CMPT 412 – Assignment 1

Jeremy J. Hartmann – 301144721 – jhartman@sfu.ca

1 INTRODUCTION

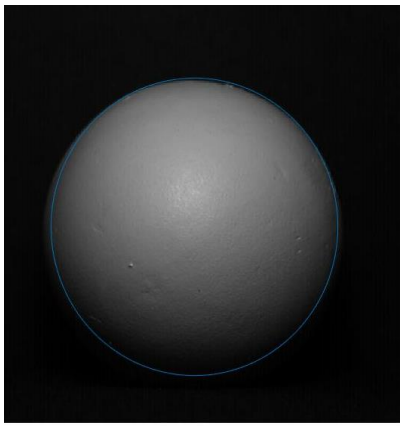
The photometric stereo algorithm has the following stages: 1) Calibration sphere phase, 2) lookup table phase, 3) F-G to P-Q Conversion phase, and 4) Integration phase.

NOTE: Make sure to run .m scripts in order: First, PhotometricStereo_CreateLookUp.m; Second, PhotometricStereo_BuildObject.m. The lookup table creation and object analysis are two separate .m files.

2 ALGORITHM

The following are the 4 stages:

2.1 CALIBRATION SPHERE



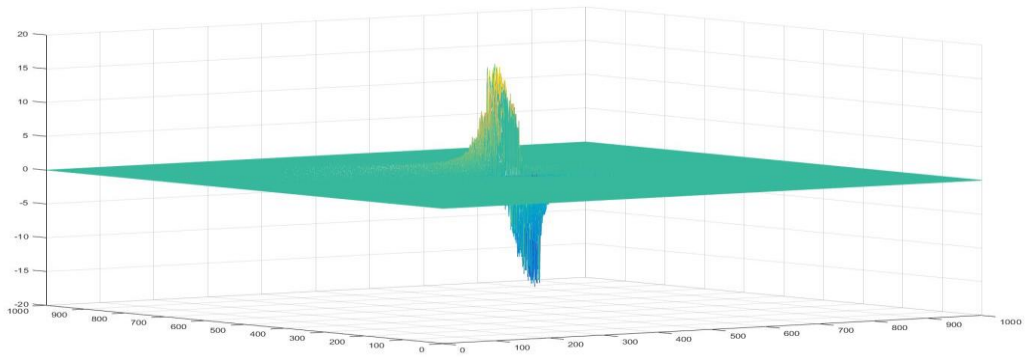
When the application starts, the user will input the center and end of the sphere with the ginput() method in Matlab. This will calculate the dimension of the sphere to create the stereo graphic projection (f, g) from the (x,y,z) coordinates from the sphere.

2.2 CREATE LOOKUP TABLE

After the sphere has extracted all (f, g) pairs from the sphere, the algorithm goes and starts building a lookup table based on the information obtained from the sphere. To handle collision in the data, the lookup table is comprised of small Matlab struct. These structs hold the values for f and g, and can handle multiple variations of f and g based on the given indices.

The indices for the table are determined by the ratios of the three images. A logarithmic calculation of the ratios are taken and the result is scaled by a scalar value (1000). The resulting table is 1000 * 1000.

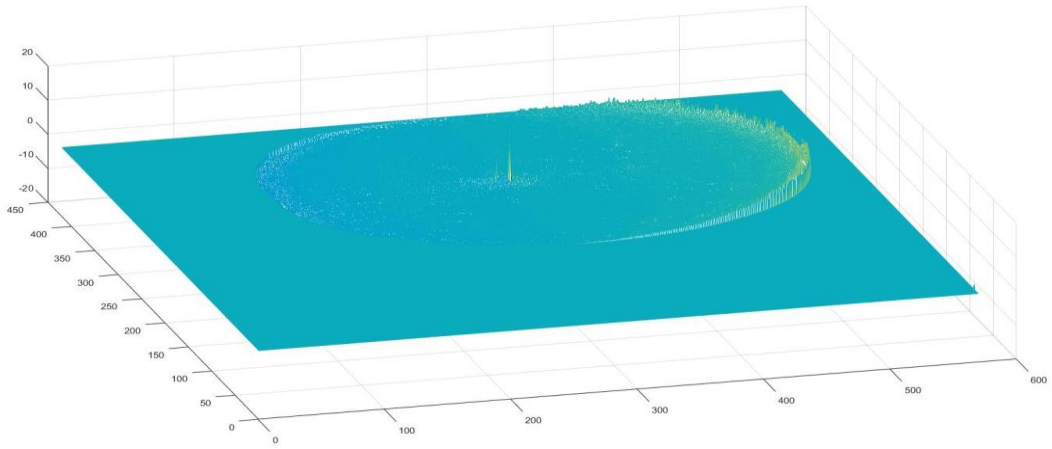
Once all information has been extracted and placed in the table, interpolation is performed across the 2D array using the griddata function in Matlab. Here, any missing information will be filled in.



2.3 Object Analysis FG Conversion

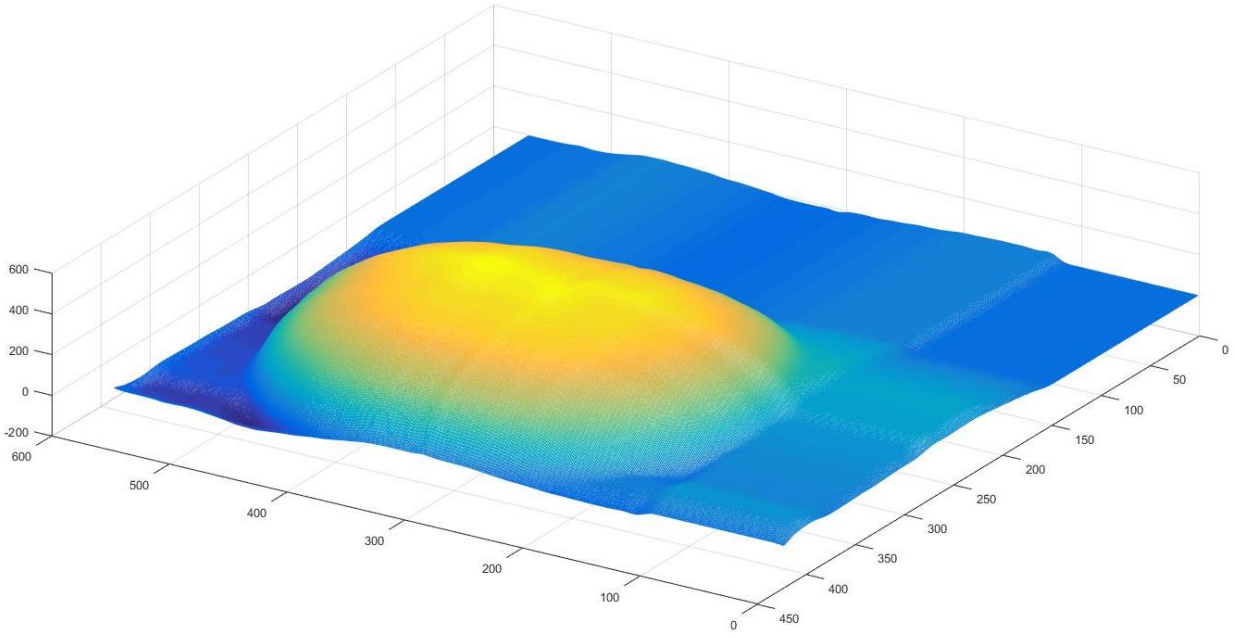
For the object analysis and <F G> pair conversion, the images are iterated over and the indices are computed based on the ratios produced by each images albedo. These X-Y value pairs are then used to index the lookup table to find the corresponding <F G> values.

Once the values are obtained, they are converted into <P Q> value pairs by taking the partial derivative of x/z and y/z. These are then placed into separate 2D arrays titled P and Q.



2.4 Integration Phase

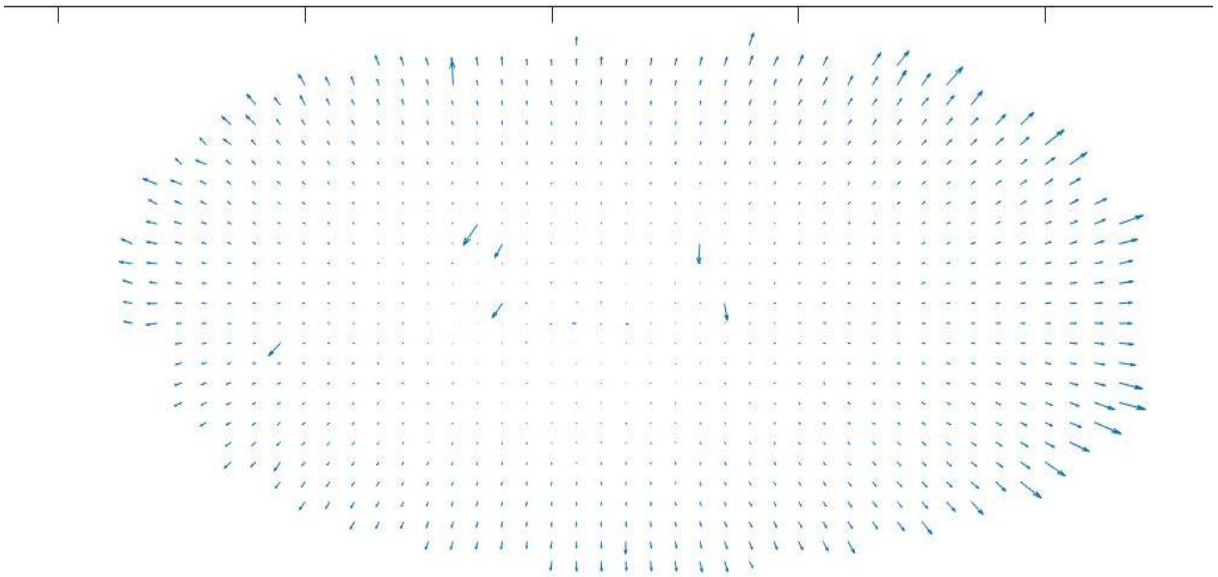
The 2D arrays for P and Q are then Integration over to reconstruct the image in the photo. The Integration adds paths form the left to right, right to left, up to down, and down to up directions to get the reconstructed image.



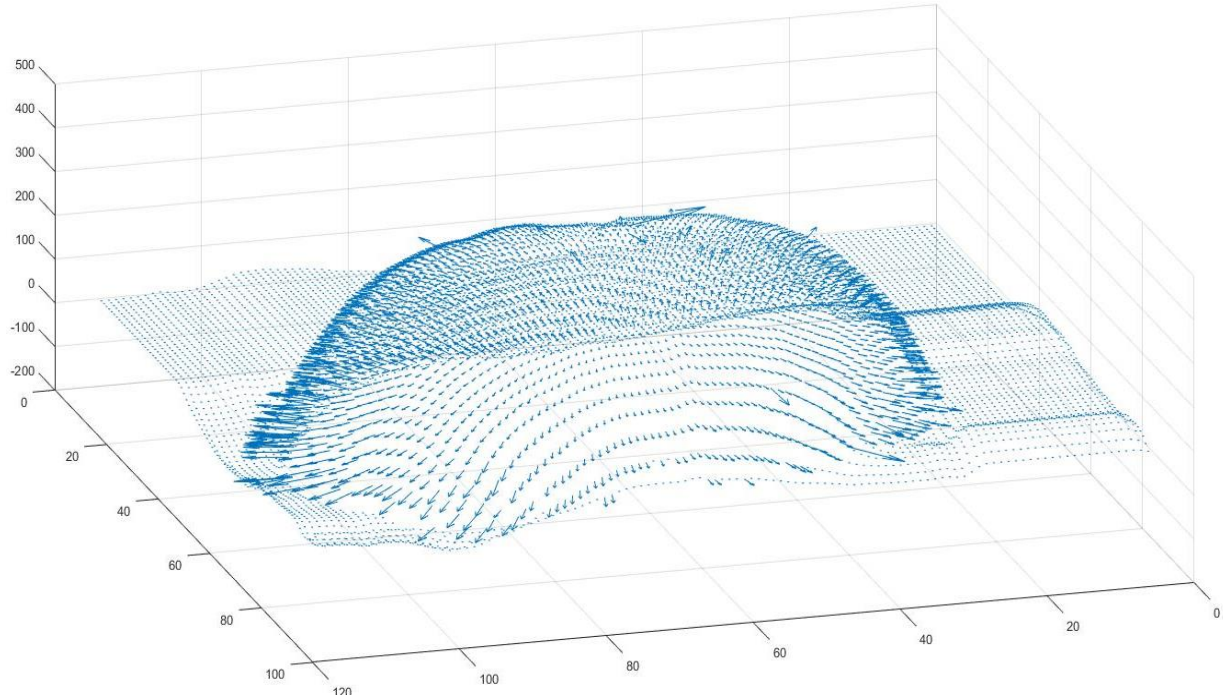
3 RESULTS

Here are some of the following results for the Photometric stereo algorithm.

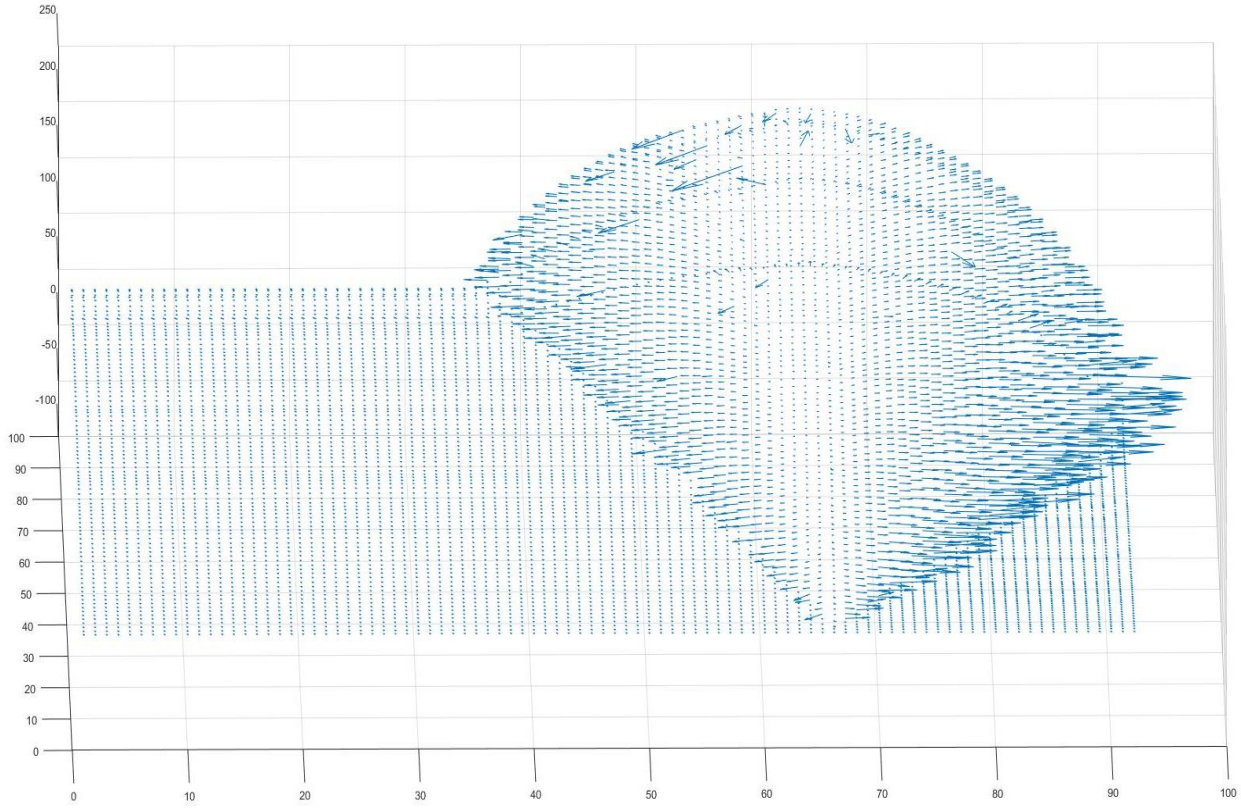
2D Quiver Plot of Ellipsoid



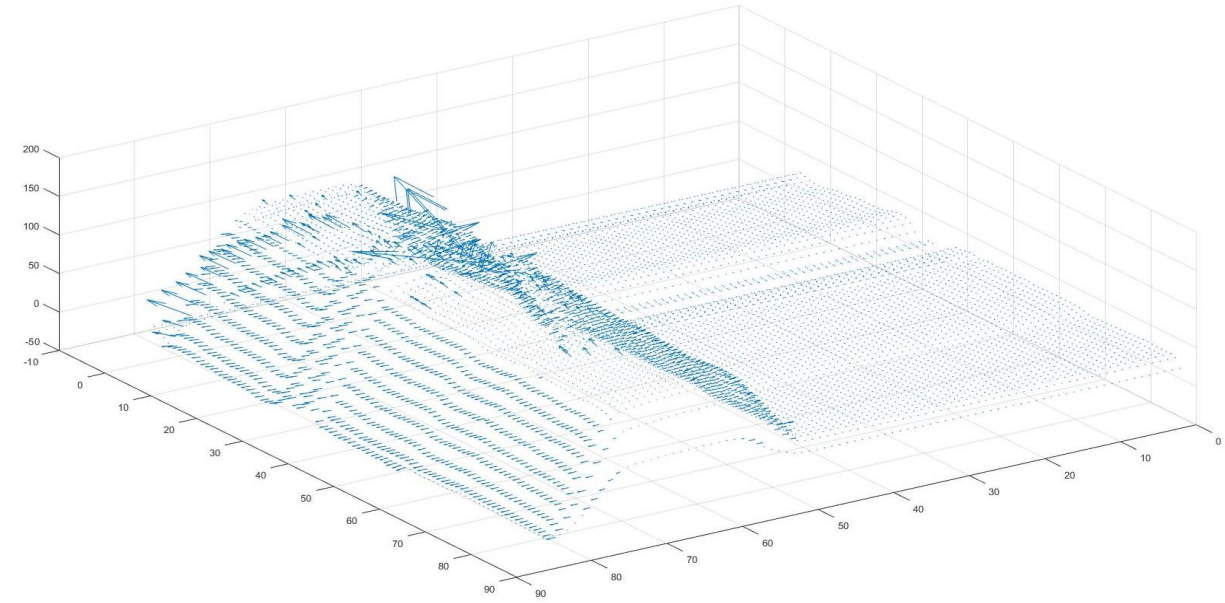
3D Quiver Plot of Ellipsoid



3D Quiver Plot Cone



3D Quiver Plot Hexagon



4 CONCLUSION AND COMMENTS

The photometric stereo method is absolutely fascinating. I really enjoyed doing this project and experimenting with different methods. My results are not always entirely accurate, but in general, I was surprised at how accurate the results could be.

Definitely will be exploring this some more in my spare time.

Photometric Stereo Code

5 CREATE LOOKUP TABLE:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Assingment 2 - Photostereo Imaging

% Calab Sphere Array
data = ['Photostereo_RealImages/sphere-lamp1.tif';
'Photostereo_RealImages/sphere-lamp2.tif'; 'Photostereo_RealImages/sphere-
lamp3.tif'];
sphereArray = cellstr(data);

%% Init calabration Sphere.
imgSphere = imread(sphereArray{3});

%Convert to gray
imgSphereGray = rgb2gray(imgSphere);
imshow(imgSphereGray)
[cx, cy] = ginput(1);
[px, py] = ginput(1);

% Find the radius of circle
radius = floor(sqrt((cx - px)^2 + (cy - py)^2));

% Draw circle on image
theta = 0 : (2 * pi /10000) : (2 * pi);
pline_x = radius * cos(theta) + cx;
pline_y = radius * sin(theta) + cy;
hold on;
plot(pline_x, pline_y)
hold off;

% Crop the image based on circle.
dim = [(cx - radius - 10) (cy - radius - 10) (2*radius + 10) (2*radius +
10)];
imgCropSphere3 = imcrop(imgSphereGray, dim);
imgCropSphere2 = rgb2gray(imcrop(imread(sphereArray{2}), dim));
imgCropSphere1 = rgb2gray(imcrop(imread(sphereArray{1}), dim));
figure(1)
imshow(imgCropSphere3)

% Get new center
[h, w] = size(imgCropSphere3);
ccx = w/2;
ccy = h/2;

%% Collect fg value pairs and E1, E2, and E3.

% Sphere 1: Iterate over inter image x and y coordinates and build f, g and E
values
```

```
fge = double(zeros(w * h, 7));
indRow = [];
index = 1;
testmat = zeros(h, w);
for x = 1:w
   for y = 1:h

       xx = ceil(x - ccx);
       yy = ceil(y - ccy);
       % check to make sure x,y is in cicle.
       rtmp = ceil(sqrt(xx^2 + yy^2));
       if (rtmp < radius)
          % Use sterographic projection to detect f and g
           zz = ceil(sqrt(radius^2 - (xx^2 + yy^2)));
           testmat(y, x) = zz;

           nxx = xx/radius;
           nyy = yy/radius;
           nzz = zz/radius;

           f = nxx/(1 - nzz);
           g = nyy/(1 - nzz);
           E1 = imgCropSphere1(y, x);
           E2 = imgCropSphere2(y, x);
           E3 = imgCropSphere3(y, x);

           % Check for infinit
           if (f == inf || isnan(f) || f == -inf)
              continue;
           end
           if (g == inf || isnan(g) || g == -inf)
              continue;
           end

           %Add to matrix
           % fge = [fge; double(x) double(y) double(f) double(g) double(E1)
double(E2) double(E3)];
           fge((x - 1) * (h - 1) + y, 1) = double(x);
           fge((x - 1) * (h - 1) + y, 2) = double(y);
           fge((x - 1) * (h - 1) + y, 3) = double(f);
           fge((x - 1) * (h - 1) + y, 4) = double(g);
           fge((x - 1) * (h - 1) + y, 5) = double(E1);
           fge((x - 1) * (h - 1) + y, 6) = double(E2);
           fge((x - 1) * (h - 1) + y, 7) = double(E3);
       else
           % build indeces of empty rows
            indRow(index) = (x - 1) * (h - 1) + y;
            index = index + 1;
       end
   end
end
fge = removerows(fge, indRow);

%% Build Lookup Table indexed by E1/E2 E2/E3
[fgesize, w]  = size(fge);
```

```
% TODO: Change look-up table to use log. Map the values between -5 to 5.
epsilon = 20;
BinScale = 100;
xsize = 10;
ysize = 10;
LookUpTable = [];
LookUpTable(ysize * BinScale, xsize * BinScale).f = [0 0];
LookUpTable(ysize * BinScale, xsize * BinScale).g = [0 0];
AvgFG = double(zeros(ysize * BinScale, xsize * BinScale));

E1E2Vec = [];
E2E3Vec = [];
fv = [];
gv = [];
for y = 1:fgesize
    E1E2 = ceil((log((fge(y,5) + 1)/(fge(y,6) + 1)) + 5) * BinScale);
    E2E3 = ceil((log((fge(y,6) + 1)/(fge(y,7) + 1)) + 5) * BinScale);

    f = fge(y,3);
    g = fge(y,4);

    curr = LookUpTable(E2E3, E1E2);
    if (isempty(curr.f))
        % Populate Container
        LookUpTable(E2E3, E1E2).f = f;
        LookUpTable(E2E3, E1E2).g = g;
        AvgFG(E2E3, E1E2) = double((f + g)/2);

        % build sample data
        E1E2Vec = [E1E2Vec; E1E2];
        E2E3Vec = [E2E3Vec; E2E3];
        fv = [fv;  double(f)];
        gv = [gv;  double(g)];
    else
        %% Check values and averge or new spot
        epsilon = 15;
        tmpf = LookUpTable(E2E3, E1E2).f(1);
        tmpg = LookUpTable(E2E3, E1E2).g(1);
        deltaf = double(abs(tmpf) - abs(f));
        deltag = double(abs(tmpg) - abs(g));

        % check value for f
        if ((f >= 0 && tmpf >= 0 && deltaf < epsilon) || (f <= 0 && tmpf < 0
&& deltaf < epsilon) || (abs(f) + abs(tmpf)) < epsilon)
            % Values are similar
             LookUpTable(E2E3, E1E2).f(1) = (tmpf + double(f))/2;
        else
            % store new value
            [t s] = size(LookUpTable(E2E3, E1E2).f);
            if (s < 2)
                LookUpTable(E2E3, E1E2).f(2) = 0;
            end
```

```
            tmpf = LookUpTable(E2E3, E1E2).f(2);
            LookUpTable(E2E3, E1E2).f(2) = (tmpf + double(f))/2;
        end

         % check value for g
        if ((g >= 0 && tmpg >= 0 && deltag < epsilon) || (g <= 0 && tmpg < 0
&& deltag < epsilon) || (abs(g) + abs(tmpg)) < epsilon)
            % Values are similar
             LookUpTable(E2E3, E1E2).g(1) = (tmpg + double(g))/2;
        else
            % store new value
          [t s] = size(LookUpTable(E2E3, E1E2).g);
          if (s < 2)
               LookUpTable(E2E3, E1E2).g(2) = 0;
          end

          tmpg = LookUpTable(E2E3, E1E2).g(2);
          LookUpTable(E2E3, E1E2).g(2) = (tmpg + double(g))/2;
        end

    end

end


%% Create Grid and intrpolate sparse matrix.
[h, w] = size(LookUpTable);
[gridx, gridy] = meshgrid(1:w, 1:h);
interpFV = griddata(E1E2Vec, E2E3Vec, fv, gridx, gridy, 'cubic');
interpGV = griddata(E1E2Vec, E2E3Vec, gv, gridx, gridy, 'cubic');

% Smooth interpolation with gaussian filters
interpFV = imgaussfilt3(interpFV, 6);
interpGV = imgaussfilt3(interpGV, 6);

%% Fill data into lookup table.
for i = 1:w
   for j = 1:h
      curr = LookUpTable(j, i);
      if (isempty(curr.f))
         % Fill in with interpolated data
         LookUpTable(j, i).f = interpFV(j, i);
         LookUpTable(j, i).g = interpGV(j, i);
      end
   end
end
```

6 OBJECT ANALYSIS SCRIPT

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Assingment 2 - Photostereo Imaging: Building 3D Objects.

% Load TableLookUp
% load('LookUpTable.mat');
% load ('radius.mat');
% Load image data
cylinderdata = {'Photostereo_RealImages/cylinder-lamp1.tif';
'Photostereo_RealImages/cylinder-lamp2.tif';
'Photostereo_RealImages/cylinder-lamp3.tif'};
hexlightdata = {'Photostereo_RealImages/hex1-lamp1.tif';
'Photostereo_RealImages/hex1-lamp2.tif'; 'Photostereo_RealImages/hex1-
lamp3.tif'};
hexlight2data = {'Photostereo_RealImages/hex2-lamp1.tif';
'Photostereo_RealImages/hex2-lamp2.tif'; 'Photostereo_RealImages/hex2-
lamp3.tif'};
ellipsoiddata = {'Photostereo_RealImages/ellipsoid-lamp1.tif';
'Photostereo_RealImages/ellipsoid-lamp2.tif';
'Photostereo_RealImages/ellipsoid-lamp3.tif'};
spheredata = {'Photostereo_RealImages/sphere-lamp1.tif';
'Photostereo_RealImages/sphere-lamp2.tif'; 'Photostereo_RealImages/sphere-
lamp3.tif'};
coneLightdata = {'Photostereo_RealImages/cone-lamp1.tif';
'Photostereo_RealImages/cone-lamp2.tif'; 'Photostereo_RealImages/cone-
lamp3.tif'};
conedarkdata = {'Photostereo_RealImages/cone2-lamp1.tif';
'Photostereo_RealImages/cone2-lamp2.tif'; 'Photostereo_RealImages/cone2-
lamp3.tif'};


img1 = rgb2gray(imread(hexlightdata{1}));
img2 = rgb2gray(imread(hexlightdata{2}));
img3 = rgb2gray(imread(hexlightdata{3}));

%% Build 3D mesh

% Iterate over all three images.
[h, w] = size(img1);
[lw, lw] = size(LookUpTable);
th = 35;
BinScale = 100; % TODO: Create Global Static Vars to share.

TDMap = [];
P = [];
Q = [];
Z = [];
prevE1E2 = 0;
prevE2E3 = 0;
prevf = 0;
prevg = 0;
prevz = 0;
prevp = 0;
prevq = 0;
```

```
for i = 1:h
   for j = 1:w

      val = max([img1(i, j), img2(i, j), img3(i, j)]);
      if (val > th)
         % Process pixel.
         E1 = img1(i, j);
         E2 = img2(i, j);
         E3 = img3(i, j);

         % Create index for lookup table
         E1E2 = ceil((log(double(E1 + 1)/double(E2 + 1)) + 5) * BinScale);
         E2E3 = ceil((log(double(E2 + 1)/double(E3 + 1)) + 5) * BinScale);

         % Get previous values
%          if (prevf > 0 && prevg > 0)
%             prevf = LookUpTable(E2E3, E1E2).f;
%             prevg = LookUpTable(prevE2E3, prevE1E2).g;
%          end

         %% Search lookup table
         f = LookUpTable(E2E3, E1E2).f;
         g = LookUpTable(E2E3, E1E2).g;

         % Find similar value
         [ft, fs] = size(f);
         if (fs > 1)
            deltaf1 = abs(f(1) - prevf);
            deltaf2 = abs(f(2) - prevf);
            minf = min(deltaf1, deltaf2);
            if (minf == deltaf1)
                f = f(1);
            else
                f = f(2);
            end
         end

         [gt, gs] = size(g);
         if (gs > 1)
            deltag1 = abs(g(1) - prevg);
            deltag2 = abs(g(2) - prevg);
            ming = min(deltag1, deltag2);
            if (ming == deltag1)
                g = g(1);
            else
                g = g(2);
            end
         end

         %% Build p and q
         x = ceil((((2 * f)/(1 + f^2 + g^2)) * radius));
         y = ceil((((2 * g)/(1 + f^2 + g^2)) * radius));
         z = ceil(((-1 + f^2 + g^2)/(1 + f^2 + g^2)) * radius);
         p = double(x/z);
         q = double(y/z);
```

```
            deltap = abs(prevp - p);
            deltaq = abs(prevq - q);
%           if (deltap > 2)
%               p = prevp;
%           end
%           if (deltaq > 2)
%               q = prevq;
%           end
%           hops = 1000;
%           while (deltap > 2 && deltaq > 2 && hops > 1)
%               z = z * 1.2;
%               deltap = abs(prevp - p);
%               deltaq = abs(prevq - q);
%               hops = hops - 1;
%           end

%           if (z < 1)
%               z = abs(z);
%           end

            P(i, j) = p;
            Q(i, j) = q;
            Z(i, j) = z;

            TDMap(i, j) = z;

            % set Previous
            prevE1E2 = E1E2;
            prevE2E3 = E2E3;
            prevf = f;
            prevg = g;
            prevz = z;
            prevp = p;
            prevq = q;

        end
    end
end

%% Build Quiver Plot
PSubSample = P(1:5:end, 1:5:end);
QSubSample = Q(1:5:end, 1:5:end);
figure(2)
quiver(PSubSample, QSubSample);

%% EXPERIMENTS: Integrate along multiple paths.
[h, w] = size(P);
P2 = zeros(size(P));
Q2 = zeros(size(Q));
Z2 = zeros(size(P));
previ = 0;
prevj = 0;
for i = 1:h
    for j = 1:w
```

```
        tmp = P(i, j);
        tmpPrevP = 0;
        tmpPrevZ = 0;
        if(previ < h && prevj < w && previ > 1 && prevj > 1)
            tmpPrevP = P2(previ, prevj);
            tmpPrevZ = Z2(previ, prevj);
        else
            tmp = 0;
        end

        % Check for inf or nans
        if (tmp == -inf || isnan(tmp) || tmp == inf)
           tmp = 0;
        end
        P2(i, j) =  tmpPrevP - tmp;
        Z2(i, j) =  double(tmpPrevZ + tmpPrevP)/2 - tmp;

        prevj= j;
        previ = i;
    end
end

previ = 0;
prevj = 0;
P3 = zeros(size(P));
for i = 1:h
    for j = w:-1:1
        tmp = P(i, j);
        tmpPrevP = 0;
        tmpPrevZ = 0;
        if(previ < h && prevj < w && previ > 1 && prevj > 1)
            tmpPrevP = P3(previ, prevj);
            tmpPrevZ = Z2(previ, prevj);
        else
            tmp = 0;
        end

        % Check for inf or nans
        if (tmp == -inf || isnan(tmp) || tmp == inf)
           tmp = 0;
        end
        P3(i, j) =  tmpPrevP + tmp;
        Z2(i, j) =  double(tmpPrevZ + tmpPrevP)/2 + double(tmp + Z2(i, j))/2;

        prevj = j;
        previ = i;
    end
end

%% Build Quiver form Results
Z2G = imgaussfilt3(Z2, 5);
Z2GSUB = Z2G(1:5:end, 1:5:end);
PSUB = P(1:5:end, 1:5:end);
QSUB = Q(1:5:end, 1:5:end);
WSUB = ones(size(PSUB));
```

```
figure(3);
quiver3(Z2GSUB, PSUB, QSUB, WSUB)


%% Q INterpolate over Q Values
previ = 0;
prevj = 0;
for j = 1:w
    for i = 1:h
        tmp = Q(i, j);
        tmpPrevQ = 0;
        tmpPrevZ = 0;
        if(previ < h && prevj < w && previ > 1 && prevj > 1)
            tmpPrevQ = Q2(previ, prevj);
            tmpPrevZ = Z2(previ, prevj);
        else
            tmp = 0;
        end

        % Check for inf or nans
        if (tmp == -inf || isnan(tmp) || tmp == inf)
           tmp = 0;
        end
        Q2(i, j) =  tmpPrevQ - tmp;
        Z2(i, j) =  double(tmpPrevZ + tmpPrevQ)/2 + double(tmp + Z2(i, j))/2;

        prevj= j;
        previ = i;
    end
end

previ = 0;
prevj = 0;
Q3 = zeros(size(P));
for j = 1:w
    for i = h:-1:1
        tmp = Q(i, j);
        tmpPrevQ = 0;
        tmpPrevZ = 0;
        if(previ < h && prevj < w && previ > 1 && prevj > 1)
            tmpPrevQ = Q3(previ, prevj);
            tmpPrevZ = Z2(previ, prevj);
        else
            tmp = 0;
        end

        % Check for inf or nans
        if (tmp == -inf || isnan(tmp) || tmp == inf)
           tmp = 0;
        end
        Q3(i, j) =  tmpPrevQ + tmp;
        Z2(i, j) =  double(tmpPrevZ + tmpPrevQ)/2 + double(tmp + Z2(i, j))/2;

        prevj = j;
```

```
        previ = i;
    end
end
```