

Computer Graphics and Scientific Computing

Computer Science Day 2011

June 15, 2011

Thomas Sangild Sørensen, Associate Professor: Medical Imaging,
Real-Time MRI Reconstruction, Surgical Simulation.

Lau Brix, Ph.D. Student: Real-Time MRI Reconstruction

Christian P. V. Christoffersen, Ph.D. Student: Reconstructing 4D
Tomography Imaging

Allan Rasmusson, Ph.D. Student: Quantitative Tissue Analysis,
Computational Stereology

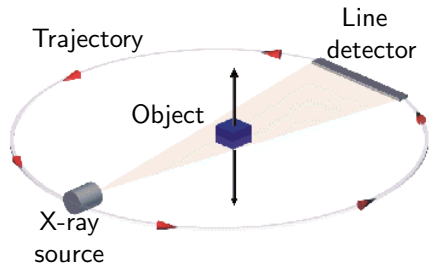
[Unknown] Fall2011, Associate Professor: Computer Graphics

Courses 2011:

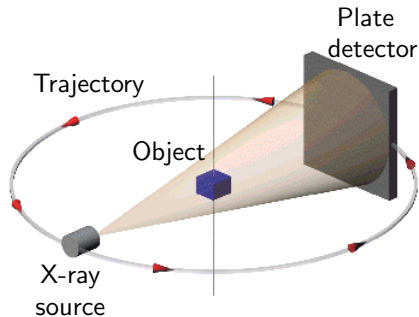
- ▶ Introduction to computer graphics and image processing (Q1)
- ▶ Data-parallel computing (Q1)
- ▶ Advanced image processing (Q2)
- ▶ Advanced real-time graphics effects (Q2)

CT Scanning Project

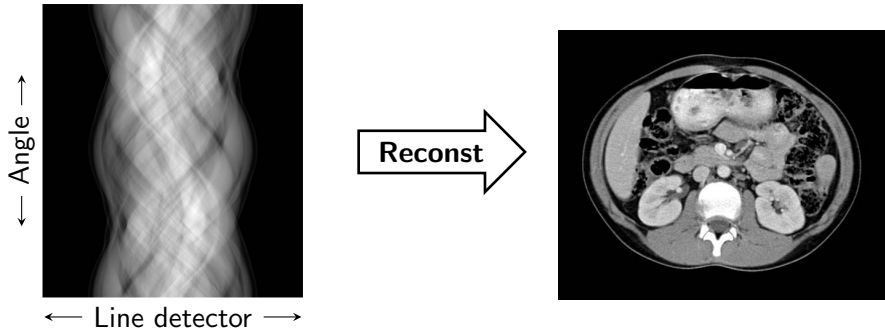
CT Scanning:



Cone-Beam CT:



- ▶ Line detector records row of intensities for every angle
- ▶ From this, an image of the object can be reconstructed

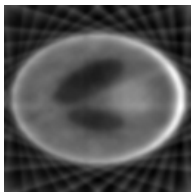


- ▶ Similar for the plate detector.

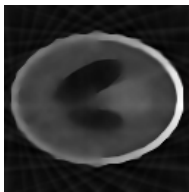
- ▶ Radiation is not good for you
 - ▶ The radiation from survey image is included in dose for tumor.
 - ▶ Also affects healthy tissue.
- ▶ Decrease radiation by using fewer samples



(a) Phantom



(b) 1 iteration



(c) 10 iterations



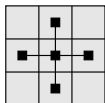
(d) 60 iterations

Connected Component Labeling on the GPU

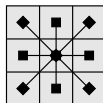
The problem:

- ▶ Given graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consisting of a set \mathcal{V} of n vertices, and a set \mathcal{E} of m undirected edges between vertices,
- ▶ A *connected component* is a sub-graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ of \mathcal{G} , $\mathcal{V}' \subseteq \mathcal{V}$ and $\mathcal{E}' \subseteq \mathcal{E}$ and all pairs $v'_i, v'_j \in \mathcal{V}'$ have a path $p_{ij} = \text{path}(v'_i, v'_j)$ that run along vertices in \mathcal{V}' only.
 $\mathcal{G} = \bigcup_s \mathcal{G}'_s$ and $\mathcal{G}'_s \cap \mathcal{G}'_t = \emptyset, s \neq t.$
- ▶ The connected components labeling is the problem of identifying *all* connected components *and* assigning all nodes within each component a label which must be unique to the component.
- ▶ We want to solve it on the GPU.

- ▶ Restrict vertices to be regular grid of pixels
- ▶ Edges defined *implicitly* using a connectivity scheme Γ :

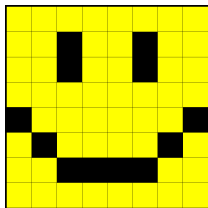


(a) 4-con.

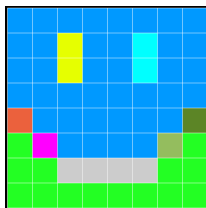


(b) 8-con.

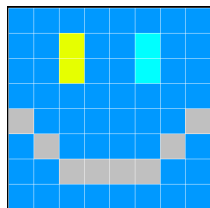
- ▶ Choice of Γ matters:



(c) Input



(d) CCL by 4-con



(e) CCL by 8-con

General idea:

- ▶ Initialize pixels with unique labels
- ▶ Rearrange labels until components are uniquely labeled.
 - ▶ Labels inside components can be equated using Union/Find, but this fits the GPU badly.
- ▶ Instead, using label propagation:

```
BasicPropagation(Image, T, F,  $\Gamma$ ){  
  Initialization(Image, ...)  
  while(!T){  
     $\forall$  vertices  $v$  in Image do in parallel{  
       $\forall$  neighbors  $v_n$  in  $\Gamma(v)$  do{  
         $v.label = F(v, v_n)$   
      }  
    }  
  }
```

T = termination criterion, Γ = Connectivity scheme and function F to perform on neighboring vertices.

For the parallel CCL on the GPU:

Initialization: Calculate unique label using lexicographical ordering of pixel coords: $v.l = v.x \ll 16 \mid v.y$;

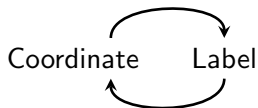
Function: $F(v, v_n)$ {
 if (v, v_n are connected)
 return $\max(v.l, v_n.l)$
 else
 return $v.l$
}

Connectivity Γ : 4-con or 8-con.

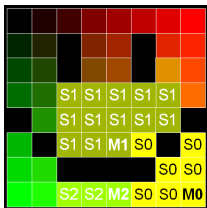
Termination: No label has changed.

- ▶ Propagates maximum label throughout connected components.
- ▶ Propagated only between neighbors, potentially slow.

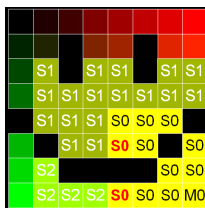
Calculation of labels allows mapping back to pixel coordinates:



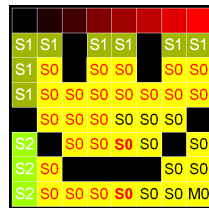
- ▶ If a pixel has its original label it's a *master*
- ▶ - otherwise it's a *slave*.
- ▶ Slaves can include label of master in propagation.



(a) Iteration i

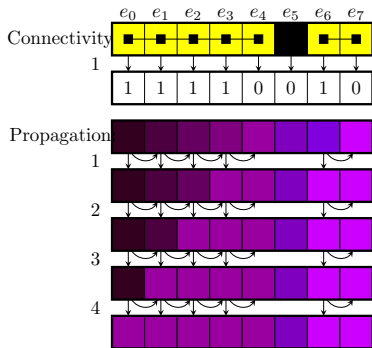


(b) Iteration $i + 1$

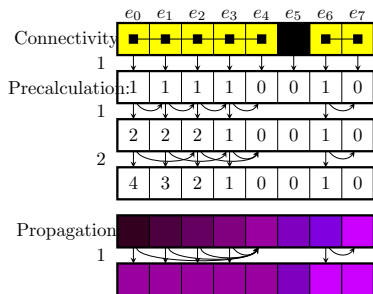


(c) Including Master label

- ▶ Storing connectivity explicitly lowers computations
 - ▶ also allows for calculations on gather sizes.
- ▶ Done for all directions, example for right direction:



(a) Gather size 1: Total 5 iterations

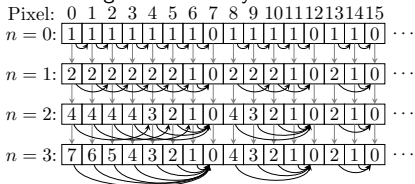


(b) Precalc: Total 4 iterations

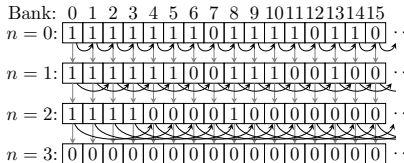
- ▶ Precalc done in log number of steps.
- ▶ Still need to do gathering of 1 size.

- ▶ In practice, explicit storage is *not* an option.
- ▶ Need to move to on-chip gpu memory, but must resolve address conflicts for this.

Addressing same memory:



No address conflicts:



Calculation on integer lengths:

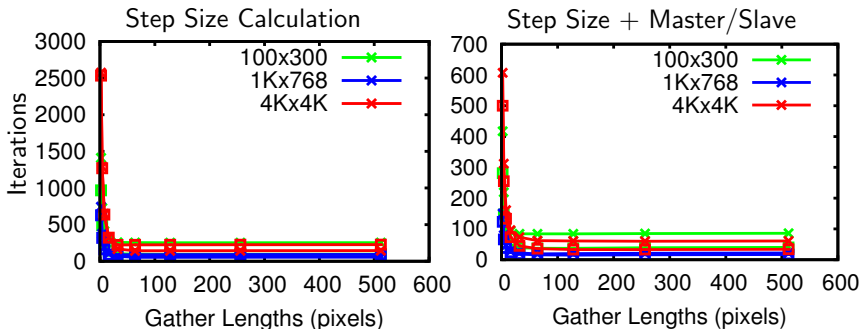
$$\nabla_d^{n+1}(v_i) = \begin{cases} 0, & n = 0, v_i, v_{i+1} \text{ !neighbors} \\ 1, & n = 0, v_i, v_{i+1} \text{ neighbors} \\ \nabla_d^n(v_i) + \nabla_d^n(v_{i+\nabla_d^n(v_i)}) \end{cases}$$

- ▶ Connectivity stored in a few bits in the labels
- ▶ Calculation of step lengths done in every iteration of label propagation

Calculations on *booleans*:

$$\Lambda_d^{n+1}(v_i) = \begin{cases} t, & n = 0, v_i, v_{i+1} \text{ neighbors} \\ f, & n = 0, v_i, v_{i+1} \text{ !neighbors} \\ \Lambda_d^n(v_i) \wedge \Lambda_d^n(v_{i+2^n}) \end{cases}$$

Number of iterations for various image sizes



Other properties calculated using label propagation

- ▶ Voronoi diagrams by using storing coordinates of closest Voronoi source and propagating label with distance to pixel.

Initialization:

$$v.l = \begin{cases} \infty & \text{if pixel not Voronoi source} \\ \text{encode}(v.\text{coords}) & \text{if Voronoi source;} \end{cases}$$

F : Minimum over distances:

```

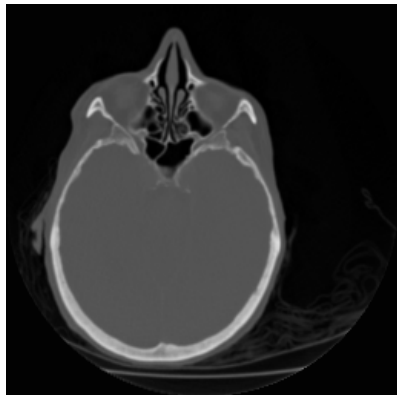
F(v, v_n){
  vsource.coord = decode(v.l);
  v_nsource.coord = decode(v_n.l);
  return label for which
      min( dist(v.coord, vsource.coord),
           dist(v.coord, v_nsource.coord) );
}

```

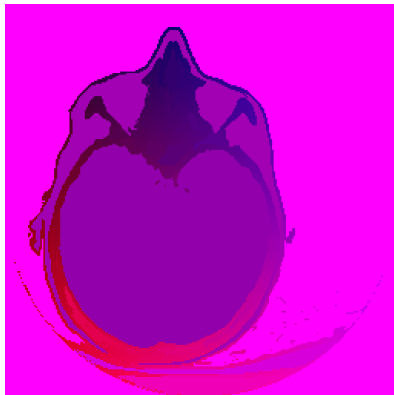
Termination T : No label has changed.

Optimizations: No Master/Slave

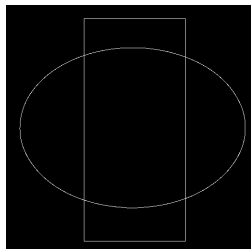
- ▶ Calculating VoD means calculating Distance fields.
- ▶ Propagate information about edge of image, ie. signed distance fields



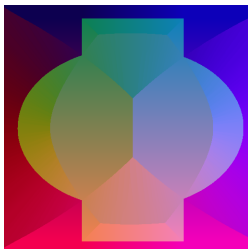
(a) Slice from CT-Scan



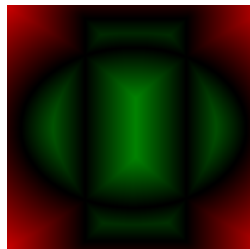
(b) Segmented using CCL



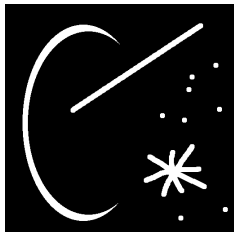
(a) Overlapping regions



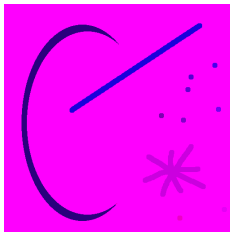
(b) VoD with edge sign



(c) Signed distance field



(a) Input



(b) CCL



(c) VoD



(d) Gen. VoD

CCL:

	No Master/Slave		Master/Slave	
Image	Time (ms)	Block	Time (ms)	Block
CTSlice	4.89	128	2.57	128
Maze	210.00	120	121.83	128

In general $\sim 15ms$ for 1024×1024 image

VoD on 3 image sizes, times in *ms*:

Points	256^2	Block	512^2	Block	1024^2	Block
10	1.55	128	6.39	256	28.73	256
1000	1.17	128	4.03	128	18.00	128
10K	0.87	128	3.85	128	23.25	64

Volumetric Datasets:

Algorithm	Dataset	Size	Time (s)	Block
CCL_3D	CTHead	$256^2 \times 100$	1.88	128
VOD_3D	CTHead	$256^2 \times 100$	0.36	64
VOD_3D	3DVor	256^3	0.73	128

We are looking for dedicated masters and PhD students!

What we can offer:

- ▶ Fun! and challenging projects
- ▶ Many applications are applied in practise
- ▶ Unsolved problems
- ▶ International and interdisciplinary collaboration

What we expect:

- ▶ You are dedicated to your project
- ▶ You work actively with the rest of the group

Thank You!

