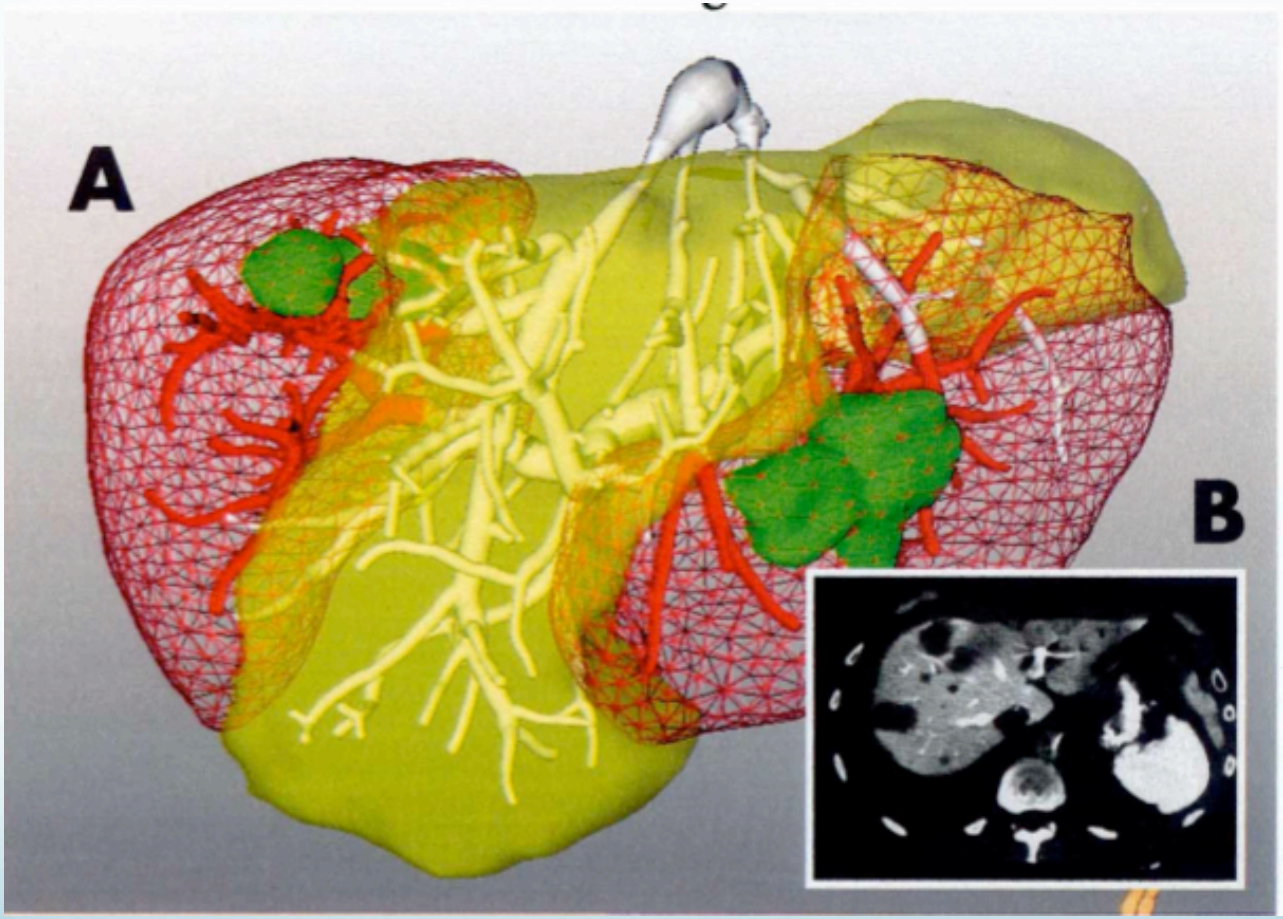
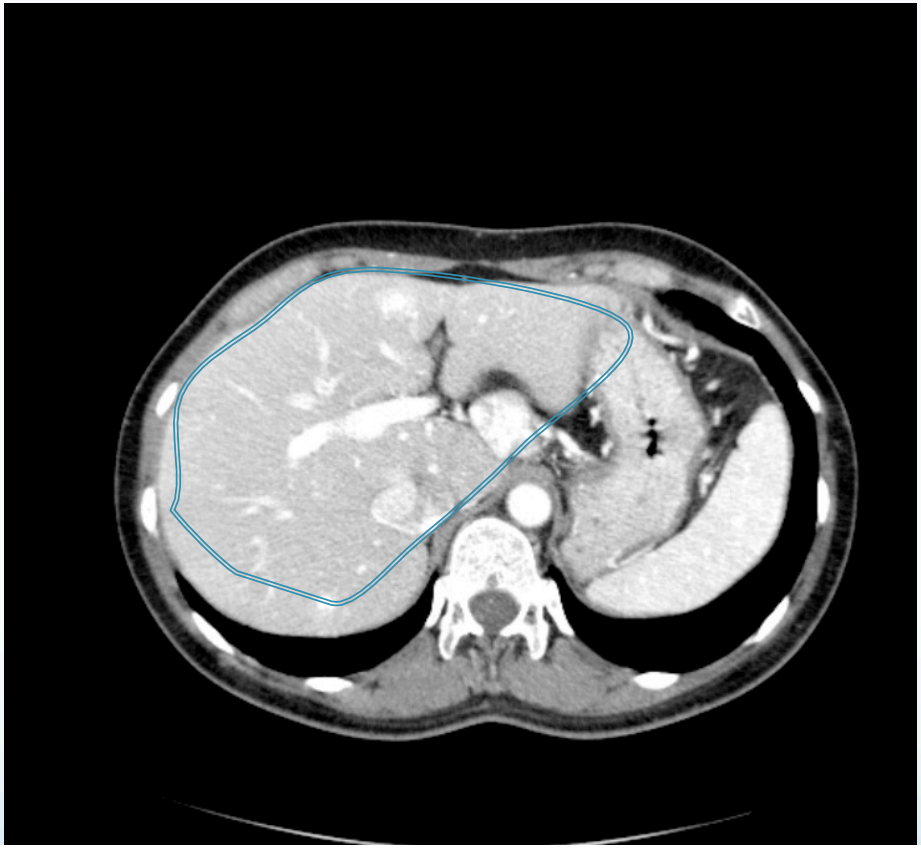
Narrow Band Level set on GPU

Chunliang Wang
CMIV, Linköping University

Image Segmetation



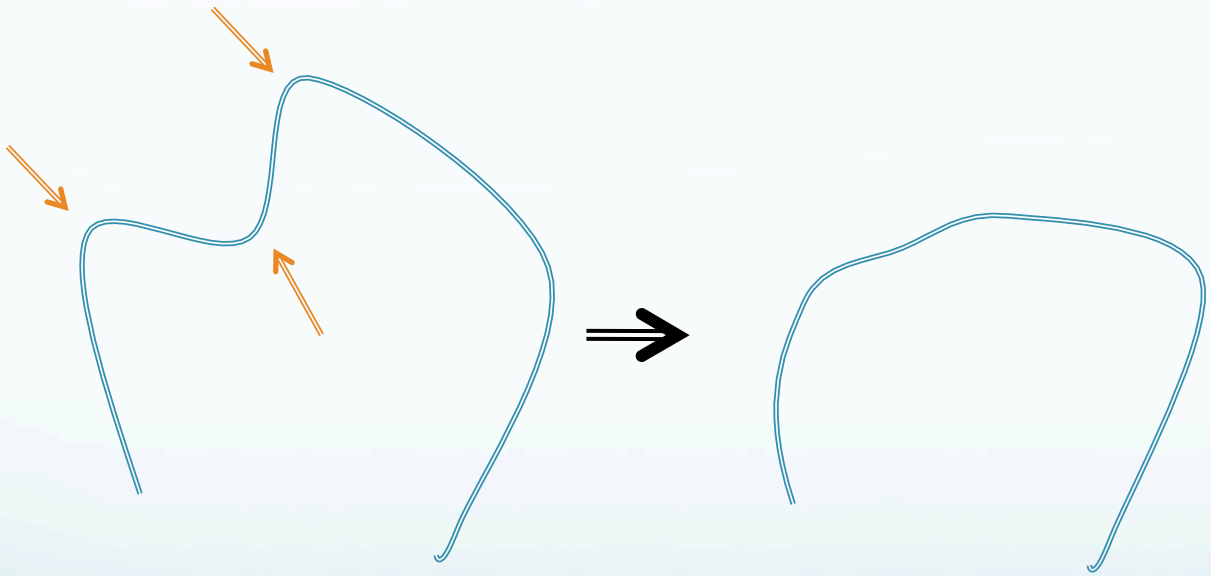
Contour based Segmentation



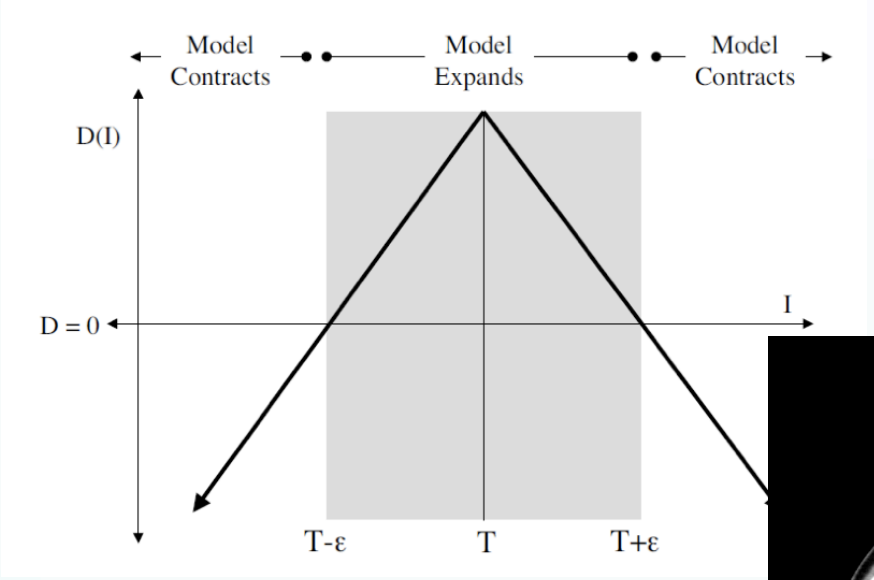
$V(s)=[x_i,y_i]^T$

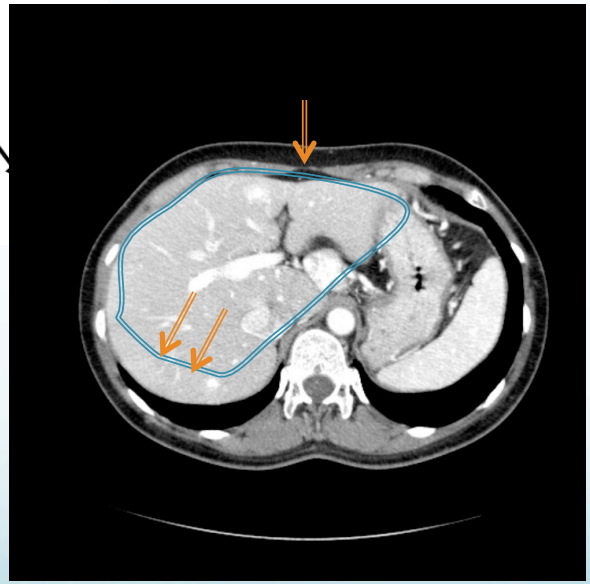
$$\min_{\mathbf{v}} E = \int_0^1 E_{int}(\mathbf{v}(s)) + E_{ext}(\mathbf{v}(s)) + E_{const}(\mathbf{v}(s))ds$$

Internal Force - Curvature

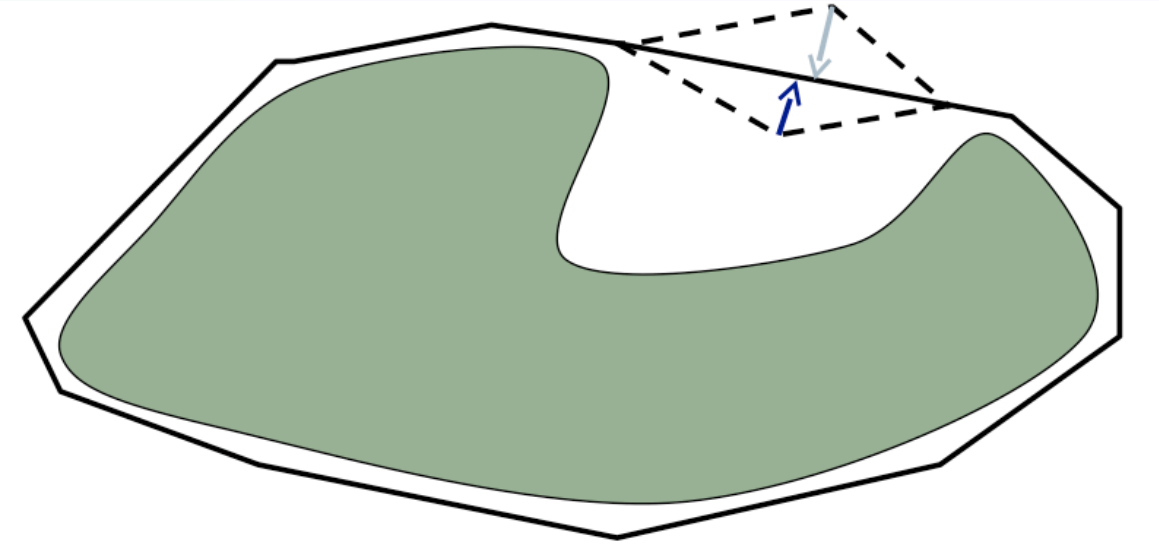


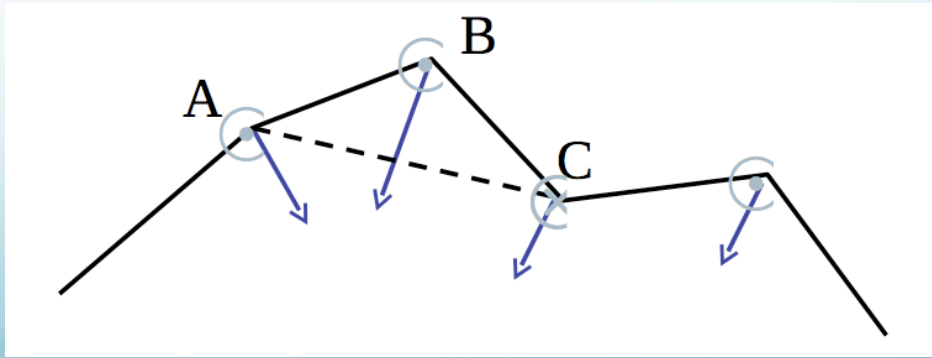
External Force-Intensity



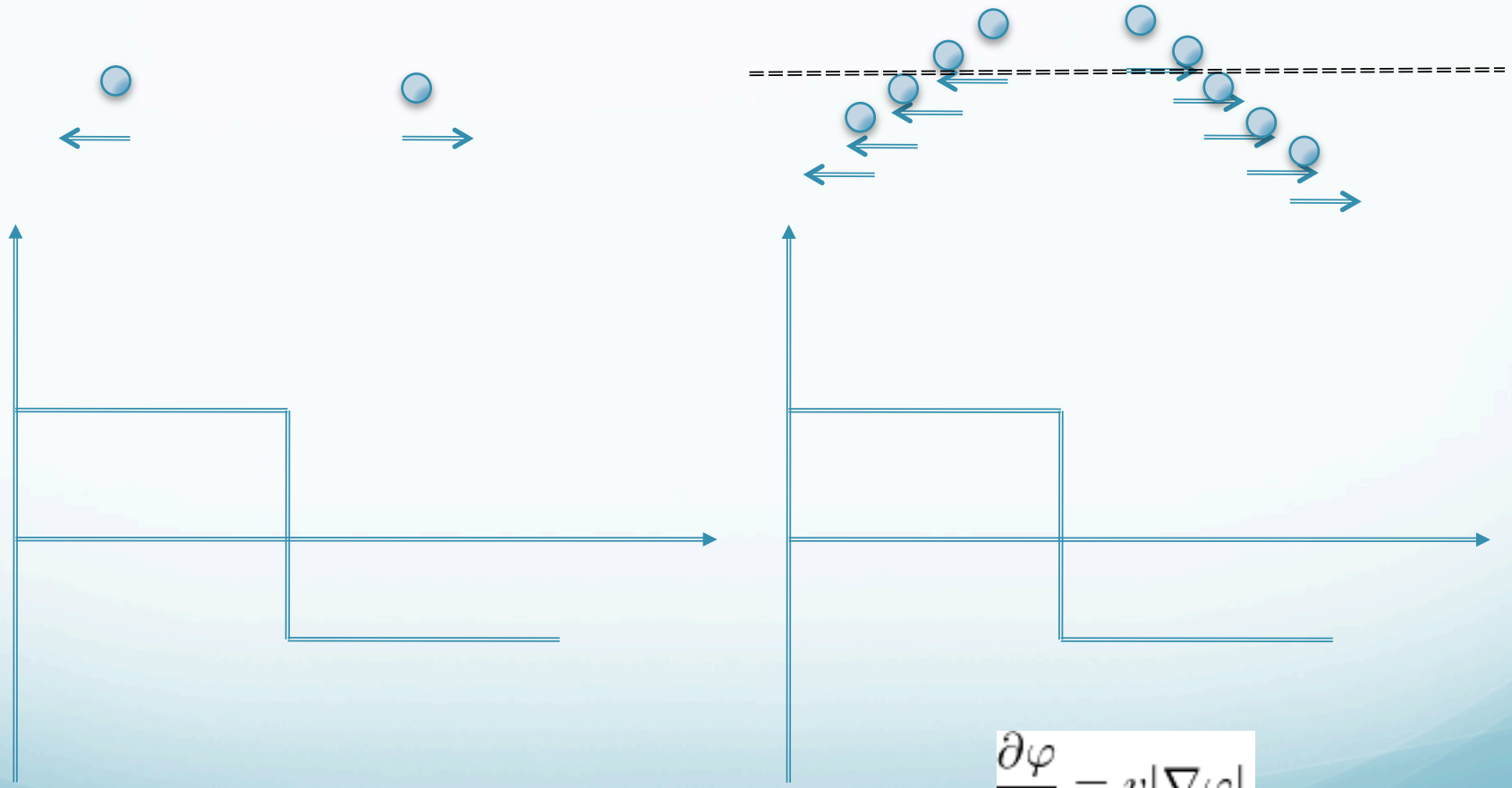


Explicit Contour





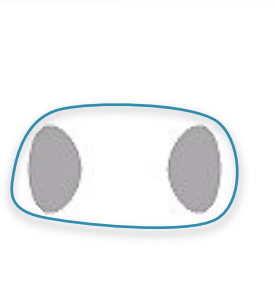
Implicit Contour: Level-set

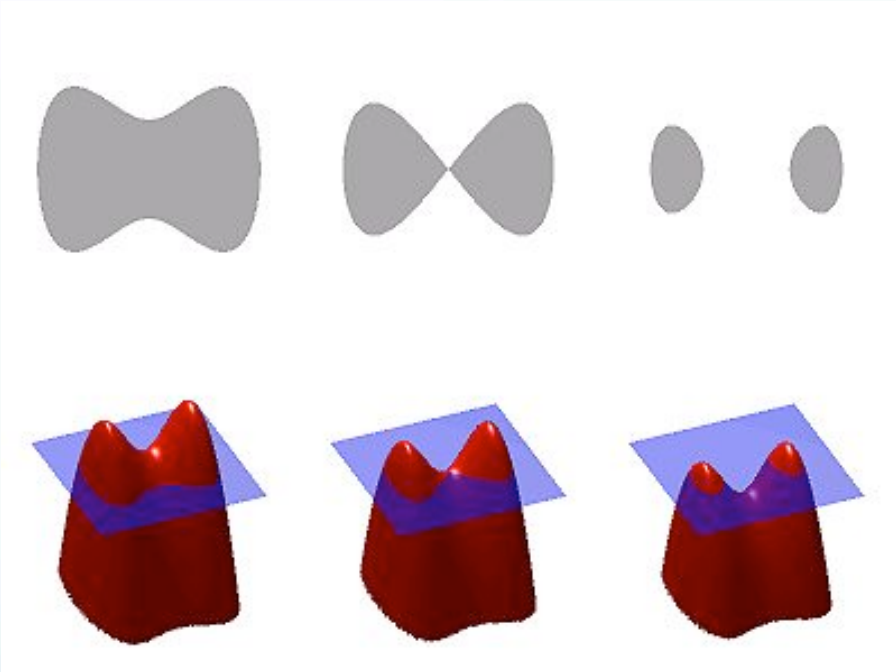


$$\frac{\partial \varphi}{\partial t} = v|\nabla \varphi|.$$

[1]S. Osher and R.P. Fedkiw, *Level set methods and dynamic implicit surfaces*, Springer, 2003.

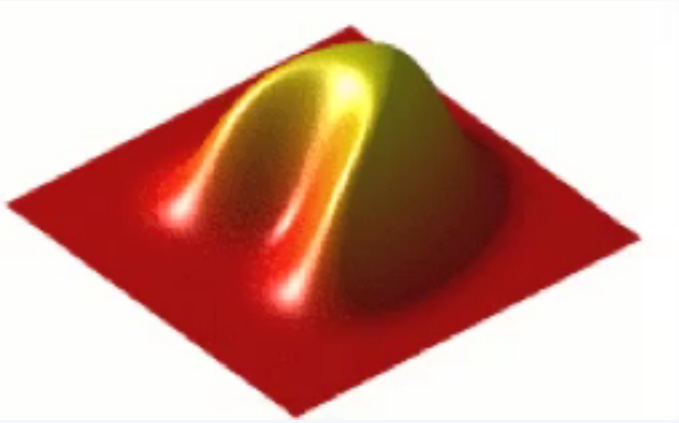
Explicit Contour->Level Set





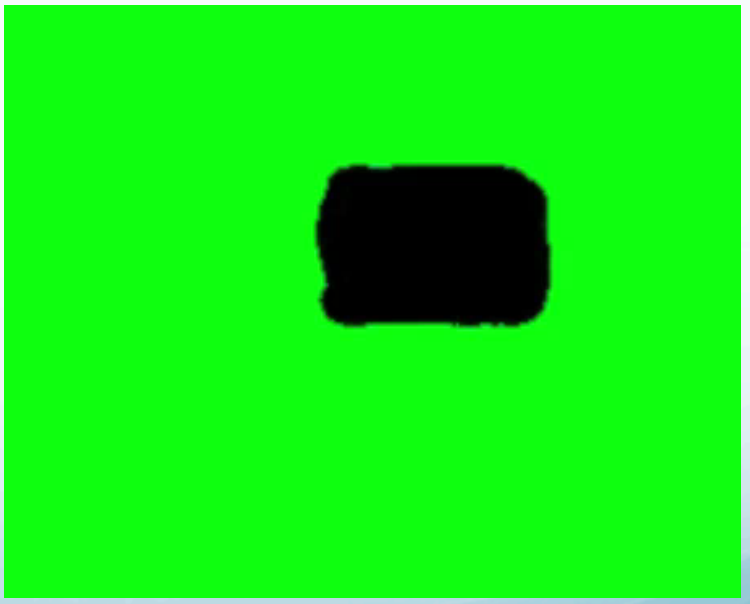
[1]S. Osher and R.P. Fedkiw, *Level set methods and dynamic implicit surfaces*, Springer, 2003.

Level set in Physics



Level set For Image segmenation

$$\frac{\partial \phi}{\partial t} = -|\nabla \phi| \left[\alpha D(I) + (1-\alpha) \nabla \cdot \frac{\nabla \phi}{|\nabla \phi|} \right]$$



What is done in each iteration

$$\frac{\partial \phi}{\partial t} = -|\nabla \phi| \left[\alpha D(I) + (1-\alpha) \nabla \cdot \frac{\nabla \phi}{|\nabla \phi|} \right]$$

- Calculate external speed (intensity)
- Calculate internal speed (curvature)
- Calculate local gradient
- Calculate the local step length (sum of speeds*gradient* *d*t)
- Add the step length to the level set function

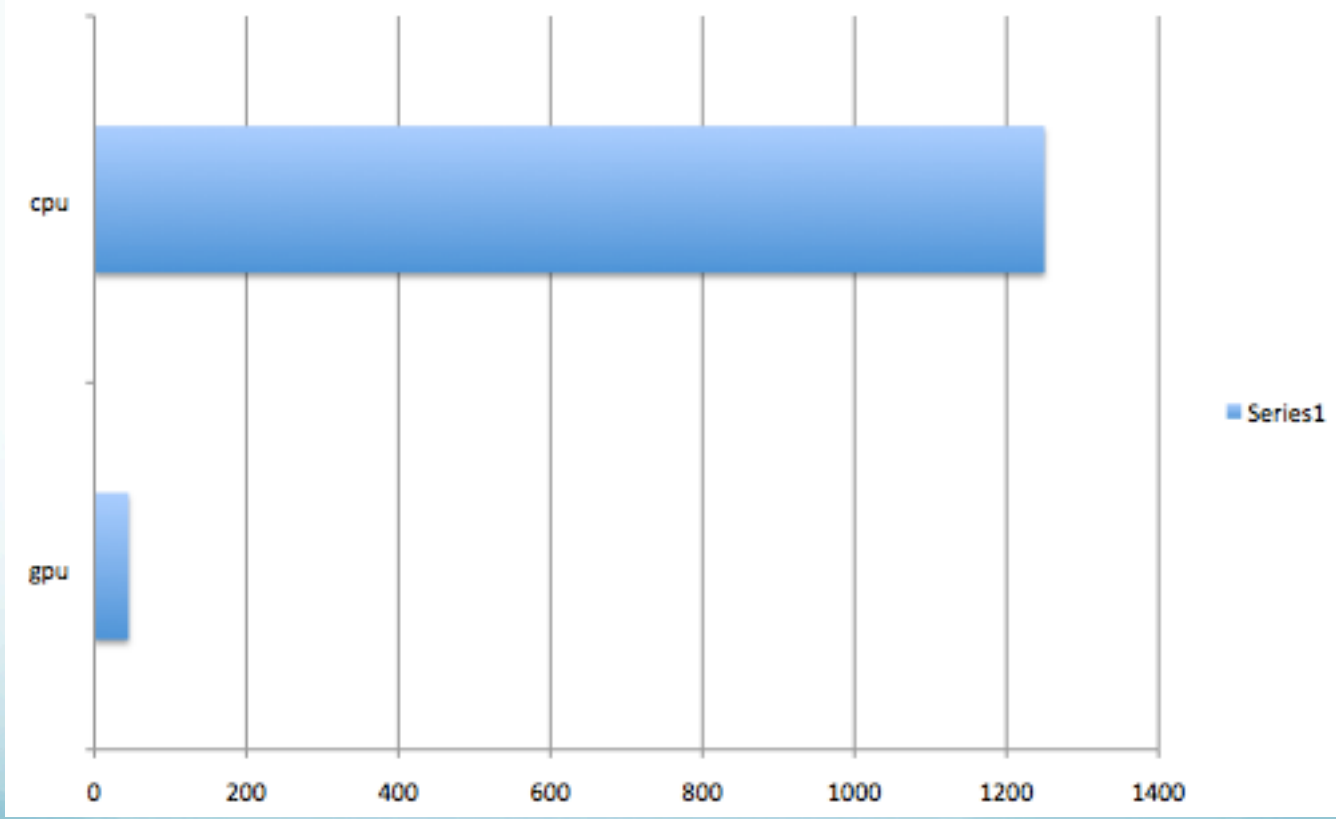
GPU implementation

```
__global__ void updatephi( float *d_phi, float *d_phi1, float *d_D, int imageW, int imageH)

{
    int c= blockIdx.x * blockDim.x + threadIdx.x;
    int r= blockIdx.y * blockDim.y + threadIdx.y;
    int ind= r*imageW+c;

    if(ind<imageW*imageH){

        float dx,dxplus,dxminus,dxplusy,dxminusy;

        float dy, dyplus,dyminus,dyplusx,dyminusx;

        float gradphimax, gradphimin, nplusx, nplusy, nminusx, nminusy, curvature;
        float F, gradphi;

        if(c==0||c==imageW-1){dx=0;} else {dx=(d_phi1[ind+1]-d_phi1[ind-1])/2;}
        if(c==imageW-1){dxplus=0;} else {dxplus=(d_phi1[ind+1]-d_phi1[ind]);}
        if(c==0){dxminus=0;} else {dxminus=(d_phi1[ind]-d_phi1[ind-1]);}
        if(r==0||c==0||c==imageW-1){dxplusy=0;} else {dxplusy=(d_phi1[ind-imageW+1]-d_phi1[ind-imageW-1])/2;}
        if(r==imageH-1||c==0||c==imageW-1){dxminusy=0;} else {dxminusy=(d_phi1[ind+imageW+1]-d_phi1[ind+imageW-1])/2;}

        if(r==0||r==imageH-1){dy=0;} else {dy=(d_phi1[ind-imageW]-d_phi1[ind+imageW])/2;}
        if(r==0){dyplus=0;} else {dyplus=(d_phi1[ind-imageW]-d_phi1[ind]);}
        if(r==imageH-1){dyminus=0;} else {dyminus=(d_phi1[ind]-d_phi1[ind+imageW]);}
        if(r==0||c==imageW-1||r==imageH-1){dyplusx=0;} else {dyplusx=(d_phi1[ind-imageW+1]-d_phi1[ind+imageW+1])/2;}
        if(r==0||c==0||r==imageH-1){dyminusx=0;} else {dyminusx=(d_phi1[ind-imageW-1]-d_phi1[ind+imageW-1])/2;}

        gradphimax=sqrt((sqrt(max(dxplus,0)*max(dxplus,0)+max(-dxminus,0)*max(-dxminus,0)))*(sqrt(max(dxplus,0)*max(dxplus,0)+max(-dxminus,0)*max(-dxminus,0)))
                        +(sqrt(max(dyplus,0)*max(dyplus,0)+max(-dyminus,0)*max(-dyminus,0)))*(sqrt(max(dyplus,0)*max(dyplus,0)+max(-dyminus,0)*max(-dyminus,0))));

        gradphimin=sqrt((sqrt(min(dxplus,0)*min(dxplus,0)+min(-dxminus,0)*min(-dxminus,0)))*(sqrt(min(dxplus,0)*min(dxplus,0)+min(-dxminus,0)*min(-dxminus,0)))
                        +(sqrt(min(dyplus,0)*min(dyplus,0)+min(-dyminus,0)*min(-dyminus,0)))*(sqrt(min(dyplus,0)*min(dyplus,0)+min(-dyminus,0)*min(-dyminus,0))));

        nplusx= dxplus / sqrt(1.192092896e-07F + (dxplus*dxplus) + ((dyplusx + dy)*(dyplusx + dy)*0.25) );
        nplusy= dyplus / sqrt(1.192092896e-07F + (dyplus*dyplus) + ((dxplusy + dx)*(dxplusy + dx)*0.25) );
        nminusx= dxminus / sqrt(1.192092896e-07F + (dxminus*dxminus) + ((dyminusx + dy)*(dyminusx + dy)*0.25) );
        nminusy= dyminus / sqrt(1.192092896e-07F + (dyminus*dyminus) + ((dxminusy + dx)*(dxminusy + dx)*0.25) );
        curvature= ((nplusx-nminusx)+(nplusy-nminusy))/2;

        F = (-ALPHA * d_D[ind]) + ((1-ALPHA) * curvature);
        if(F>0) {gradphi=gradphimax;} else {gradphi=gradphimin;}
        d_phi[ind]=d_phi1[ind] + (DT * F * gradphi);

}
}
```

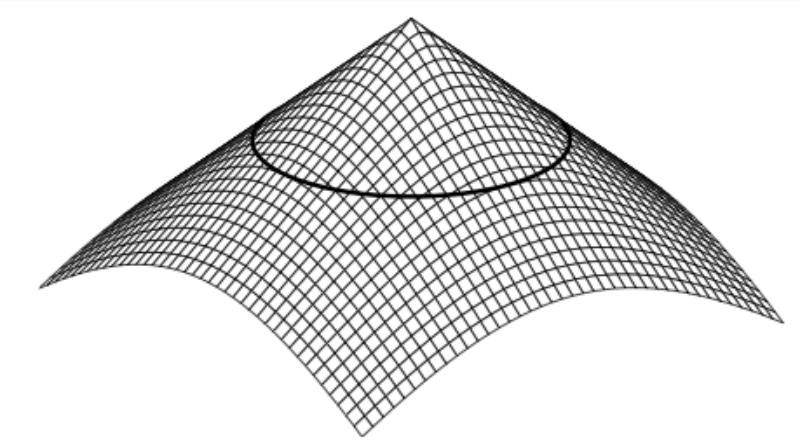
http://code.google.com/p/cudaseg/

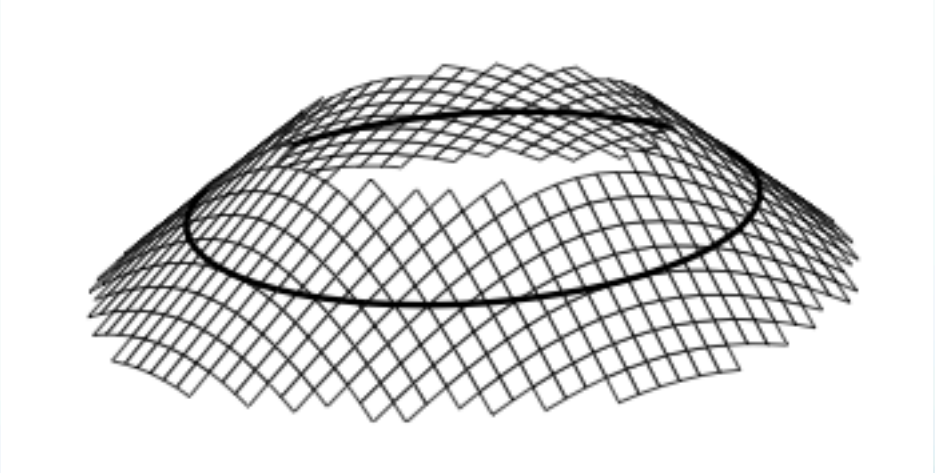
GPU shared mem vs. CPU

- Test image 512*512 (30 time faster)



Narrow Band level set





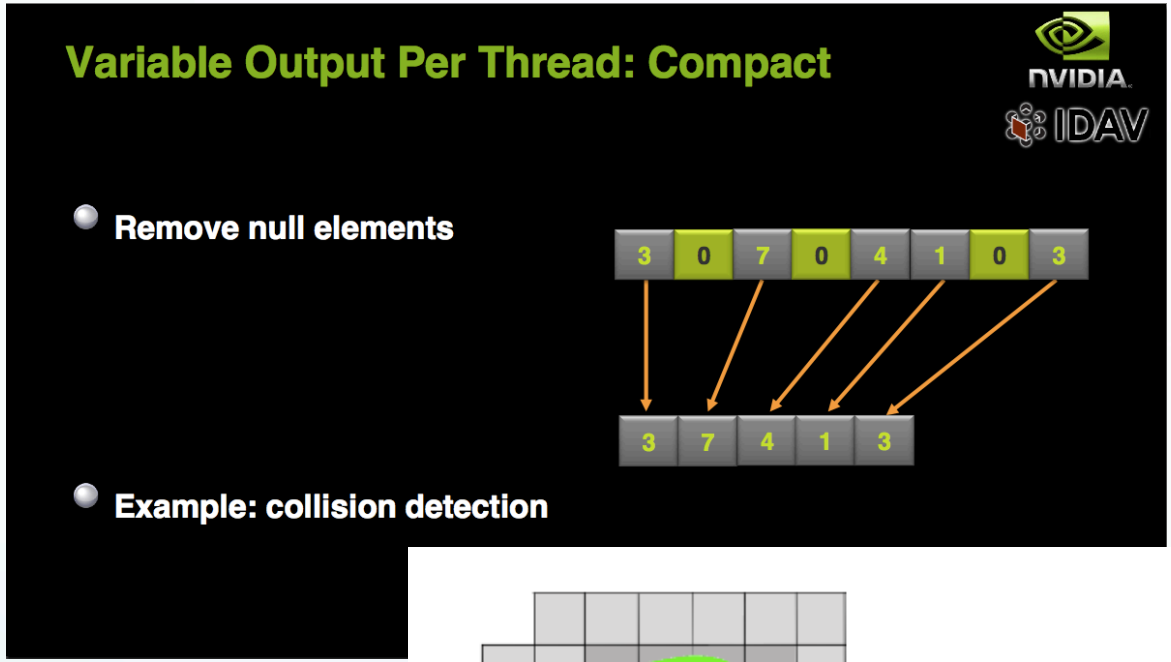
26s on CPU

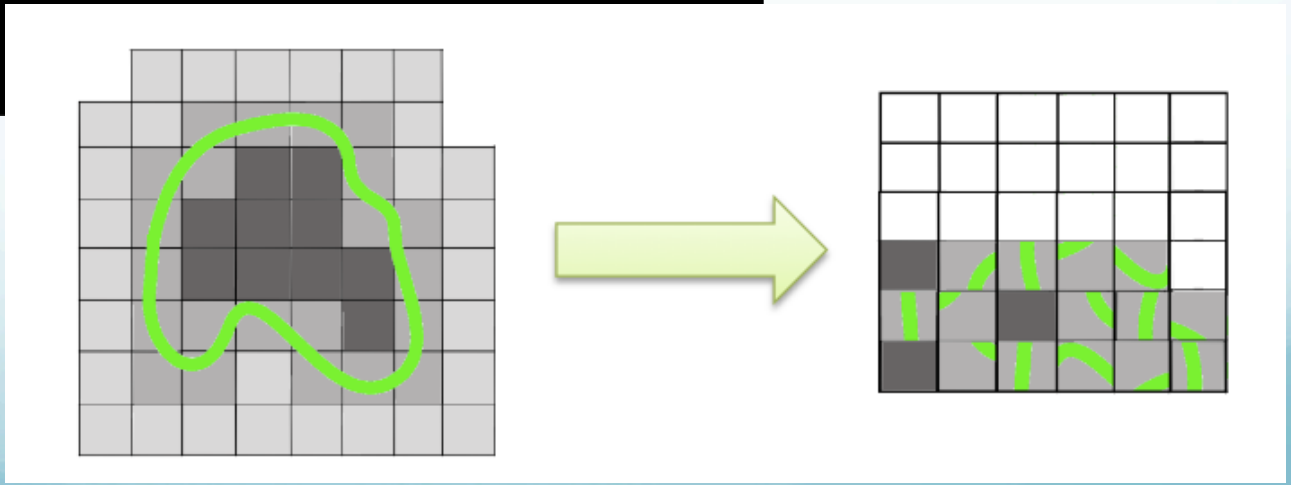
[1]D. Adalsteinsson, D. Adalsteinsson, J.A. Sethian, and J.A. Sethian, “A Fast Level Set Method for Propagating Interfaces,” *JOURNAL OF COMPUTATIONAL PHYSICS*, vol. 118, 1994, pp. 269--277.

Drawbacks of Narrow band on GPU

- No shared memory
- No coalescing
- List operation

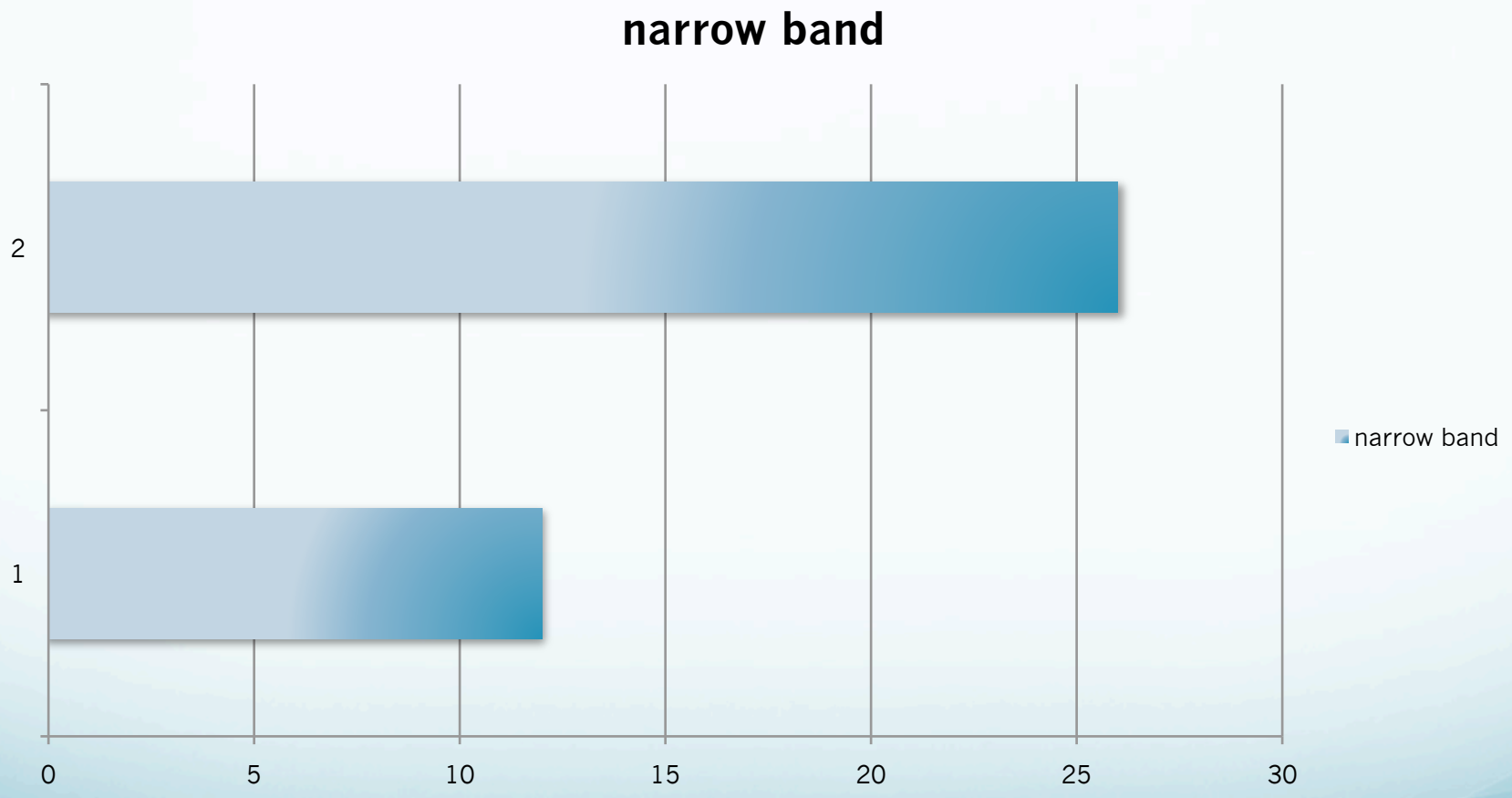
Generating Active Points list using CUDPP on GPU





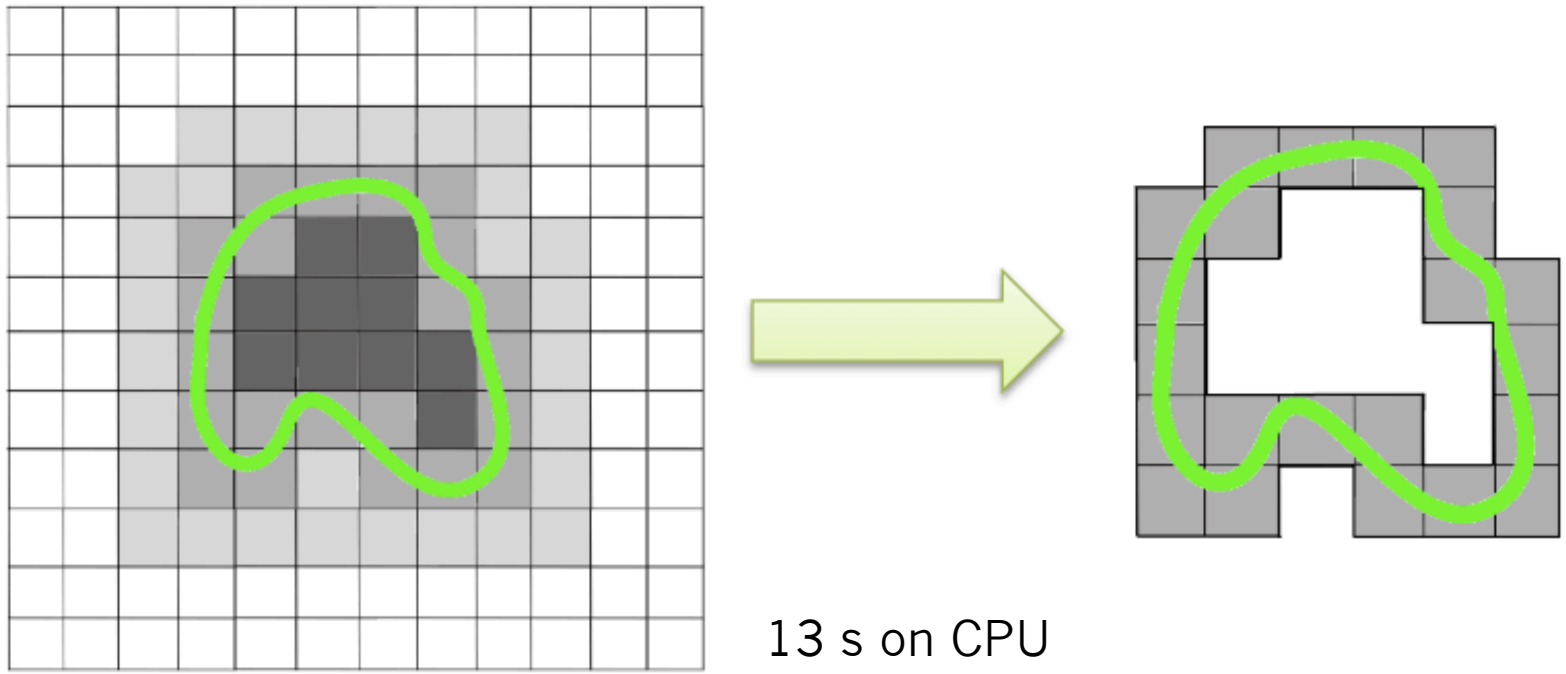
http://cudpp.googlecode.com/svn/trunk/cudpp/doc/CUDPP_slides.pdf

GPU vs. CPU



Ultra-Narrow Band

Sparse field level set



13 s on CPU

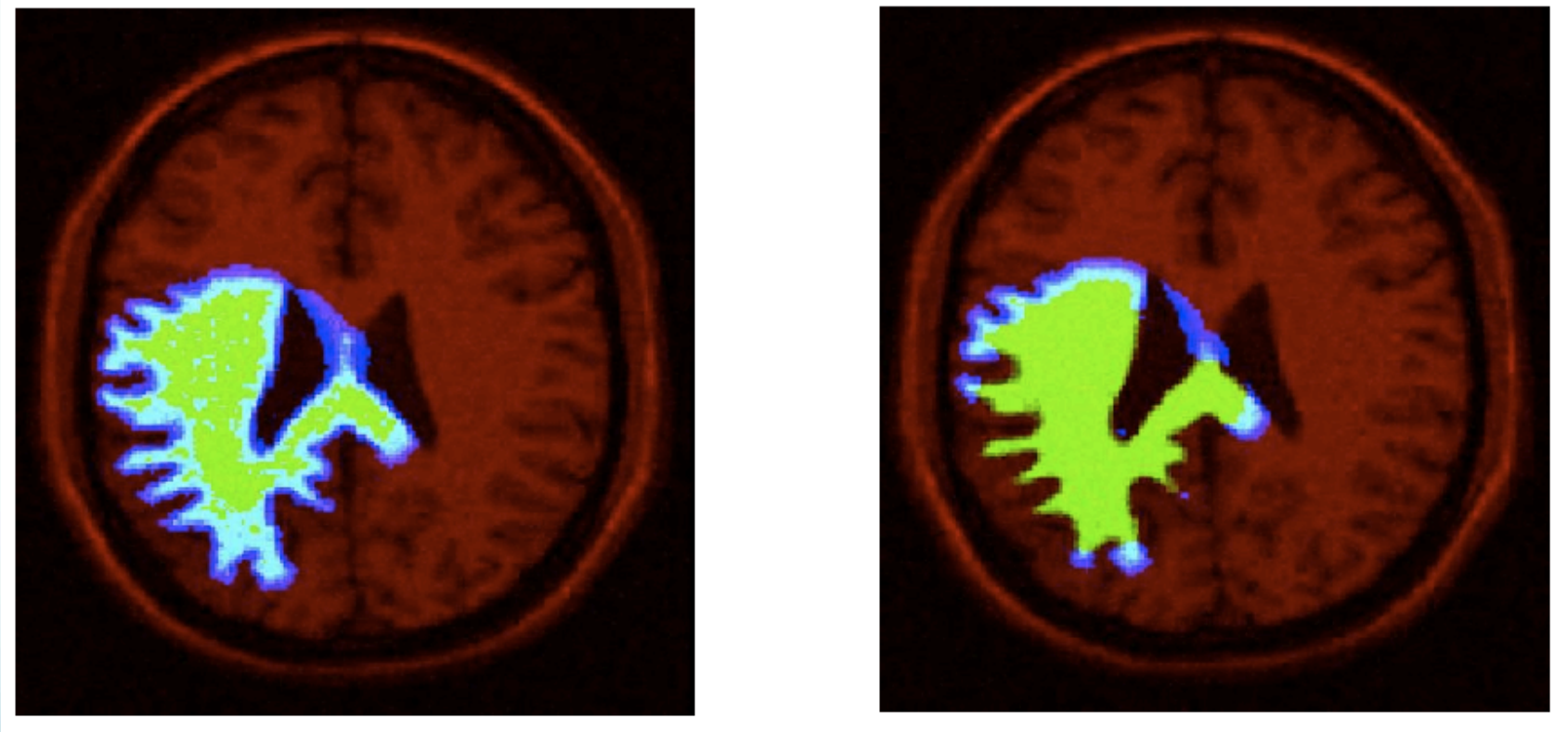
[1]R.T. Whitaker, “A level-set approach to 3d reconstruction from range data,” *INTERNATIONAL JOURNAL OF COMPUTER VISION*, vol. 29, 1998, pp. 203--231.

Generate list from list

index1	index2	index3	index4	index5
neigbor1	neigbor1	neigbor1	neigbor1	neigbor1
neigbor2	neigbor2	neigbor2	neigbor2	neigbor2

		Index2's neighbor		
	Index1's neighbor	index2	Index2's neighbor	
Index1's neighbor	index1	Index2's neighbor		
	Index1's neighbor			

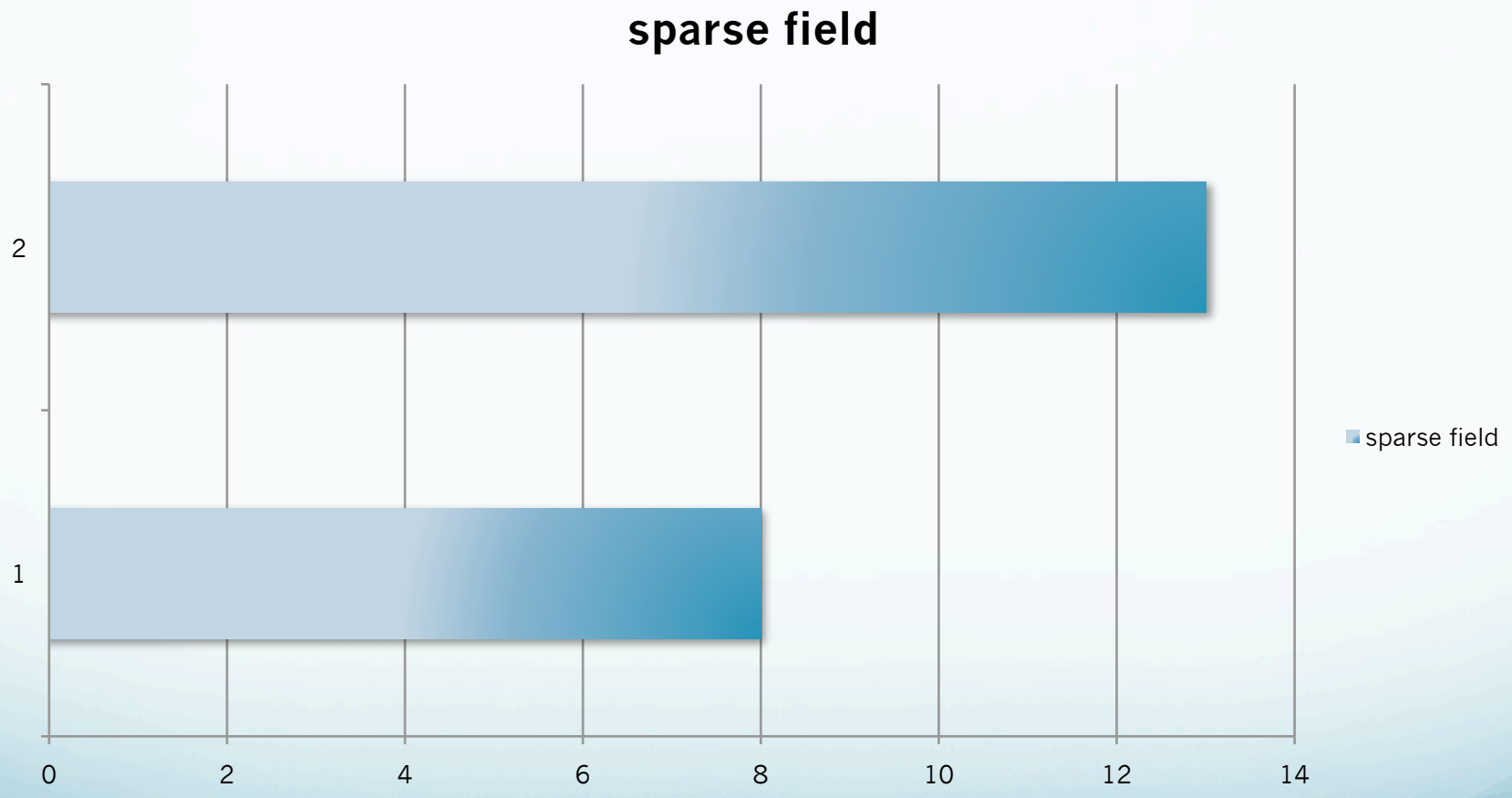
Lazy narrow band levelset



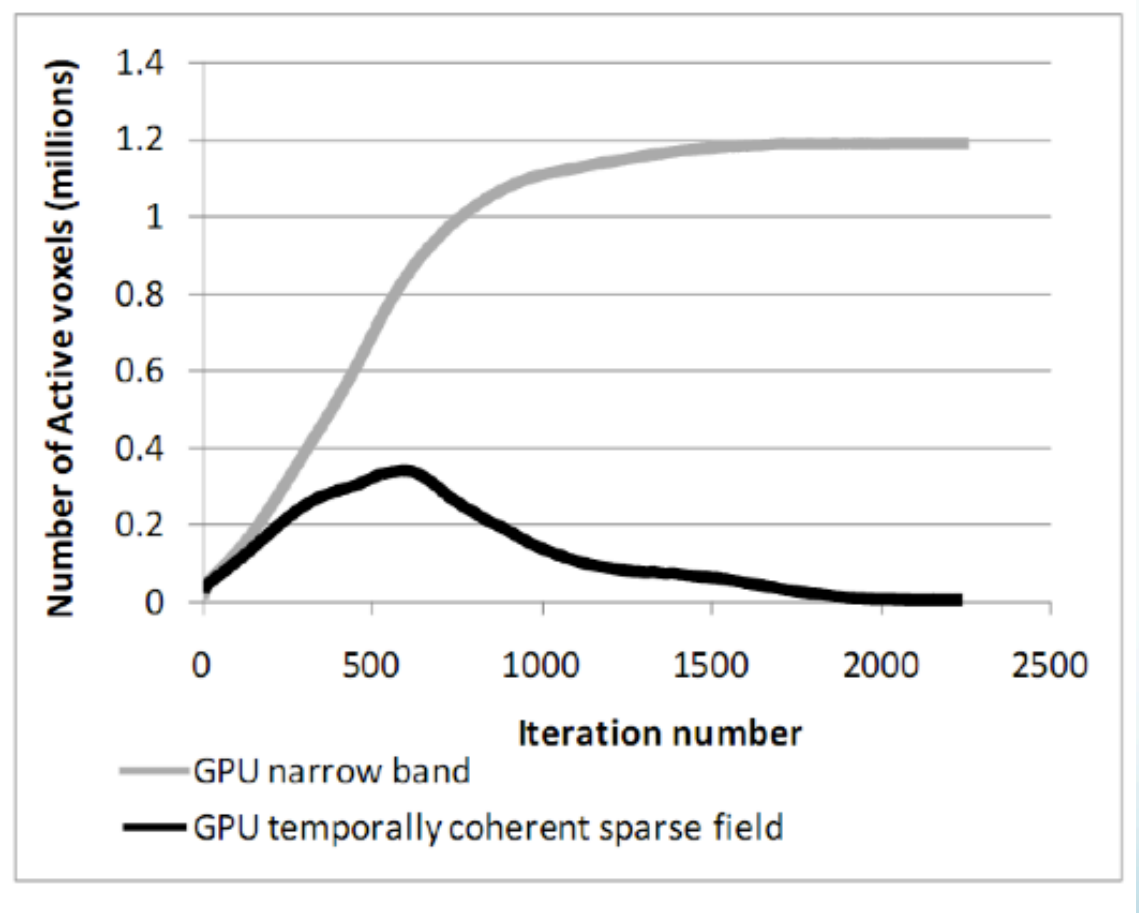
0.8 s on CPU

[1]M. Roberts, J. Packer, M.C. Sousa, and J.R. Mitchell, “A Work-Efficient GPU Algorithm for Level Set Segmentation.”

GPU vs. CPU



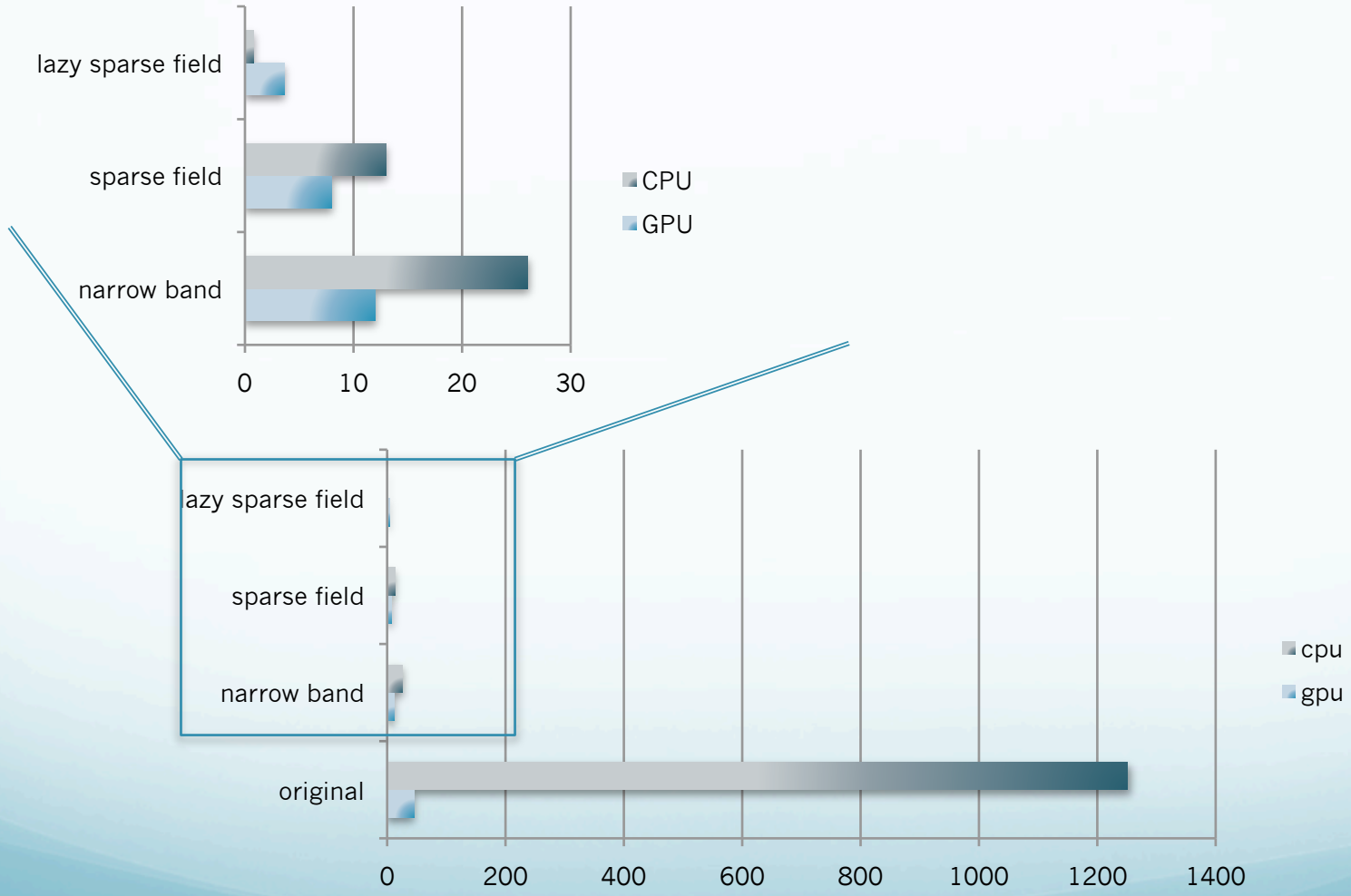
Lazy narrow band levelset



0.8 s on CPU

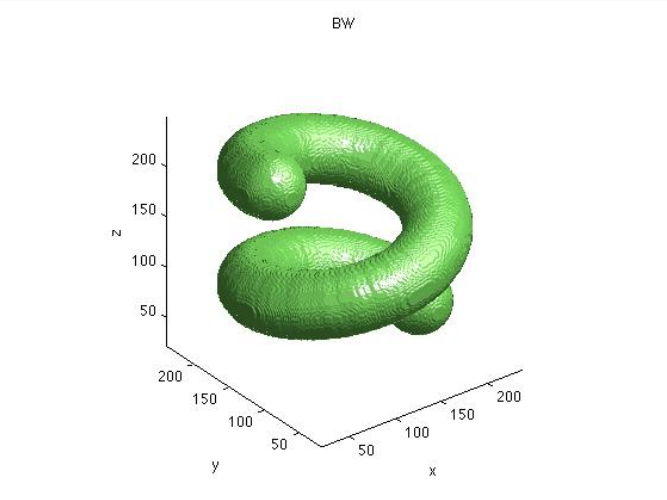
[1]M. Roberts, J. Packer, M.C. Sousa, and J.R. Mitchell, “A Work-Efficient GPU Algorithm for Level Set Segmentation.”

Summary


lazy sparse field
sparse field
narrow band
0
10
20
30
CPU
GPU
lazy sparse field
sparse field
narrow band
original
0
200
400
600
800
1000
1200
1400
cpu
gpu

On 3D

- 256*256*256 dataset



	gpu	cpu
original	360	-
sparse field	70	450
lazy sparse field	14[1]	90

1. I haven't be able to make my 3D version works on GPU, the 14s record is from Roberts' paper.

[1]M. Roberts, J. Packer, M.C. Sousa, and J.R. Mitchell, "A Work-Efficient GPU Algorithm for Level Set Segmentation."

Thank You!