Precise Thresholding

… or, a Small Matlab Package Called Abmask

Anders Brun
Centre for Image Analysis
Swedish University of Agricultural Sciences
Uppsala University

Basic Ideas…

- Given a function u(**x**)
- **Thresholding:**
  if u(**x**) > 0: T(u(**x**)) = 1
  else: T(u(**x**)) = 0
- **Soft version:**
  if u(**x**) > 0 inside pixel **x:** T(u(**x**)) = 1
  else if u(**x**) <= 0 inside pixel **x:** T(u(**x**)) = 0
  else: T(u(**x**)) in]0,1[

What is u(x) inside a pixel?

- Function Value
- Function Gradient

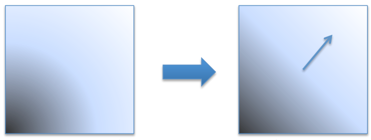
$\left.\begin{array}{l}\text{Function Value}\\ \text{Function Gradient}\end{array}\right\}$ Linear Model

- ü(**x**+d**x**) = u(**x**) + grad(u(**x**)) * d**x**

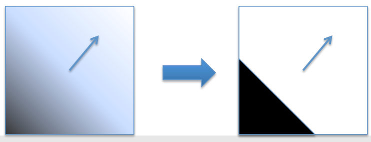
Centre for Image Analysis
Swedish University of Agricultural Sciences
Uppsala University
UPPSALA UNIVERSITET SLU

What is u(x) inside a pixel?

- Cons: A linear model introduces a bias



- Pros: We can estimate a precise "coverage"



Centre for Image Analysis
Swedish University of Agricultural Sciences
Uppsala University
UPPSALA UNIVERSITET SLU

abmask1

- abmask1(u, softness)
- In 1D we have partial coverage iff:
  |u'(**x**) / 2| > |u(**x**)|
- Precise coverage from simple code:

```
fill = zeros(size(u));
fill((u - 0.5*gradx > 0) & (u + 0.5*gradx > 0)) = 1;
fill((u - 0.5*gradx < 0) & (u + 0.5*gradx < 0)) = 0;
idx = (u - 0.5*gradx > 0) & (u + 0.5*gradx < 0);
fill(idx) = (0.5*gradx(idx)-u(idx))./gradx(idx);
idx = (u - 0.5*gradx < 0) & (u + 0.5*gradx > 0);
fill(idx) = (0.5*gradx(idx)+u(idx))./gradx(idx);
```

Centre for Image Analysis
Swedish University of Agricultural Sciences
Uppsala University
UPPSALA UNIVERSITET SLU

abmask2

- abmask2(u, softness)
- In 2D we have / have not partial coverage if:
  ||grad u(**x**) / 2|| > |u(**x**)| or ||grad u(**x**) / sqrt(2)|| < |u(**x**)|
- Precise coverage from trigonometry tricks:

```
theta = pi/4 - abs(mod(angle(gradx+i*grady),pi/2)-pi/4);
x = u./sqrt(gradx.^2 + grady.^2);
a = -1/sqrt(2)*cos(pi/4-theta); d = -a;
b = -1/sqrt(2)*sin(pi/4-theta); c = -b;

fill = zeros(size(u));
m = x <= a;
fill(m) = double(u(m)>0);
m = (x > a) & (x <= b);
fill(m) = 0.5*(x(m)-a(m)).^2./(cos(theta(m)).*(b(m)-a(m)));
m = (x > b) & (x <= c);
fill(m) = 0.5*(b(m)-a(m))./cos(theta(m)) + (x(m)-b(m))./cos(theta(m));
m = (x > c) & (x < d);
fill(m) = 1 - 0.5*(-x(m)-a(m)).^2./(cos(theta(m)).*(b(m)-a(m)));
m = x >= d;
fill(m) = double(u(m)>0) ;
```

Centre for Image Analysis
Swedish University of Agricultural Sciences
Uppsala University
UPPSALA UNIVERSITET SLU

abmask3

- abmask3(u, softness)
- In 3D we have / have not partial coverage if:
  ||grad u(**x**) / 2|| > |u(**x**)| or ||grad u(**x**) * sqrt(3)/2|| < |u(**x**)|
- Precise coverage from divide and conquer…
  - Divide voxel into 5 tetrahedra (simplices)
  - Compute precise coverage for each simplex & sum

```
for k = 1:length(ii)
    f1 = tetragradvol([u000(k),u001(k),u010(k),u100(k)], [0 0 0; 0 0 1; 0 1 0; 1 0 0]);
    f2 = tetragradvol([u110(k),u100(k),u010(k),u111(k)], [1 1 0; 1 0 0; 0 1 0; 1 1 1]);
    f3 = tetragradvol([u111(k),u100(k),u001(k),u101(k)], [1 1 1; 1 0 0; 0 0 1; 1 0 1]);
    f4 = tetragradvol([u111(k),u001(k),u010(k),u011(k)], [1 1 1; 0 0 1; 0 1 0; 0 1 1]);
    f5 = tetragradvol([u001(k),u100(k),u111(k),u010(k)], [0 0 1; 1 0 0; 1 1 1; 0 1 0]);
    fill(ii(k)) = fill(ii(k)) + f1 + f2 + f3 + f4 + f5;
end
```


Sub Pixel Precision is non-linear!

- Because of all the geometric cases involved, essentially the rotation variance of the pixel (it is a square, it is not round), sub pixel accuracy using *linear* models inside pixels yields a *non-linear* expression for the coverage inside a pixel.
- Could there be another representation of the image / gradient where the coverage is a linear function?

Softness, what?

- Softness:
  - Multiplies the gradient with a factor. High gradient yields a higher probability of partial coverage.
  - The mismatch between original function values and artificially larger gradients makes the fuzzy border bigger! Bug or feature?
- If softness > 1: soft border wider than > 1 pixel
- If 0 < softness < 1: more crisp border

Gradient, what?

- The gradient is either
  - Estimated from numerical differentiation of the function or
  - Provided analytically, because it is know to the user and then we can avoid the extra smoothing a numerical differentiation might give

"Precise" Enables Differentiation

- Enables numerical differentiation:
  - Compute volume of sphere with radius 0.50000001
  - Compute volume of sphere with radius 0.50000000
  - Divide the difference with 0.00000001
  - This is an estimate of the surface area
- Applies to surface area (3D) and circumference (2D) of arbitrary shapes
- Thresholding or sampled coverage… try! :-)

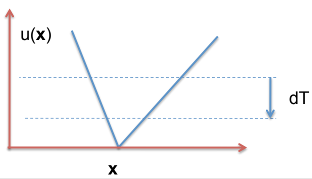
"Precise" Enables Differentiation

Increasing threshold moves the levelset curve

"The Eikonal equation"

Going from threshold T to T - dt:

moves curve segment dN = dt/||grad u(**x**)||

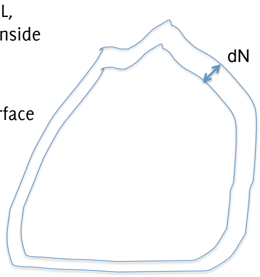


"Precise" Enables Differentiation

Thus, in 2D, area increases locally by dN*dL, where dL is the curve segment length inside the pixel.

So … we can measure circumference or surface area by this simple expression:

Sum((abmask(u-dT,1)-abmask(u,1)) ./ dN)



"Divide the band with its width and integrate"

Open Questions

- Generalization to N-D (N>3) and other grids
  - Divide and Conquer via N-D simplices is one way to go here...
- And hey… didn´t we throw away a little too much when we forgot the gradient direction?
- Given both coverage (a bitmap with values 0...1) and gradient direction, we have all information about the linear model inside every pixel. Useful?