Assignment 3 & 5

Computer Vision - IT-524

Milind Padalkar
ID: 201121015

12 October 2012

Assignment 3

1. Consider a hemispherical surface $z(x,y) = \sqrt{(r^2 - x^2 - y^2)} + z_0$. Select an image of size 64×64. With the origin as the centre, and $r = 24$ (say), find the surface normal representing the gradients $p(x,y)$ and $q(x,y)$ analytically. Assume the surface to be Lambertian. The object is illuminated by a point light source with $(p_s, q_s) = 0, 0$. Generate the corresponding image.

2. Use regularization based approach in (p,q) and (f,g) domains to recover the shape. Use the gradient at $x^2 + y^2 = r^2$ with $r = 24$ or 18 as the initial conditions. How do the performances compare (w.r.t. actual p&q)? From the estimated map recover the depth $z(x,y)$. How does the recovered depth compare with the actual depth?

3. Add noise to the image and redo the experiment. Compare the MSE as a function of the variances of noise. Repeat the experiment with different p_s, q_s (See papers for using p_s and q_s).

4. Assume that the exact knowledge of p_s, q_s is not available i.e. you only know a noisy estimate of p_s, q_s. Repeat the experiment with varying amount of errors in source localization.

Assignment 5

1. Recover Depth using an initial estimate of depth (iterative method)

Code and Results

```
%   Computer Vision (IT524)
%   Assignment 3 & 5
%--------------------------------------------------------------------------
%   Part 1:
%   Statement: Consider a hemispherical surface z(x,y) = sqrt(r^2-x^2-y^2) + z0.
%   Select an image of size 64x64. With the origin as the centre, and r=24 (say),
%   find the surface normal representing the gradients p(x,y) and q(x,y) analytically.
%   Assume the surface to be Lambertian. The object is illuminated by a point
%   light source with (ps,qs)=(0,0). Generate the corresponding image.
%--------------------------------------------------------------------------
%   Part 2:
%   Statement: Use regularization based approach in (p,q) and (f,g) domains
%   to recover the shape. Use the gradient at x^2+y^2=r^2 with r=24 or 18 as
%   the initial conditions. How do the performances compare (w.r.t. actual p & q)?
%   From the estimated map recover the depth z(x,y).
%   How does the recovered depth compare with the actual depth?
%--------------------------------------------------------------------------
```

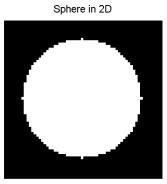
```
%   Part 3:
%   Add noise to the image and redo the experiment.
%   Compare the MSE as a function of the variances of noise.
%   Repeat the experiment with different ps,qs (See papers for using ps and ps).
%-------------------------------------------------------------------------
%   Part 4:
%   Assume that the exact knowledge of ps,qs is not available i.e. you only
%   know a noisy estimate of ps,qs. Repeat the experiment with varying
%   amount of errors in source localization
%-------------------------------------------------------------------------
%   Created by: Milind Padalkar (201121015)
%   Date: 13-09-2012
%-------------------------------------------------------------------------
%   Assignment 5: Recover Depth using an initial estimate of depth (iterative method)

%% Clear workspace
clear;
close all;
clc;

%% Part 1
%-------------------------------------------------------------------------
%% Set Parameters
rds = 24;                   % Sphere's radius
sz = 64;                    % Image size
p_s = [0,-1];               % Source p_s
q_s = [0,-1];               % Source q_s
max_itrn = 500;             % maximum iterations (500)
reg_lambda = 0.1;           % Regularization parameter (0.1)
z0 = 50;                     % Average depth
fn_pts = 5;                 % To calculate MSE as a funtion of noise variance

%% Circle initialization
if(mod(sz,2)==0)
    mid = sz/2;
else
    mid = floor(sz/2) + 1;
end
sphr = zeros(sz,sz);

% Draw the sphere
for j = 1:sz
    y = mid - j;
    for i = 1:sz
        x = mid - i;
        if(x*x + y*y <= rds*rds)
            sphr(i,j) = 1;
        end
    end
end
hdl_sphr = figure; imshow(sphr), title('Sphere in 2D');
```

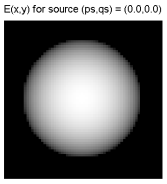
```
%% Calculating p and q and finally R(p(x,y),q(x,y)) = E(x,y)
%   Given
%   1. p = -x/(z-z0)
%   2. q = -y/(z-z0)
%   3. (z-z0)^2 + (x^2+y^2) = r^2
%   Thus,
%   E(x,y) = (p*p_s + q*q_s + 1)/(sqrt(p_s^2+q_s^2+1)*sqrt(p^2+q^2+1))
%
%          = ((-x/(z-z0))*ps + (-y/(z-z0))*qs + 1)
%            ---------------------------------------------------------------
%            ((sqrt(p_s^2+q_s^2+1)*sqrt((-x/(z-z0))^2+(-y/(z-z0))^2+1))
%
%   Therefore,
%   E(x,y) = (-x*p_s - y*q_s + (z-z0))/(r*sqrt(p_s*p_s+qs*q_s+1))
for srcs = 1:1%size(p_s,2)
    E = zeros(sz,sz);
    ps = p_s(srcs);
    qs = q_s(srcs);
    for j = 1:sz
        y = mid - j;
        for i = 1:sz
            x = mid - i;
            if(x*x + y*y < rds*rds)
                z_z0 = sqrt(rds*rds-(x*x + y*y));
                E(i,j) = (-x*ps-y*qs+z_z0)/(rds*sqrt(ps*ps+qs*qs+1));
            end
        end
    end

    % Display
    wintitle = sprintf('E(x,y) for source (ps,qs) = (%.1f,%.1f)',ps,qs);
    hdl_E = figure; imshow(E), title(wintitle);
end
%-------------------------------------------------------------------------

%-------------------------------------------------------------------------
%% Actual p and q and z
% Assumption: Value of z0
ps = p_s(1);
qs = q_s(1);
sqrt_psqs = sqrt(ps.^2+qs.^2+1);
p_mat = zeros(sz,sz);
q_mat = zeros(sz,sz);
for j = 1:sz
    y = mid - j;
    for i = 1:sz
        x = mid - i;
        if(x*x + y*y < rds*rds)
            z_z0 = sqrt(rds*rds-(x*x + y*y));
            p_mat(i,j) = -x/z_z0;
```

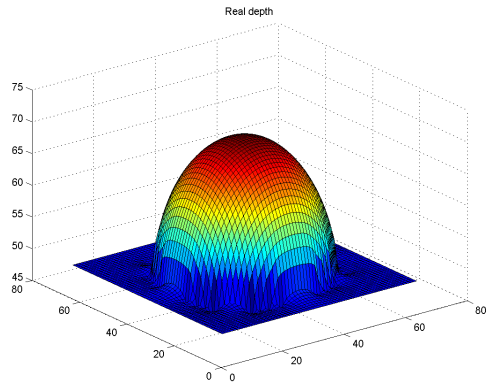
```
            q_mat(i,j) = -y/z_z0;
        end
    end
end

%% Actual F and G from actual P and Q
f_mat = zeros(sz,sz);
g_mat = zeros(sz,sz);
for j = 1:sz
    for i = 1:sz
        f_mat(i,j) = 2*p_mat(i,j)/(1+sqrt(p_mat(i,j).^2+q_mat(i,j).^2+1));
        g_mat(i,j) = 2*q_mat(i,j)/(1+sqrt(p_mat(i,j).^2+q_mat(i,j).^2+1));
    end
end

%%  Calculation of Real Depth using Real values of p and q
%   Assuming the image plane to be kept at a distance z0 from the source,
%   we can consider z0 to be greater than the radius of the sphere.
%   Because p(x,y) = z(x+1,y) - z(x,y), p(x-1,y) = z(x,y) - z(x-1,y) and
%   q(x,y) = z(x,y+1) - z(x,y), q(x,y) = z(x,y) - z(x,y-1), we can write,
%   4*z(x,y) = (p(x-1,y)-p(x,y) + q(x,y-1) - q(x,y)) + ...
%               (z(x+1,y) + z(x-1,y) + z(x,y+1) + z(x,y-1))
%   Assuming some initial value for z matrix (let it be z0), after some
%   iterations the z matrix will not update, where one can stop.
z_old = ones(sz,sz)*z0;
z_mat = z_old;
itrn = 0;
while(1)
    itrn = itrn + 1;
    for j = 2:sz-1
        for i = 2:sz-1
            z_mat(i,j) = (p_mat(i,j)-p_mat(i-1,j)+q_mat(i,j)-q_mat(i,j-1)+...
                    z_old(i+1,j)+z_old(i-1,j)+z_old(i,j+1)+z_old(i,j-1))*(1/4);
        end
    end
    if(z_mat==z_old)
        fprintf('Actual depth calcuation converged in %d iterations.\n',itrn);
        break;
    else
        z_old = z_mat;
    end
end

hdl_zreal = figure; surf(z_mat), title('Real depth');

%% PQ Initial conditions
% pn = ones(sz,sz);
% qn = ones(sz,sz);
pn = zeros(sz,sz);
qn = zeros(sz,sz);
rds2 = rds-1;
```

```
for j = 1:sz
    y = mid - j;
    for i = 1:sz
        x = mid - i;
        if(x*x + y*y < rds2*rds2)
            z_z0 = sqrt(rds2*rds2-(x*x + y*y));
            pn(i,j) = -x/z_z0;
            qn(i,j) = -y/z_z0;
        end
    end
end
i_pn = pn;          % Starting P
i_qn = qn;          % Starting Q

%% FG Initial conditions
fn = zeros(sz,sz);
gn = zeros(sz,sz);
for j = 1:sz
    for i = 1:sz
        fn(i,j) = 2*i_pn(i,j)/(1+sqrt(i_pn(i,j).^2+i_qn(i,j).^2+1));
        gn(i,j) = 2*i_qn(i,j)/(1+sqrt(i_pn(i,j).^2+i_qn(i,j).^2+1));
    end
end
i_fn = fn;
i_gn = gn;
%--------------------------------------------------------------------------

%% Save all figures (print command)
print(hdl_sphr,'-v','-dpng', './Results (a)/sphere.png');
print(hdl_E,'-v','-dpng', './Results (a)/shade.png');
print(hdl_zreal,'-v','-dpng', './Results (a)/depth.png');
close all;
%--------------------------------------------------------------------------
```

Results








```
%-------------------------------------------------------------------------
%%  Part 2(a) : PQ domain
%% Iterative calculation of p and q
pn1 = pn.*0;        % New P
qn1 = qn.*0;        % New Q
orgE = E;           % Input image
itrn = 0;
lambda = reg_lambda;
while(itrn<max_itrn)
    itrn = itrn + 1;
    for j = 2:sz-1
        for i = 2:sz-1
            % Neighbourhood
            p_ngbh = (1/4)*(pn(i-1,j)+pn(i+1,j)+pn(i,j-1)+pn(i,j+1));
            q_ngbh = (1/4)*(qn(i-1,j)+qn(i+1,j)+qn(i,j-1)+qn(i,j+1));

            sqrt_pq = sqrt(pn(i,j).^2+qn(i,j).^2+1);

            % Rpq = R(p,q) = (p*ps+q*qs+1)/(sqrt(ps^2+qs^2+1)*(sqrt(p^2+q^2+1)))
            Rpq = (pn(i,j)*ps + qn(i,j)*qs + 1)/(sqrt_psqs*sqrt_pq);

            % dR/dp = Rp = (ps(q^2+1)-p(q*qs+1))/(sqrt(ps^2+qs^2+1)*(sqrt(p^2+q^2+1))^3)
            Rp = (ps*(qn(i,j).^2+1) - pn(i,j)*(qn(i,j)*qs+1))/(sqrt_psqs*(sqrt_pq^3));

            % dR/dq = Rq = (qs(p^2+1)-q(p*ps+1))/(sqrt(ps^2+qs^2+1)*(sqrt(p^2+q^2+1))^3)
            Rq = (qs*(pn(i,j).^2+1) - qn(i,j)*(pn(i,j)*ps+1))/(sqrt_psqs*(sqrt_pq^3));

            % New p and q
            pn1(i,j) = p_ngbh + (lambda/4)*(orgE(i,j) - Rpq)*Rp;
            qn1(i,j) = q_ngbh + (lambda/4)*(orgE(i,j) - Rpq)*Rq;
        end
    end
    if(pn1==pn & qn1==qn)
        fprintf('PQ recovery converged in %d iterations.\n',itrn);
        break;
    else
        pn = pn1;
```

```
        qn = qn1;
%         lambda = lambda - (lambda/10);
    end
end

hdl_2recP = figure; imshow(pn1), title('Recovered P');
hdl_2recQ = figure; imshow(qn1); title('Recovered Q');

%% Depth calculation using recovered p and q in PQ domain
zn = ones(sz,sz)*z0;
zn1 = zn;
itrn = 0;
while(1)
    itrn = itrn + 1;
    for j = 2:sz-1
        for i = 2:sz-1
            zn1(i,j) = (1/4)*(zn(i+1,j) + zn(i-1,j) + zn(i,j+1) + zn(i,j-1)) + (pn1(i,j)-pn1(i-1,j)) +
        end
    end
    if(zn1==zn)
        fprintf('PQ Depth recovery converged in %d iterations.\n',itrn);
        break;
    else
        zn = zn1;
    end
end

hdl_2recZpq = figure; surf(zn1), title('Recovered depth in PQ domain');

%% Save all figures (print command)
print(hdl_2recP,'-v','-dpng', './Results (b)/2_recP.png');
print(hdl_2recQ,'-v','-dpng', './Results (b)/2_recQ.png');
print(hdl_2recZpq,'-v','-dpng', './Results (b)/2_recZ.png');
close all;

%% MSE calculation
PQp_err_mat = (p_mat - pn1).^2;
PQp_err = sum(PQp_err_mat(:))/numel(PQp_err_mat);
PQq_err_mat = (q_mat - qn1).^2;
PQq_err = sum(PQq_err_mat(:))/numel(PQq_err_mat);
PQz_err_mat = (z_mat - zn1).^2;
PQz_err = sum(PQz_err_mat(:))/numel(PQz_err_mat);
%--------------------------------------------------------------------------

%%  Part 2(b) : FG domain

%% Iterative calculation of F and G
% Now, f = 2p/(1+sqrt(p^2+q^2+1)), g = 2q/(1+sqrt(p^2+q^2+1))
% p = 4f/(4-(f^2+g^2)), q = 4g/(4-(f^2+g^2))
% R(p,q) = (p*ps+q*qs+1)/(sqrt(ps^2+qs^+1)*sqrt(p^2+q^2+1))
%        = (4*f*ps+4*g*qs+4-f^2-g^2)/(sqrt(ps^2+qs^2+1)*(4+f^2+g^2))
```

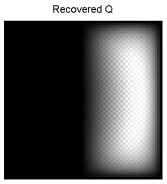
```
%
% dRf = (16*ps-4*ps*f^2+4*ps*g-16*f)/(sqrt(ps^2+qs^2+1)*((4+f^2+g^2)^2))
% dRg = (16*qs-4*qs*g^2+4*qs*g-16*g)/(sqrt(ps^2+qs^2+1)*((4+f^2+g^2)^2))
% fn1(i,j) = fn_avg(i,j) + (lambda/4)*(E-R)*dRf
% gn1(i,j) = gn_avg(i,j) + (lambda/4)*(E-R)*dRg
fn1 = fn.*0;
gn1 = gn.*0;
itrn = 0;
lambda = reg_lambda;
while(itrn<max_itrn)
    itrn = itrn + 1;
    for j = 2:sz-1
        for i = 2:sz-1
            % Neighbourhood
            f_ngbh = (1/4)*(fn(i-1,j)+fn(i+1,j)+fn(i,j-1)+fn(i,j+1));
            g_ngbh = (1/4)*(gn(i-1,j)+gn(i+1,j)+gn(i,j-1)+gn(i,j+1));

            sqr_sum_fg = fn(i,j)^2 + gn(i,j)^2;

            % Rpq = R(p,q) = = (4*f*ps+4*g*qs+4-f^2-g^2)/(sqrt(ps^2+qs^2+1)*(4+f^2+g^2))
            Rfg = (4*fn(i,j)*ps + 4*gn(i,j)*qs + 4 - sqr_sum_fg)/(sqrt_psqs*(4 + sqr_sum_fg));

            % dR/df = Rf = (16*ps-4*ps*f^2+4*ps*g-16*f)/(sqrt(ps^2+qs^2+1)*((4+f^2+g^2)^2))
            Rf = (16*ps - 4*ps*(fn(i,j)^2) + 4*ps*gn(i,j) - 16*fn(i,j))/(sqrt_psqs*((4+sqr_sum_fg)^2));

            % dR/dg = Rg = (16*qs-4*qs*g^2+4*qs*g-16*g)/(sqrt(ps^2+qs^2+1)*((4+f^2+g^2)^2))
            Rg = (16*qs - 4*qs*(gn(i,j)^2) + 4*qs*fn(i,j) - 16*gn(i,j))/(sqrt_psqs*((4+sqr_sum_fg)^2));

            % New p and q
            fn1(i,j) = f_ngbh + (lambda/4)*(orgE(i,j) - Rfg)*Rf;
            gn1(i,j) = g_ngbh + (lambda/4)*(orgE(i,j) - Rfg)*Rg;
        end
    end

%     if(fn1==fn & gn1==gn)
%         fprintf('FG recovery converged in %d iterations.\n',itrn);
%         break;
%     else
        fn = fn1;
        gn = gn1;
%         lambda = lambda - (lambda/10);
%     end
end

hdl_2recF = figure; imshow(fn1), title('Recovered F');
hdl_2recG = figure; imshow(gn1), title('Recovered G');

%% Recover P and Q from F and G
pn1f = fn.*0;
qn1g = gn.*0;
for i = 1:sz
```

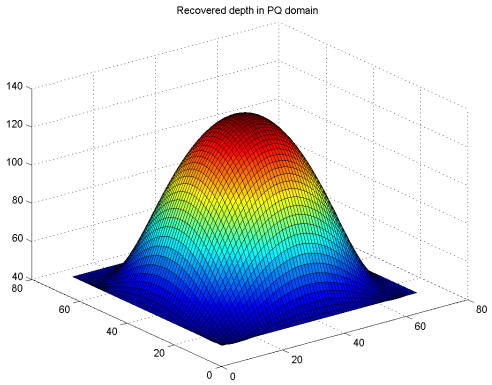
```
    for j = 1:sz
        pn1f(i,j) = 4*fn1(i,j)/(4-(fn1(i,j)^2)-(gn1(i,j)^2));
        qn1g(i,j) = 4*gn1(i,j)/(4-(fn1(i,j)^2)-(gn1(i,j)^2));
    end
end

hdl_2recPfg = figure; imshow(pn1f), title('Recovered P from FG');
hdl_2recQfg = figure; imshow(qn1g), title('Recovered Q from FG');

%% Depth calculation in FG domain using estimated PQ
znfg = ones(sz,sz)*z0;
zn1fg = znfg;
itrn = 0;
while(1)
    itrn = itrn + 1;
    for j = 2:sz-1
        for i = 2:sz-1
            zn1fg(i,j) = (1/4)*(znfg(i+1,j) + znfg(i-1,j) + znfg(i,j+1) + znfg(i,j-1)) + (pn1f(i,j)-pn1
        end
    end
    if(zn1fg==znfg)
        fprintf('FG Depth recovery converged in %d iterations.\n',itrn);
        break;
    else
        znfg = zn1fg;
    end
end

hdl_2recZfg = figure; surf(zn1fg), title('Recovered depth in FG domain');

%% Save all figures (print command)
print(hdl_2recF,'-v','-dpng', './Results (b)/2_recF.png');
print(hdl_2recG,'-v','-dpng', './Results (b)/2_recG.png');
print(hdl_2recPfg,'-v','-dpng', './Results (b)/2_recPfg.png');
print(hdl_2recQfg,'-v','-dpng', './Results (b)/2_recQfg.png');
print(hdl_2recZfg,'-v','-dpng', './Results (b)/2_recZfg.png');
close all;

%% MSE calculation
FGp_err_mat = (p_mat - pn1f).^2;
FGp_err = sum(FGp_err_mat(:))/numel(FGp_err_mat);
FGq_err_mat = (q_mat - qn1g).^2;
FGq_err = sum(FGq_err_mat(:))/numel(FGq_err_mat);
FGz_err_mat = (z_mat - zn1fg).^2;
FGz_err = sum(FGz_err_mat(:))/numel(FGz_err_mat);

%% Show error comparison
fprintf('PQp_err = %f\tFGp_err = %f\n',PQp_err,FGp_err);
fprintf('PQq_err = %f\tFGq_err = %f\n',PQq_err,FGq_err);
fprintf('PQz_err = %f\tFGz_err = %f\n',PQz_err,FGz_err);
fprintf('-------------------------------------------\n');
```

%----------

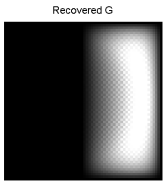
Results

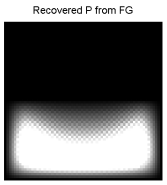

Recovered P

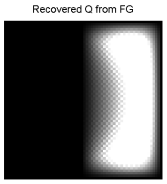

Recovered Q

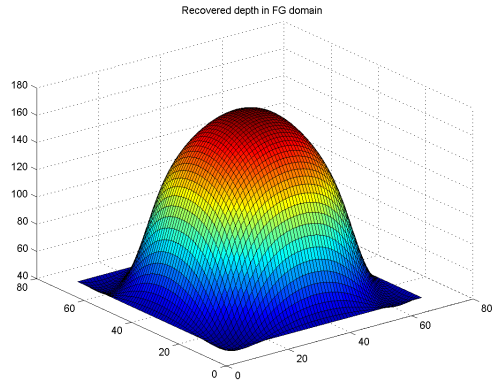

Recovered depth in PQ domain
140
120
100
80
60
40
80
60
40
20
0
0
20
40
60
80


Recovered F


Recovered G


Recovered P from FG



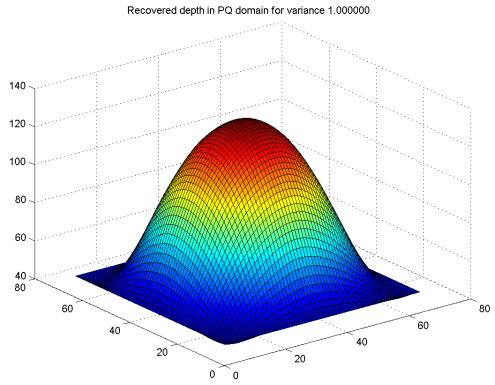


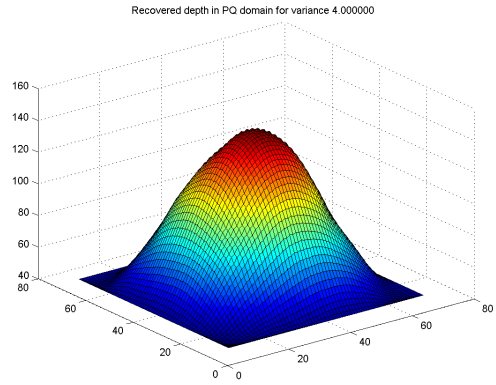

```
%-------------------------------------------------------------------------
%%   Part 3(a): Recovery from noisy image
%-------------------------------------------------------------------------
mse_var_fn = zeros(fn_pts,4);
for pt = 1:fn_pts
    cur_var = pt^2;
    mse_var_fn(pt,1) = cur_var;
    curE = orgE + cur_var*randn(size(orgE));
    cur_pn = i_pn;
    cur_qn = i_qn;
    cur_pn1 = cur_pn;
    cur_qn1 = cur_qn;

    % PQ caluclation
    itrn = 0;
    lambda = reg_lambda;
    while(itrn<max_itrn)
        itrn = itrn + 1;
        for j = 2:sz-1
            for i = 2:sz-1
                % Neighbourhood
                p_ngbh = (1/4)*(cur_pn(i-1,j)+cur_pn(i+1,j)+cur_pn(i,j-1)+cur_pn(i,j+1));
                q_ngbh = (1/4)*(cur_qn(i-1,j)+cur_qn(i+1,j)+cur_qn(i,j-1)+cur_qn(i,j+1));

                sqrt_pq = sqrt(cur_pn(i,j).^2+cur_qn(i,j).^2+1);

                % Rpq = R(p,q) = (p*ps+q*qs+1)/(sqrt(ps^2+qs^2+1)*(sqrt(p^2+q^2+1)))
                Rpq = (cur_pn(i,j)*ps + cur_qn(i,j)*qs + 1)/(sqrt_psqs*sqrt_pq);

                % dR/dp = Rp = (ps(q^2+1)-p(q*qs+1))/(sqrt(ps^2+qs^2+1)*(sqrt(p^2+q^2+1))^3)
                Rp = (ps*(cur_qn(i,j).^2+1) - cur_pn(i,j)*(cur_qn(i,j)*qs+1))/(sqrt_psqs*(sqrt_pq^3));

                % dR/dq = Rq = (qs(p^2+1)-q(p*ps+1))/(sqrt(ps^2+qs^2+1)*(sqrt(p^2+q^2+1))^3)
                Rq = (qs*(cur_pn(i,j).^2+1) - cur_qn(i,j)*(cur_pn(i,j)*ps+1))/(sqrt_psqs*(sqrt_pq^3));

                % New p and q
                cur_pn1(i,j) = p_ngbh + (lambda/4)*(curE(i,j) - Rpq)*Rp;
                cur_qn1(i,j) = q_ngbh + (lambda/4)*(curE(i,j) - Rpq)*Rq;
```

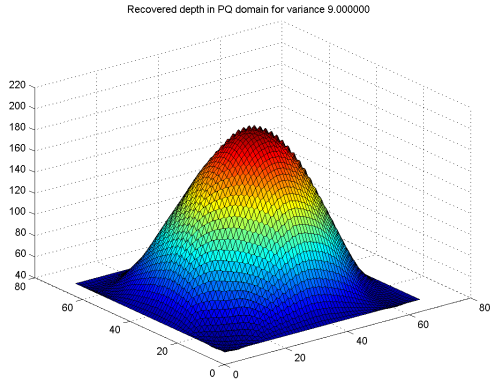
```
            end
        end
        if(cur_pn1==cur_pn & cur_qn1==cur_qn)
            fprintf('curPQ recovery converged in %d iterations for variance %f.\n',itrn,cur_var);
            break;
        else
            cur_pn = cur_pn1;
            cur_qn = cur_qn1;
%             lambda = lambda - (lambda/10);
        end
    end

    % Depth estimation
    cur_zn = ones(sz,sz)*z0;
    cur_zn1 = cur_zn;
    itrn = 0;
    while(1)
        itrn = itrn + 1;
        for j = 2:sz-1
            for i = 2:sz-1
                cur_zn1(i,j) = (1/4)*(cur_zn(i+1,j) + cur_zn(i-1,j) + cur_zn(i,j+1) + cur_zn(i,j-1)) +
            end
        end
        if(cur_zn1==cur_zn)
            fprintf('curPQ Depth recovery converged in %d iterations for variance %f.\n',itrn,cur_var);
            break;
        else
            cur_zn = cur_zn1;
        end
    end

    hdl_varZ = figure; surf(cur_zn1), title(sprintf('Recovered depth in PQ domain for variance %f',cur_

    %% Save figure (print command)
    print(hdl_varZ,'-v','-dpng', sprintf('./Results (c)/Z_var_%d.png',cur_var));
    close all;

    %% MSE calculation
    curPQp_err_mat = (p_mat - cur_pn1).^2;
    mse_var_fn(pt,2) = sum(curPQp_err_mat(:))/numel(curPQp_err_mat);
    curPQq_err_mat = (q_mat - cur_qn1).^2;
    mse_var_fn(pt,3) = sum(curPQq_err_mat(:))/numel(curPQq_err_mat);
    curPQz_err_mat = (z_mat - cur_zn1).^2;
    mse_var_fn(pt,4) = sum(curPQz_err_mat(:))/numel(curPQz_err_mat);

    %% Show error comparison
    fprintf('Errors for noisy image with variance = %d\n',cur_var);
    fprintf('PQp_err = %f\n',mse_var_fn(pt,2));
    fprintf('PQq_err = %f\n',mse_var_fn(pt,3));
    fprintf('PQz_err = %f\n',mse_var_fn(pt,4));
    fprintf('-------------------------------------------\n');
```

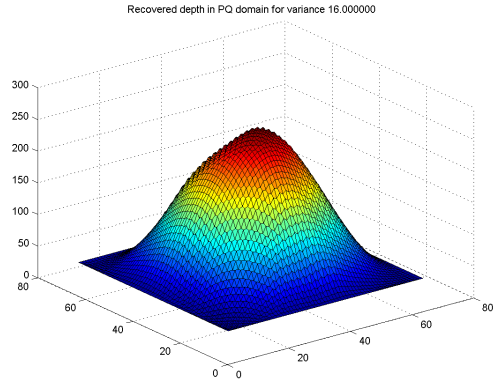
```
end
%--------------------------------------------------------------------------
```

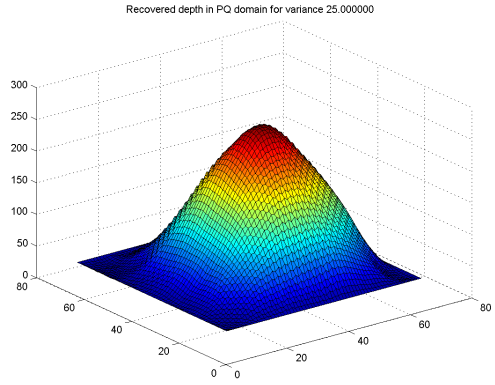
Results











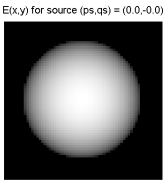

```
%--------------------------------------------------------------------------
%%   Part 4: Recovery from varying error in ps and qs
%--------------------------------------------------------------------------

% Calculating p and q and finally R(p(x,y),q(x,y)) = E(x,y)
```

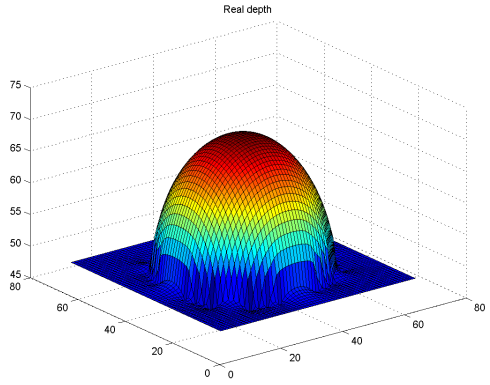
```
E = zeros(sz,sz);
ps = ps+0.001;
qs = qs-0.001;
sqrt_psqs = sqrt(ps^2+qs^2+1);
p_mat = zeros(sz,sz);
q_mat = zeros(sz,sz);
z_mat = zeros(sz,sz);
for j = 1:sz
    y = mid - j;
    for i = 1:sz
        x = mid - i;
        if(x*x + y*y < rds*rds)
            z_z0 = sqrt(rds*rds-(x*x + y*y));
            p_mat(i,j) = -x/z_z0;
            q_mat(i,j) = -y/z_z0;
            E(i,j) = (-x*ps-y*qs+z_z0)/(rds*sqrt(ps*ps+qs*qs+1));
            z_mat(i,j) = z_z0 + z0;
        end
    end
end

% Display
wintitle = sprintf('E(x,y) for source (ps,qs) = (%.1f,%.1f)',ps,qs);
hdl_E = figure; imshow(E), title(wintitle);

%  Calculation of Real Depth using Real values of p and q
z_old = ones(sz,sz)*z0;
z_mat = z_old;
itrn = 0;
while(1)
    itrn = itrn + 1;
    for j = 2:sz-1
        for i = 2:sz-1
            z_mat(i,j) = (p_mat(i,j)-p_mat(i-1,j)+q_mat(i,j)-q_mat(i,j-1)+...
                    z_old(i+1,j)+z_old(i-1,j)+z_old(i,j+1)+z_old(i,j-1))*(1/4);
        end
    end
    if(z_mat==z_old)
        fprintf('Actual depth calcuation converged in %d iterations.\n',itrn);
        break;
    else
        z_old = z_mat;
    end
end

hdl_Zreal = figure; surf(z_mat), title('Real depth');

% PQ Initial conditions
pn = zeros(sz,sz);
qn = zeros(sz,sz);
rds2 = rds-1;
```

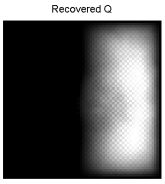
```
for j = 1:sz
    y = mid - j;
    for i = 1:sz
        x = mid - i;
        if(x*x + y*y < rds2*rds2)
            z_z0 = sqrt(rds2*rds2-(x*x + y*y));
            pn(i,j) = -x/z_z0;
            qn(i,j) = -y/z_z0;
        end
    end
end
i_pn = pn;          % Starting P
i_qn = qn;          % Starting Q

% Iterative calculation of p and q
pn1 = pn.*0;        % New P
qn1 = qn.*0;        % New Q
curE = E + 1*randn(size(E));
itrn = 0;
lambda = reg_lambda;
while(itrn<max_itrn)
    itrn = itrn + 1;
    for j = 2:sz-1
        for i = 2:sz-1
            % Neighbourhood
            p_ngbh = (1/4)*(pn(i-1,j)+pn(i+1,j)+pn(i,j-1)+pn(i,j+1));
            q_ngbh = (1/4)*(qn(i-1,j)+qn(i+1,j)+qn(i,j-1)+qn(i,j+1));

            sqrt_pq = sqrt(pn(i,j).^2+qn(i,j).^2+1);

            % Rpq = R(p,q) = (p*ps+q*qs+1)/(sqrt(ps^2+qs^2+1)*(sqrt(p^2+q^2+1)))
            Rpq = (pn(i,j)*ps + qn(i,j)*qs + 1)/(sqrt_psqs*sqrt_pq);

            % dR/dp = Rp = (ps(q^2+1)-p(q*qs+1))/(sqrt(ps^2+qs^2+1)*(sqrt(p^2+q^2+1))^3)
            Rp = (ps*(qn(i,j).^2+1) - pn(i,j)*(qn(i,j)*qs+1))/(sqrt_psqs*(sqrt_pq^3));

            % dR/dq = Rq = (qs(p^2+1)-q(p*ps+1))/(sqrt(ps^2+qs^2+1)*(sqrt(p^2+q^2+1))^3)
            Rq = (qs*(pn(i,j).^2+1) - qn(i,j)*(pn(i,j)*ps+1))/(sqrt_psqs*(sqrt_pq^3));

            % New p and q
            pn1(i,j) = p_ngbh + (lambda/4)*(curE(i,j) - Rpq)*Rp;
            qn1(i,j) = q_ngbh + (lambda/4)*(curE(i,j) - Rpq)*Rq;
        end
    end
    if(pn1==pn & qn1==qn)
        fprintf('PQ recovery converged in %d iterations.\n',itrn);
        break;
    else
        pn = pn1;
        qn = qn1;
%         lambda = lambda - (lambda/10);
```

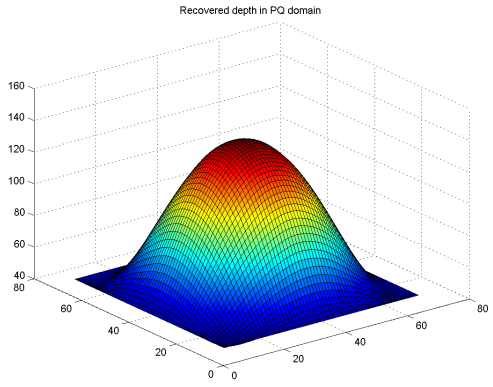
```
    end
end

hdl_P = figure; imshow(pn1), title('Recovered P');
hdl_Q = figure; imshow(qn1), title('Recovered Q');

% Depth calculation using recovered p and q in PQ domain
zn = ones(sz,sz)*z0;
zn1 = zn;
itrn = 0;
while(1)
    itrn = itrn + 1;
    for j = 2:sz-1
        for i = 2:sz-1
            zn1(i,j) = (1/4)*(zn(i+1,j) + zn(i-1,j) + zn(i,j+1) + zn(i,j-1)) + (pn1(i,j)-pn1(i-1,j)) +
        end
    end
    if(zn1==zn)
        fprintf('PQ Depth recovery converged in %d iterations.\n',itrn);
        break;
    else
        zn = zn1;
    end
end

hdl_Zrec = figure; surf(zn1), title('Recovered depth in PQ domain');

%% Save all figures (print command)
print(hdl_E,'-v','-dpng', './Results (d)/diffshade2.png');
print(hdl_Zreal,'-v','-dpng', './Results (d)/diffreal_depth.png');
print(hdl_P,'-v','-dpng', './Results (d)/diffPrec.png');
print(hdl_Q,'-v','-dpng', './Results (d)/diffQrec.png');
print(hdl_Zrec,'-v','-dpng', './Results (d)/diffZrec.png');
close all;

% MSE calculation
PQp_err_mat = (p_mat - pn1).^2;
PQp_err = sum(PQp_err_mat(:))/numel(PQp_err_mat);
PQq_err_mat = (q_mat - qn1).^2;
PQq_err = sum(PQq_err_mat(:))/numel(PQq_err_mat);
PQz_err_mat = (z_mat - zn1).^2;
PQz_err = sum(PQz_err_mat(:))/numel(PQz_err_mat);

 %% Show error comparison
 fprintf('Errors for image formed due to source (%d,%d)\n',ps,qs);
 fprintf('PQp_err = %f\n',PQp_err);
 fprintf('PQq_err = %f\n',PQq_err);
 fprintf('PQz_err = %f\n',PQz_err);
 fprintf('-------------------------------------------\n');

%-------------------------------------------------------------------------
```

Results









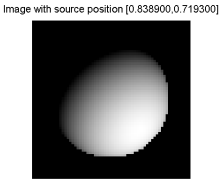

```
%-------------------------------------------------------------------------

%% Depth recovery given many source positions
% Source positions
ps_ary = [0.8389;0.5773;0.3638;0.1763;-0.1763;-0.3638;-0.5773;-0.8389];
qs_ary = [0.7193;0.6363;0.5865;0.5596;-0.5596;-0.5865;-0.6363;-0.7193];
num_src = length(ps_ary);
```

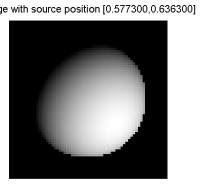
```
%%  Calculation of E(x,y) for each source position
%   p = -x/(z-z0) and q = -y/(z-z0);    (z-z0)^2 + (x^2+y^2) = r^2
%   E(x,y) = (p*ps + q*qs + 1)/(sqrt(ps^2+qs^2+1)*sqrt(p^2+q^2+1))
%          = ((-x/(z-z0))*ps + (-y/(z-z0))*qs + 1)
%            ------------------------------------------------------------------
%            ((sqrt(ps^2+qs^2+1)*sqrt((-x/(z-z0))^(-x/(z-z0))+(-y/(z-z0))*(-y/(z-z0))+1))
%          = (-x*ps - y*qs + (z-z0))/(r*sqrt(ps*ps+qs*qs+1))
E = zeros(sz,sz,num_src);
for pln = 1:num_src
    ps = ps_ary(pln,1);
    qs = qs_ary(pln,1);
    for j = 1:sz
        y = mid - j;
        for i = 1:sz
            x = mid - i;
            if(x*x + y*y < rds*rds)
                z_z0 = sqrt(rds*rds-(x*x + y*y));
                E(i,j,pln) = (-x*ps-y*qs+z_z0)/(rds*sqrt(ps*ps+qs*qs+1));
            end
        end
    end
    hdl_E = figure; imshow(E(:,:,pln)), title(sprintf('Image with source position [%f,%f]',ps,qs));
    print(hdl_E,'-v','-dpng', sprintf('./Results (d)/src_%d.png',pln));
    close all;
end

%%  Calculation of S matrix which is (num_src)x(3)
S = zeros(num_src,3);
for pln = 1:length(ps_ary)
    ps = ps_ary(pln,1);
    qs = qs_ary(pln,1);
    denom = sqrt(ps*ps+qs*qs+1);
    S(pln,1) = (-1)*ps/denom;
    S(pln,2) = (-1)*qs/denom;
    S(pln,3) = 1/denom;
end

%%  Calculation of S_inverse using SVD for solving S*n = Exy
[u s v] = svd(S);
st = s';
inv_st = 1./st;
inv_st(~isfinite(inv_st)) = 0;
inv_S = v*inv_st*u';

%%  Calculation of p_cap and q_cap using E(x,y) and S matrix
p_cap = zeros(sz,sz);
q_cap = zeros(sz,sz);
for j = 1:sz
    y = mid - j;
    for i = 1:sz
```

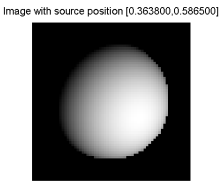
```
        x = mid - i;
        if(x*x+y*y<rds*rds)
            Exy = reshape(E(i,j,:),num_src,1);
            %%  Now Exy = S*n   assuming rho = 1,   n = inv_S*Exy
            %   n(1,1) = (-1)*p/(p*p+q*q+1)
            %   n(2,1) = (-1)*q/(p*p+q*q+1)
            %   n(3,1) = 1/(p*p+q*q+1)
            n = inv_S*Exy;
            if(n(3,1)==0)
                n(3,1) = eps;
            end
            p_cap(i,j) = (-1)*(n(1,1)/n(3,1));
            q_cap(i,j) = (-1)*(n(2,1)/n(3,1));
        end
    end
end
hdl_P = figure; imshow(abs(p_cap)), title('p\_cap');
hdl_Q = figure; imshow(abs(q_cap)), title('q\_cap');

%%  Calculation of Depth using recovered values viz p_cap and q_cap
% Just as above, z_cap can be calculated in a manner similar to that of
% obtaining z
z_cap = ones(sz,sz)*z0;
z_old = z_cap;
itrn = 0;
while(1)
    itrn = itrn + 1;
    for j = 1:sz
        y = mid - j;
        for i = 1:sz
            x = mid - i;
            if(x*x+y*y<rds*rds)
                z_cap(i,j) = (p_cap(i,j)-p_cap(i-1,j)+q_cap(i,j)-q_cap(i,j-1)+...
                    z_old(i+1,j)+z_old(i-1,j)+z_old(i,j+1)+z_old(i,j-1))*(1/4);
            end
        end
    end
    if(z_cap==z_old)
        break;
    else
        z_old = z_cap;
    end
end
hdl_Z = figure; surf(z_cap), title('Recovered Depth');

%% Save all figures (print command)
print(hdl_P,'-v','-dpng', './Results (d)/recP.png');
print(hdl_Q,'-v','-dpng', './Results (d)/recQ.png');
print(hdl_Z,'-v','-dpng', './Results (d)/recZ.png');
close all;
```

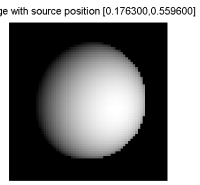
```
%%  Various squrared errors
p_error = sum(sum((p_mat-p_cap).*(p_mat-p_cap)));
q_error = sum(sum((q_mat-q_cap).*(q_mat-q_cap)));
z_error = sum(sum((z_mat-z_cap).*(z_mat-z_cap)));

%% Show error comparison
fprintf('Errors for multi-image recovery\n');
fprintf('PQp_err = %f\n',p_error);
fprintf('PQq_err = %f\n',q_error);
fprintf('PQz_err = %f\n',z_error);
fprintf('---------------------------------------------\n');
%-------------------------------------------------------------------------
```

Results









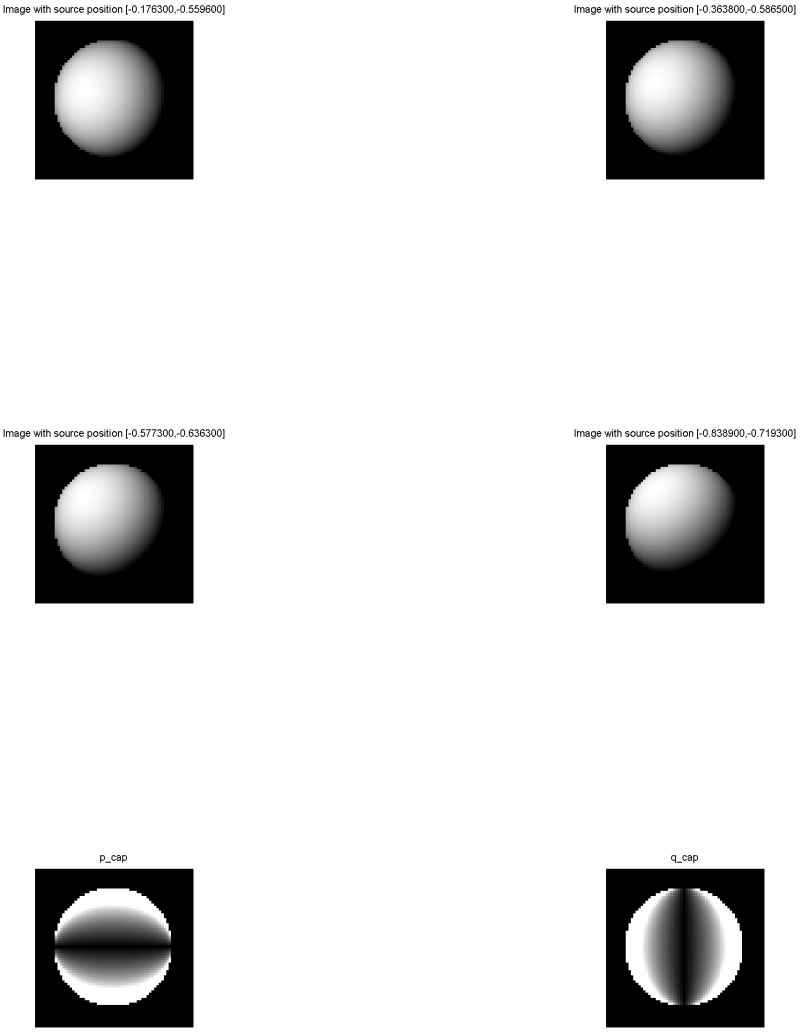
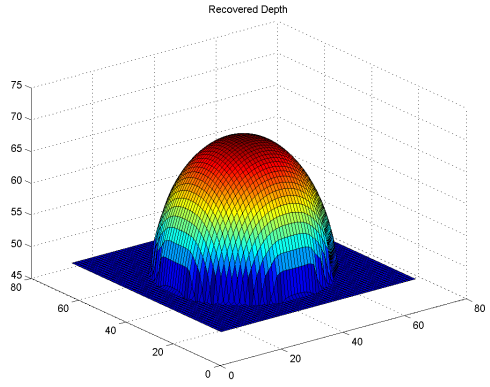


Image with source position [-0.176300,-0.559600]
Image with source position [-0.363800,-0.586500]
Image with source position [-0.577300,-0.636300]
Image with source position [-0.838900,-0.719300]
p_cap
q_cap



Command Window Output

```
Actual depth calcuation converged in 24176 iterations.
PQ Depth recovery converged in 25262 iterations.
FG Depth recovery converged in 25189 iterations.
PQp_err = 0.536680 FGp_err = 0.878690
PQq_err = 0.536680 FGq_err = 0.878690
PQz_err = 1100.221090 FGz_err = 3451.785828
-------------------------------------------
curPQ Depth recovery converged in 24975 iterations for variance 1.000000.
Errors for noisy image with variance = 1
PQp_err = 0.544947
PQq_err = 0.544551
PQz_err = 1020.414762
-------------------------------------------
curPQ Depth recovery converged in 25249 iterations for variance 4.000000.
Errors for noisy image with variance = 4
PQp_err = 0.607665
PQq_err = 0.607852
PQz_err = 1214.610618
-------------------------------------------
curPQ Depth recovery converged in 25146 iterations for variance 9.000000.
Errors for noisy image with variance = 9
PQp_err = 1.654301
PQq_err = 1.549102
PQz_err = 3229.491209
-------------------------------------------
curPQ Depth recovery converged in 25332 iterations for variance 16.000000.
Errors for noisy image with variance = 16
PQp_err = 2.706898
PQq_err = 3.789765
PQz_err = 7389.189577
-------------------------------------------
curPQ Depth recovery converged in 25249 iterations for variance 25.000000.
Errors for noisy image with variance = 25
PQp_err = 4.054465
PQq_err = 3.206611
PQz_err = 7195.349750
```

```
-------------------------------------------
Actual depth calcuation converged in 24176 iterations.
PQ Depth recovery converged in 25112 iterations.
Errors for image formed due to source (1.000000e-003,-1.000000e-003)
PQp_err = 0.538954
PQq_err = 0.542257
PQz_err = 1153.943497
-------------------------------------------
Errors for multi-image recovery
PQp_err = 0.000000
PQq_err = 0.000000
PQz_err = 270.216140
-------------------------------------------
```