Digital Image Processing, Assignment #3: Level-set Image Segmentation

고려대학교 컴퓨터학과
2015410113 신채호

1. Geodesic Active Contours implementation

먼저 edge indicator term에서 smoothing된 이미지의 gradient를 쓰므로 g에서 smoothed version of image를 먼저 구현하기로 했다. Assignment 2에서 Butterworth frequency filter를 구현했는데, 그걸 활용해 smoothed version을 만들기로 했다. 현재 input image를 처음에 모두 double 형으로 만들어 주기 때문에 tofloat함수를 빼고, 2배 padding을 해주면 되므로 인풋 사이즈에 2배를 해서 PQ를 세팅해줬다. 그리고 마지막에 I_hat의 실수부만 남겨주었다. 코드와 결과 smoothed version of input image는 다음과 같다.

```
PQ = size(Img)*2;
dimPadX = PQ(1);
dimPadY = PQ(2);

F = fft2(Img,PQ(1),PQ(2));
F = fftshift(F);

n = 2;
D0 = 70;
D = double(zeros(dimPadX,dimPadY));
for i=1:dimPadX
    for j=1:dimPadY
        D(i,j) = ((i-dimPadX/2)^2+(j-dimPadY/2)^2)^(1/2);
    end
end

H = 1./(1+(D./D0).^(2*n));
G = H.*F;
G = ifftshift(G);
I_hat = ifft2(G);
I_hat = real(I_hat(1:dimX, 1:dimY));
```

그리고 I_hat의 gradient를 구하기 위해 gradient를 구현하였는데, sobel gradient filter를 써 보았다. Sobel gradient filter는 대각 인접 픽셀에는 1만큼의 weight를 주고 수직 수평선에 위치한 인접 픽셀에는 2만큼의 weight를 줘서 central difference를 계산하는 것이다. 원래는 제일 처음과 끝 row, column에 위치한 픽셀들 때문에 padding을 해주고 진행해야 하지만, 우리가 필터링 하고자 하는 이미지는 끝 쪽이 모두 0으로 set 되어 있으므로 zeros를 통해 먼저 모두 0으로 세팅해주고 row와 column을 2부터 input size - 1까지 진행했다. 그리고 자료에 나와있는 수식을 구현해주었다.

```
[dim_Ix,dim_Iy] = size(I_hat);
dx=zeros(dim_Ix,dim_Iy);
dy=zeros(dim_Ix,dim_Iy);

for x=2:dimX-1
    for y=2:dimY-1
        dx(x,y) = (I_hat(x+1,y+1)+2.*I_hat(x+1,y)+I_hat(x+1,y-1)-2.*I_hat(x-1,y)-I_hat(x-1,y+1)-I_hat(x-1,y-1))./9;
        dy(x,y) = (I_hat(x-1,y+1)+2.*I_hat(x,y+1)+I_hat(x+1,y+1)-I_hat(x-1,y-1)-2.*I_hat(x,y-1)-I_hat(x+1,y-1))./9;
    end
end

tmp = sqrt(dx.^2+dy.^2);
p = 2;
g = 1./(1+tmp.^p);
```

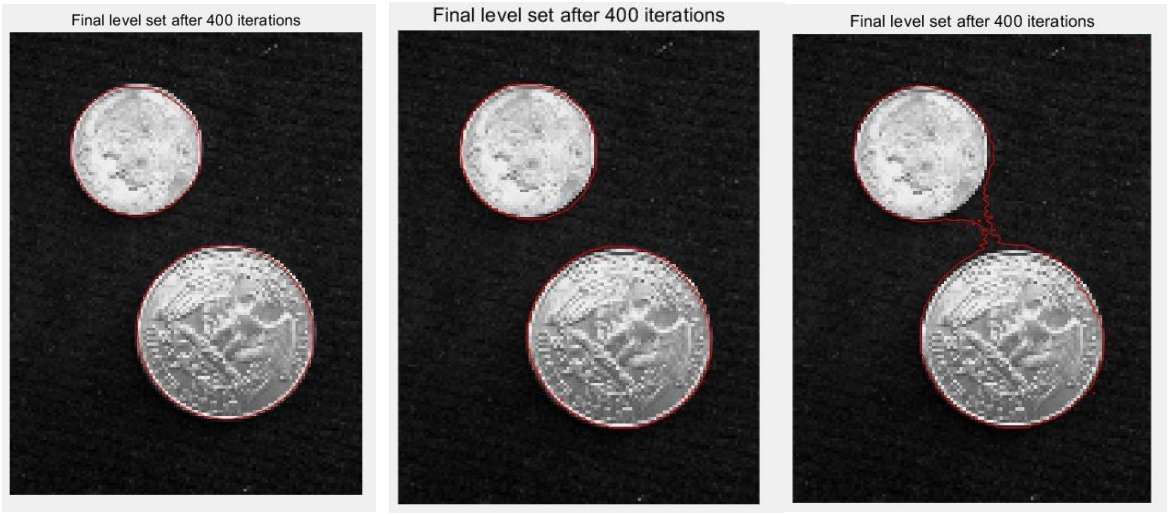
그 후 levelset_update 부분을 구현했다. Levelset-update에서도 똑같이 sobel gradient filter를 따로 find_grad()라는 함수를 구현해서 사용했다. 먼저 들어온 input phi의 gradient를 find_grad()를 통해 각각 phi_gradx, phi_grady로 놓고, 각각의 제곱 합의 제곱근을 dPhi(magnitude of gradient)로 설정해 주었다. 그리고 curvature 텀을 구해주기 위해 phi_gradx, phi_grady를 각각의 magnitude로 나눠준 것을 dx, dy로 설정해주고, dx+dy의 gradient를 구해 div_x, div_y를 구하고 둘을 더해주며 curvature 텀을 구현해 주었다.

```
[phi_gradx,phi_grady] = find_grad(phi_in);

dPhi = sqrt(phi_grady.^2+phi_gradx.^2); % mag(grad(phi))

eps = 1e-10;
dx = phi_gradx./(sqrt(phi_gradx.^2)+eps);
dy = phi_grady./(sqrt(phi_grady.^2)+eps);
[div_x, div_y] = find_grad(dx+dy);
kappa =  div_x+div_y;% curvature
```

2. Tests

다음은 다른 parameter는 바꾸지 않고, butterworth LPF의 n=2로 고정, D0를 50, 70, 100으로 높여가며 실험한 결과이다.

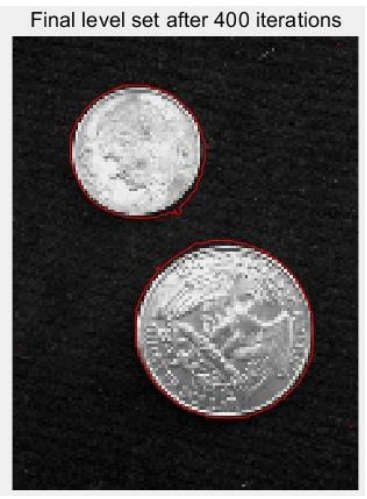
D0=50 D0=70 D0=100



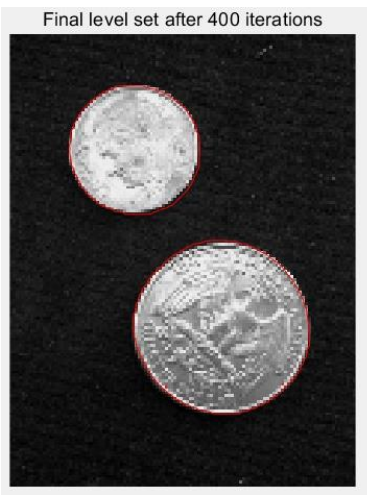
D0가 작을수록, 즉 cut-off frequency가 작아 더 작은 frequency만 통과시키며 smoothing이 잘되어 있을수록 더 빨리 converge하는 것을 확인 가능했다. D0가 50인 경우에는 400 iteration이 좀 과한 듯했고, D0가 100인 경우에는 아직 converge 하지 못해 더 많은 iteration이 필요한 것 같았다. D0가 70일 때가 딱 적당해 보였다.

다음으로 weight constant c를 0.8, 1, 1.5로 바꿔보며 실험을 해 보았다.

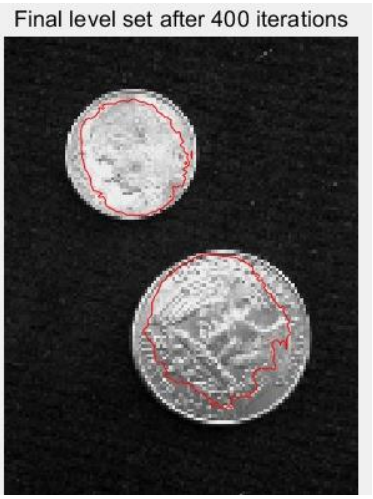
C=0.8



C=1.0



C=1.5



C가 커질수록 한 번에 expand 더 많이 expand 되므로 converge도 빨리 되지만, c가 1.5일 때를 보면 coin의 경계 안까지 뚫고 들어가 버린다. 마찬가지로 1.0일 때가 가장 적절해 보인다.

인터넷에서 다른 grayscale image를 가져와서 실행을 시켜 보았다. Edge의 인접 픽셀과의 차이가 큰 부분은 잘 detection이 되었지만 그렇지 않은 부분은 iteration이 돌면서 경계 안쪽으로 들어가는 것을 확인 가능했다.