EECE 8395: Medical Image Segmentation
Vanderbilt University
Prof. Jack H. Noble

Fast Marching

This demo is an example of a fast marching computation of a distance map. Our FastMarch function will look like this:

```
function dmapout = FastMarch (dmapi)
```

First we initialize variables

```
global dmap Active

[r,c,d] = size(dmapi);
LSHeapInit(10000);
dmap = 3e8*ones(r,c,d);
dmap(dmapi(:)==0)=0;
Active = ones(r,c,d);
```

Then we can process the foreground part of the distance map using a function `InsertBorderVoxelsIntoHeap`. The function has access to `global dmap Active`. First we insert the foreground border voxels into the heap and estimate their distances. Their `Active` status is set to 2 in the function. The second argument is +1 for finding foreground border voxels and -1 for background:

```
InsertBorderVoxelsIntoHeap(dmapi,1);

figure(2); clf; colormap(gray(256));
image(dmap(:,:,ceil(d/2))*255)
hold on;
cntr = contour(dmapi(:,:,ceil(d/2)),[0,0],'r');
```

Next we pop a voxel off of the heap and add its neighbors to the heap until we have completed the distance map. Here I've added plotting, which slows things down:

```
[node,dist] = LSHeapPop();
while ~isempty(node)

    hold off
    image(dmap(:,:,ceil(d/2))*75)
    hold on;
    plot([cntr(1,2:end),cntr(1,2)],[cntr(2,2:end),cntr(2,2)],'r');
    drawnow;
```

Now that we've popped it off the queue it is no longer Active so we shouldn't add it again in `ProcessNeighborsEikonal`

```
    Active(node)=0;
```

Now we call `ProcessNeighborsEikonal` to estimate distances for the node's active neighbors using the eikonal equation and add them to the heap. This function also has access to `global dmap Active`. The sign of voxels in `dmapi` is used to only compute distances in foreground or background. The third argument is +1 for foreground and -1 for background.

```
    ProcessNeighborsEikonal(node,dmapi,1);
```

Finally, we pop a new node from the heap and keep popping until we find an active one.

```
    [node,dist] = LSHeapPop();
    while (~isempty(node))&&Active(node)==0
        [node,dist] = LSHeapPop();
    end

end
```

That gives us our foreground distance map. Now we do it again for background.

```
dmapin = dmap;
dmap = 3e8*ones(r,c,d);
dmap(dmapi(:)==0)=0;
Active = ones(r,c,d);

% background
InsertBorderVoxelsIntoHeap(dmapi,-1);

figure(2); clf; colormap(gray(256));
image(dmap(:,:,ceil(d/2))*255)
hold on;
cntr = contour(dmapi(:,:,ceil(d/2)),[0,0],'r');

[node,dist] = LSHeapPop();
while ~isempty(node)
    hold off
    image(dmap(:,:,ceil(d/2))*25)
    hold on;
    plot([cntr(1,2:end),cntr(1,2)],[cntr(2,2:end),cntr(2,2)],'r');
    drawnow;


    Active(node)=0;
    ProcessNeighborsEikonal(node,dmapi,-1);

    [node,dist] = LSHeapPop();
    while (~isempty(node))&&Active(node)==0
        [node,dist] = LSHeapPop();
    end

end
```

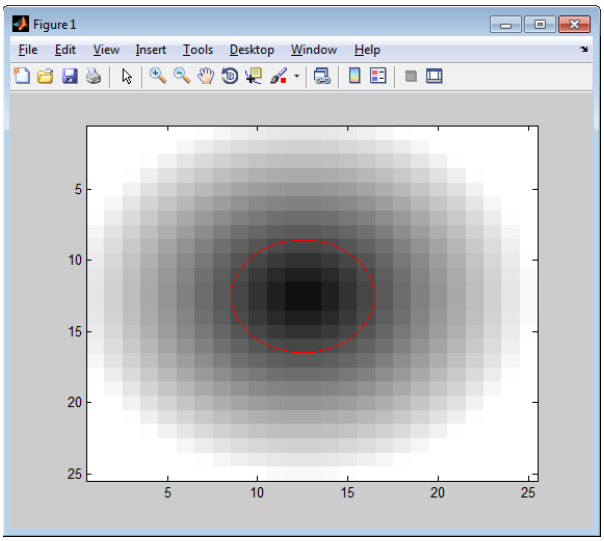
Then we combine the two results into our output distance map:

```
dmapout = dmap;
dmapout(dmapi(:)<0) = -dmapin(dmapi(:)<0);
```

```
return;
```

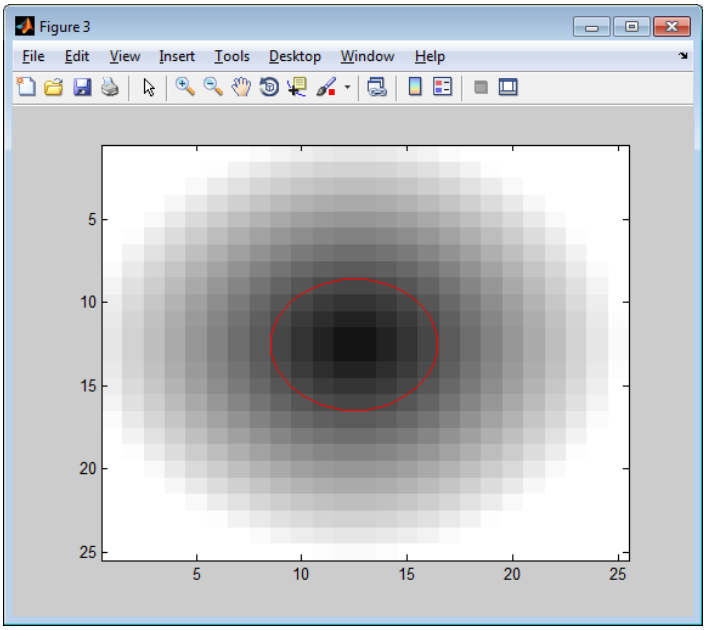
So let's try it out with a ground truth distance map for a circle

```
r=25;
c=25;
d=1;
img = zeros(r,c,d);
rad = 4;
for i=1:r
    for j=1:c
        for k=1:d
            img(i,j,k) = sqrt((i-r/2)*(i-r/2) +(j-c/2)*(j-c/2) +(k-d/2)*(k-
d/2)) - rad;
        end
    end
end
figure(1);clf; colormap(gray(256));
image(20*(img(:,:,ceil(d/2))+rad));
hold on;
contour(img(:,:,ceil(d/2)),[0,0],'r');
```

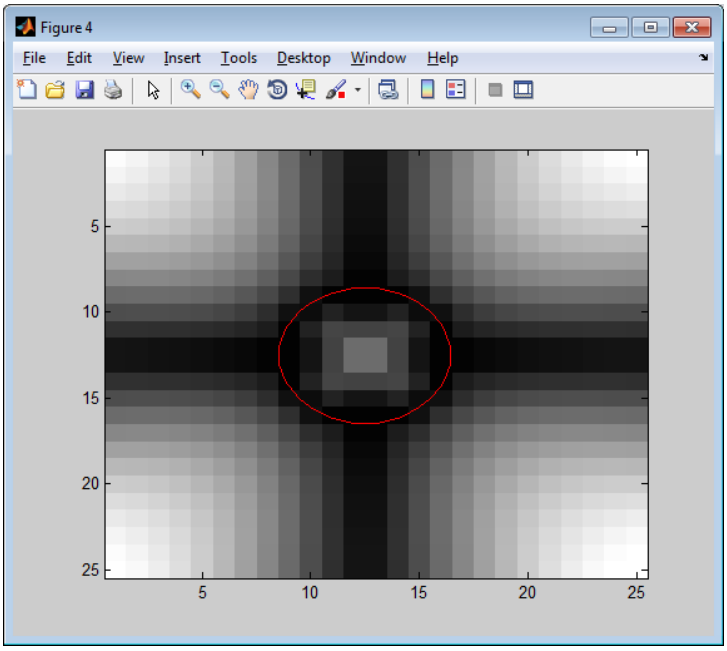



```
dmap = FastMarch(img);

mean(abs(dmap(:)-img(:)))
max(abs(dmap(:)-img(:)))
      ans =

            0.2247


      ans =

            0.5349

figure(3); clf;
```

```
colormap(gray(256));
image((dmap(:,:,ceil(d/2))+rad)*20)
hold on;
contour(dmap(:,:,ceil(d/2)),[0,0],'r');
```

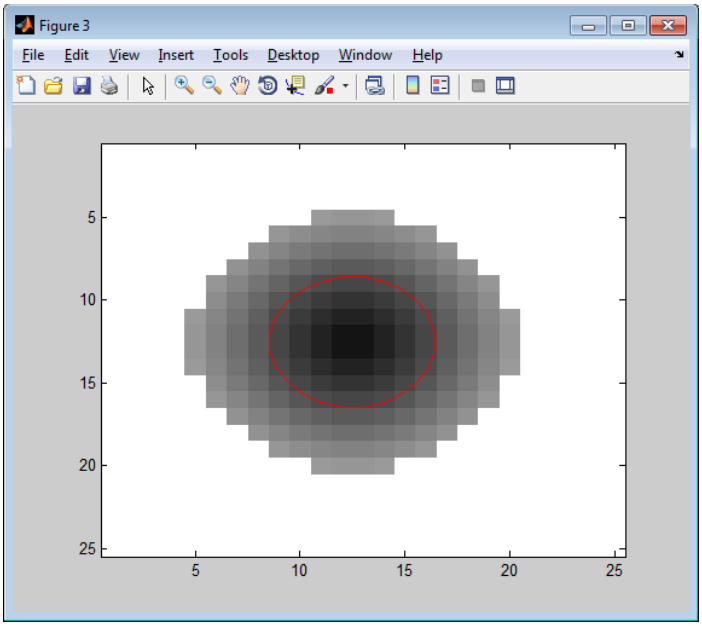



```
figure(4); clf;
colormap(gray(256));
image(abs(dmap(:,:,ceil(d/2))-img(:,:,ceil(d/2)))*500)
```



Next we want to add in a stopping criterion since for level set segmentation we do not need to compute distances over the entire image.

```
function dmapout = FastMarch(dmapi,maxdist)
…
while ~isempty(node)&&dist<maxdist
…
while ~isempty(node)&&dist<maxdist
…

tic; dmap = FastMarch(img,200); toc
      Elapsed time is 5.951386 seconds.
tic; dmap = FastMarch(img,3); toc
      Elapsed time is 1.436689 seconds.
figure(3); clf;
colormap(gray(256));
image((dmap(:,:,ceil(d/2))+rad)*20)
hold on;
contour(dmap(:,:,ceil(d/2)),[0,0],'r');
```



Next we would like to output the narrow band because LevelSet segmentation will want to update the border just in the narrow band region.

```
function [dmapout,nbin,nbout] = FastMarch (dmapi,maxdist,getnb)
…
Initialize
if getnb
    nb.q = zeros(2,r*c*d);
    nb.len=0;
end
…
while ~isempty(node)&&dist<maxdist
```

```
    if getnb
        nb.len = nb.len+1;
        nb.q(:,nb.len) = [node;dist];
    end
```

…

After the foreground processing is done, reinitialize for background:

```
if getnb
    nbin = nb;
    nb.len=0;
end
while ~isempty(node)&&dist<maxdist
    if getnb
        nb.len = nb.len+1;
        nb.q(:,nb.len) = [node;dist];
    end
...
```

And at the end, set `nbout`

```
if getnb
    nbout = nb;
end
```

…

```
[dmap,nbin,nbout] = FastMarch(img,3,1);
nbin =

      q: [2x625 double]
    len: 52
nbout =

      q: [2x625 double]
    len: 96
```

Finally, we also would like to input a narrow band if one already exists (remember level set will iteratively call FastMarch) so that when we run `InsertBorderVoxelsIntoHeap` we don't have to search the whole image, just the voxels in the narrow band.

```
function [dmapout,nbin,nbout] = FastMarch (dmapi,maxdist,getnb,nbi)
```

…

Initialize if it does not already exist

```
if nargin<4 || isempty(nbi)
    nbi.q = 1:r*c*d;
    nbi.len = length(nbi.q);
end
```

…

And we give the narrow band as a 3rd argument to `InsertBorderVoxelsIntoHeap` so that it can search just nodes within the narrow band.

```
InsertBorderVoxelsIntoHeap(dmapi,1,nbi);
```

…

```
InsertBorderVoxelsIntoHeap(dmapi,-1,nbi);
```

How much time does this save?

```
tic; [dmap,nbi,nbout] = FastMarch(img,3,1); toc
nbn = nbi;
nbn.q(:,nbn.len+1:nbout.len+nbn.len) = nbout.q(:,1:nbout.len);
nbn.len = nbn.len+nbout.len;
tic; [dmap,nbi,nbout] = FastMarch(img,3,1,nbn); toc
```

```
Elapsed time is 1.461376 seconds.
Elapsed time is 1.414145 seconds.
```

Not much here but if our image is bigger…

```
r=250;
c=250;
d=1;
img = zeros(r,c,d);
rad = 4;
voxsz = [1,1,1];
for i=1:r
    for j=1:c
        for k=1:d
            img(i,j,k) = sqrt((i-r/2)*(i-r/2)+(j-c/2)*(j-c/2) +(k-d/2)*(k-
d/2)) - rad;
        end
    end
end
figure(1);clf; colormap(gray(256));
image(20*(img(:,:,ceil(d/2))+rad));
hold on;
contour(img(:,:,ceil(d/2)),[0,0],'r');


tic; [dmap,nbi,nbout] = FastMarch(img,3,1); toc
nbn = nbi;
nbn.q(:,nbn.len+1:nbout.len+nbn.len) = nbout.q(:,1:nbout.len);
nbn.len = nbn.len+nbout.len;
tic; [dmap,nbi,nbout] = FastMarch (img,3,1,nbn); toc
```

```
Elapsed time is 2.276586 seconds.
Elapsed time is 1.478813 seconds.
```

And it we code `InsertBorderVoxelsIntoHeap` and `ProcessNeighborsEikonal` properly, the same function also works for 3D:

```
r=25;
c=25;
d=25;
img = zeros(r,c,d);
rad = 4;
for i=1:r
    for j=1:c
```

```
        for k=1:d
            img(i,j,k) = sqrt((i-r/2)*(i-r/2)+(j-c/2)*(j-c/2) +(k-d/2)*(k-
d/2)) - rad;
        end
    end
end
figure(1);clf; colormap(gray(256));
image(20*(img(:,:,ceil(d/2))+rad));
hold on;
contour(img(:,:,ceil(d/2)),[0,0],'r');
```

```
tic; [dmap,nbi,nbout] = FastMarch(img,3,1); toc
Elapsed time is 12.694496 seconds.
```

```
figure(3); clf;
colormap(gray(256));
image((dmap(:,:,ceil(d/2))+rad)*20)
hold on;
contour(dmap(:,:,ceil(d/2)),[0,0],'r');
```