

Code A: Matlab Code for Poisson Image Reconstruction from Image Gradients

```
% Read Input Gray Image
imgstr = 'test.png'; disp(sprintf('Reading Image %s',imgstr));
img = imread(imgstr);      [H,W,C] = size(img);      img = double(img);
% Find gradinets
gx = zeros(H,W); gy = zeros(H,W);      j = 1:H-1; k = 1:W-1;
gx(j,k) = (img(j,k+1) - img(j,k));      gy(j,k) = (img(j+1,k) - img(j,k));

% Reconstruct image from gradients for verification
img_rec = poisson_solver_function(gx,gy,img);

figure;imagesc(img);colormap gray;colorbar;title('Image')
figure;imagesc(img_rec);colormap gray;colorbar;title('Reconstructed');
figure;imagesc(abs(img_rec-img));colormap gray;colorbar;title('Abs error');
```

```
function [img_direct] = poisson_solver_function(gx,gy,boundary_image);
% function [img_direct] = poisson_solver_function(gx,gy,boundary_image)
% Inputs; Gx and Gy -> Gradients
% Boundary Image -> Boundary image intensities
% Gx Gy and boundary image should be of same size
[H,W] = size(boundary_image);
gxx = zeros(H,W);      gyy = zeros(H,W);
f = zeros(H,W);      j = 1:H-1;      k = 1:W-1;

% Laplacian
gyy(j+1,k) = gy(j+1,k) - gy(j,k);      gxx(j,k+1) = gx(j,k+1) - gx(j,k);
f = gxx + gyy;      clear j k gxx gyy gyyd gxxd

% boundary image contains image intensities at boundaries
boundary_image(2:end-1,2:end-1) = 0;
disp('Solving Poisson Equation Using DST');      tic
j = 2:H-1;      k = 2:W-1;      f_bp = zeros(H,W);
f_bp(j,k) = -4*boundary_image(j,k) + boundary_image(j,k+1) +
            boundary_image(j,k-1) + boundary_image(j-1,k) + boundary_image(j+1,k);
clear j k

f1 = f - reshape(f_bp,H,W);% subtract boundary points contribution
clear f_bp f

% DST Sine Transform algo starts here
f2 = f1(2:end-1,2:end-1);      clear f1
%compute sine transform
tt = dst(f2);      f2sin = dst(tt)';      clear f2

%compute Eigen Values
[x,y] = meshgrid(1:W-2,1:H-2);      denom = (2*cos(pi*x/(W-1))-2) + (2*cos(pi*y/(H-1)) - 2) ;

%divide
f3 = f2sin./denom;      clear f2sin x y

%compute Inverse Sine Transform
tt = idst(f3);      clear f3;      img_tt = idst(tt)';      clear tt

time_used = toc;      disp(sprintf('Time for Poisson Reconstruction = %f secs',time_used));

% put solution in inner points; outer points obtained from boundary image
img_direct = boundary_image;
img_direct(2:end-1,2:end-1) = 0;
img_direct(2:end-1,2:end-1) = img_tt;
```

```

function b=dst(a,n)
%DST Discrete sine transform (Used in Poisson reconstruction)
% Y = DST(X) returns the discrete sine transform of X.
% The vector Y is the same size as X and contains the
% discrete sine transform coefficients.
% Y = DST(X,N) pads or truncates the vector X to length N
% before transforming.
% If X is a matrix, the DST operation is applied to each
% column. This transform can be inverted using IDST.

error(nargchk(1,2,nargin));

if min(size(a))==1
    if size(a,2)>1
        do_trans = 1;
    else
        do_trans = 0;
    end
    a = a(:);
else
    do_trans = 0;
end
if nargin==1, n = size(a,1); end
m = size(a,2);

% Pad or truncate a if necessary
if size(a,1)<n,
    aa = zeros(n,m); aa(1:size(a,1),:) = a;
else
    aa = a(1:n,:);
end

y=zeros(2*(n+1),m); y(2:n+1,:)=aa; y(n+3:2*(n+1),:)=flipud(aa);
yy=fft(y); b=yy(2:n+1,:)/(-2*sqrt(-1));

if isreal(a), b = real(b); end
if do_trans, b = b.'; end

```

```

function b=idst(a,n)
%IDST Inverse discrete sine transform (Used in Poisson reconstruction)
%
% X = IDST(Y) inverts the DST transform, returning the
% original vector if Y was obtained using Y = DST(X).
% X = IDST(Y,N) pads or truncates the vector Y to length N
% before transforming.
% If Y is a matrix, the IDST operation is applied to
% each column.

if nargin==1
    if min(size(a))==1
        n=length(a);
    else
        n=size(a,1);
    end
end

nn=n+1; b=2/nn*dst(a,n);

```

Code B: Matlab Code for Graph Cuts on Images

```
% Read gray scale image
I = imread('test.png');      [H,W,C] = size(I);

% Find graph cut
Ncut = graphcuts(I,33,255);

figure;imagesc(I);colormap gray;title('Image');
figure;imagesc(Ncut);colormap gray;title('Segmentation');
```

```
function [Ncut] = graphcuts(I,pad,MAXVAL)
% function [Ncut] = graphcuts(I)
% Input: I image
%      pad: spatial connectivity; eg. 3
%      MAXVAL: maximum image value
% Output: Ncut: Binary map 0 or 1 corresponding to image segmentation

I = double(I);      [H,W] = size(I);

% Find weights between nodes I1 and I2, w = exp(a*abs(I1-I2));
% Set a to have a weight of 0.01 for diff = MAXVAL
a = log(0.01)/MAXVAL;      x = [0:MAXVAL/100:MAXVAL]';      y = exp(a*x);
figure;plot(x,y);xlabel('intensity diff');ylabel('weights');      title('weights')

ws = 2*pad + 1;
if(ws <= 3)      ws = 3;      end

%Build the weight matrix
disp('Building Weight Matrix');      close all;      tic

WM = zeros(H*W,H*W);      countWM = 0;
for kk = 1:W
    for jj = 1:H
        mask = logical(zeros(H,W));
        cs = kk-pad;      ce = kk+pad;      rs = jj-pad;      re = jj+pad;
        if(cs<1)      cs = 1;      end;
        if(ce>W)      ce = W;      end;
        if(rs<1)      rs = 1;      end;
        if(re>H)      re = H;      end;
        mask(rs:re,cs:ce) = 1;
        idx = find(mask==1);
        p = abs(I(idx) - I(jj,kk));      p = exp(a*p);
        countWM = countWM + 1;      WM(countWM,idx) = p(:)';
    end
end
ttime = toc; disp(sprintf('Time for generating weight matrix = %f',ttime)); clear countWM

% Weight between a node and itself is 0
for jj = 1:H*W      WM(jj,jj) = 0;      end;      WM = sparse(WM);

% Shi and Malik Algorithm: second smallest eigen vector
disp('Finding Eigen Vector');
d = sum(WM,2);      D = diag(d);      tic
B = (D-WM);      B = (B+B')/2;      OPTS.disp = 0;
[v,d,flag] = eigs(B,D,2,'SA',OPTS);      ttime = toc;
disp(sprintf('Time for finding eigen vector = %f',ttime));      clear OPTS
y = v(:,2);
Ncut = reshape(y,H,W);
Ncut = Ncut > 0;
```

Code C: Matlab Code for Bilateral Filtering on Images

```
function [img1] = bilateral_filtering(img,winSize,sigma)

% Bilateral Filtering(img,winSize,sigma)
% Input      -> Image img
%           -> winSize: spatial filter width
%           -> sigma for intensity diff gaussian filter
%           -> sigma for spatial filter = winSize/6
% Output     -> Filtered Image
% Author: Amit Agrawal, 2004

disp('Bilateral Filtering');

[H,W] = size(img);

%Gaussian spatial filter
g_filter = fspecial('gaussian',winSize,winSize/6);
padnum = (winSize-1)/2;

A = padarray(img,[padnum padnum],'replicate','both');
img1 = zeros(size(img));

for jj = padnum+1:(padnum+1+H-1)
    for kk = padnum+1:(padnum+1+W-1)

        % Get a local neighborhood
        imgwin = A(jj-padnum:jj+padnum, kk-padnum:kk+padnum);

        % Find weights according to intensity diffs
        Wwin = exp(-abs(imgwin - imgwin(padnum+1,padnum+1))/sigma^2);

        % Find composite filter
        newW = Wwin.*g_filter;

        t = sum(sum(newW));
        if(t>0)
            newW = newW/t;
        end

        img1(jj-padnum, kk-padnum) = sum(sum(imgwin.*newW));
    end
end
```
