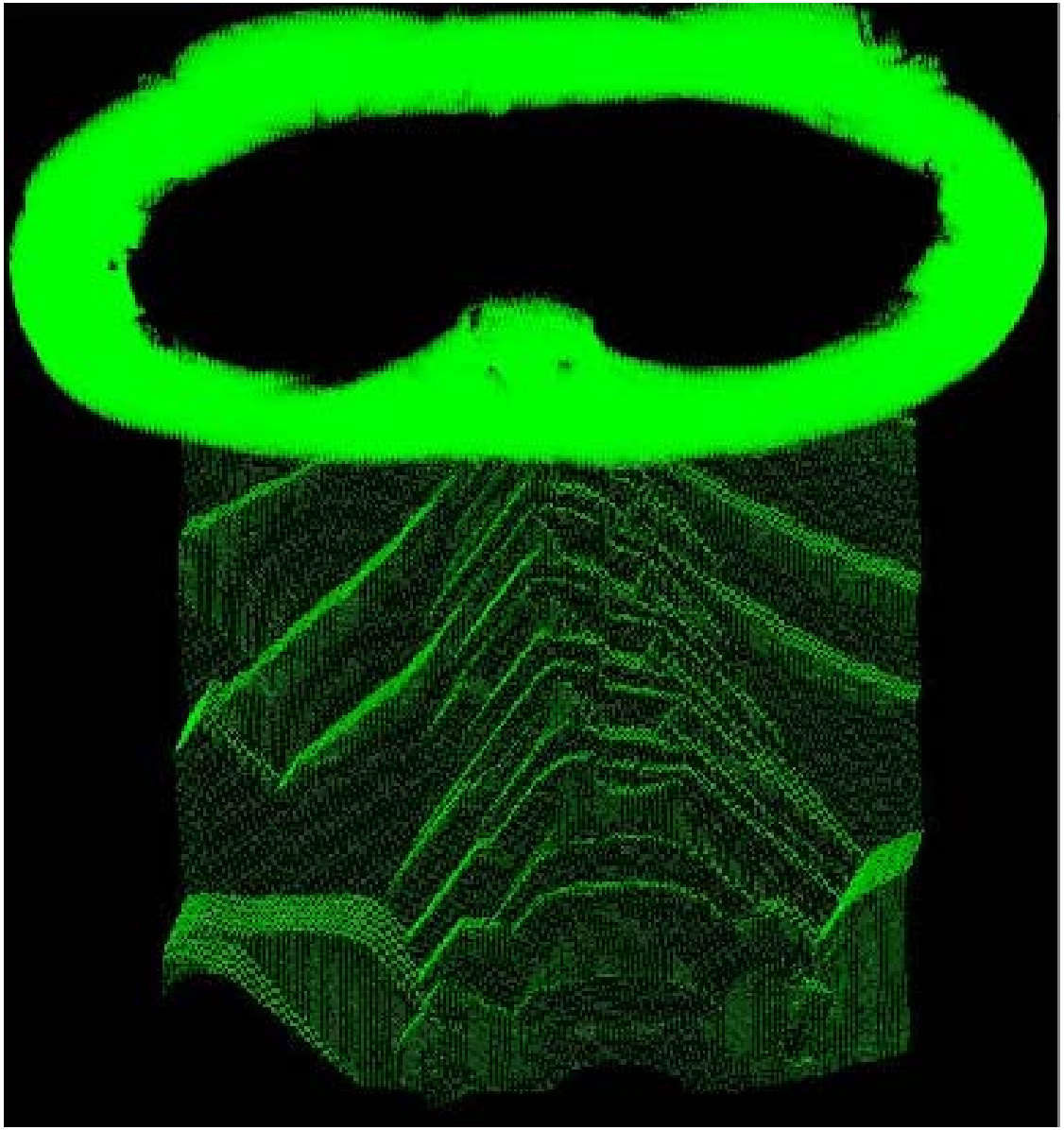
ine Tethering

Write to temp buffer

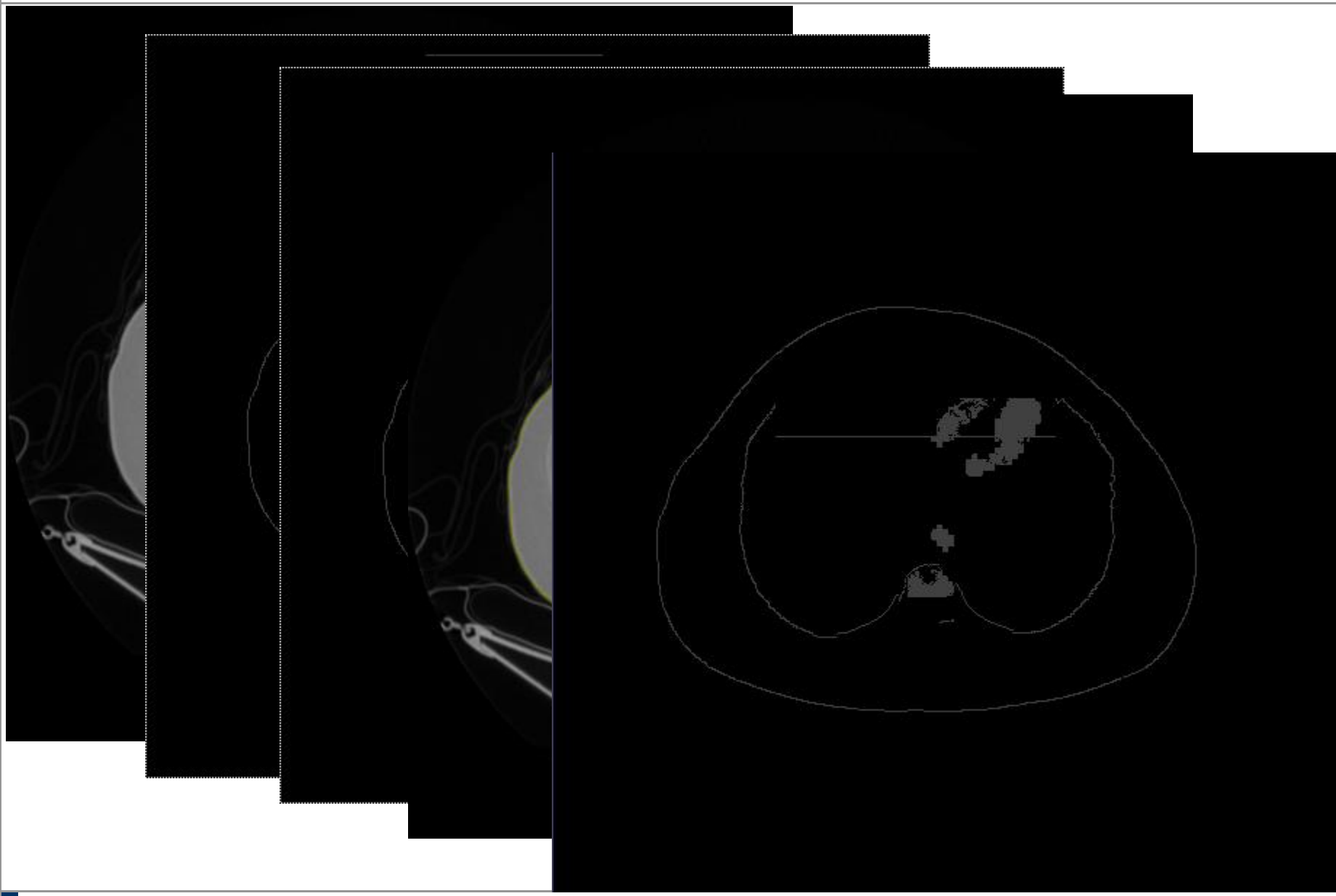
Identifying Chest Wall Margins



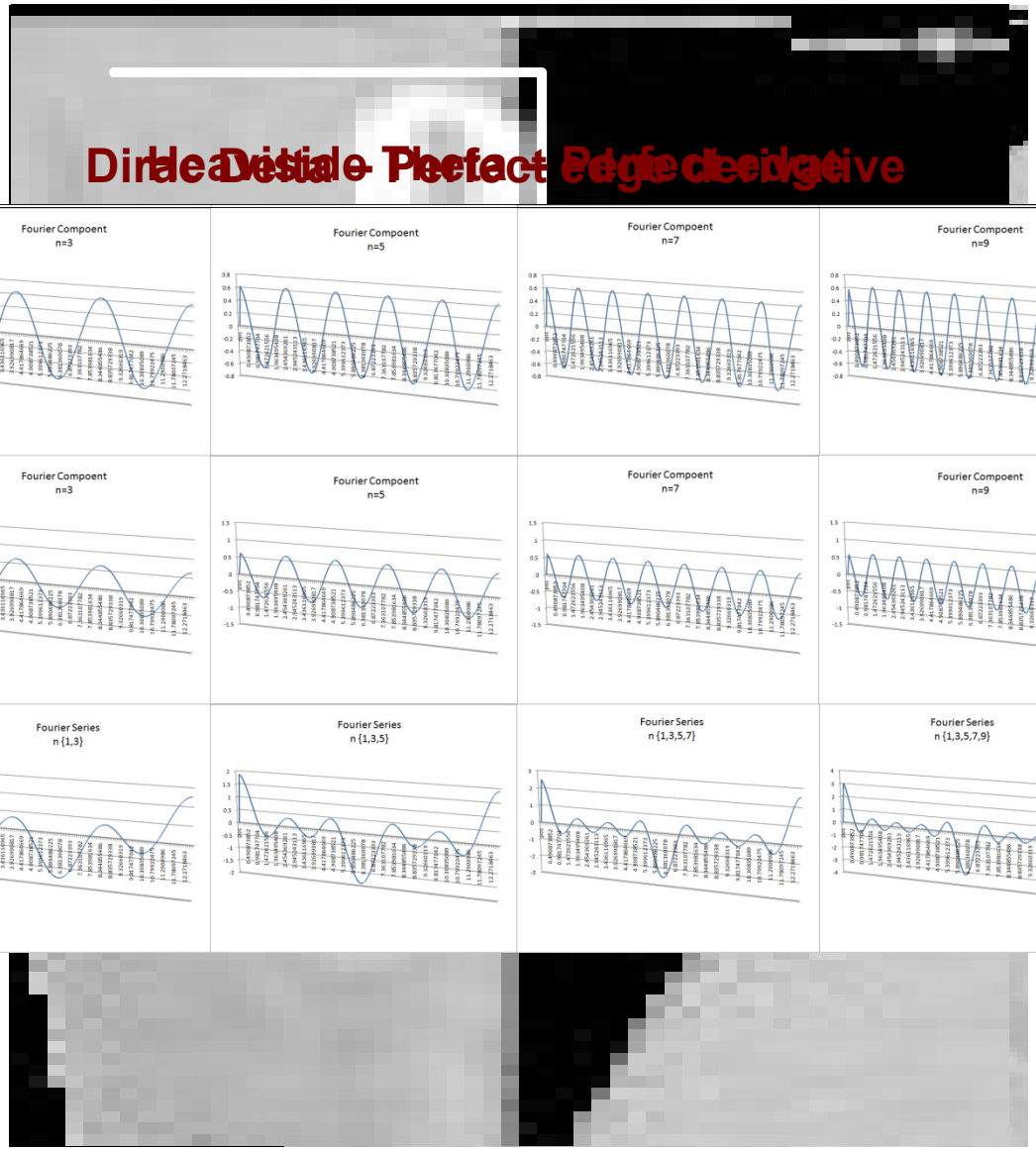
Reconstructing body wall margins



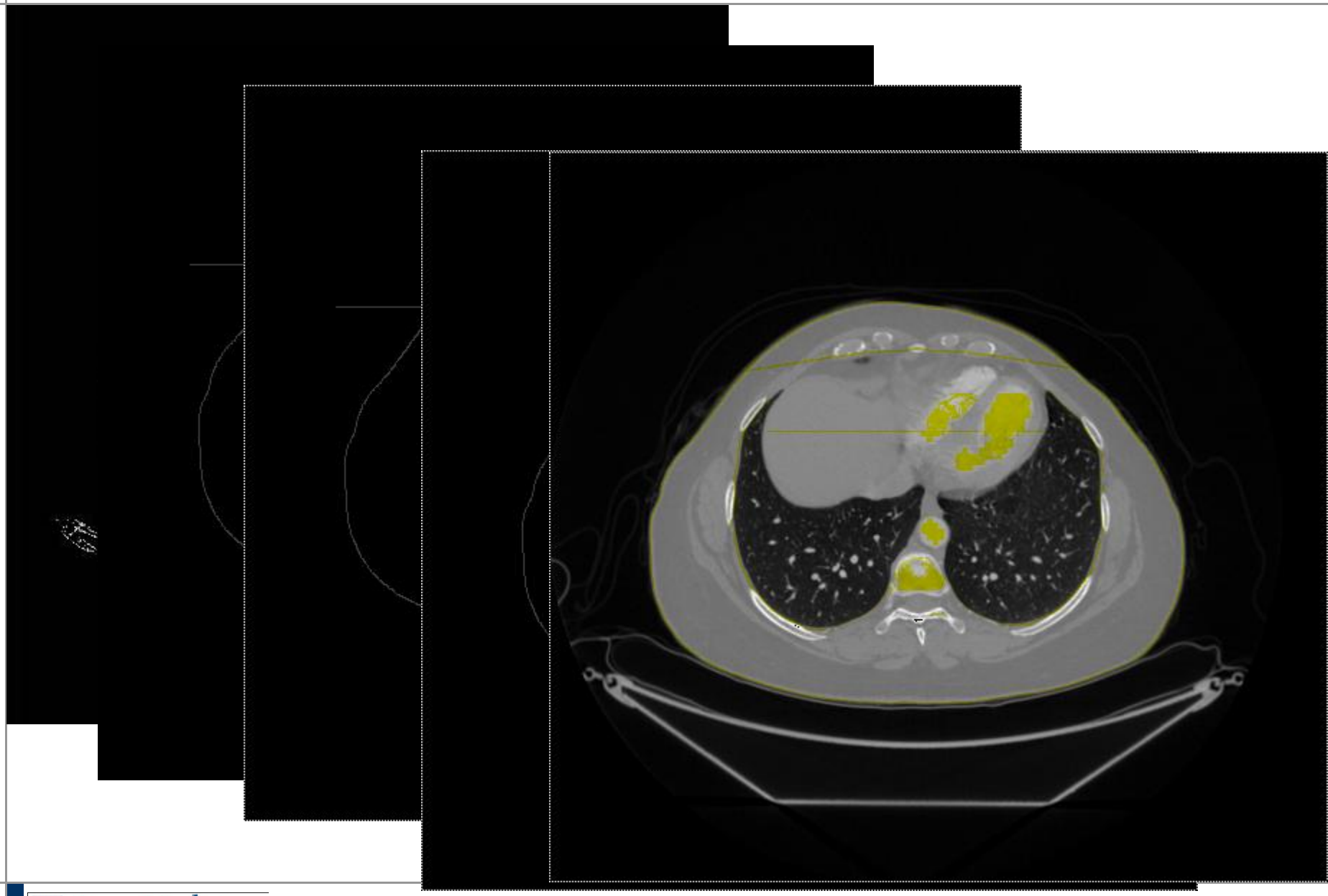
Issues of Differentiating Soft Tissue Interfaces

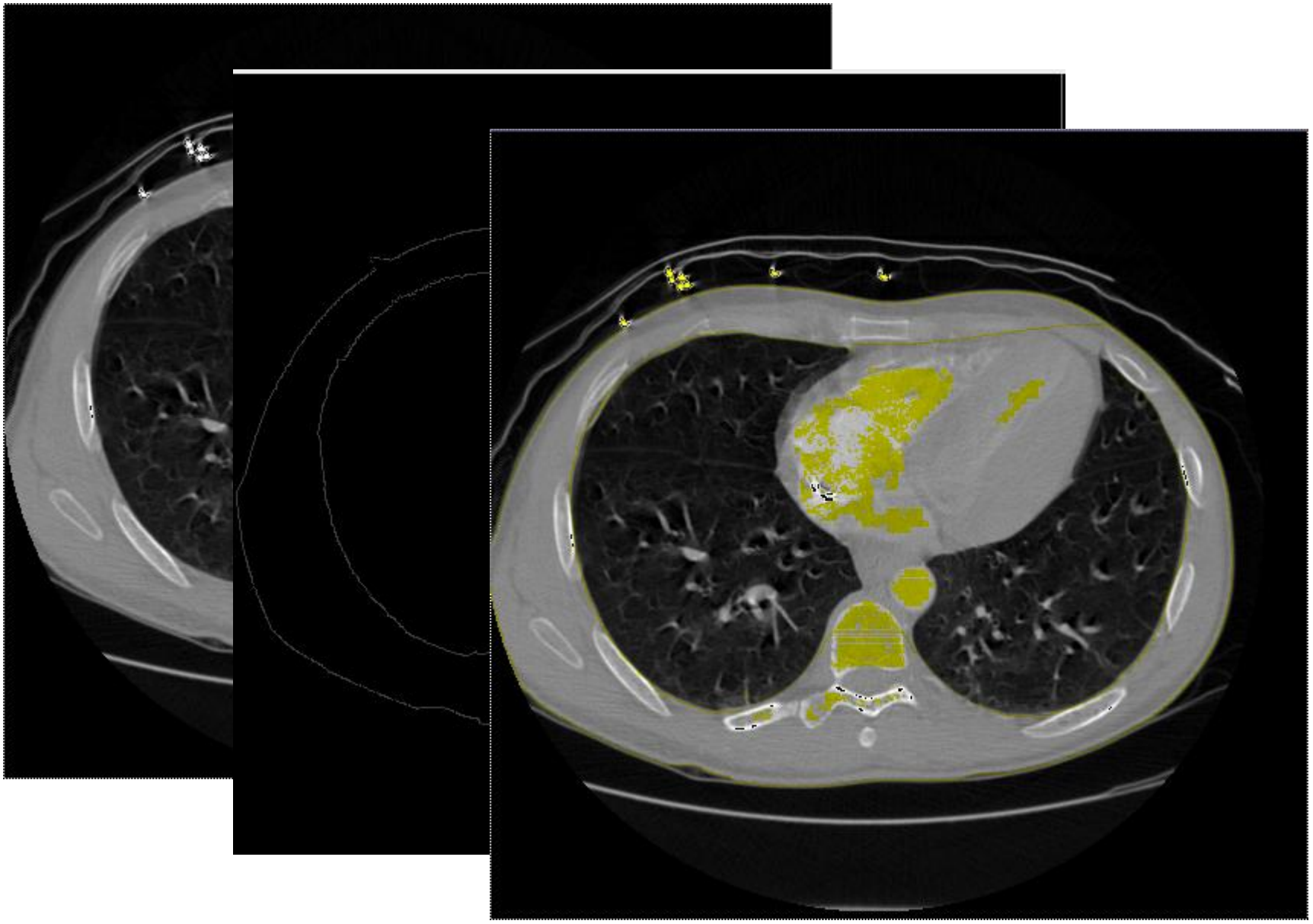


Suppressing Noise in Edge Detection

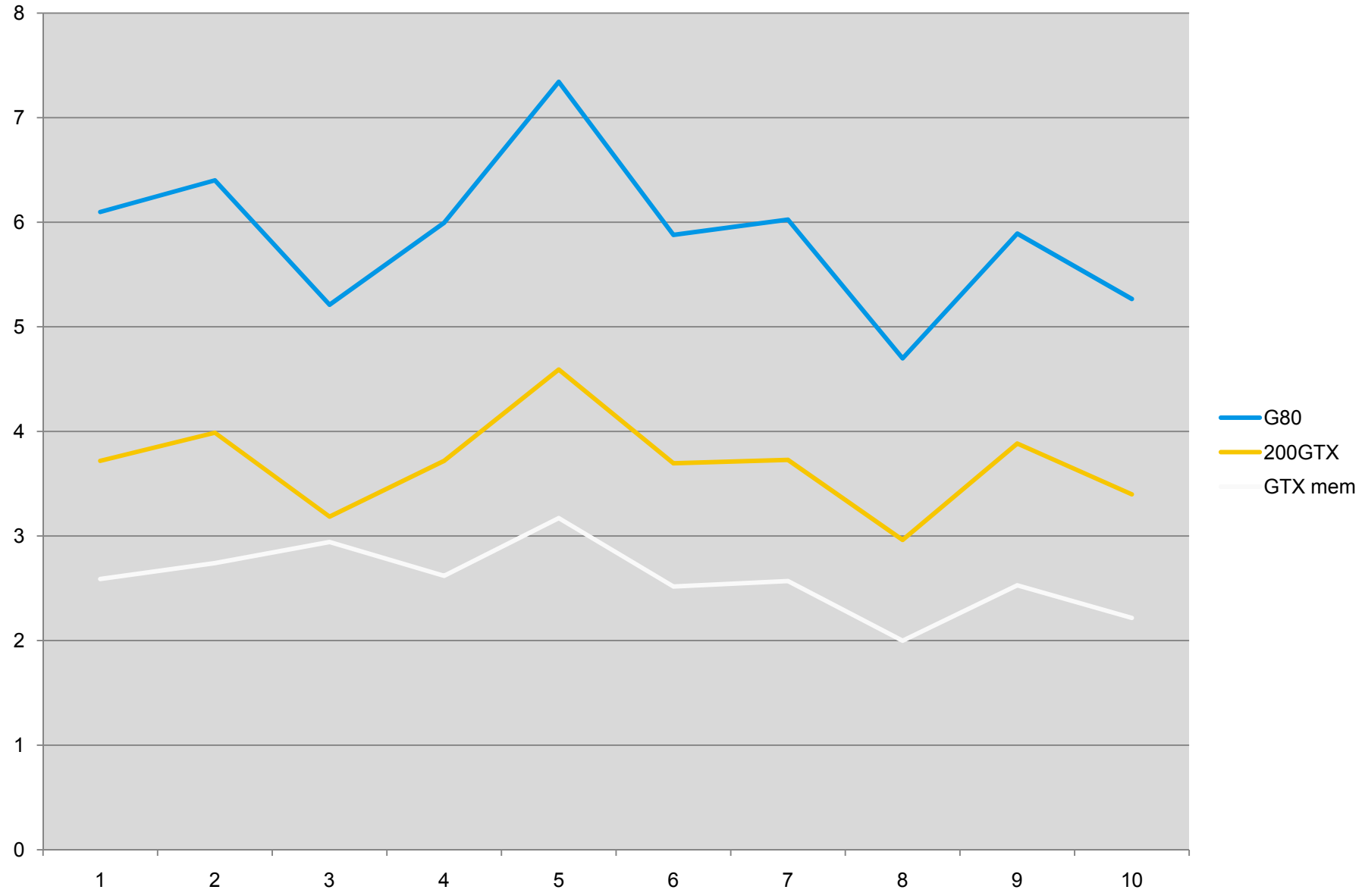


First two fourier coefficients





CPU/GPU Comparison

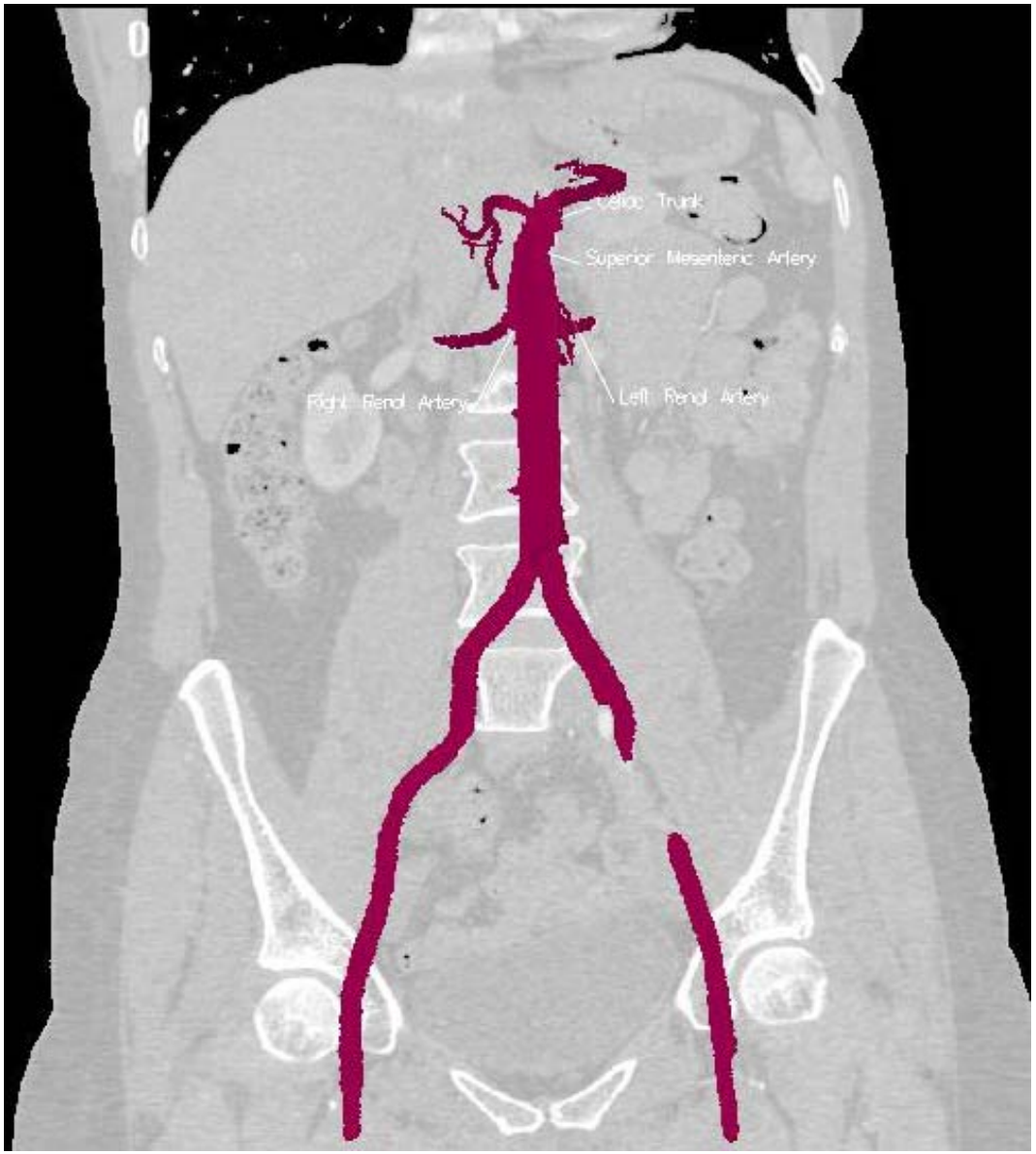


Known Issues contd…

- **Difficult to create coalesced memory access.**
  - Nearly random at times (CPU and GPU code)
- **Difficult to determine when movement stops, results in wasted cycles.**
  - Individual kernels fast enough that no significant change in overall run time
- **Cycling memory copies are major time factor**

Bear Learns to Play the Drums

Vessel Tracking

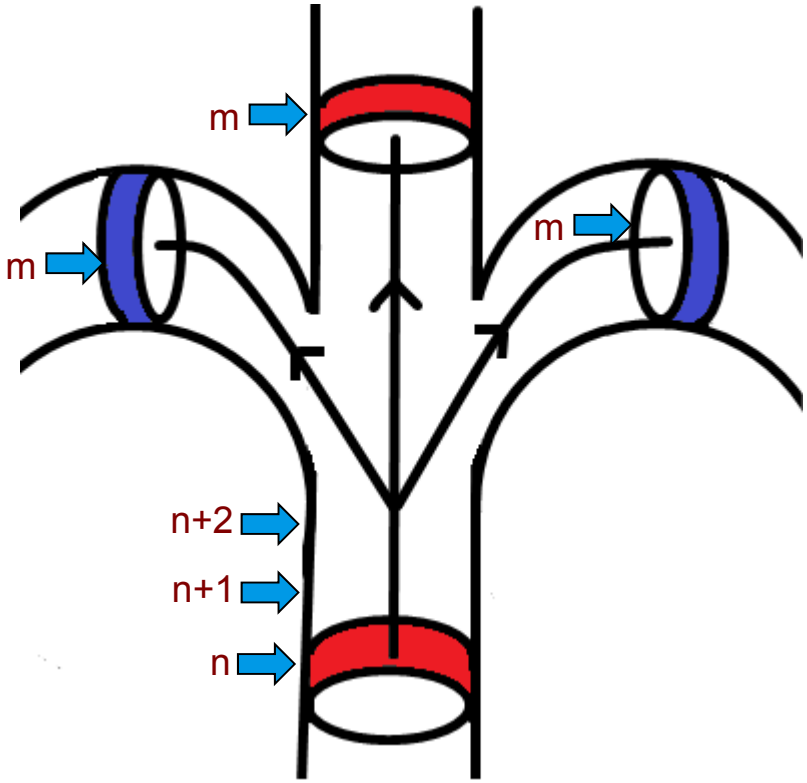


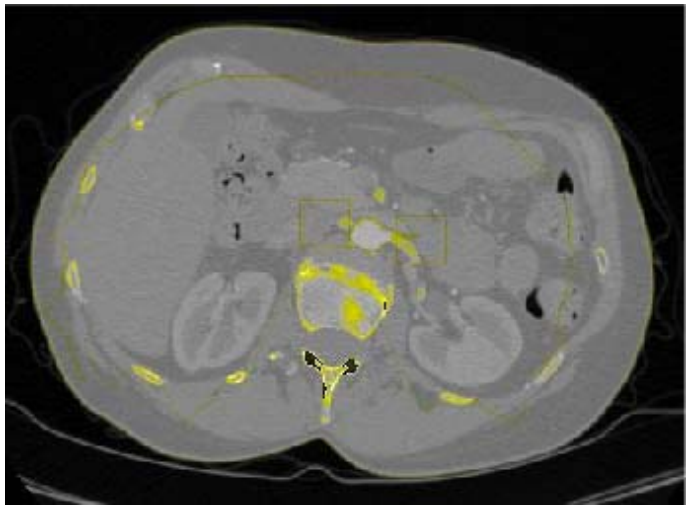
Vessel Tracking

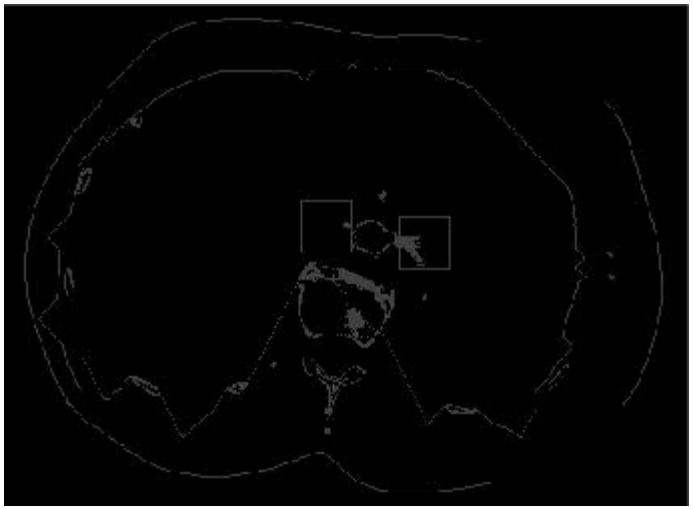
- **A subset of the more general concept of lumen tracking**
- **Initial algorithms functioned on analyzing 2D angiographic images.**
- **Threshold based region growth algorithms**
  - Widely available commercially.
  - Typically require a manual entry of seed point
  - Fail with non-standard anatomy
- **Predictive algorithms which can identify individual heart (coronary) arteries and neck (carotid) arteries**
  - Highly dependant on presence of standard (textbook) anatomy
  - Limited ability to segment individual downstream branches
- **What is important?**
  - “Is this a vessel?”
  - “what vessel is it”
  - “is the vessel normal in coarse and caliber”

Virtual Angiogram utilizing CUDA

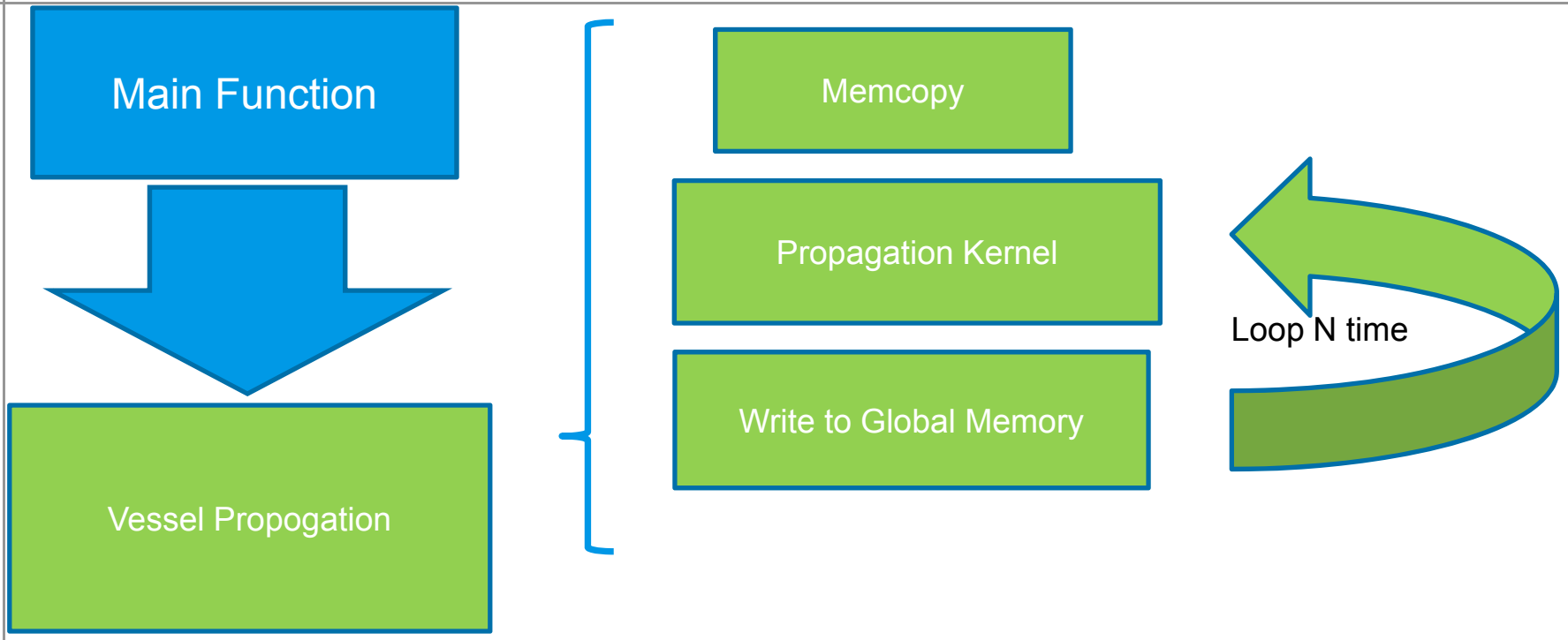
Virtual Angiogram







.cu Structure for Region Growth



Region Growth kernel

```
__global__ void Vessel_Axial_Evolve(float* CURR_VB, float* PREV_VB, float* NEXT_VB,  short2* CURR_D, short2* PREV_D, short2 *NEXT_D, int* SLICE)
{
  int tx = threadIdx.x;
  int ty = threadIdx.y;
  int bx_a  = threadIdx.z;
  int bx_b = blockIdx.x;
  int by = blockIdx.y;

  int bx = bx_b + bx_a*EVOLUTION_GRID_X_MOD;
  int bx_mod = bx+tx-2;
  int by_mod = by+ty-2;

  if(bx_mod <1){bx_mod = 1;}else if(bx_mod >511){bx_mod = 511;}
  if(by_mod <1){by_mod = 1;}else if(by_mod >511){by_mod = 511;}

  __shared__ int Slice;if(tx==2 && ty==2){Slice = *SLICE;}
  __syncthreads();
  __shared__ float Data_BLK[EVOLUTION_NET_BLOCK_SIZE][EVOLUTION_NET_BLOCK_SIZE][EVOLUTION_NET_OFFSET];
  __shared__ short2 D_BLK[EVOLUTION_NET_BLOCK_SIZE][EVOLUTION_NET_BLOCK_SIZE][EVOLUTION_NET_OFFSET];
  __shared__ float Center[EVOLUTION_NET_OFFSET];

  Data_BLK[tx][ty][bx_a] = CURR_VB[bx_mod + (y_dim*by_mod)];    //BW value
  D_BLK[tx][ty][bx_a].x = CURR_D[bx_mod + (y_dim*by_mod)].x;    //pixel value
  D_BLK[tx][ty][bx_a].y = CURR_D[bx_mod + (y_dim*by_mod)].y;    //extrema

  if(tx==2 && ty==2){Center[bx_a] = Data_BLK[tx][ty][bx_a];}
  __syncthreads();

  if(Center[bx_a] < 1 && Center[bx_a]> -1 && bx_mod==0 && by_mod==0)
    {
      int cnt=0;
      for(int i=-1;i<=1;i++){for(int j=-1;j<=1;j++){if(Data_BLK[i][j][bx_a] >=1){cnt+=1;}}}
      if(cnt >= 5){CURR_VB[bx_mod + (y_dim*by_mod)] = Slice;}
    }

  else if(Center[bx_a]>=1 )
    {
      if(Data_BLK[tx][ty][bx_a] <1)
    {
      if(bx_mod !=0 && by_mod!=0)
        {
          if(D_BLK[tx][ty][bx_a].x > cont_min-50 && D_BLK[tx][ty][bx_a].x < cont_max+50 && D_BLK[tx][ty][bx_a].y==0)
            {Data_BLK[tx][ty][bx_a] = Slice;}
          else if (D_BLK[tx][ty][bx_a].y!=0 && D_BLK[tx][ty][bx_a].x < cont_min-50)
            {Data_BLK[tx][ty][bx_a] = -1;}
          else if (D_BLK[tx][ty][bx_a].y!=0 && D_BLK[tx][ty][bx_a].x > cont_max+50)
            {Data_BLK[tx][ty][bx_a] = -1;}
          else {Data_BLK[tx][ty][bx_a] = 0;}
        }
    }
      CURR_VB[bx_mod + (y_dim*by_mod)] = Data_BLK[tx][ty][bx_a];
    }

  __syncthreads();
}
```

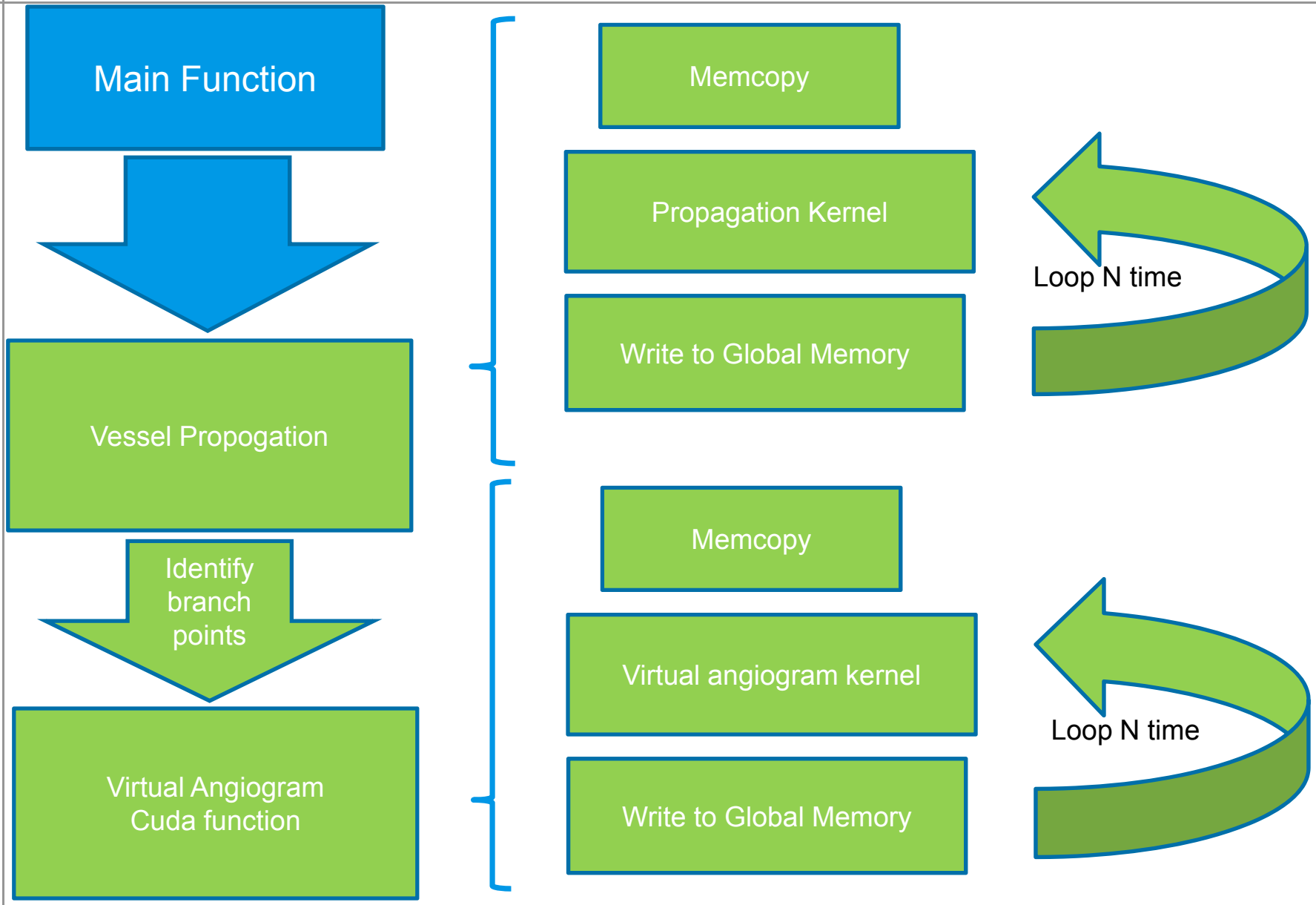
Indexing

Global Memory Copy

Perform Work

Result to Global Memory

.cu Structure for Virtual Angiogram



Virtual Angiogram kernel

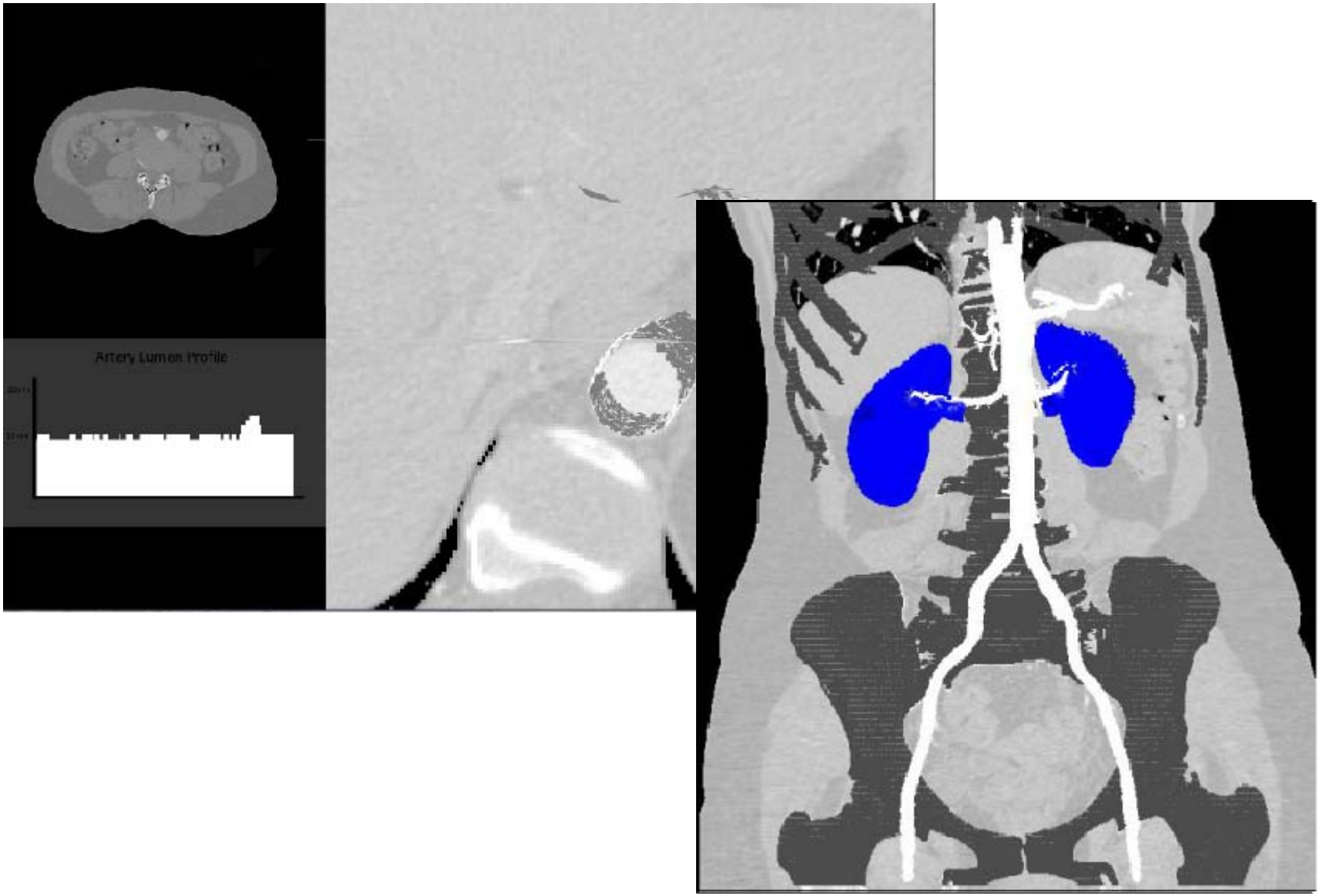
```
__global__ void VVessel_Connect(short2* DATA, float* VR, int* CNT)
{
  int tx = threadIdx.x;
  int ty = threadIdx.y;
  int bx = blockIdx.x;
  int by = blockIdx.y;
  int modx = truncf(bx/8);
  int mody = bx - modx * 8;
  int surr_dist=1;
  int3 pos, tpos;
  pos.x = tx + modx * 16;
  pos.y = ty + mody * 16;
  pos.z = by;

  int thd_index = pos.x + pos.y*128 + pos.z * 128*128;


  float pt_res = VR[thd_index];
  int cnt = CNT[tx];

  if(pt_res == cnt)
    {
      int go_flag=1; if(pos.x <=0 || pos.x >= 128){go_flag=0;}
      else if(pos.y <=0 || pos.y >= 127){go_flag=0;}
      else if(pos.z <=0 || pos.z >= 63){go_flag=0;}
      short2 tpt;
      float tr;
      int ti;
      int res = cnt + 1;

      if(go_flag==1)
        {
        for(int k=-surr_dist;k<=surr_dist;k++)
            {
            for(int j=-surr_dist;j<=surr_dist;j++)
                {
                for(int i=-surr_dist;i<=surr_dist;i++)
                    {
                    tpos = pos; tpos.x += i; tpos.y += j; tpos.z += k;
                    ti =  tpos.x + tpos.y*128 + tpos.z * 128*128;
                    if(k!=0 || j!=0 || k!=0)
                        {
                        tr = VR[ti];
                        tpt = DATA[ti];

                        if(tpt.y == 1 && tr ==0){VR[ti] = res;}
                        }
                    }
                }
            }
        }
    }
}
```

n=>n+1

Indexing

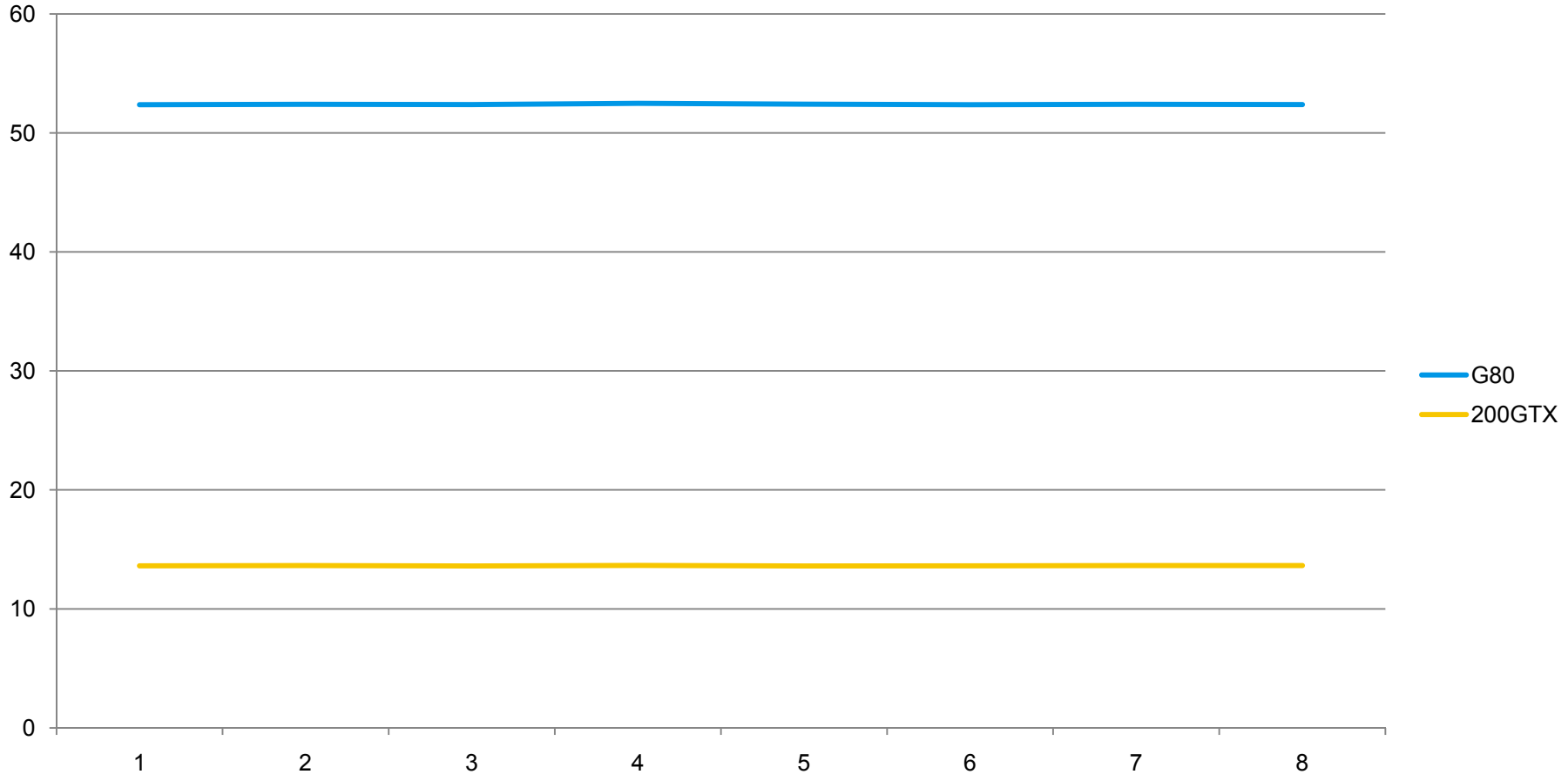
Global Memory Access

Perform analysis
&
Copy to Global Memory


Artery Lumen Profile

Memory coalescing

- Initial phases well suited for G80 and newer GTX200/400
- Work on isolated blocks of data
- Vessels less than 0.2 % of total voxels



Known Issues

- **Only works with contrast enhanced studies**
- **Partially depends on accuracy of body wall/bone removal algorithms**
- **Initial stages non-specific**
  - “enhance” non-vascular structures

2 Losers at a Windows 7 House Party

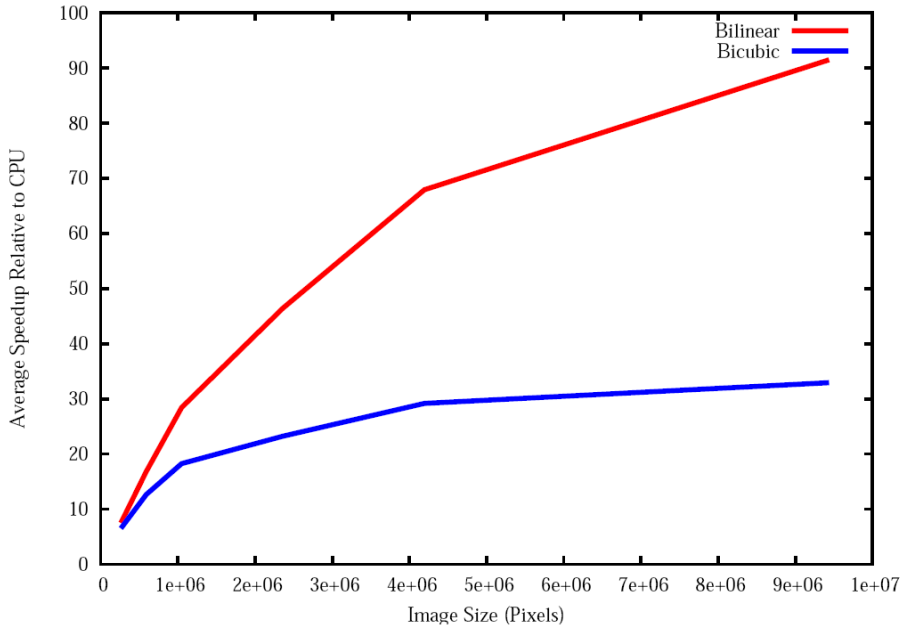
Rigid and Non-Rigid Registration

Rigid and Non-Rigid Registration

- **Full scan deformable registration**
  - Application in lung nodule mapping
  - Sequential lesion comparison
  - Ideally suited for GPU computing
- **Isolated bone matching**
  - Application in segmenting axial skeleton
  - Targeted anatomic analysis (e.g. femoral neck fractures)
  - Suffers from difficulties in creating coalesced memory access

Full Scan Registration on G80 architecture

- **Multiple optimized algorithms**
  - Demons algorithm most common
  - Gaussian recursive filtering
- **Bi-linear and bi-cubic algorithms**
  - Coalesced memory access patterns
  - Ideal for early CUDA hardware.



Performance Analysis of Accelerated Image Registration Using GPGPU

Peter Bui <pbui@cse.nd.edu>
Jay Brockman <jbb@cse.nd.edu>

University of Notre Dame, IN, USA

Workshop on General Purpose Processing on Graphics Processing Units, 2009

System Configuration

- **Hardware:**
  - Intel Quad-Core Q6700 2.66 GHz CPU, 8.0 GB
  - NVIDIA Tesla C870, 128 Stream Processors, 1.0GB
- **Software:**
  - Ubuntu 8.04 (kernel 2.6.20)
  - GCC 4.1.2
  - NVIDIA CUDA 1.1 SDK

Demons algorithm on GPU

- **Memcopy data to texture memory**
- **Run optimization kernel**
  - Assume separable function
  - Compute demons f(Δ(t,x))
  - Operate on each element of volume
- **Identify convergence by MSE differential < threshold**
- **Apply and iterate (Gauss-Newton resolution scheme)**

Accelerating 3D Non-Rigid Registration using Graphics Hardware

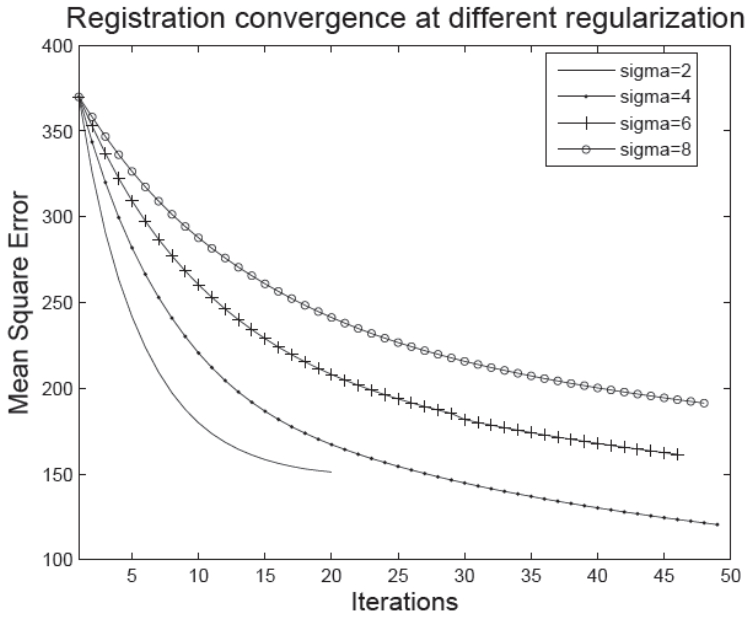
Nicolas Courty[1]
[1]UBS, Laboratoire Valoria
Campus de Tohannic, 56000 Vannes, France
NICOLAS.COURTY@UNIV-UBS.FR

Pierre Hellier[2,3,4]
[2]INRIA, Visages project, Campus de Beaulieu, 35042 Rennes cedex, France
PIERRE.HELLIER@IRISA.FR

[3]INSERM, Visages-U746, Campus de Beaulieu, 35042 Rennes cedex, France

[4]University of Rennes1 - CNRS, Campus de Beaulieu, 35042 Rennes cedex, France

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)



Segmentation of the axial skeleton

- Complicated by variation of anatomy between individuals
- Difficult to coalesce memory access
- Operating based primarily on density which makes initial positioning critical

Manual Segmentation (10min run time)

Optimization Scheme Choices

A

- Alpha-beta pruning
- Artificial Bee Colony Algorithm
- Artificial neural network
- Artificial immune system
- Auction algorithm
- Automatic label placement

B

- BFGS method
- Bees algorithm
- BHHH algorithm
- Big M method
- Bin packing problem
- Biologically inspired algorithms
- Bland's rule
- Boid (computer science)
- Branch and bound
- Branch and cut

C

- CMA-ES
- Compositional pattern-producing network
- Conjugate gradient method
- Cooperative optimization
- Crew scheduling
- Cross-entropy method
- Cuckoo search
- Cutting-plane method

D

- Davidon–Fletcher–Powell formula
- Delayed column generation
- Derivation of the conjugate gradient method
- Destination dispatch
- Differential evolution
- Divide and conquer algorithm
- Dynamic programming

E

- Evolutionary algorithm
- Evolutionary multi-modal optimization

E cont.

- Evolutionary programming
- Expectation-maximization algorithm
- Extremal optimization

F

- Firefly algorithm
- Fourier–Motzkin elimination
- Frank–Wolfe algorithm

G

- Gauss–Newton algorithm
- Genetic algorithm
- Genetic algorithms in economics
- Golden section search
- Gradient descent
- Graduated optimization
- Great Deluge algorithm
- Greedy algorithm
- Group method of data handling
- Guided Local Search

H

- Harmony search
- Hill climbing

I

- IOSO
- Interior point method

J

- Job Shop Scheduling

K

- K-approximation of k-hitting set
- Karmarkar's algorithm

L

- L-BFGS
- Levenberg–Marquardt algorithm
- Local search (optimization)
- Local unimodal sampling

M

- MCS algorithm
- Matrix chain multiplication
- Maximum subarray problem
- Mehrotra predictor-corrector method
- Meta-optimization
- Minimax

N

- Nearest neighbor search
- Negamax
- Nelder–Mead method
- Newton's method
- Newton's method in optimization
- Nonlinear conjugate gradient method

O

- Ordered subset expectation maximization

P

- Particle swarm optimization
- Pattern search (optimization)
- Penalty method
- Powell's method

Q

- Quantum annealing
- Quasi-Newton method

R

- Radial basis function network
- Random optimization
- Random search
- Reactive search optimization
- Rosenbrock methods
- Rprop

S

- SR1 formula
- Sequence-dependent setup
- Sequential Minimal Optimization
- Simplex algorithm
- Simulated annealing
- Stochastic hill climbing
- Successive parabolic interpolation
- Swarm intelligence

T

- Tabu search
- Tree rearrangement
- Types of artificial neural networks

V

- Very large-scale neighborhood search

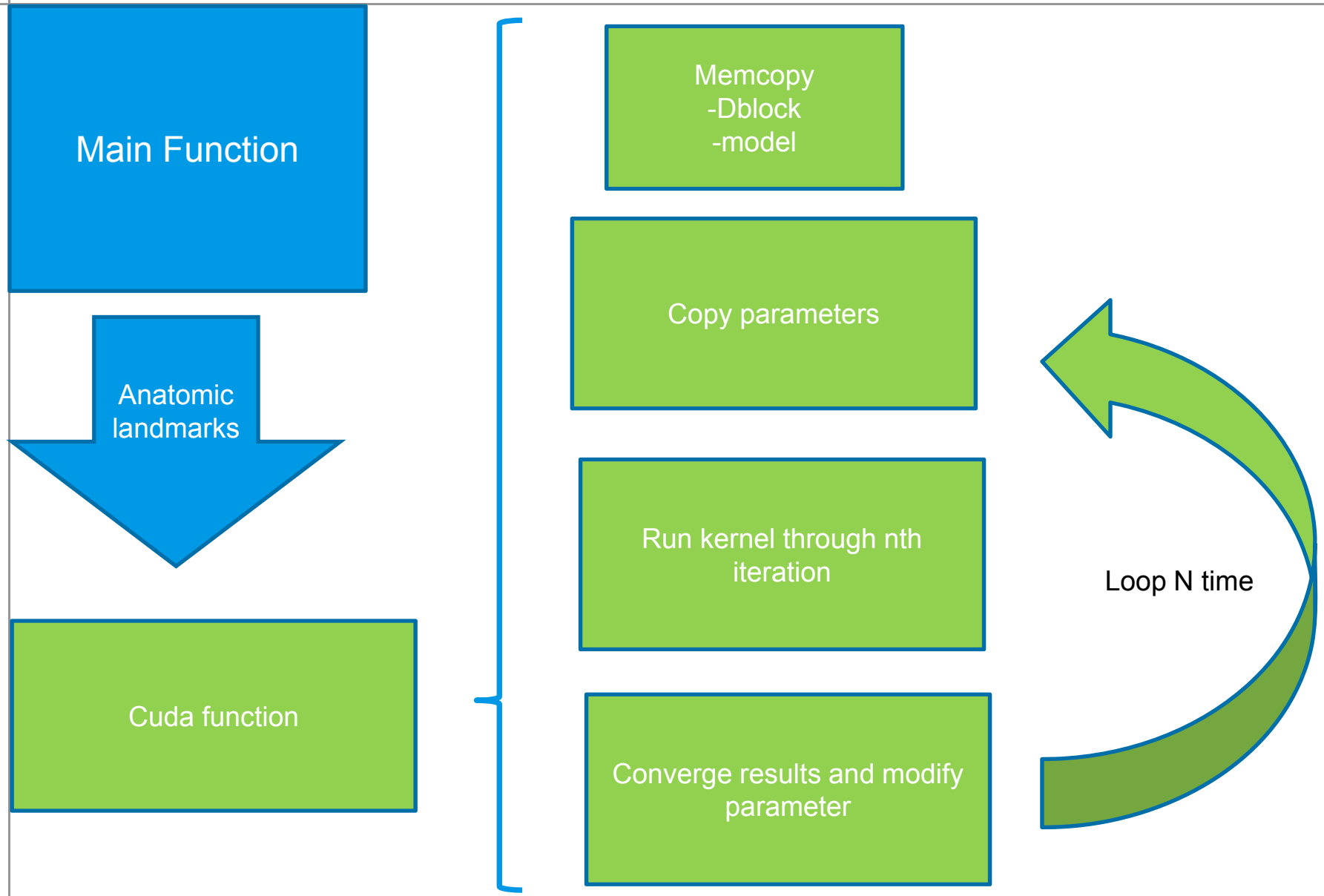
Z

- Zionts–Wallenius method

Guided local search

- **Metaheuristic search method**
  - Iteratively improve match
  - Match model and scan
  - No theoretical guarantee of extrema
- **Penalties to escape unwanted extrema**
- **Calculate initial positioning**
- **Repeat algorithm until match criteria met**

.cu structure



Kernel

```
__global__ void SCALE_MODEL_VARIATIONS(float4 *SMODEL, short2 *DATA, float3 *TRANSLATION, float3 *RSCALE, int2 *RESULT, int *COARSE, int *PTCNT, int *CBCNT)
{
  int tx = threadIdx.x;//512
  int bx = blockIdx.x; //512
  int by = PTCNT[tx];

  int coarse = COARSE[tx];
  int res_index = tx + bx * 512;
  float tpi = 6.2831853071795864, angle_rat=0, scl_rat=0;
  float3 vpos, vrs;
  int4 resi;
  int tmpi;

  vpos.z = truncf(tx/64);
  tmpi = tx - vpos.z * 64;
  vpos.y = truncf(tmpi/8);
  vpos.x = tmpi - vpos.y * 8;
  vrs.z = truncf(bx/64);
  tmpi = bx - vrs.z * 64;
  vrs.y = truncf(tmpi/8);
  vrs.x = tmpi - vrs.y * 8;

  vrs.x -= 4; vrs.y -= 4; vrs.z -= 4;
  vpos.x -= 4; vpos.y -= 4; vpos.z -= 4;

  if(coarse==0){vpos.x *= 16; vpos.y *= 16; vpos.z *= 16; angle_rat = 32; scl_rat=0.04;}
  else if(coarse ==1){vpos.x *= 2; vpos.y *= 2; vpos.z *= 2; angle_rat = 64; scl_rat=0.01;}
  else if(coarse ==2){vpos.x *= 1; vpos.y *= 1; vpos.z *= 1; angle_rat = 256; scl_rat=0.0025;}
  else if(coarse ==3){vpos.x *= 0.5; vpos.y *= 0.5; vpos.z *= 0.5; angle_rat = 512; scl_rat=0.00125;}


  float3 trans = TRANSLATION[tx];
  float3 irscale = RSCALE[tx];
  int cbcnt = CBCNT[tx];
  short2 sdata;
  float2 trigp, trigt;
  float3 tres, res;
  int go_flag=1;
  float tmpf=1;
  short2 cb_range; cb_range.x = 1350; cb_range.y = 3000; if(cbcnt < 1100000){cb_range.x=1200;}

  __shared__ float4 smdlpt;
  if(tx==0){smdlpt = SMODEL[by];}__syncthreads();
```

Indexing

Threads

⇩

Variations of position (x,y,z)

Block

⇩

Variation of rotation and scale

Copy model point to shared memory

Kernel contd

```
if(smdlpt.w > 0)
  {
    trans.x += vpos.x;
    trans.y += vpos.y;
    trans.z += vpos.z;

    irscale.x += (vrs.x * tpi/angle_rat) * 0.5;
    irscale.y += vrs.y * tpi/angle_rat;
    irscale.z += vrs.z * scl_rat;

    trigp.x = cos(irscale.x);
    trigp.y = sin(irscale.x);

    trigt.x = cos(irscale.y);
    trigt.y = sin(irscale.y);


    tres.x = smdlpt.x;
    tres.y = smdlpt.y;
    tres.z = smdlpt.z;

    res.x = tres.x * trigt.x + tres.y * trigt.y;
    res.y = -tres.x * trigt.y * trigp.x + tres.y * trigt.x * trigp.x + tres.z * trigp.y;
    res.z = tres.x * trigt.y * trigp.y - tres.y * trigt.x * trigp.y + tres.z * trigp.x;

    res.x *= irscale.z;
    res.y *= irscale.z;
    res.z *= irscale.z;


    tmpf = irscale.z;

    res.x += trans.x;
    res.y += trans.y;
    res.z += trans.z;

    if(res.x < 0 || res.x >255){go_flag=0;}
    if(res.y < 0 || res.y >255){go_flag=0;}
    if(res.z < 0 || res.z >255){go_flag=0;}

    resi.x = float2int(res.x);
    resi.y = float2int(res.y);
    resi.z = float2int(res.z);

    if(go_flag==1)
  {
    resi.w = resi.x + resi.y * 256 + resi.z * 256 * 256;
    sdata = DATA[resi.w];

    if(sdata.y==0 && sdata.x > cb_range.x && sdata.x < cb_range.y){sdata.y = 1;}
    else if(sdata.y==0){smdlpt.w = -1;}
    else if(sdata.y==-1){smdlpt.w = 0;}

    RESULT[res_index].y += sdata.y * smdlpt.w * tmpf;
  }
    }
}
```

Apply variation to model point

Not interpolated or texture value

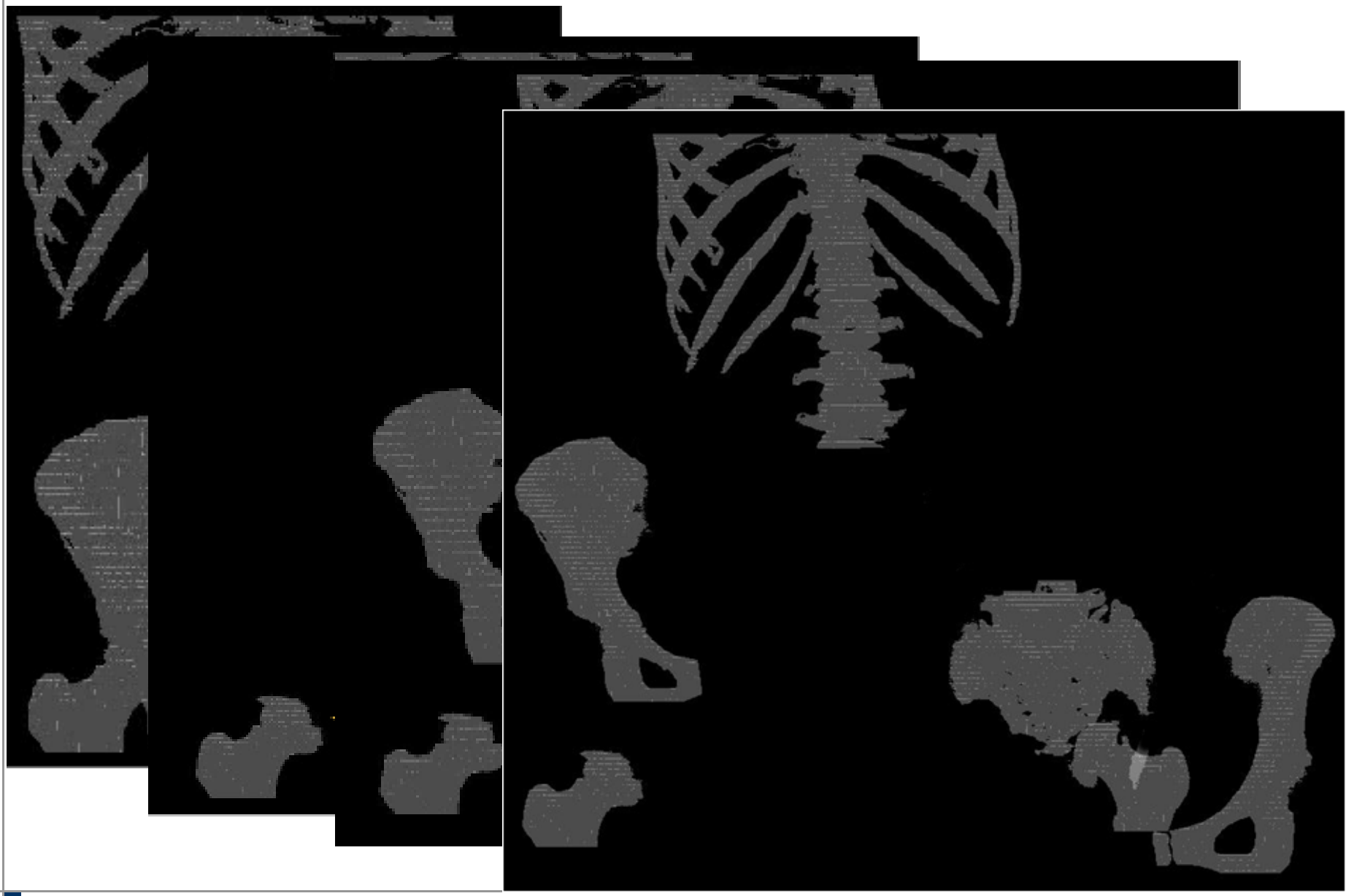
Get Voxel Value

Apply conditions

Write result

Hybrid view of axial skeleton

Full Pelvic Segmentation



Known Issues

- **Determinants of success**
  - global extrema within Δ of the initial guess
  - Finite sampling of a quasi continuous parametric space
- **Bone specific parameters**
  - Humeral head
    - Spherical weighting for large radius
    - Penalties for Iiac components
  - Humeral shaft
    - Weighting towards point near trochanters
    - Allow metallic points
  - Iliac crest
    - Weighting towards point matching near iliac spine
    - Penalties for unmatched point

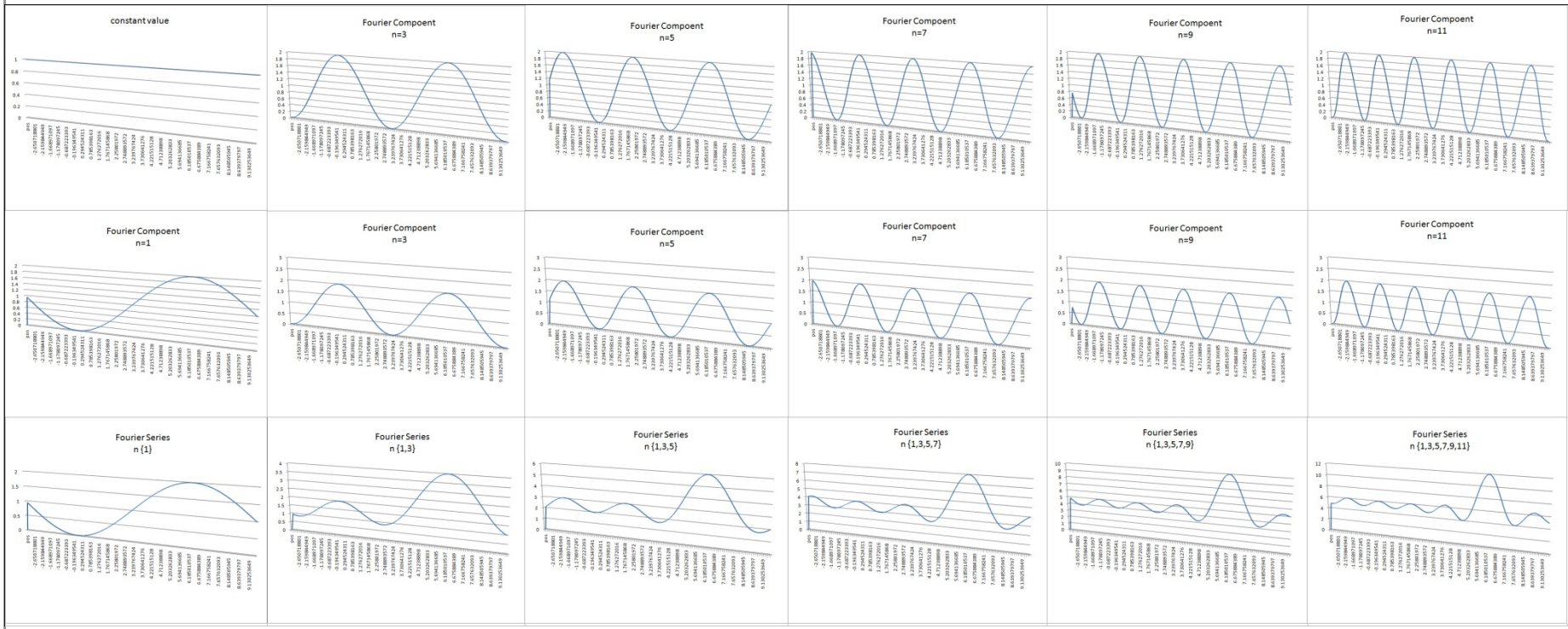
How many computers can you fit in a 1B1B?

Overview

I. Modalities In Medical Imaging
II. Isotropic Voxels and Volumetric Imaging
III. Utilizing CUDA for Image Analysis
IV. Outstanding Challenges in Medical Imaging
    A. Quantum Mottle
    B. Unexpected Dense Objects
    C. Vessel Discontinuity
VI. Future of GPU Computing in Informatics

Fundamental Issues

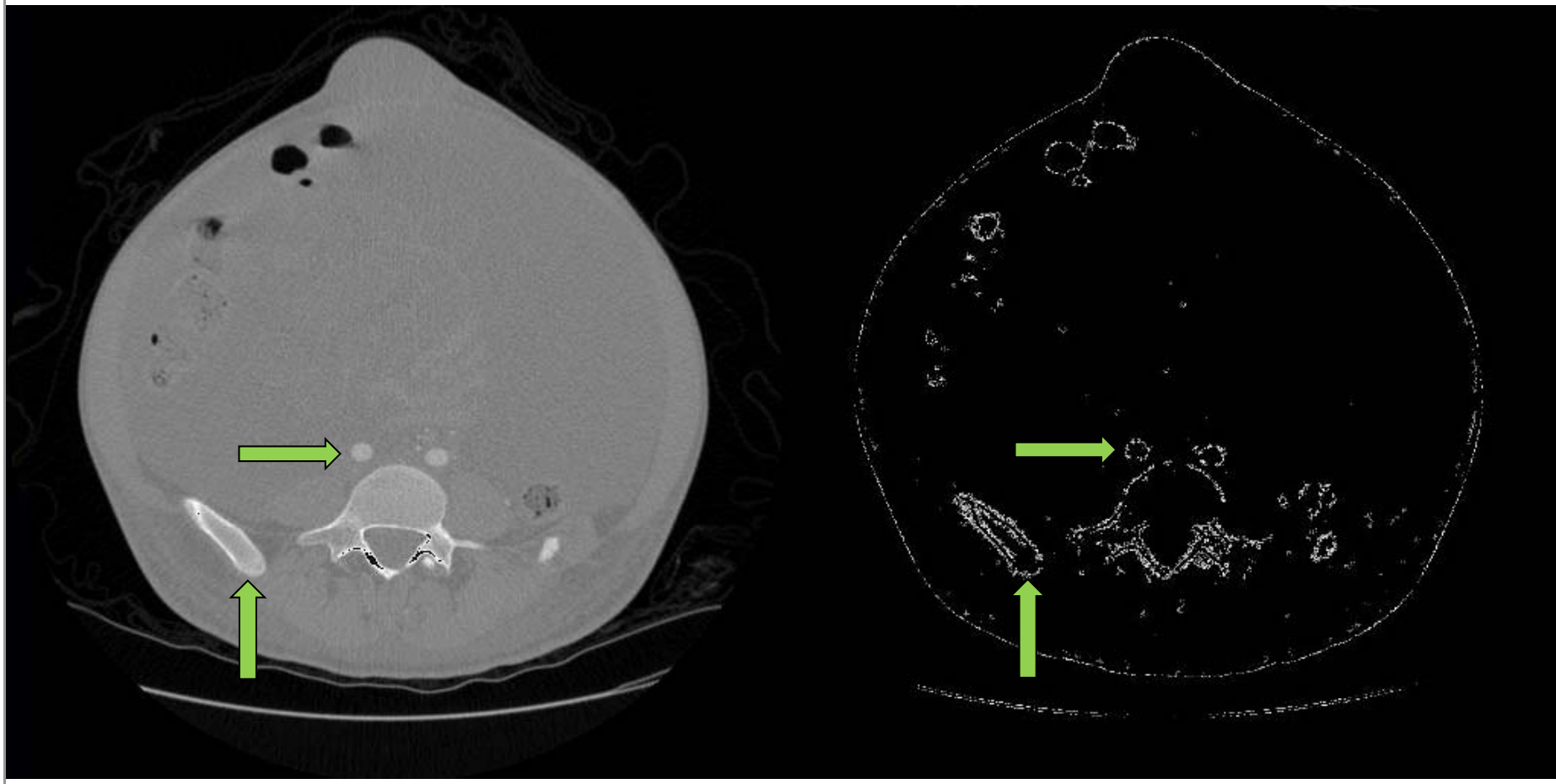
Quantum Mottle and the Issue of Lost Edges



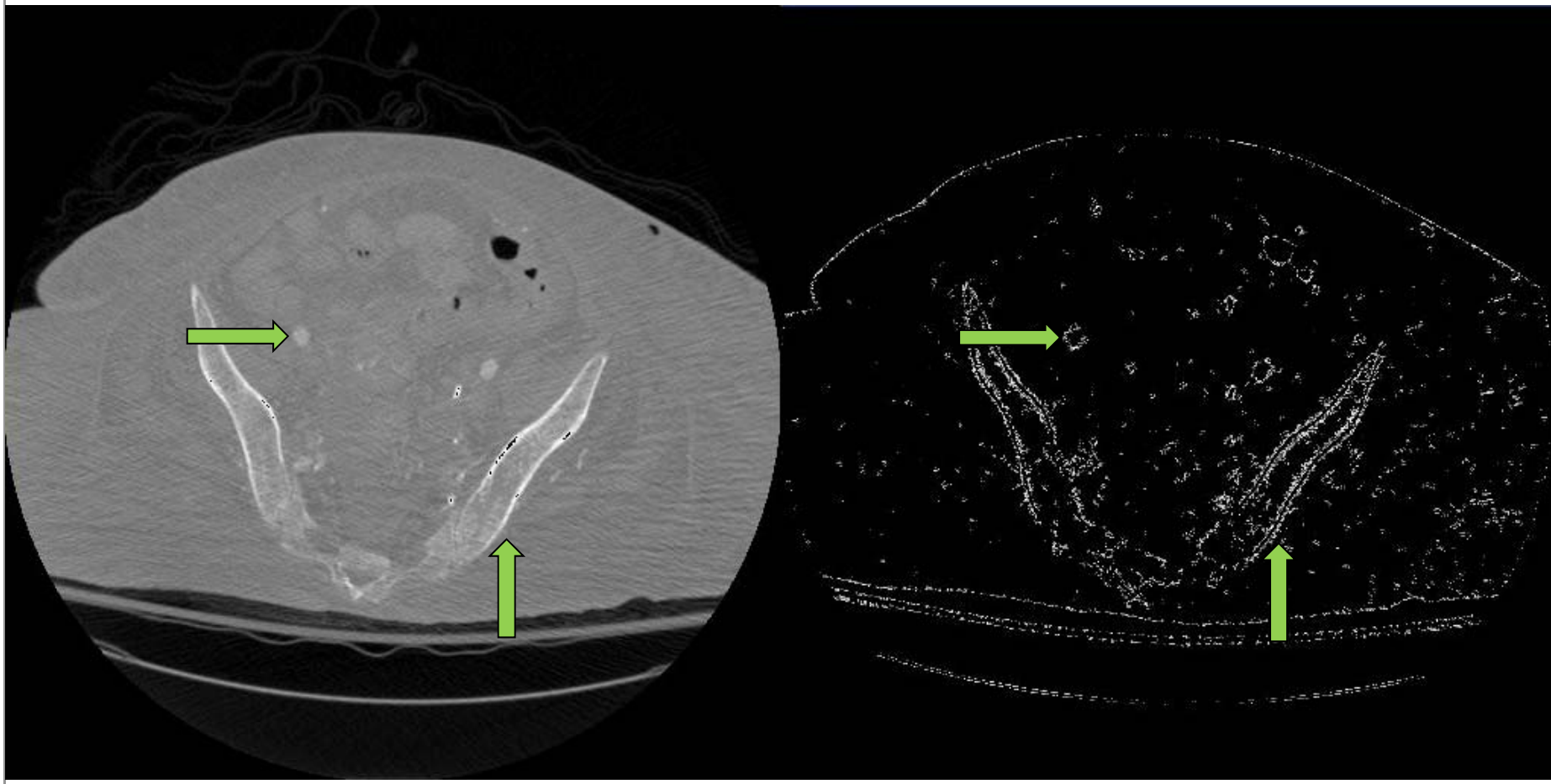
Quantum Mottle and Lost Edges

- Generalized body wall edema
- Pleural Effusions
- Poor contrast enhancement
- Obesity

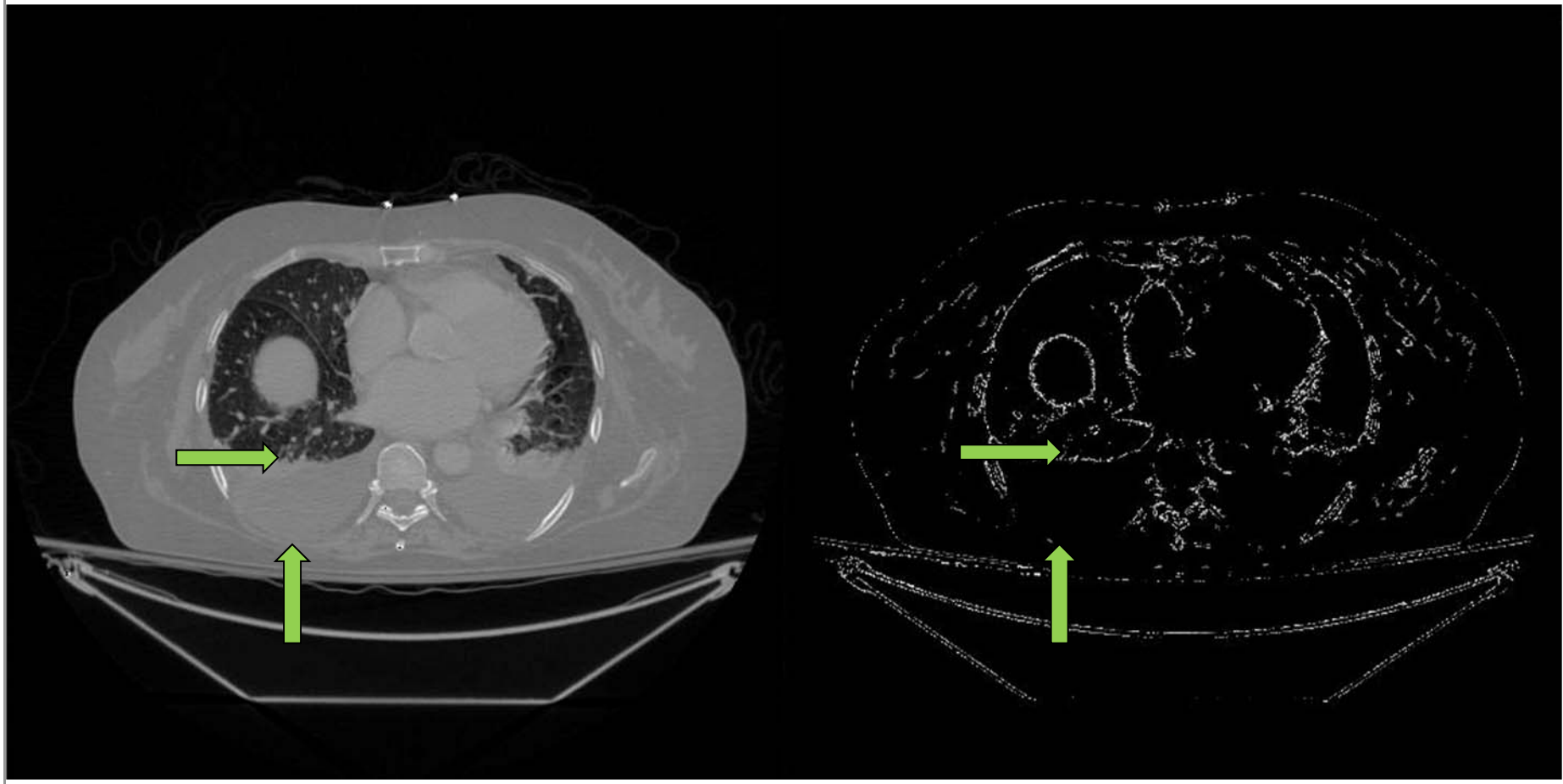
Generalized Edema



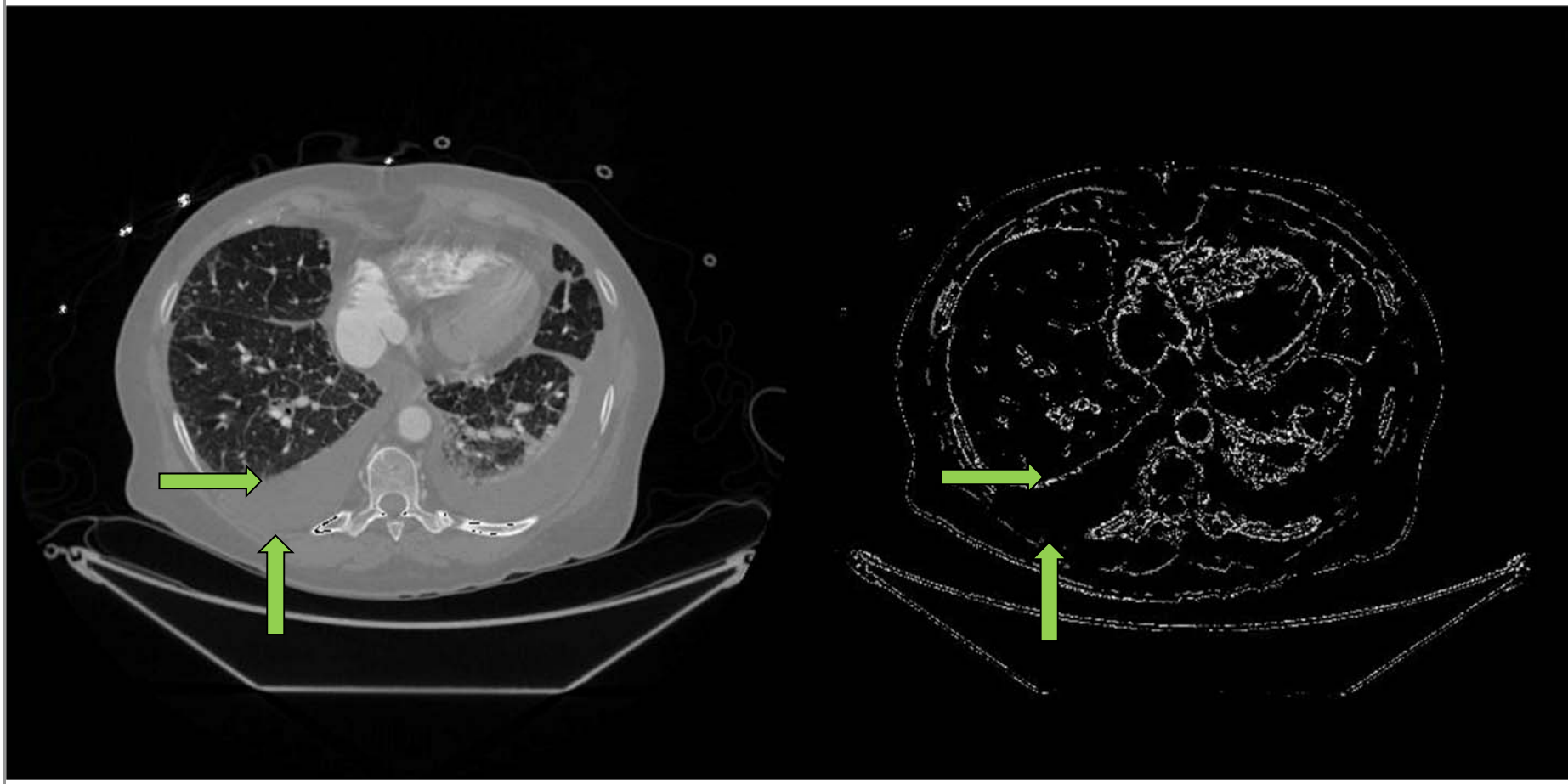
Generalized Edema



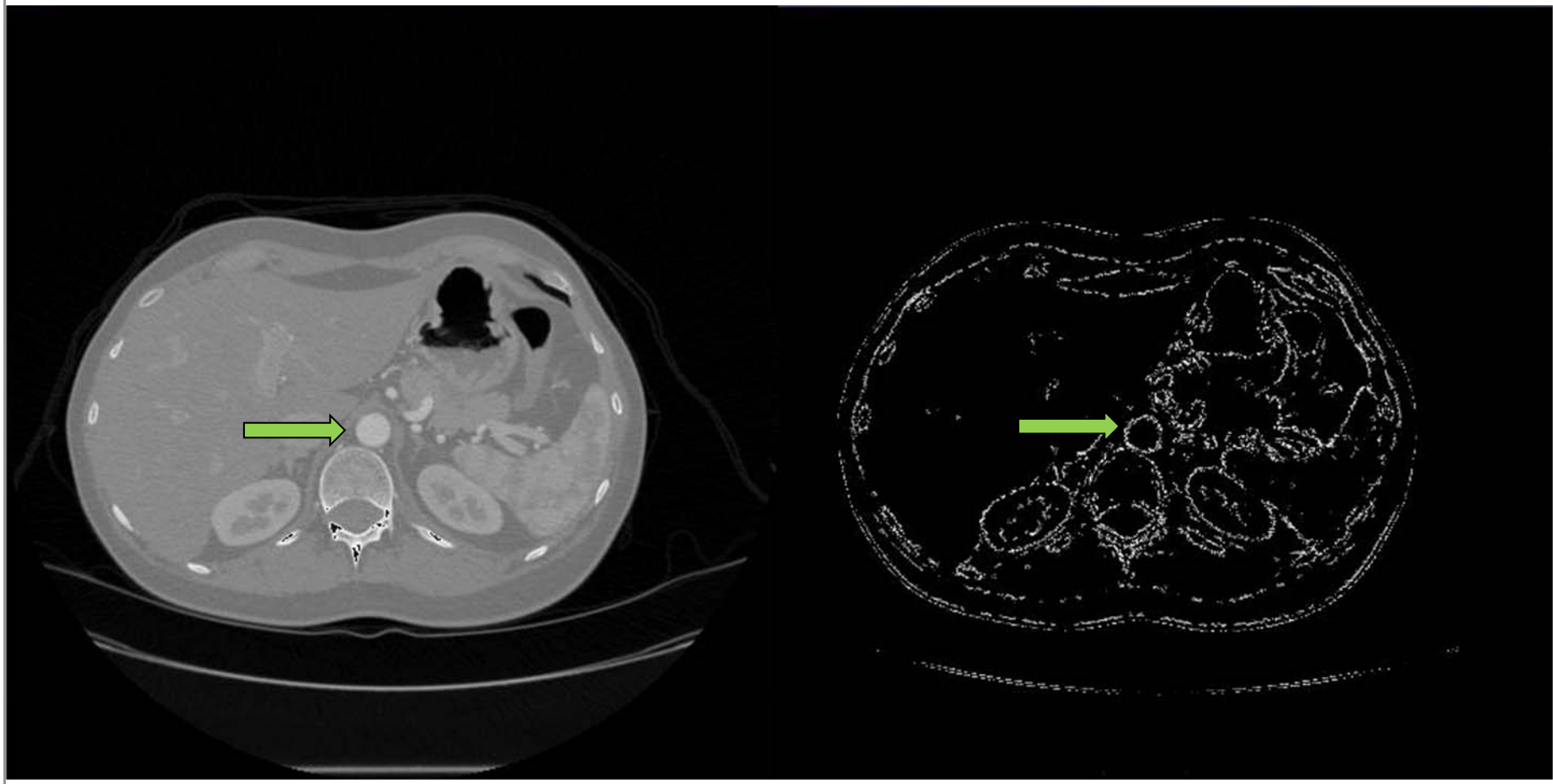
Pleural Effusions



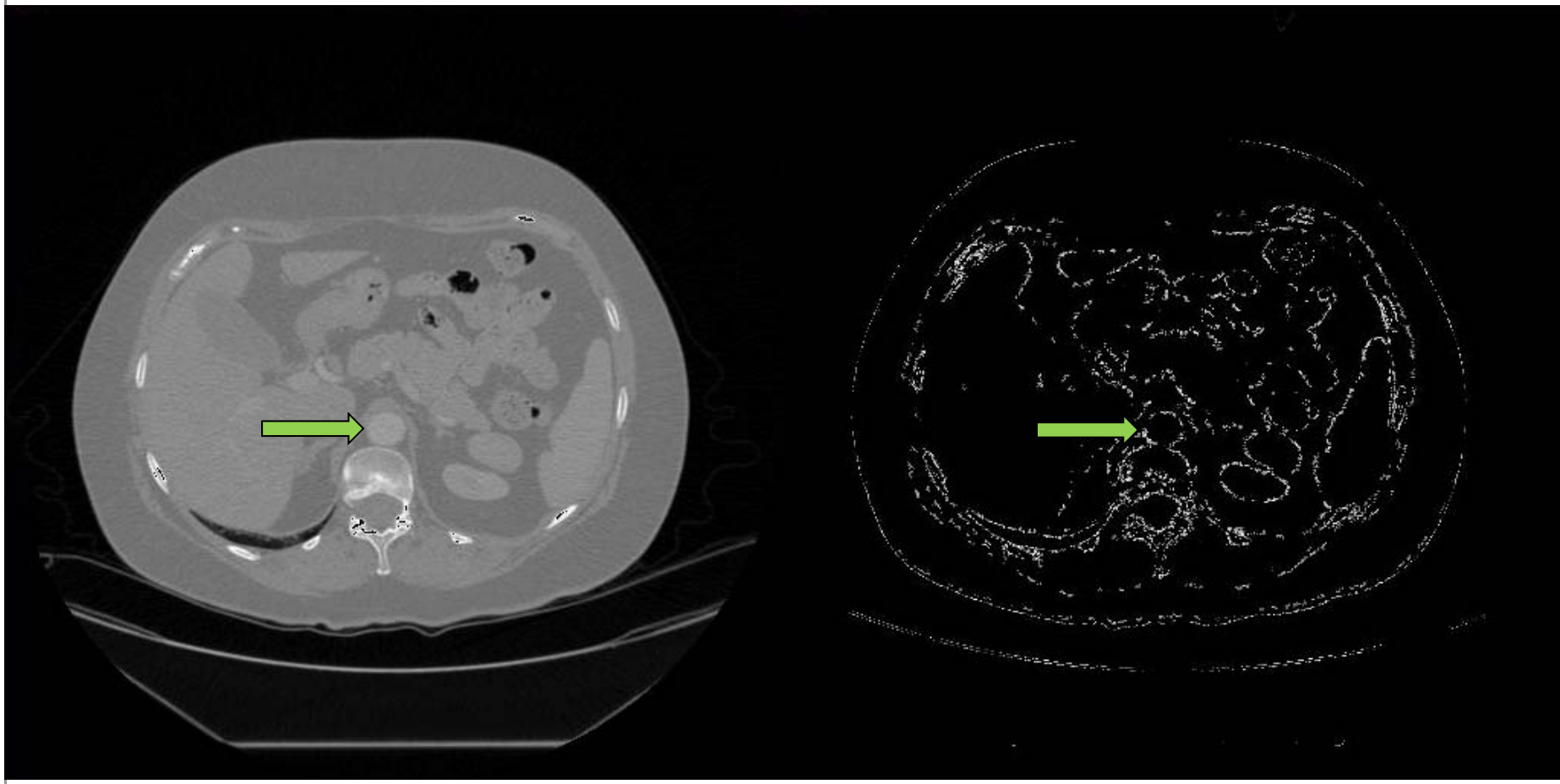
Pleural Effusions



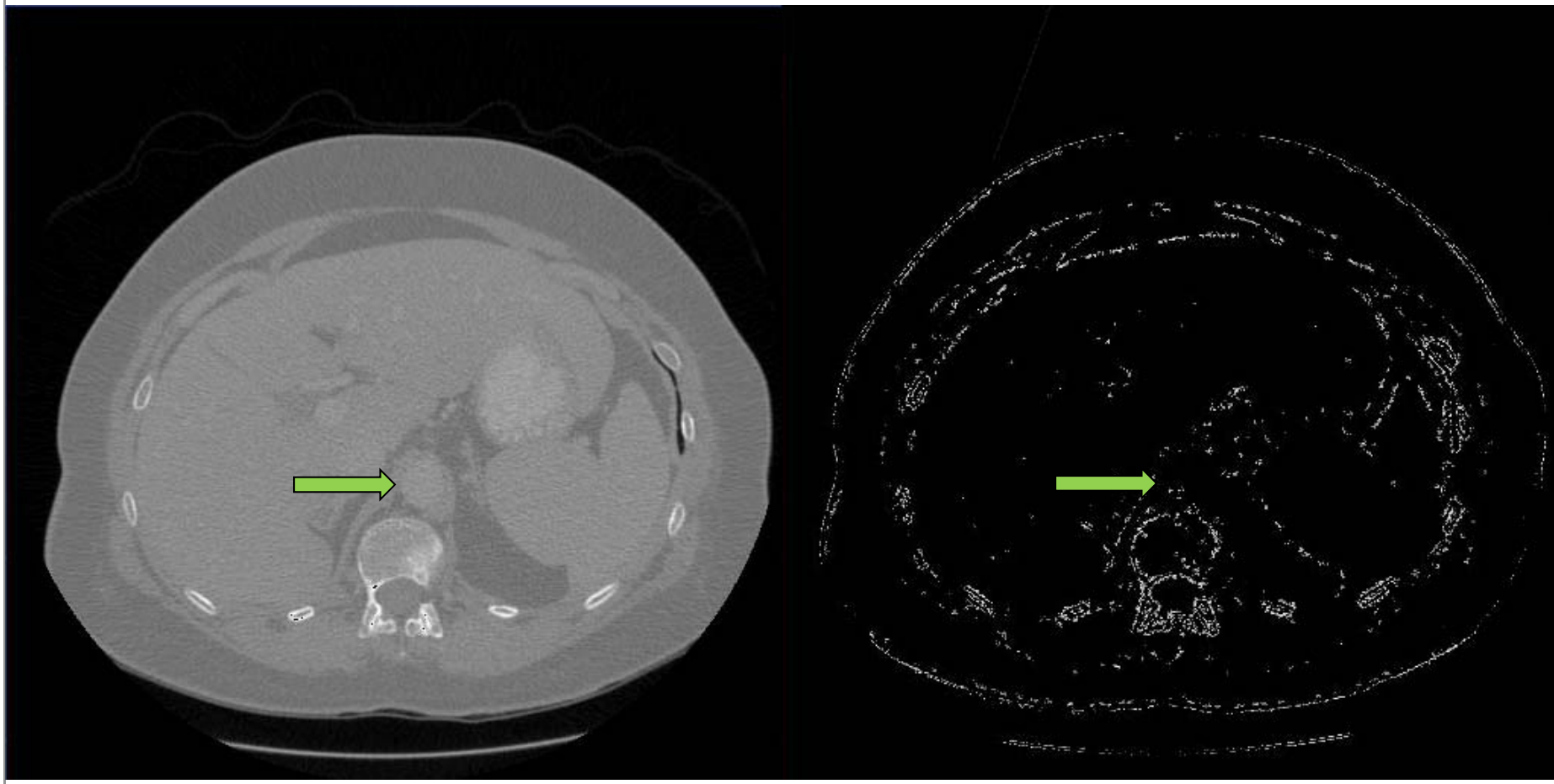
Poor Contrast Enhancement



Poor Contrast Enhancement

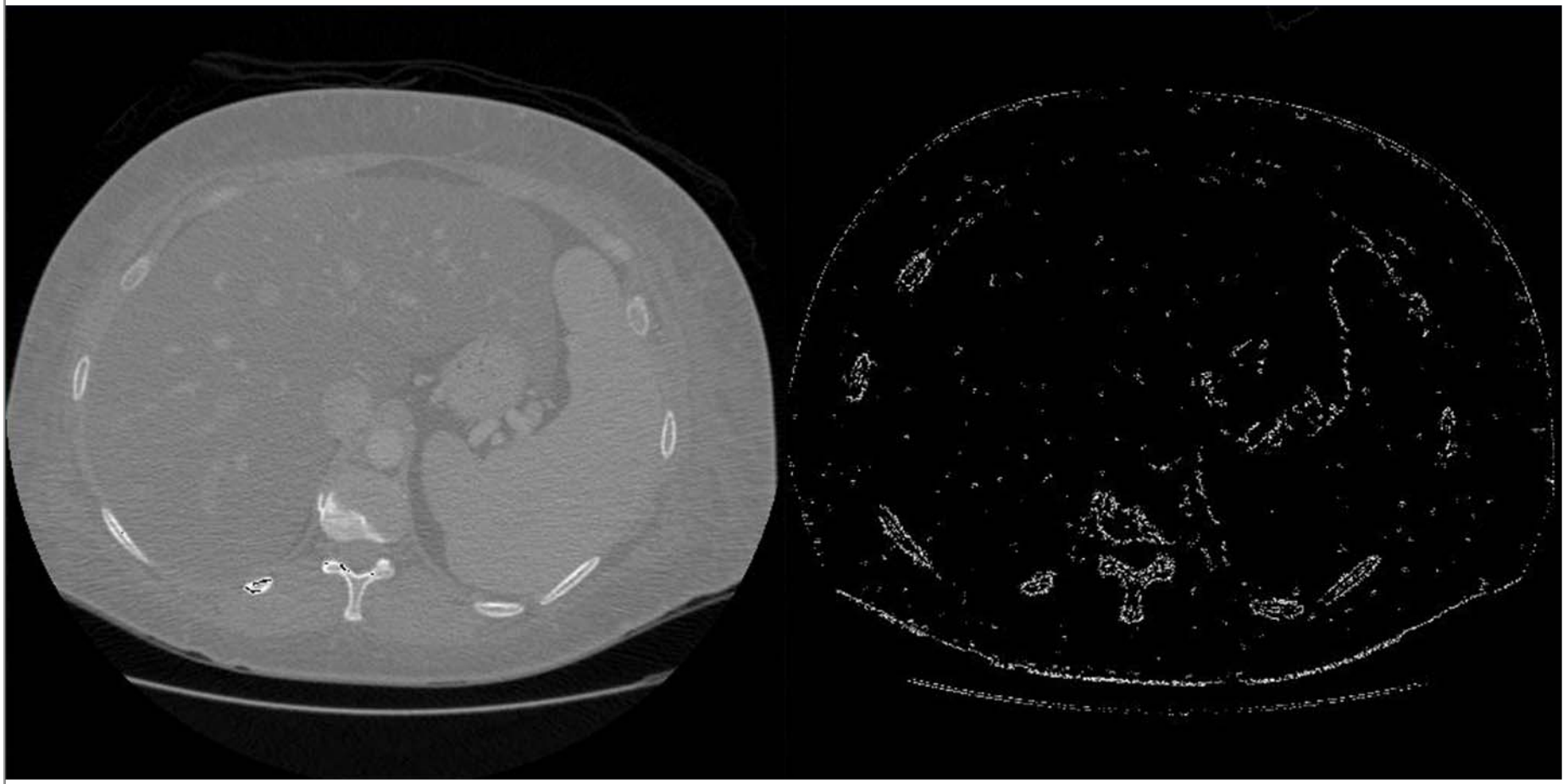


Poor Contrast Enhancement

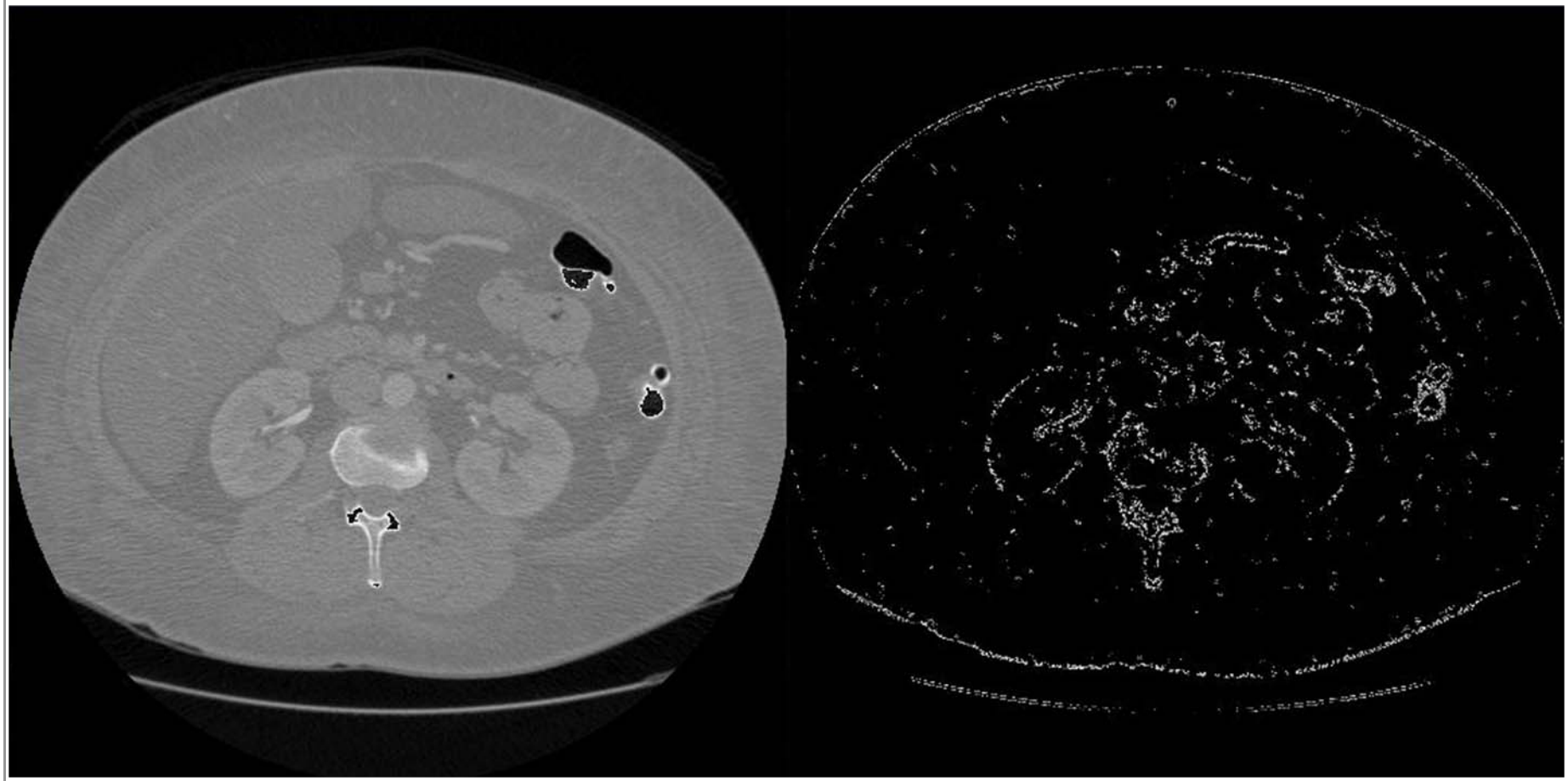


Quantum Mottle and the issue of Lost Edges

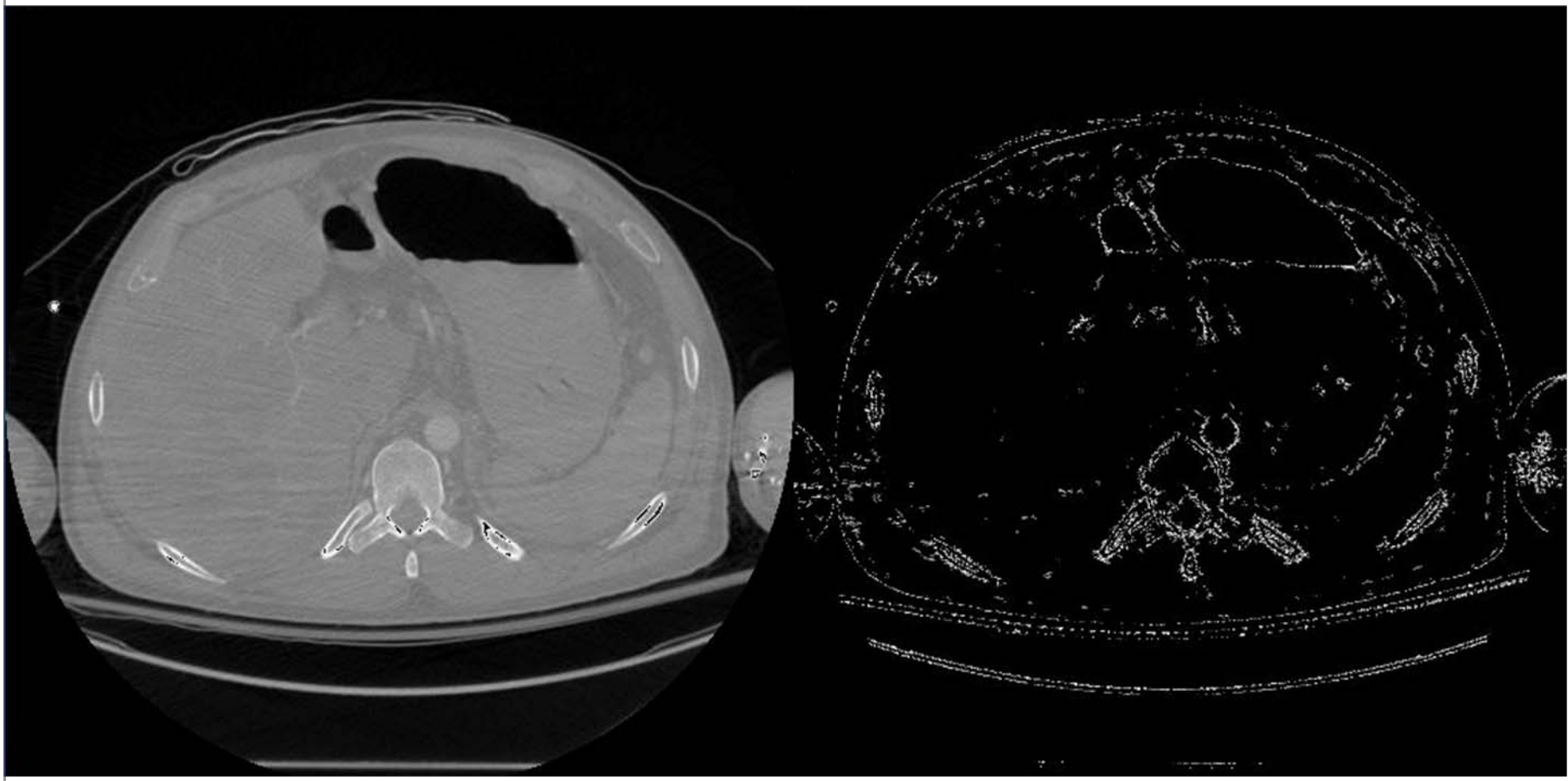
Obesity



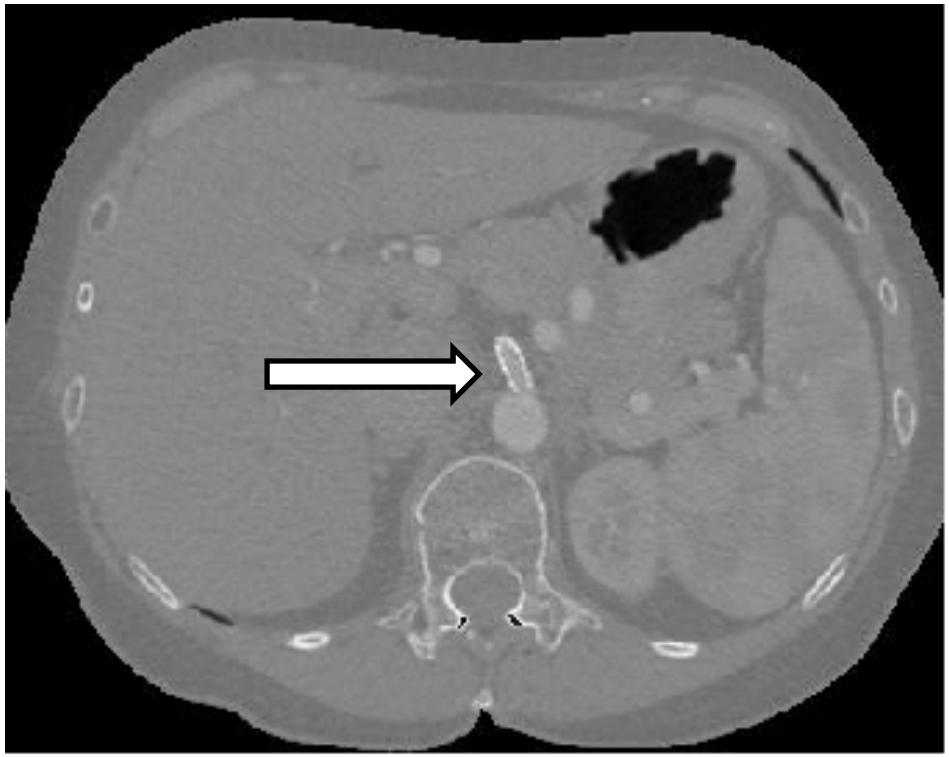
Obesity



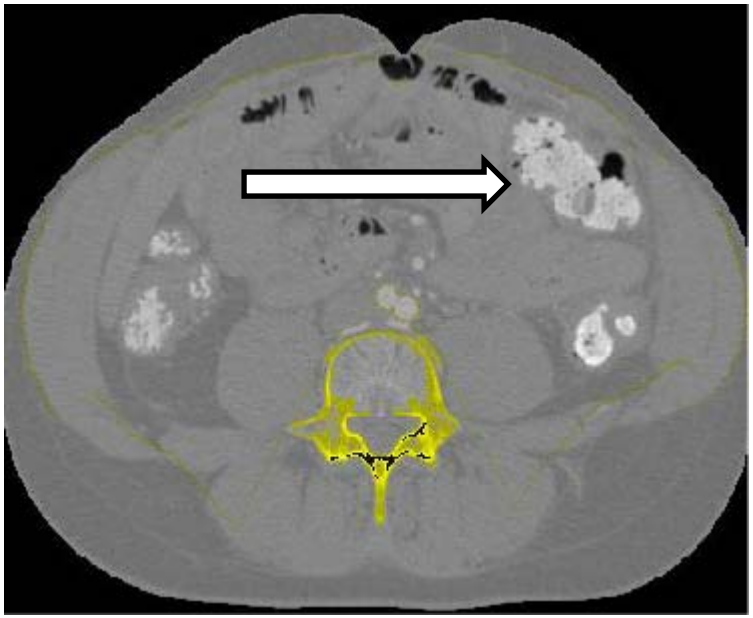
Obesity

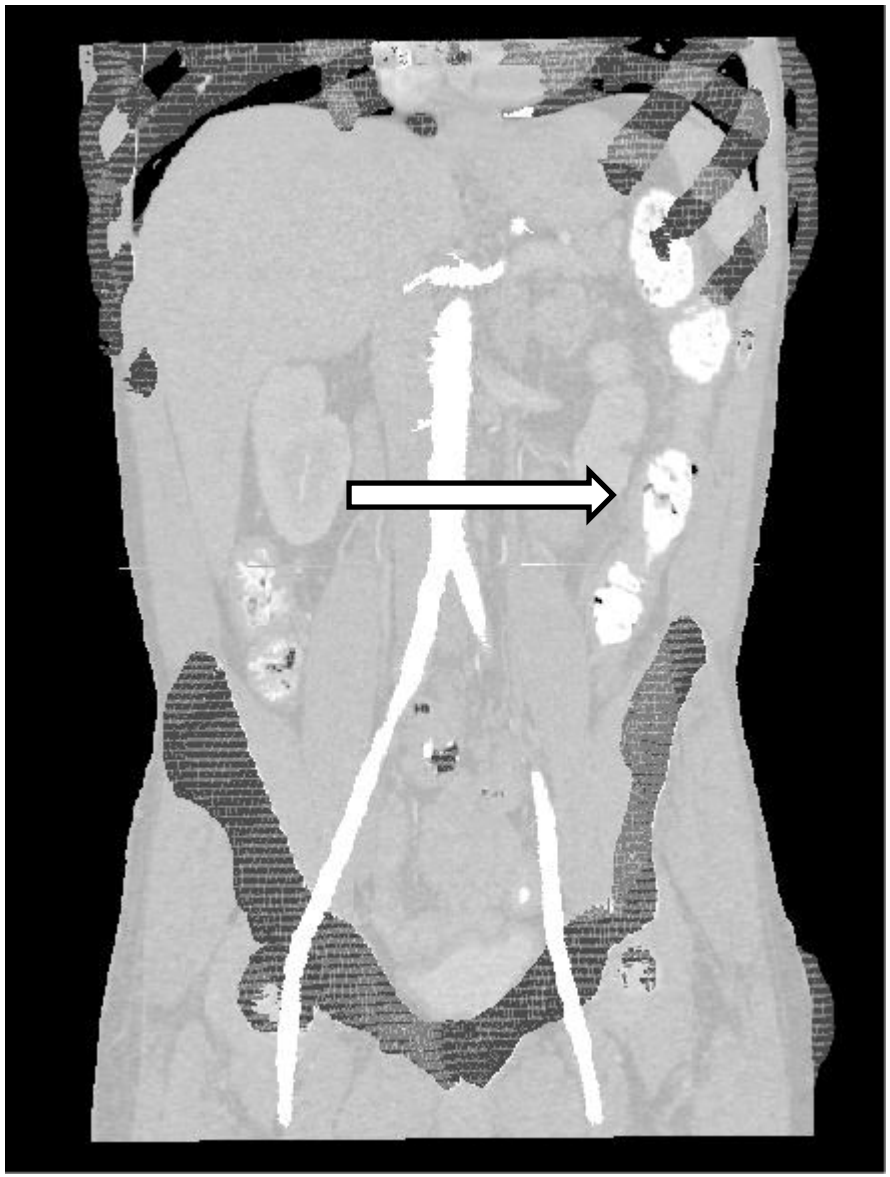


Unexpected Dense Objects

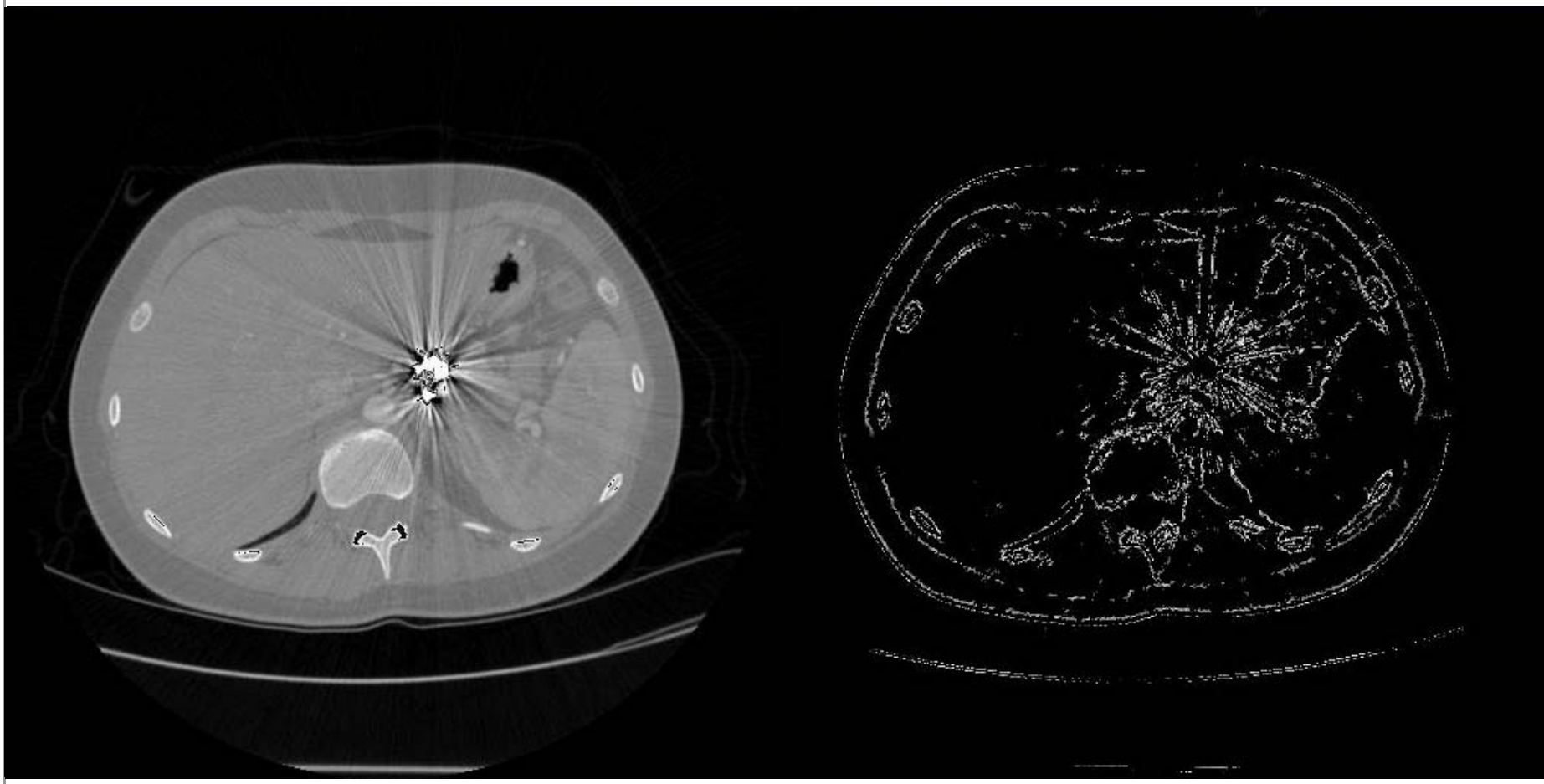


Enteric Contrast

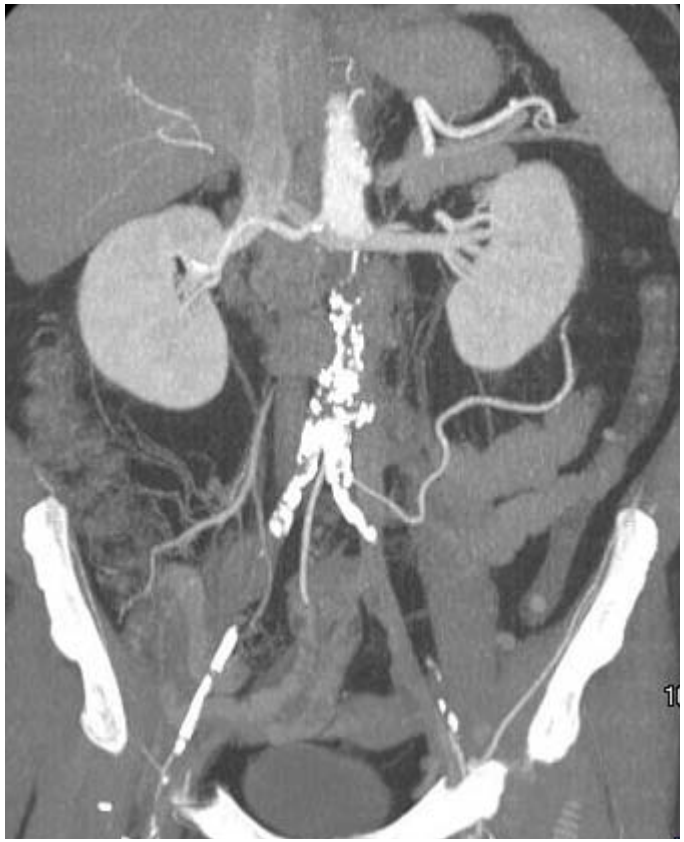




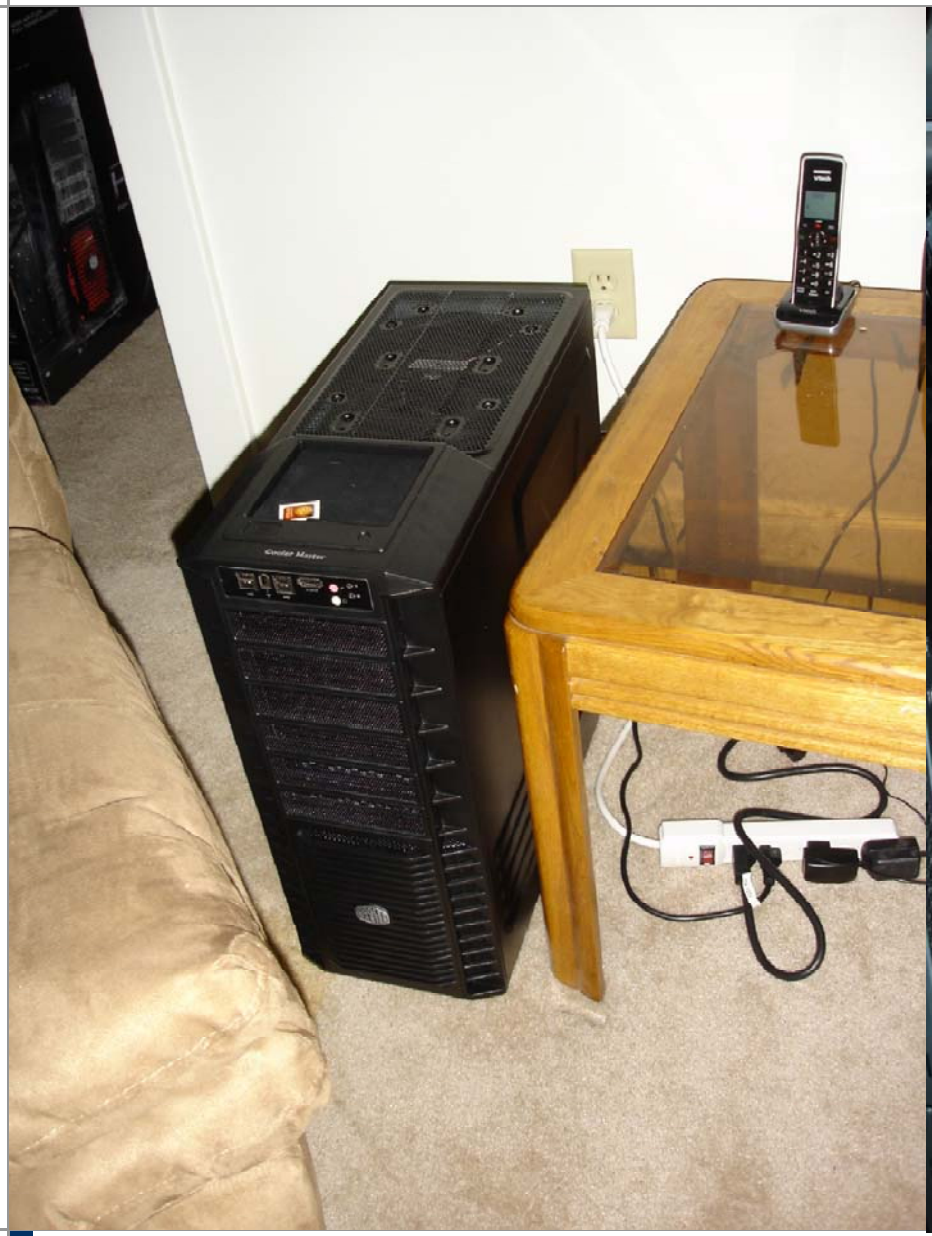
Implanted Metal



Vascular Discontinuity



Running out of space

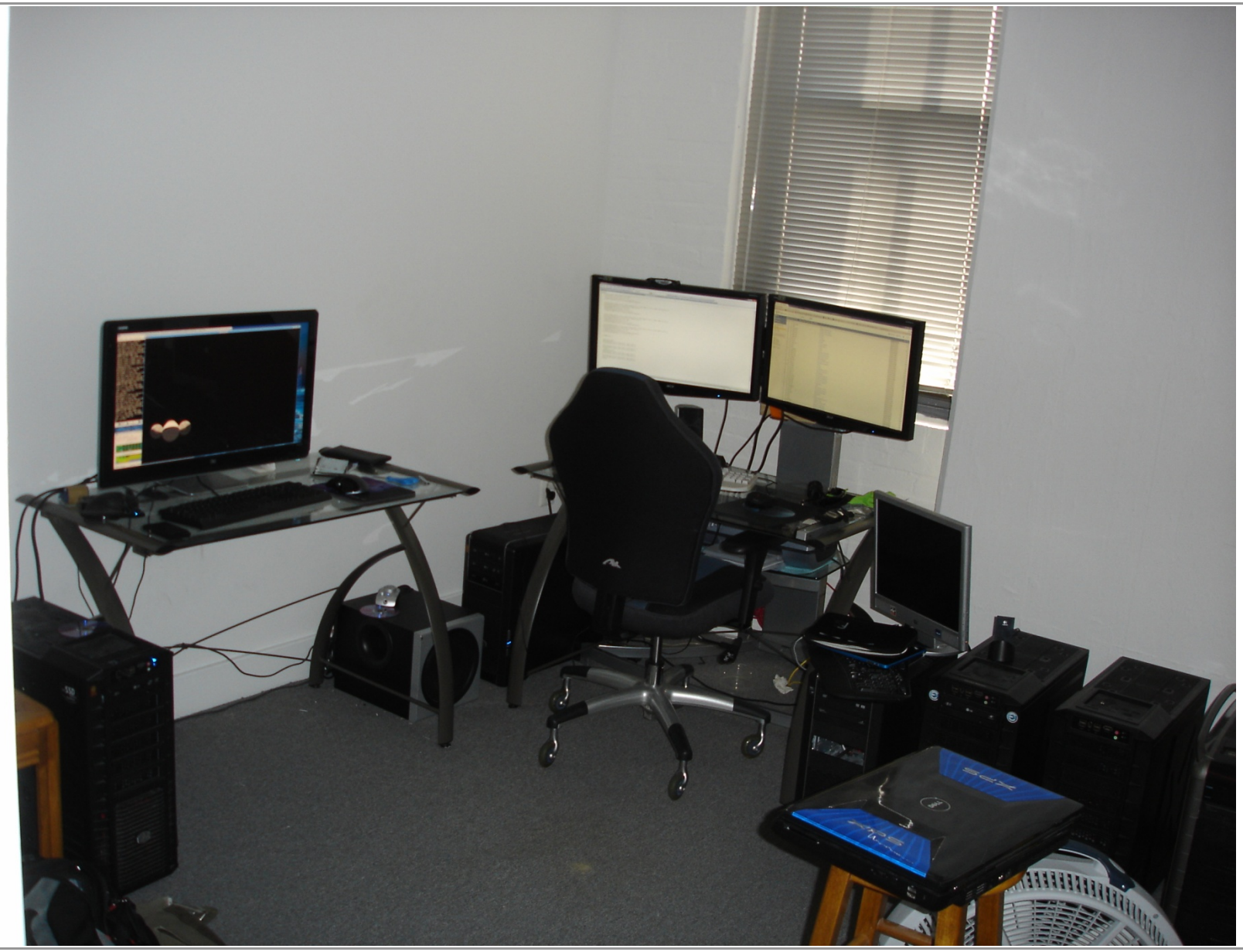


Overview

I. Modalities In Medical Imaging
II. Isotropic Voxels and Volumetric Imaging
III. Utilizing CUDA for Image Analysis
IV. Outstanding Challenges in Medical Imaging
VI. Future of GPU Computing in Informatics

- **Real-time vessel and lesion characterization**
  - Link segmented data to allow organ specific lesion characterization
  - Registration with prior exams/segmentation to refine analysis
  - Incorporate multiple modalities (MRI, CT)
  - Interactive vascular maps for surgical planning
- **Preliminary/Quantitative report generation**
  - Automatically provide lesion characterization addendum to reports
  - Analyze segmented data for common pathologies
- **Hardware**
  - Memory access patterns do not lend themselves to the CUDA architecture
  - Integration of a RISC/x86 core which has direct access to Global Memory
  - Increase thread block size and shared memory

Heating Solution for Philadelphia Winters



Acknowledgements

- Dr. Litt, Dr. Boonn, and the Department of Radiology at University of Pennsylvania for supporting my work.
- Dr. Knopp and the Department of Radiology at The Ohio State University Medical Center for their continued support and collaboration.
- The 3D labs at UPenn and OSU for their help and guidance
- NVIDIA for providing a forum for the exchange of ideas and for creating a GPU computing paradigm worthy of its own conference.

?? QUESTIONS ??