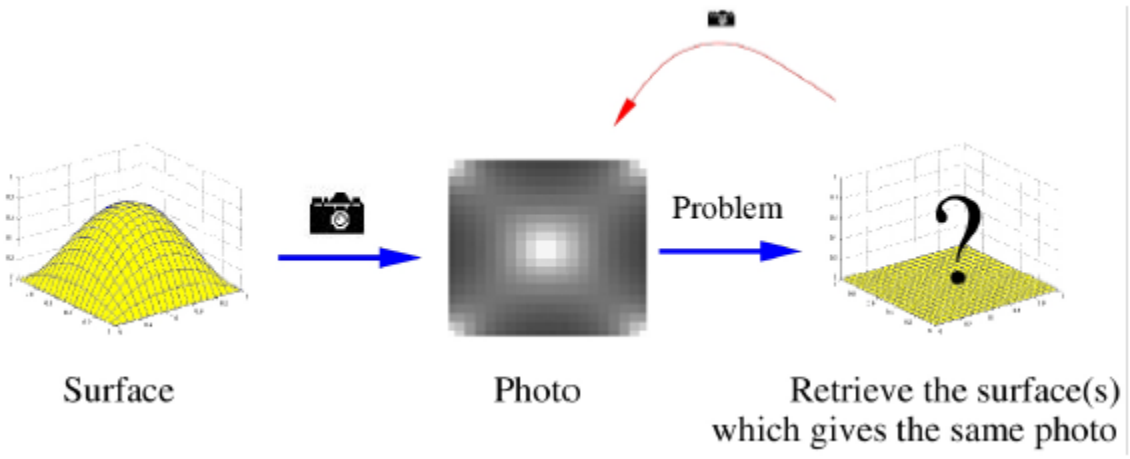
Shape From Shading

Name-Harish Chand

Roll No.-22b2218

Assignment -1

Basic Idea



Given an input gray-scale image, we want to recover the 3D information of the captured (imaged) object. The observed (measured/captured) intensity of an image depends on four main factors:

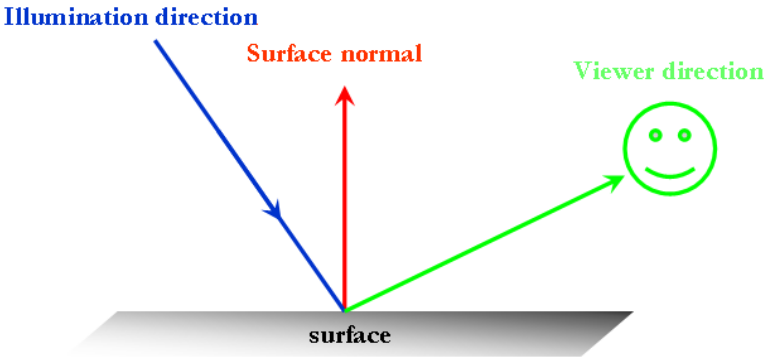
- **Illumination,** which is defined by its position, direction and spectral energy distribution

- **Surface reflectivity of the object,** which is known in shape from shading literature as albedo, it gives the information about how the object reacts with light in terms of the amount of incident light being reflected, in computer graphics literature, it is denoted as surface reflectance coefficients or more commonly as surface material.
- **Surface geometry of an object,** which we want to recover given the object's 2D gray-scale image.
- **Camer a**

the image formation process is ,let say we are given the 3D surface geometry, illumination direction and surface reflectivity, an image is rendered to simulate the camera process. However, in principle, shape from shading is the inverse problem, given an image, a model of the illumination and a model of the surface reflectivity; it is required to recover the surface geometry.

because of the inverse problem, we have more unknown parameters than equations (given information). To solve such problems, we should impose some assumptions.

If the surface normal is known at every point on the surface, surface geometry can be known up to scale; hence most of the shape from shading algorithms are seeking the recovery of surface normals.



Surface normal

The orientation of a surface can be completely defined by defining the normal to the surface at every point *(x,y,z)*. Let **n** = (n_x,n_y,n_z) be a unit vector giving the surface normal at point *(x,y,z)*.

it is useful to define the normal by its spherical not Cartesian coordinates, hence **n** can be defined by its tilt τ_n and slant σ_n, where the tilt (or azimuth angle) is define as the angle between the *x*-axis and the projection of **n** on the *xy*-plane, and the slant (or polar angle) is defined as the angle between the *z*-axis and the normal. Hence the surface normal is defined as $\mathbf{n}(\sigma_n,\tau_n)$ where $\sigma_n \in [0,\pi/2]$ is the slant angle and $\tau_n \in [0,2\pi]$ is the tilt angle.

From the relation between the Cartesian and spherical coordinates, we can obtain the slant and tilt angles from (x_n,y_n,z_n) as follows;

$$\tau_n = \tan^{-1}\left(\frac{n_y}{n_x}\right) \qquad (1)$$

$$\sigma_n = \cos^{-1}(n_z) \tag{2}$$

Hence the normal vector can be defined as:

$$n = (n_x, n_y, n_z) = [\sin\sigma_n \cos\tau_n, \sin\sigma_n \sin\tau_n, \cos\sigma_n]^T \tag{3}$$

Another representation of surface orientation is by its gradient, let the surface be given by $Z = Z(x,y)$, hence the gradient $(p,q)^T$ at point $(x,y,Z(x,y))$ is simply the partial derivative in x and y directions and can be given by the surface slopes $(1,0,p)$ and $(0,1,q)$ where;

$$p = \frac{\partial Z(x,y)}{\partial x} \tag{4}$$

$$q = \frac{\partial Z(x,y)}{\partial y} \tag{5}$$

The relation between the surface gradient and the surface normal can be derived from the fact that surface slopes $(1,0,p)$ and $(0,1,q)$ are two vectors lying on the tangent plane at point (x,y,z), hence the surface normal can be obtained by the cross product of these vectors to result in a vector perpendicular to the tangent plane, then normalizing it to be a unit vector, hence the normal vector can be expressed in terms of surface gradients as a point (p,q) in the gradient space as follows;

$$n = \frac{1}{\sqrt{1+p^2+q^2}}[-p,-q,1]^T \tag{6}$$

Illumination direction

This is the direction from which the object is illuminated. Similar to the representation of the surface normal, we can represent the illumination direction in terms of its slant and tilt as $I(\sigma,\tau)$ where $\sigma \in [0,\pi/2]$ is the slant angle and $\tau \in [0,2\pi]$ is the tilt angle. Hence the illumination direction can be defined as follows;

$$I = [\sin\sigma \cos\tau, \sin\sigma \sin\tau, \cos\sigma]^T \tag{7}$$

Surface types

In order to see an object, there should be a light source; emitting light, then the object will either absorb the entire incident light or reflect all of it, or even reflect some and absorb the rest. This phenomena defines the surface reflectivity, which is the fraction of incident light (radiation) reflected by a surface.

In general, surface reflectivity is a function of the reflected direction and the incident direction; hence it is a directional property. Most of the surfaces can be divided into those that are specular and those that are diffuse. For specular surfaces, such as glass or polished metal, reflectivity will be nearly zero at all angles except at the appropriate reflected angle where the surface shines. For diffuse surfaces, such as matte white paint, reflectivity is uniform, light is reflected in all angles equally or near-equally, such surfaces are said to be Lambertian. Most real objects have some mixture of diffuse and specular reflective properties.

For a Lambertian surface, the intensity only depends on the angle α between the illumination direction and the surface normal, hence under Lambertian assumption, the reflectance at point *(x,y,z)* is a function of the angle α and the surface reflectivity (albedo) ρ defined as follows;

$$R(x,y,z) = \rho(x,y,z)\cos\theta \tag{8}$$

In (8) the angle θ between the illumination direction and the surface normal is assumed to be fixed with respect to the surface point *(x,y,z)* under the assumption that all visible surface points will receive the same amount of light from the same direction, i.e. the illumination direction will not be changed from a point to another on the same surface. However according to the surface of interest, the albedo might change from one point to another, however there are some shape from shading algorithms which assume constant albedo over the surface to facilitate problem solution, yet this is not a realistic assumption.

Using unit vectors for the illumination direction and the surface normal, eqn (8) can be rewritten as follows; for simplicity we will refer to $\rho(x,y,z)$ as simply ρ since we are mainly interested in surface geometry under the assumption of either given or estimate surface albedo.

$$R_{\rho,I}(p,q) = \rho\mathbf{I}^T\mathbf{n} = \frac{\rho}{\sqrt{1+p^2+q^2}}\mathbf{I}^T[-p,-q,1]^T \tag{9}$$

Eq(9) implies that given the surface albedo and the illumination direction, the reflectance map $R_{\rho,I}$ at a point *(p,q)* defined in the gradient space can be expressed in terms of the dot product of the surface normal and the illumination direction, scaled by the surface albedo.

The viewer (camera)

A camera is a projective system; it projects 3D point to 2D image pixels. To model the camera process, we use projections models such as perspective and orthographic projection. Under the assumption of orthographic projection, a point *(X,Y,Z)* can be projected to a 2D point *(x,y)* such that $x = mX$ and $y = mY$, where m is a magnification factor of our choice. While using perspective projection, the 2D point will be defined as $x = X/Z$ and $y = Y/Z$.

The simplest scenario

We will be take the following assumptions ;

- Camera; orthographic projection.
- Surface reflectivity; Lambertian surface
- Known/estimated illumination direction.
- Known/estimated surface albedo/
- The optical axis is the Z axis of the camera and the surface can be parameterized as $Z = Z(x,y)$.

The image irradiance (amount of light received by the camera to which the gray-scale produced is directly proportional) can be defined as follows;

$$E(x,y) = R_{\rho,I}(p,q) = \rho \mathbf{I}^T \mathbf{n} = \frac{\rho}{\sqrt{1+p^2+q^2}} \mathbf{I}^T [-p,-q,1]^T \qquad (10)$$

Eq(10) is the typical starting point of many shape from shading techniques, yet it is of a great mathematical complexity, it is a non-linear partial differential equation in $p = p(x,y)$ and $q = q(x,y)$, which are the gradients of the unknown surface $Z = Z(x,y)$ and depends on quantities not necessarily known (albedo, illumination direction and boundary conditions on which the partial differential equation is dependent). In essence the solution is determined uniquely only if certain a priori information on the

solution is available. This information is typically given at the boundary of the domain of interest, in our case, the image boundaries.

Synthetic image generation

The best way to grasp the meaning of (10) is to produce a synthetic shaded image ourselves. To do this, first we assume Lambertian surface, we then write the equation of our favorite surface in the form of $Z = Z(x,y)$, choose value(s) for the surface albedo, specify illumination direction, then compute the numerical partial derivatives of the surface Z over a pixel grid, then compute the corresponding image brightness using (10).

Lets do it ...

Lets generate the 2D image of a sphere, the equation of a sphere of center (0,0,0) and radius r can be given as follows;

$$x^2 + y^2 + z^2 = r^2 \tag{11}$$

Step 1:

Write the above equation in the form of $Z = Z(x,y)$:

$$\therefore z^2 = r^2 - \left(x^2 + y^2\right) \Rightarrow z = \pm\sqrt{r^2 - \left(x^2 + y^2\right)} \tag{12}$$

Since when a sphere is imaged, only a hemisphere appears in the captured image, and the other hemisphere is not visible, hence we will take either the positive or the negative root. Hence our surface equation can be written as follows;

$$z = \sqrt{r^2 - \left(x^2 + y^2\right)} \tag{13}$$

code-

```
% sphere radius ...
r = 50;

% lets define the xy domain (pixel grid)... x and y are data grids
% (matrices) where the our surface Z = Z(x,y) can be calculated at
% each point (x,y).
```

```
% use finner grid to enhance the resolution
[x,y] = meshgrid(-1.5*r:0.1:1.5*r,-1.5*r:0.1:1.5*r);
```

Step 2:

Assume constant albedo over the points of the sphere surface, let ρ=0.5, this means that our sphere will reflect 50% of the incident light.

```
% surface albedo ...
albedo = 0.5;
```

Step 3:

Choose the illumination direction, let, I = $[0.2, 0, 0.98]^T$.

```
% illumination direction ...
I = [0.2 0 0.98]';
```

Step 4:

Derive the surface partial derivates as follows; in the *x*-direction

$$p = \frac{\partial Z(x,y)}{\partial x} = \frac{1}{2}\left(r^2 - \left(x^2 + y^2\right)\right)^{-1/2}\left(-2x\right) = -x\left(r^2 - \left(x^2 + y^2\right)\right)^{-1/2} \qquad (14)$$

Likewise in the *y*-direction,

$$q = \frac{\partial Z(x,y)}{\partial y} = \frac{1}{2}\left(r^2 - \left(x^2 + y^2\right)\right)^{-1/2}\left(-2y\right) = -y\left(r^2 - \left(x^2 + y^2\right)\right)^{-1/2} \qquad (15)$$

```
% surface partial derivates at each point in the pixel grid ...
p = -x./sqrt(r^2-(x.^2 + y.^2)); % in the x direction
q = -y./sqrt(r^2-(x.^2 + y.^2)); % in the y direction
```

Step 5:

Compute the image brightness using (10), it can be rewritten as follows;

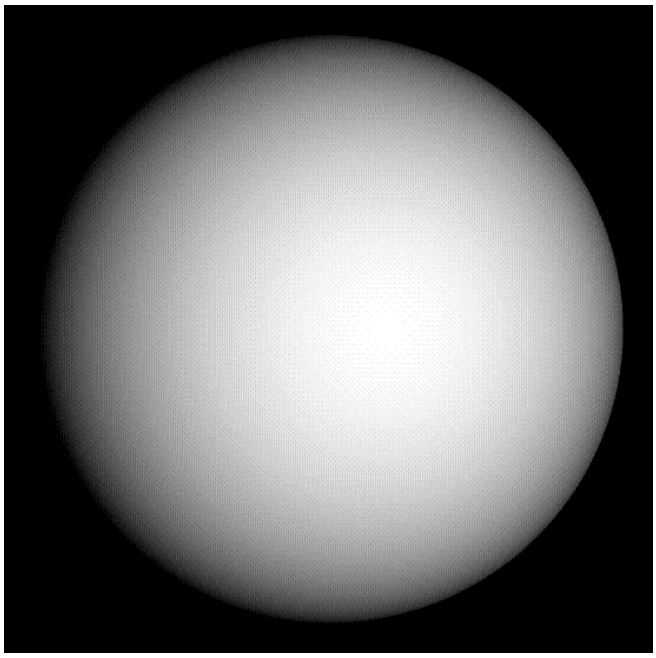
$$E(x,y) = \frac{\rho}{\sqrt{1+p^2+q^2}}[i_x, i_y, i_z][-p,-q,1]^T = \frac{\rho(-i_x p - i_y q + i_z)}{\sqrt{1+p^2+q^2}} \quad (16)$$

where $I = [i_x, i_y, i_z]^T$

Note that for certain illuminants, some image locations might violate the assumption that all visible surface points receive direct illumination, e.g. donut being illuminated from one side, hence the image brightness computed from (10) become negative, in this case the brightness should be set to zero.

```
% now lets compute the image brightness at each point in the pixel grid ...
% the reflectance map will be ...
R = (albedo.*(-I(1).* p - I(2).* q + I(3))./sqrt(1 + p.^2 + q.^2);

% the points of the background are those who don't satisfy the sphere
% equation leading to a negative under the square root of the equation of Z
% = Z(x,y)
mask = ((r^2 - (x.^2 + y.^2) >= 0));

% now we can mask out those points
R = R .* mask;

% converting the reflectance map to image irradiance (by setting negative
% values to zeros)
E = max(0,R);

% converting the image irradiance to a gray scale image
E = E ./ max(E(:));

% displaying our synthetic sphere image ...
figure;
imshow(E);
axis off;

% saving our image
imwrite(E,'sphere.jpg');
```

we will end up with this sphere, we might experiment different values for surface albedo and illumination directions to see their effect, more over using the same methodology we can generate images of different 3D surfaces other than spheres, the main idea is formulating their equations.



Synthesized sphere image with surface albedo 0.5 and illumination direction [0.2 0 0.98] ^{T}

Problem statement

The number of unknowns in (10) seems to suggest that this equation does not provide enough constraints to reconstruct p and q at all pixels. Having $N \times N$ image, it seems that we end up with N^2 equations, one for each pixels, in $2N^2$ unknowns the p and the q for each pixel. But this is not true, because the p and the q are not independent since $Z_{xy} = Z_{yx}$, hence we have N further equations of the kind $p_y = q_x$, where p_y is the derivative of the x-gradient in the y-direction, and q_x is the derivative of the y-gradient in the x-direction.

Now we can formalize the problem statement as follows;

Given the reflectance map of the viewed surface R = R(p,q) and the full knowledge of the surface albedo and illumination direction relative to the available image, it is required to reconstruct the surface gradients p and q for which E(x,y) = R(p,q) and the surface Z = Z(x,y) such that $p = \frac{\partial Z(x,y)}{\partial x}$ *and* $q = \frac{\partial Z(x,y)}{\partial y}$.

Finding surface albedo and illumination direction

We are one last step away from a shape from shading method; we still have to deal with the problems that the parameters of the reflectance map (albedo and illumination direction) can be unknown..

Assumptions

Similarly to shape from shading itself, the problem of estimating albedo and illumination direction is only apparently simple; in fact, an effective way to provide good estimates under general circumstances, or even in the simpler Lambertian case, must still be found. The major difficulty is the fact that the problem is heavily under constrained. To overcome this difficulty, we must make assumptions on either the shape of the viewed surface or the distribution of the surface normals;

Our task is therefore to derive a method which, under appropriate assumptions estimates albedo and illumination direction. Hence the problem statement can be formalized as follows;

Under the same assumptions of shape from shading and with further assumptions that the surface images is Lambertian and the directions of the surface normals are distributed uniformly in 3D space, determine albedo and illumination direction from a single image of the surface.

Let's select an appropriate distribution of the surface normals in a generic scene,
from (3) where surface normals can be represented by means of the tilt and slant angles, for simplicity let us denote the surface normal as follows;

$$n = [\sin\sigma_n \cos\tau_n, \sin\sigma_n \sin\tau_n, \cos\sigma_n]^T = [\sin\beta\cos\alpha, \sin\beta\sin\alpha, \cos\beta]^T \qquad (17)$$

where $\beta \in [0,\pi/2]$ is the slant angle and $\alpha \in [0,2\pi]$ is the tilt angle, we will assume that normal vectors follows uniform distribution in α as seen from the image plane, yet due to foreshortening (the illusion of seeing something shorter than it is due to illumination effect), the number of normal vectors with slant equal to β will be proportional to $\cos\beta$, hence the distribution of the normal vectors can be assumed as follows;

$$f(\alpha,\beta) = \frac{\cos\beta}{2\pi} \qquad (18)$$

Simple method for Lambertian surfaces

In This section will discuss a simple method for recovering albedo and illumination direction. We are given the gray-scale image, under Lambertian assumption, its relation to the surface albedo and surface illumination is given by (10), representing the illumination direction as $I(\sigma,\tau)$ $\sigma\in[0,\pi/2]$ is the slant angle and $\tau\in[0,2\pi]$ is the tilt angle, we obtain (7), substituting (7) and (18) in (10), we will have the image irradiance (captured gray-scale image) expressed in terms of the slant and tilt angles of the surface normals as follows, taking into consideration that we still assume that the illumination direction is fixed for the whole image, hence what vary with respect to the image pixels are surface normals;

$$E(\alpha,\beta)=\rho\left(\cos\alpha\sin\beta\cos\tau\sin\sigma+\sin\alpha\sin\beta\sin\tau\sin\sigma+\cos\beta\cos\sigma\right) \tag{19}$$

Now lets find the first moment of the image irradiance $E(\alpha,\beta)$, which can be defined as follows where $\mathrm{E}(.)$ is the expectation operator;

$$\mu_1=\mathrm{E}\left(E(\alpha,\beta)\right)=\int_0^{2\pi}\int_0^{\pi/2}E(\alpha,\beta)\,f(\alpha,\beta)\,d\beta\,d\alpha \tag{20}$$

This integral breaks down into three additive terms, of which the first two vanish (because the tilt α is integrated over a full period) and the third yields;

$$\mu_1=\frac{\pi}{4}\rho\cos\sigma \tag{21}$$

The interesting fact about (21) is that, as we assumed that the surface normals are distributed according to (18), we can compute μ_1 as the average of the brightness values of the given image, and hence we can consider (21) as an equation for albedo and slant.

A similar derivation for the second moment of the image irradiance, which will boil to the average of the square of the image brightness and can be related to the albedo and slant as follows;

$$\mu_2=\mathrm{E}\left(E^2(\alpha,\beta)\right)=\int_0^{2\pi}\int_0^{\pi/2}E^2(\alpha,\beta)\,f(\alpha,\beta)\,d\beta\,d\alpha=\frac{1}{6}\rho^2\left(1+3\cos^2\sigma\right) \tag{22}$$

From (21) and (22), we can estimate the surface albedo and slant as follows;

$$\rho = \frac{\sqrt{6\pi^2\mu_2 - 48\mu_1^2}}{\pi} \tag{23}$$

$$\cos\sigma = \frac{4\mu_1}{\sqrt{6\pi^2\mu_2 - 48\mu_1^2}} \tag{24}$$

We are still left with the problem of estimating the tilt τ of the illumination, this can be obtained from the spatial derivates of the image brightness, E_x and E_y which entails the illumination tilt direction, the tilt angle can be obtained as follows; refer to [4] for analytical proof of (25).

$$\tan\tau = \frac{average(E_y)}{average(E_x)} \tag{25}$$

Hence we can now summarize the method of estimating the surface albedo and illumination direction as follows; given an intensity image of a Lambertian surface:

Step 1:

Compute the average of the image brightness μ_1 and of its square μ_2, we will use the image we have just generated to compare our results with the groundtruth parameters used in the image generation.

```
% read the image
E = double(imread('sphere.jpg'));

% normalizing the image to have maximum of one
E = E ./ max(E(:));

% compute the average of the image brightness
Mu1 = mean(E(:));

% compute the average of the image brightness square
Mu2 = mean(mean(E.^2));
```

Step 2:

Compute the spatial image gradients in x and y directions. Then we normalize them to have unit vectors.

```
% now lets compute the image's spatial gradient in x and y directions
[Ex,Ey] = gradient(E);

% normalize the gradients to be unit vectors
Exy = sqrt(Ex.^2 + Ey.^2);
nEx = Ex ./(Exy + eps); % to avoid dividing by zero
nEy = Ey ./(Exy + eps);
```

Step 3:

Compute the averages of the normalized gradients in x and y directions.

```
% computing the average of the normalized gradients
avgEx = mean(Ex(:));
avgEy = mean(Ey(:));
```

Step 4:

Estimate the albedo, slant and tilt using (23), (24) and (25)

```
% now lets estimate the surface albedo
gamma = sqrt((6 *(pi^2)* Mu2) - (48 * (Mu1^2)));
albedo = gamma/pi;

% estimating the slant
slant = acos((4*Mu1)/gamma);

% estimating the tilt
tilt = atan(avgEy/avgEx);

if tilt < 0
    tilt = tilt + pi;
end

% the illumination direction will be ...
I = [cos(tilt)*sin(slant) sin(tilt)*sin(slant) cos(slant)];
```

The computed parameters are :

```
Estimated Albedo: 0.8788
Estimated Slant: 1.2342
Estimated Tilt: 3.1378
Estimated Illumination Direction: [-0.9439 0.0036 0.3302]
```

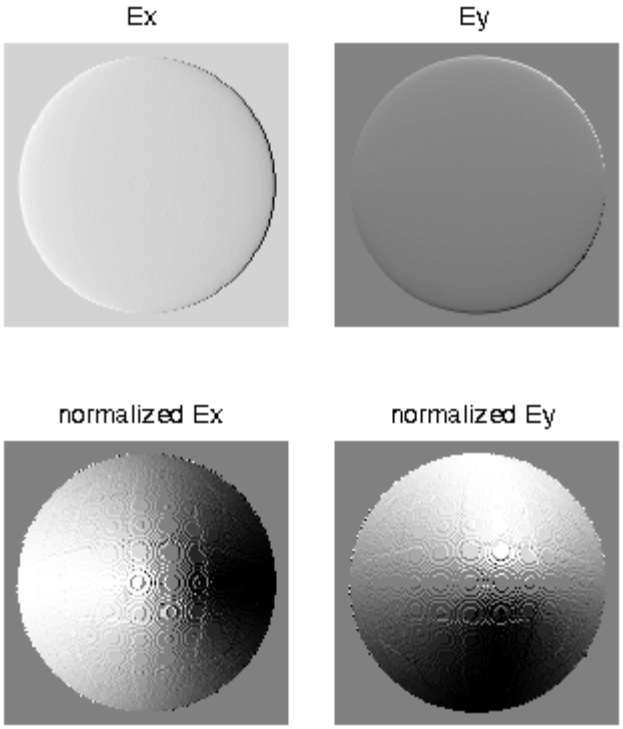
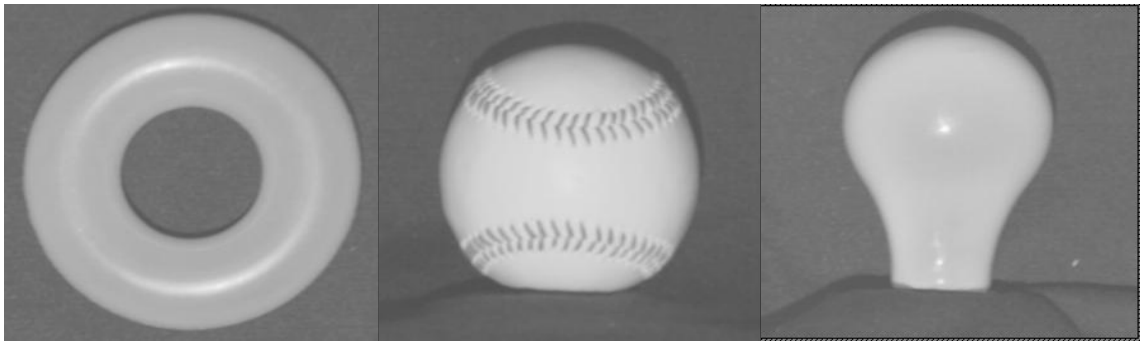


Image gradients in x and y directions and their normalization

As noted by [1], this method gives reasonably good results, though it fails for very small and very large slants.

below are the test images

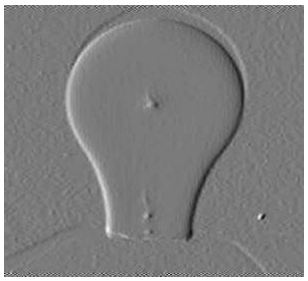


gradient_x_man

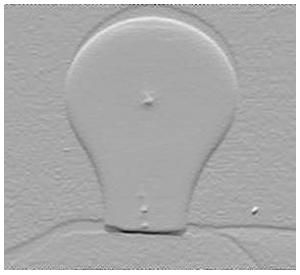
gradient_y_man

normalized_gradient_x_man

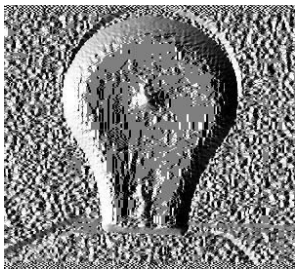
normalized_gradient_y_man



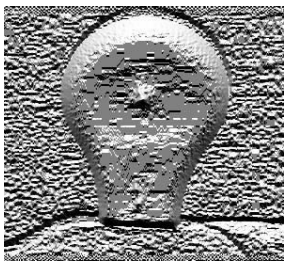
gradient_x



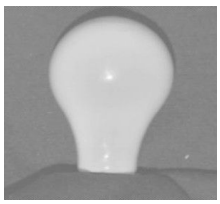
gradient_x

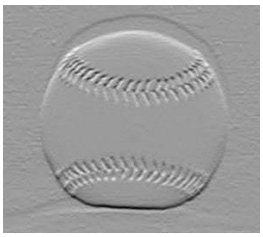


normalized_gradient_x

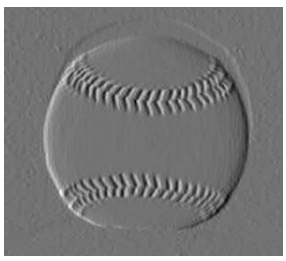


normalized_gradient_y

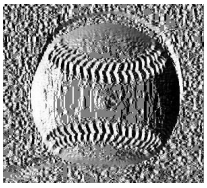




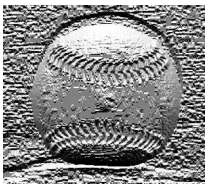
gradient_x



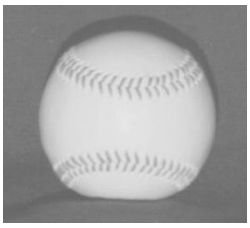
gradient_y

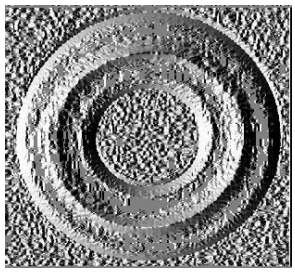


normalized_gradient_x

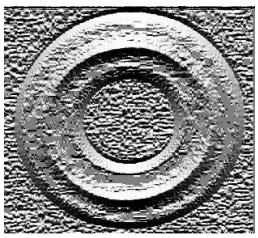


normalized_gradient_y

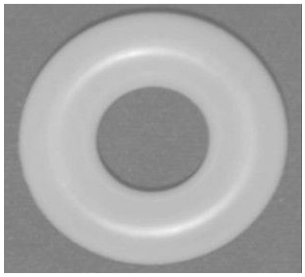




normalized_gradient_x



normalized_gradient_y



Above are the results for test images in the right side

Classical images in shape from shading literature

Shape from shading as a minimization problem

We are now ready to search for a solution to the shape from shading problem. Even under the simplifying Lambertian assumption, the direct inversion of (10) to solve for surface normals is a very difficult task, which involves the solution of a nonlinear partial differential equation in the presence of uncertain boundary conditions. Moreover a slight amount of noise in the image can cause (a) non-existing solution (b) not unique solution or (c) a solution that does not depend continuously on the data. Hence we are not looking for an exact solution for (10), instead we are seeking an approximation in some sense.

In literature, shape from shading algorithms can be classified into global and local approaches, where in global approaches; the surface is recovered by minimizing some energy functional over the entire image. On the other hand, the local approaches make use of local brightness information to recover surface patches and then glue them together to obtain the whole surface.

From another perspective, the shape from shading algorithms can be categorized according to the method used to solve the image irradiance equation (10). There are approaches that try to recover the surface by minimizing the error between the two sides of (10), i.e. minimization approaches. A second class of approaches, called propagation approaches, starts from a set of points where the solution is known and propagates the shape information to the entire image. The last class includes approaches that reduce the complexity of the problem by either converting the nonlinear image irradiance equation into a linear one (linear approaches) or by approximating the surface locally by simple shapes (local shading analysis approaches). In what follows we will concentrate on minimization techniques.

7.1 The functional to be minimized

In minimization approaches, the solution of the shape from shading problem is obtained by minimizing an energy function over the entire image. Two aspects should be considered in this class of approaches: (1) the choice of the functions which has to be minimized, (2) the method of minimization. In order to have a unique solution, the functional should be constrained in different ways. In what follows we will discuss the most important constraints used.

The brightness constraint is the most important constraint. The main idea is to solve for surface normals, such that when you use these normals with the given surface albedo and illumination direction, you can form the corresponding 2D image using the image irradiance equation (10), hence we want to find the surface normals such that there is a minimum deviation from the original given image and the formed image, this constraint can be defined as follows;

$$\varepsilon_1 = \int\int \big(E(x,y) - R(p,q)\big)^2 dxdy \tag{26}$$

where *E(x,y)* is the given image and *R(p,q)* is the computed image using the estimated surface normals.

The smoothness constraint is used to obtain a smooth surface that is free from discontinuities, hence the smoothness constraint can be expressed in terms of the second derivatives of surface normals to ensure surface smoothness as follows;

$$\begin{aligned}\varepsilon_2 &= \iint \left[\left(\frac{\partial^2 Z}{\partial x^2}\right)^2 + \left(\frac{\partial^2 Z}{\partial y^2}\right)^2 + \left(\frac{\partial^2 Z}{\partial x \partial y}\right)^2 + \left(\frac{\partial^2 Z}{\partial y \partial x}\right)^2 \right] dxdy \\ &= \int\int \big(p_x^2 + q_y^2 + p_y^2 + q_x^2\big) dxdy \end{aligned} \tag{27}$$

Hence the energy function to be minimized can be formulated to minimize the brightness deviation between the original image and the estimated one, and enforce a smoothness constraint, one possible way to implement this idea is to look for the minimum of the following functional;

$$\varepsilon = \int\int (\varepsilon_1 + \lambda \varepsilon_2) dxdy = \int\int \big((E(x,y) - R(p,q))^2 + \lambda (p_x^2 + q_x^2 + p_y^2 + q_y^2)\big) dxdy \tag{28}$$

Where the parameter λ is always positive, in the order of 1000, and controls the relative influence of the two terms in the minimization process. Obviously, a large λ encourages a very smooth solution that is not necessarily close to the data, while a small λ promotes a more irregular solution closer to the data.

The Euler-Lagrange equations

The minimization of (28) can not be effectively performed by the means of a greedy algorithm. However the calculus of variations gives us a straightforward procedure to derive equations minimizing a generic functional, the Euler-Lagrange equations. This section will focus on how to set up these equations, in a matter of fact, the real problem is not the derivation of the equations, but finding a good numerical algorithm for solving them.

For a functional ε which depends on two functions *p* and *q* of two real variables *x* and *y*, and on their first order spatial derivatives, the Euler-Lagrange equations by definition can be stated as follows;

$$\frac{\partial \varepsilon}{\partial p} - \frac{\partial}{\partial x}\frac{\partial \varepsilon}{\partial p_x} - \frac{\partial}{\partial y}\frac{\partial \varepsilon}{\partial p_y} = 0 \qquad (29)$$

and

$$\frac{\partial \varepsilon}{\partial q} - \frac{\partial}{\partial x}\frac{\partial \varepsilon}{\partial q_x} - \frac{\partial}{\partial y}\frac{\partial \varepsilon}{\partial q_y} = 0 \qquad (30)$$

since *R* is the only function of *p* and *q* in (28), and neither *E* nor *R* depend on p_x , p_y, q_x and q_y, the Euler-Lagrange equations associated with (28) become;

$$-2(E-R)\frac{\partial R}{\partial p} - 2\lambda p_{xx} - 2\lambda p_{yy} = 0 \qquad (31)$$

and

$$-2(E-R)\frac{\partial R}{\partial q} - 2\lambda q_{xx} - 2\lambda q_{yy} = 0 \qquad (32)$$

which can be simplified to give;

$$\nabla^2 p = \frac{1}{\lambda}(R - E)\frac{\partial R}{\partial p} \tag{33}$$

$$\nabla^2 q = \frac{1}{\lambda}(R - E)\frac{\partial R}{\partial q} \tag{34}$$

Where $\nabla^2 p = p_{xx} + p_{yy}$ and $\nabla^2 q = q_{xx} + q_{yy}$ are the Laplacians of p and q respectively. Our next task now is to solve (33) and (34) for p and q.

7.3 From the continuous to the discrete case

Since the given image is already discrete, we need somehow to find a way to solve (33) and (34) in a discrete way. We start by denoting $p_{i,j}$ and $q_{i,j}$ as the samples of p and q over the pixel grid at the location *(i,j)*. Using the formula for the numerical approximation of the second derivative, (33) and (34) become ;

$$-4p_{i,j} + p_{i+1,j} + p_{i-1,j} + p_{i,j+1} + p_{i,j-1} = \frac{1}{\lambda}\left(R\left(p_{i,j}, q_{i,j}\right) - E(i,j)\right)\frac{\partial R}{\partial p} \tag{35}$$

and

$$-4q_{i,j} + q_{i+1,j} + q_{i-1,j} + q_{i,j+1} + q_{i,j-1} = \frac{1}{\lambda}\left(R\left(p_{i,j}, q_{i,j}\right) - E(i,j)\right)\frac{\partial R}{\partial q} \tag{36}$$

Now the problem is reduced to finding the gradients $p_{i,j}$ and $q_{i,j}$ and hence determining the unknown surface $Z = Z(x,y)$ from them.

The algorithm

These equations can be described in the discrete form as:

$$p_{i,j} = \bar{p}_{i,j} + \frac{1}{4\lambda}(E - R)\frac{\partial R}{\partial p} \tag{37}$$

and

$$q_{i,j} = \bar{q}_{i,j} + \frac{1}{4\lambda}(E - R)\frac{\partial R}{\partial q} \tag{38}$$

Where $\bar{p}_{i,j}$ and $\bar{q}_{i,j}$ are the averages of $p_{i,j}$ and $q_{i,j}$ over the four nearest neighbors and are defined as follows:

$$\bar{p}_{i,j} = \frac{p_{i+1,j} + p_{i-1,j} + p_{i,j+1} + p_{i,j-1}}{4} \tag{39}$$

and

$$\bar{q}_{i,j} = \frac{q_{i+1,j} + q_{i-1,j} + q_{i,j+1} + q_{i,j-1}}{4} \tag{40}$$

An iterative scheme can be used to solve for p and q, starting from some initial configurations for the $p_{i,j}$ and $q_{i,j}$, advances from the step k to the step $k+1$ can be defined according to the following updating rules;

$$p_{i,j}^{k+1} = \bar{p}_{i,j}^{k} + \frac{1}{4\lambda}(E - R)\frac{\partial R}{\partial p}|^{k} \tag{41}$$

and

$$q_{i,j}^{k+1} = \bar{q}_{i,j}^{k} + \frac{1}{4\lambda}(E - R)\frac{\partial R}{\partial q}|^{k} \tag{42}$$

Enforcing integrability

The solution obtained by iterating (41) and (42) is inconsistent, since the functional was not told that p and q were the partial derivates of the same function Z, hence there is no Z such that $Z_x = p$ and $Z_y = q$. to overcome this inconvenience, a good idea is to insert a step enforcing integrability after each iteration, to guarantee that when we integrate p and q we will obtain the surface Z.

The discrete Fourier transform can be used to find the depth Z and enforce the integrability constraint from the estimated p and q as follows; At each iteration, we compute the Fast Fourier Transform (FFT) of p and q. Let c_p and c_q are the Fourier transform of p and q respectively, then

$$p = \sum c_p(\omega_x, \omega_y) e^{j(\omega_x x + \omega_y y)} \tag{43}$$

$$q = \sum c_q(\omega_x, \omega_y) e^{j(\omega_x x + \omega_y y)} \tag{44}$$

Then Z can be computed as the inverse Fourier transform of $c(\omega_x, \omega_y)$ where:

$$Z = \sum c(\omega_x, \omega_y) e^{j(\omega_x x + \omega_y y)} \tag{45}$$

Such that;

$$c(\omega_x, \omega_y) = \frac{-j(\omega_x c_p(\omega_x, \omega_y) + \omega_y c_q(\omega_x, \omega_y))}{\omega_x^2 + \omega_y^2} \tag{46}$$

The function Z in (45) has three important properties;

- It provides a solution to the problem of reconstructing a surface from a set of non-integrable p and q.
- Since the coefficients $c(\omega_x, \omega_y)$ do not depend on x and y, (45) can be easily differentiated with respect to x and y to give a new set of integrable p and q, say p' and q', such that:

$$p' = \frac{\partial Z}{\partial x} = \sum j\omega_x c(\omega_x, \omega_y) e^{j(\omega_x x + \omega_y y)} = \sum c'_p(\omega_x, \omega_y) e^{j(\omega_x x + \omega_y y)} \tag{47}$$

and

$$q' = \frac{\partial Z}{\partial y} = \sum j\omega_y c(\omega_x, \omega_y) e^{j(\omega_x x + \omega_y y)} = \sum c'_q(\omega_x, \omega_y) e^{j(\omega_x x + \omega_y y)} \tag{48}$$

- Most importantly, p' and q' are the integrable pair closest to the old pair of p and q.

This can be viewed as if we have projected the old p and q onto a set which contains only integrable pairs.

Let's do it

We now can summarize the algorithm of shape from shading as follows; given the gray-scale image of the object, do the following;

Step 1:

Compute surface albedo and illumination direction using the algorithm described in section 6.

```
% read the image
E = imread('sphere.jpg');

% making sure that it is a grayscale image
if isrgb(E)
    E = rgb2gray(E);
end

% downsampling to speedup
E = E(1:2:end,1:2:end);

E = double(E);

% normalizing the image to have maximum of one
E = E ./ max(E(:));

% first compute the surface albedo and illumination direction ...
[albedo,I] = estimate_albedo_illumination (E);
```

Step 2:

Initializations: surface normals, the second order derivatives of surface normals, the estimated reflectance map, the controlling parameter λ, the maximum number of iterations, the filter to be used to get the second order derivates of the surface normals, ω_x in (47) and ω_y in (48) .

```
% Initializations ...
[M,N] = size(E);

% surface normals
p = zeros(M,N);
q = zeros(M,N);

% the second order derivatives of surface normals
p_ = zeros(M,N);
q_ = zeros(M,N);

% the estimated reflectance map
R = zeros(M,N);

% the controling parameter
lamda = 1000;

% maximum number of iterations
```

```
maxIter = 2000;

% The filter to be used to get the second order derivates of the surface
% normals.
w = 0.25*[0 1 0;1 0 1;0 1 0]; % refer to equations (39) and (40)

% wx and wy in (47) and (48)
[x,y] = meshgrid(1:N,1:M);
wx = (2.* pi .* x) ./ M;
wy = (2.* pi .* y) ./ N;
```

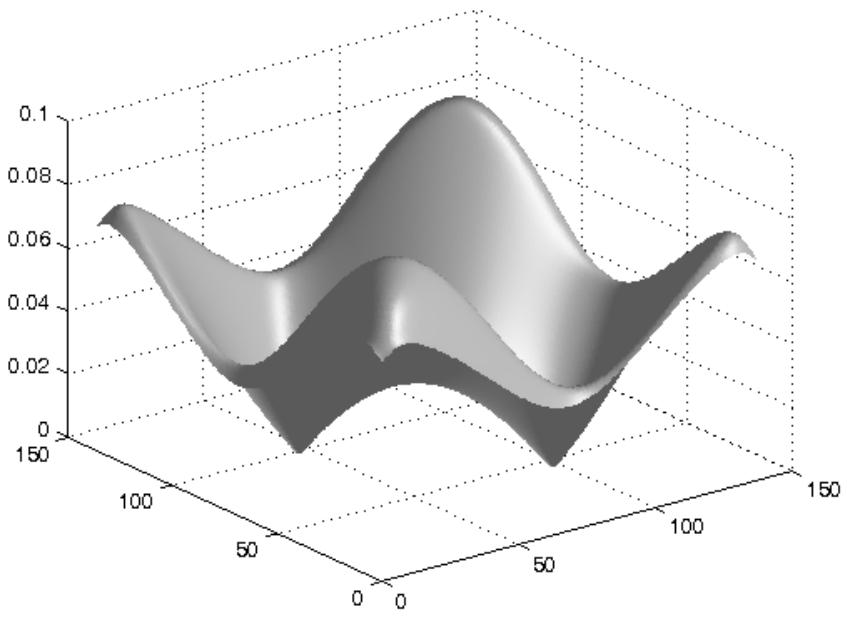
Step 3:

Iterate the following steps until a suitable stopping criterion is met;

- Compute the second order derivates of the current surface normals.
- Using the computed surface albedo, illumination direction and surface normals, compute estimation for the reflectance map.
- Compute the partial derivatives of the reflectance map with respect to *p* and *q*.
- Compute the newly estimated surface normals.
- Compute the Fast Fourier Transform of the surface normals.
- Compute the Fourier Transform of the surface *Z* from the Fourier Transform of the surface normals.
- Compute the surface *Z* using inverse Fourier Transform.
- Compute the integrable surface normals.

```
for k = 1 : maxIter
    % compute the second order derivates of the surface normals
    p_ = conv2(p,w,'same');
    q_ = conv2(q,w,'same');

    % Using the computed surface albedo, illumination direction and
    % surface normals, compute estimation for the reflectance map.
    % refer to (16)
    R = (albedo.*(-I(1).* p - I(2).* q + I(3)))./sqrt(1 + p.^2 + q.^2);

    % Compute the partial derivatives of the reflectance map with respect
    % to p and q. it will be the differenation of (16) with respect to p
    % and q
    pq = (1 + p.^2 + q.^2);

    dR_dp =
      (-albedo*I(1) ./ (pq .^(1/2))) + (-I(1) * albedo .* p - I(2) * albedo .* q +
      I(3) * albedo) .* (-1 .* p .* (pq .^(-3/2)));
```

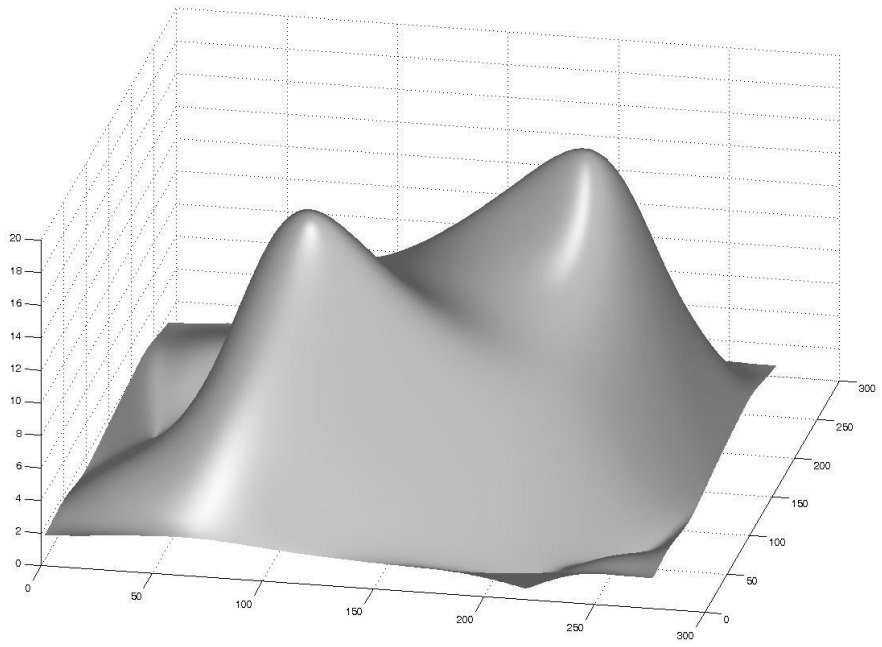
```
    dR_dq =
       (-albedo*I(2) ./ (pq .^(1/2))) + (-I(1) * albedo .* p - I(2) * albedo .* q +
       I(3) * albedo) .* (-1 .* q .* (pq .^(-3/2)));

     % Compute the newly estimated surface normals ... refer to (41) and
     % (42)
     p = p_ + (1/(4*lamda))*(E-R).*dR_dp;
     q = q_ + (1/(4*lamda))*(E-R).*dR_dq;

     % Compute the Fast Fourier Transform of the surface normals.
     Cp = fft2(p);
     Cq = fft2(q);

     % Compute the Fourier Transform of the surface Z from the Fourier
     % Transform of the surface normals ... refer to (46) ...
     C = -i.*(wx .* Cp + wy .* Cq)./(wx.^2 + wy.^2);

     % Compute the surface Z using inverse Fourier Transform ... refer to
     % (45)
     Z = abs(ifft2(C));

     % Compute the integrable surface normals .. refer to (47) and (48)
     p = ifft2(i * wx .* C);
     q = ifft2(i * wy .* C);

     % saving intermediate results
     if (mod(k,100) == 0)
         save(['Z_after_' num2str(k) '_iterations.mat'],'Z');
     end
end

% visualizing the result
figure;
surfl(Z);
shading interp;
colormap gray(256);
lighting phong;
```



The reconstructed surface using the algorithm described

Apparently the reconstructed surface is not our sphere, therefore, this algorithm is not an optimal algorithm. Coming to think of it, this algorithm mainly depends on the solution of partial differential equation, which needs boundary conditions in order to be solved, yet we have not incorporated any boundary conditions. What we mean by boundary conditions is that there are some points we know approximately their depth, since shape from shading mainly rely on shading, this means intensity variations have a meaning to the algorithm, hence we can assume the knowledge of the depth of either the brightest or darkest points in the image, let's see if this idea will enhance the result of this algorithm. This only modification will be in the initialization step, as follows;

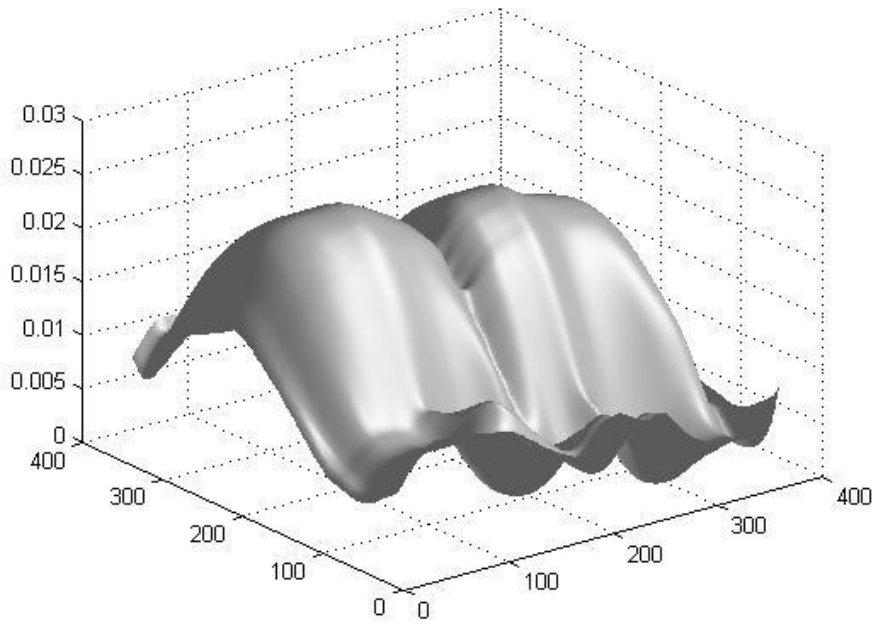
```
% surface normals
Z = zeros(M,N);

% assign very bright pixels with large depth value, the negative is used because it
% was observed that this algorithm always generate concave surfaces (curved
% inward), this is an ambiguity in shape from shading in general, SFS algorithm can
% not distinguish between concave and convex surfaces, both are the same.
Z(find(E>0.75)) = -100 .* E(find(E>0.75));

% getting the surface normals from the initial depth map

[p,q] = gradient(Z);
```



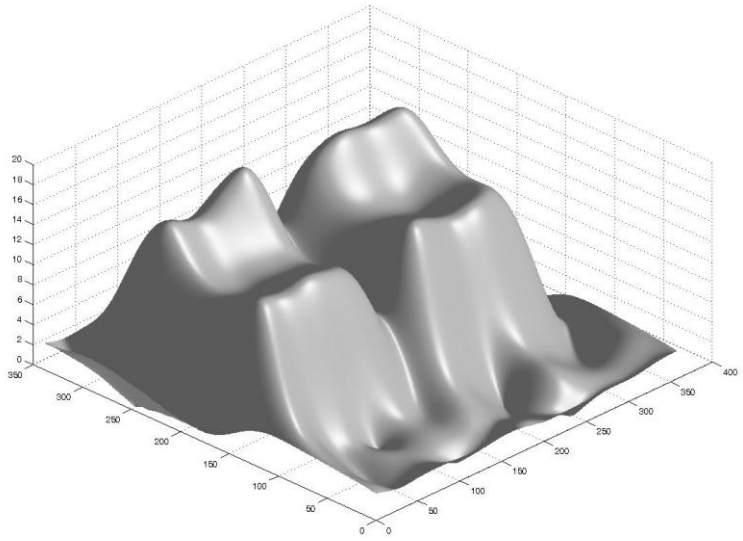
The reconstructed surface using the algorithm , after imposing the boundary conditions

Obviously this result is much better, yet there are other algorithms which lead to more accurate results. Just out of curiosity, let's experiment this algorithm on another image.

another image to experiment the algorithm described in this section.



the reconstructed mug without imposing the boundary conditions



the reconstructed mug with imposing the boundary conditions

Let us now discuss other types of algorithms which will recover our sphere. we can experiment other values for illumination direction to discover the cases where this algorithm yields acceptable results.

Linear approaches

Linear approaches reduce the non-linear problem into a linear through the linearization of the image irradiance equation (10). The idea is based on the assumption that the lower order components in the reflectance map dominate. Therefore, these algorithms only work well under this assumption.

Pentland [6] approach

Under the assumptions of Lambertian surface, orthographic projections, the surface being illuminated by distant light sources, and the surface is not self-shadowing, Pentland [6] defined the image irradiance equation as follows;

$$E(x,y) = R(p,q) = \frac{\rho\left(i_x p + i_y q - i_z\right)}{\sqrt{1+p^2+q^2}} = \frac{p\sin\sigma\cos\tau + q\sin\sigma\sin\tau + \cos\sigma}{\sqrt{1+p^2+q^2}} \qquad (49)$$

Comparing (46) with (10), we can conclude the following;

- It is assumed that the surface has constant albedo, hence it can be ignored as it is not a function of surface points anymore.
- In (10) the surface normal is obtained from the cross product *(1,0,p)x(0,1,q)*, yet in (46) the cross product is applied in the inverse way, i.e. *(0,1,q)x(1,0,p)*, hence the normal vector becomes *(p,q,-1)* instead of *(-p,-q,1).*
- Moreover when using the representation of the illumination direction in terms of its slant and tilt, the negative sign in *(p,q,-1)* is ignored since $\cos\sigma = \cos(-\sigma)$.

By taking the Taylor series expansion of (49) about $p = p_o$ and $q = q_o$, and ignoring the higher order terms, the image irradiance equation will be reduced to;

$$E(x,y) = R(p,q) \approx R(p_0,q_0) + (p-p_0)\frac{\partial R}{\partial p}(p_0,q_0) + (q-q_0)\frac{\partial R}{\partial q}(p_0,q_0) \qquad (50)$$

For Lambertian surface, the above equation at $p_0 = q_0 = 0$ reduces to;

$$E(x,y) \approx \cos\sigma + p\cos\tau\sin\sigma + q\sin\tau\sin\sigma \qquad (51)$$

Next, Pentland takes the Fourier transform of both sides of (51). Since the first term on the right is a DC term, i.e. constant with respect to the variables we are looking for (surface normals), it can be dropped. Using Fourier transform identities, we have the following;

$$p = \frac{\partial}{\partial x} Z(x,y) \xleftrightarrow{\Im} (-j\omega_x) F_Z(\omega_x, \omega_y) \tag{52}$$

$$q = \frac{\partial}{\partial y} Z(x,y) \xleftrightarrow{\Im} (-j\omega_y) F_Z(\omega_x, \omega_y) \tag{53}$$

Where F_z is the Fourier transform of *Z(x,y)*, taking the Fourier transform of (51), we will get the following;

$$F_E = (-j\omega_x) F_Z(\omega_x, \omega_y) \cos\tau \sin\sigma + (-j\omega_y) F_Z(\omega_x, \omega_y) \sin\tau \sin\sigma \tag{54}$$

Where F_E is the Fourier transform of the given image *E(x,y)*. The depth map (our sought surface) *Z(x,y)* can be computed by rearranging the terms in (54), and then taking the inverse Fourier transform as follows;

$$\begin{aligned} F_E &= F_Z(\omega_x, \omega_y)\left[-j\omega_x \cos\tau \sin\sigma - j\omega_y \sin\tau \sin\sigma\right] \\ &\Rightarrow F_Z(\omega_x, \omega_y) = \frac{F_E}{-j\omega_x \cos\tau \sin\sigma - j\omega_y \sin\tau \sin\sigma} \end{aligned} \tag{55}$$

Hence,

$$Z(x,y) = \Im^{-1}\left\{F_Z(\omega_x, \omega_y)\right\} \tag{56}$$

This algorithm gives a non-iterative, closed-form solution using Fourier transform. The problem lies in the linear approximation of the reflectance map, which causes trouble when the non-linear terms are dominant. As pointed out by Pentland, when the quadratic terms in the reflectance map dominate, the *frequency doubling* occurs, in this case, the recovered surface will not be consistent with the illumination conditions.

Let's do it …

We now can summarize the algorithm of shape from shading using Pentland approach as follows; given the gray-scale image of the object, do the following;

Step 1:

Compute surface albedo and illumination direction using the algorithm described in section 6.

```
% read the image
E = imread('sphere.jpg');

% making sure that it is a grayscale image
if isrgb(E)
    E = rgb2gray(E);
end

E = double(E);

% normalizing the image to have maximum of one
E = E ./ max(E(:));

% first compute the surface albedo and illumination direction ...
[albedo,I,slant,tilt] = estimate_albedo_illumination (E);
```

Step 2:

Compute the Fourier transform of the given image.

```
% compute the fourier transform of the image
Fe = fft2(E);
```

Step 3:

Compute ω_x in (47) and ω_y in (48) .

```
% wx and wy in (47) and (48)
[M,N] = size(E);
[x,y] = meshgrid(1:N,1:M);
wx = (2.* pi .* x) ./ M;
wy = (2.* pi .* y) ./ N;
```

Step 4:

Using the estimated illumination direction, compute (55).

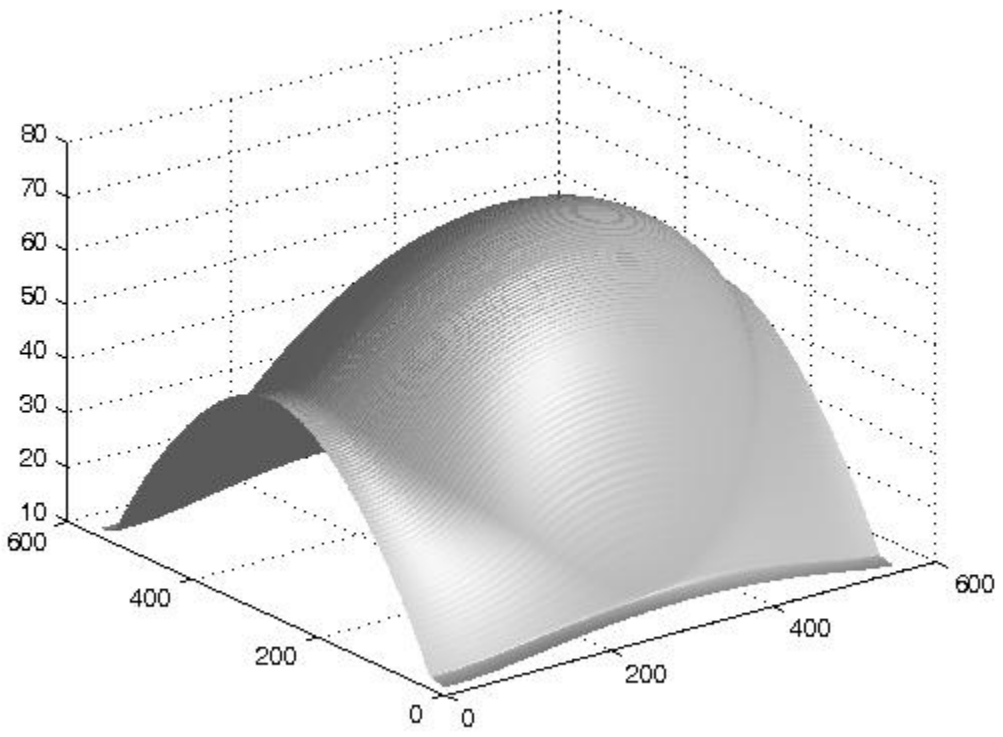
```
% Using the estimated illumination direction, compute (55).
Fz =
```

```
Fe./(-i.*wx.*cos(tilt).*sin(slant)-i.*wy.*sin(tilt).*sin(slant));
```

Step 5:

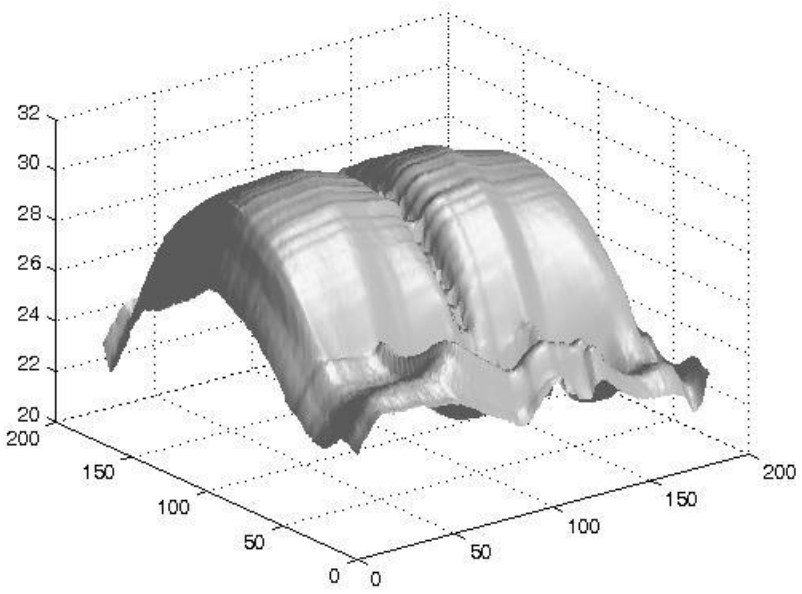
Compute the inverse Fourier transform to recover the surface and visualize the results.

```
% Compute the inverse Fourier transform to recover the surface.
Z = abs(ifft2(Fz));

% visualizing the result
figure;
surfl(Z);
shading interp;
colormap gray(256);
lighting phong;
```



Surface recovered using Pentland approach

It is obvious that somehow, it is our sphere, yet we are still eager to recover a sphere-like surface not just a dome as in Fig.7. Hence we will discuss the work of Shah[7] in the next section.



the reconstructed mug using Pentland approach

Shah [7] approach

Shah [7] employed the discrete approximations of *p* and *q* using finite differences in order to linearize the reflectance map in terms of *Z*. The reflectance function for Lambertian surfaces is defined as follows;

$$R(p,q) = \frac{-i_x p - i_y q + i_z}{\sqrt{1+p^2+q^2}} = \frac{\cos\sigma + p\cos\tau\sin\sigma + q\sin\tau\sin\sigma}{\sqrt{1+p^2+q^2}} \tag{57}$$

Where $i_x = \frac{I(1)}{I(3)} = \frac{\cos\tau\sin\sigma}{\cos\sigma} = \cos\tau\tan\sigma$, $i_y = \frac{I(2)}{I(3)} = \frac{\sin\tau\sin\sigma}{\cos\sigma} = \sin\tau\tan\sigma$. Comparing (57) with (16), two main differences, surface albedo is ignored (assumed to be constant over the whole surface).

Using the following discrete approximations for *p* and *q*;

$$p = Z(x,y) - Z(x-1,y) \tag{58}$$

and

$$q = Z(x,y) - Z(x,y-1) \tag{59}$$

Shah linearized the function $f = E - R = 0$ in terms of Z in the vicinity of Z^{k-1} which is the surface recovered in iteration $k-1$. For a fixed point (x,y) and a given image E, a linear approximation (using Taylor series expansion up through the first order terms) of the function f about a given depth map Z^{k-1}. For an N by N image, there are N^2 such equations, which will form a linear system. This system can be solved easily using the Jacobi iterative scheme, which simplifies the Taylor series expansion up to the first order of f into the following equation;

$$f(Z(x,y)) = 0 \approx f(Z^{n-1}(x,y)) + (Z(x,y) - Z^{n-1}(x,y))\frac{df(Z^{n-1}(x,y))}{dZ(x,y)} \tag{60}$$

Let $Z^n(x,y) = Z(x,y)$ then

$$Z^n(x,y) = Z^{n-1}(x,y) - \frac{f(Z^{n-1}(x,y))}{\frac{df(Z^{n-1}(x,y))}{dZ(x,y)}} \tag{61}$$

where,

$$\frac{df(Z^{n-1}(x,y))}{dZ(x,y)} = \frac{(p+q)(pi_x + qi_y + 1)}{\sqrt{(1+p^2+q^2)^3}\sqrt{1+i_x^2+i_y^2}} - \frac{(i_x + i_y)}{\sqrt{(1+p^2+q^2)}\sqrt{1+i_x^2+i_y^2}} \tag{62}$$

Assuming $Z^0(x,y) = 0$, then $Z(x,y)$ can be extracted iteratively from (61).

Let's do it …

We now can summarize the algorithm of shape from shading using Shah approach as follows; given the gray-scale image of the object, do the following;

Step 1:

Compute surface albedo and illumination direction using the algorithm described in section 6.

```
% read the image
E = imread('sphere.jpg');

% making sure that it is a grayscale image
if isrgb(E)
    E = rgb2gray(E);
end
```

```
E = double(E);

% first compute the surface albedo and illumination direction ...
[albedo,I,slant,tilt] = estimate_albedo_illumination (E);
```

Step 2:

Initializations: the surface, the surface derivatives in x and y directions and the maximum number of iterations.

```
% initializations ...
[M,N] = size(E);

% surface normals
p = zeros(M,N);
q = zeros(M,N);

% the surface
Z = zeros(M,N);

% surface derivatives in x and y directions
Z_x = zeros(M,N);
Z_y = zeros(M,N);

% maximum number of iterations
maxIter = 200;

% the normalized illumination direction
ix = cos(tilt) * tan(slant);
iy = sin(tilt) * tan(slant);
```

Step 3:

Iterate the following steps until a suitable stopping criterion is met:

- Using the illumination direction and the currently estimated surface normals, compute the corresponding reflectance map, refer to (57).
- Make sure that the reflectance map is positive at each pixel, set negative values to zero.
- Compute our function *f*, which is the deviation of the computed reflectance map from the original image.
- Compute the derivative of *f* with respect to our surface *Z*, refer to (62).
- Update our surface, refer to (61).
- Compute the surface derivatives with respect to x and y.

- Using the updated surface, compute new surface normals, refer to (58) and (59).

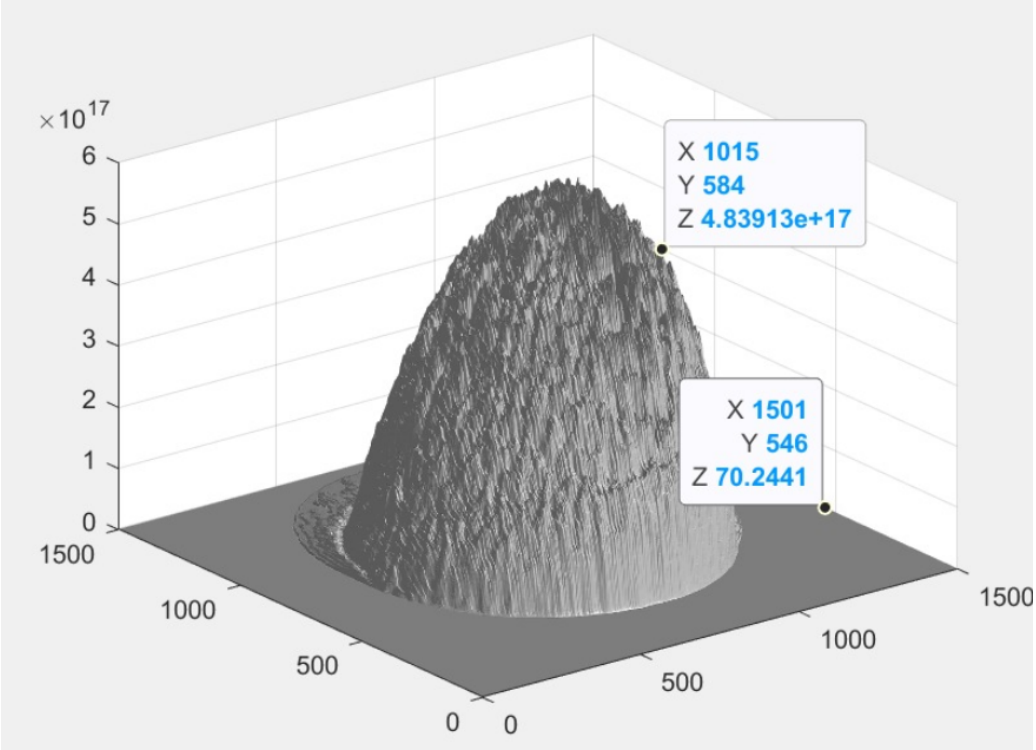
```
for k = 1 : maxIter
    % using the illumination direction and the currently estimate
    % surface normals, compute the corresponding reflectance map.
    % refer to (57) ...
    R =
       (cos(slant) + p .* cos(tilt)*sin(slant)+ q .*
                   sin(tilt)*sin(slant))./sqrt(1 + p.^2 + q.^2);

    % at each iteration, make sure that the reflectance map is positive at
    % each pixel, set negative values to zero.
    R = max(0,R);

    % compute our function f which is the deviation of the computed
    % reflectance map from the original image ...
    f = E - R;

    % compute the derivative of f with respect to our surface Z ... refer
    % to (62)
    df_dZ =
        (p+q).*(ix*p + iy*q + 1)./(sqrt((1 + p.^2 + q.^2).^3)*
            sqrt(1 + ix^2 + iy^2))-(ix+iy)./(sqrt(1 + p.^2 + q.^2)*
            sqrt(1 + ix^2 + iy^2));


    % update our surface ... refer to (61)
    Z = Z - f./(df_dZ + eps); % to avoid dividing by zero

    % compute the surface derivatives with respect to x and y
    Z_x(2:M,:) = Z(1:M-1,:);
    Z_y(:,2:N) = Z(:,1:N-1);

    % using the updated surface, compute new surface normals, refer to (58)
    % and (59)
    p = Z - Z_x;
    q = Z - Z_y;
end
```

Step 4:

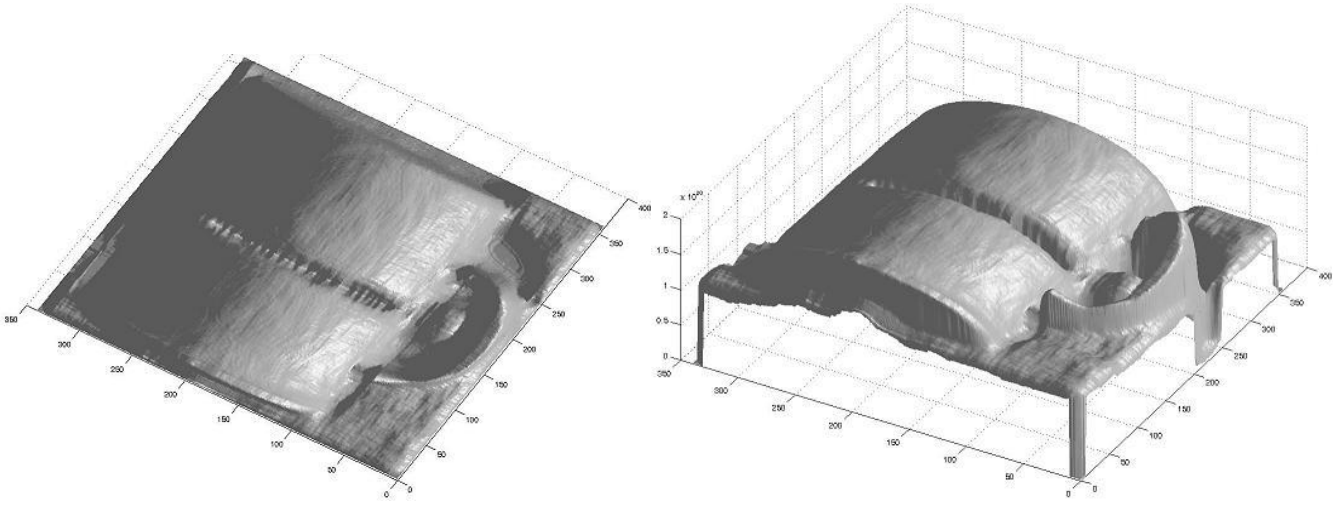
Smooth the obtained surface, then visualize your sphere … see figure (8)

```
% smoothing the recovered surface
Z = medfilt2(abs(Z),[21 21]);

% visualizing the result
figure;
surfl(Z);
shading interp;
colormap gray(256);
```

```
lighting phong;
```



Here is our sphere reconstructed using Shah's approach.



the reconstructed mug using Shah's approach.

Final Code

```
% sphere radius ...
r = 50;
% lets define the xy domain (pixel grid)... x and y are data grids
% (matrices) where the our surface Z = Z(x,y) can be calculated at
% each point (x,y).
% use finner grid to enhance the resolution
[x,y] = meshgrid(-1.5*r:0.1:1.5*r,-1.5*r:0.1:1.5*r);
% surface albedo ...
albedo = 0.5;
% illumination direction ...
I = [0.2 0 0.98]';
% surface partial derivates at each point in the pixel grid ...
p = -x./sqrt(r^2-(x.^2 + y.^2)); % in the x direction
q = -y./sqrt(r^2-(x.^2 + y.^2)); % in the y direction
% now lets compute the image brightness at each point in the pixel grid ...
% the reflectance map will be ...
R = (albedo.(-I(1). p - I(2).* q + I(3)))./sqrt(1 + p.^2 + q.^2);

% the points of the background are those who don't satisfy the sphere
% equation leading to a negative under the square root of the equation of Z
% = Z(x,y)
mask = ((r^2 - (x.^2 + y.^2) >= 0));

% now we can mask out those points
R = R .* mask;

% converting the reflectance map to image irradiance (by setting negative
% values to zeros)
E = max(0,R);

% converting the image irradiance to a gray scale image
E = E ./ max(E(:));

% displaying our synthetic sphere image ...
figure;
imshow(E);
axis off;

% saving our image
imwrite(E,'sphere.jpg');
```

```
% read the image
E = double(imread('sphere.jpg'));

% normalizing the image to have maximum of one
E = E ./ max(E(:));

% compute the average of the image brightness
Mu1 = mean(E(:));

% compute the average of the image brightness square
Mu2 = mean(mean(E.^2));
% now lets compute the image's spatial gradient in x and y directions
[Ex,Ey] = gradient(E);
% normalize the gradients to be unit vectors
Exy = sqrt(Ex.^2 + Ey.^2);
nEx = Ex ./(Exy + eps); % to avoid dividing by zero
nEy = Ey ./(Exy + eps);
% computing the average of the normalized gradients
avgEx = mean(Ex(:));
avgEy = mean(Ey(:));
% now lets estimate the surface albedo
gamma = sqrt((6 (pi^2) Mu2) - (48 * (Mu1^2)));
albedo = gamma/pi;
% estimating the slant
slant = acos((4*Mu1)/gamma);
% estimating the tilt
tilt = atan(avgEy/avgEx);
if tilt < 0
tilt = tilt + pi;
end
% the illumination direction will be ...
I = [cos(tilt)*sin(slant) sin(tilt)*sin(slant) cos(slant)];
% Display computed values
fprintf('Estimated Albedo: %.4f\n', albedo);
fprintf('Estimated Slant: %.4f\n', slant);
fprintf('Estimated Tilt: %.4f\n', tilt);
fprintf('Estimated Illumination Direction: [%.4f %.4f %.4f]\n', I);

% Display and Save Gradient Images

% Display image gradients Ex and Ey
figure;
subplot(1,2,1);
imshow(mat2gray(Ex));
title('Gradient in X Direction');
imwrite(mat2gray(Ex), 'gradient_x.jpg');

subplot(1,2,2);
imshow(mat2gray(Ey));
title('Gradient in Y Direction');
imwrite(mat2gray(Ey), 'gradient_y.jpg');
```

```
% Display normalized gradients nEx and nEy
figure;
subplot(1,2,1);
imshow(mat2gray(nEx));
title('Normalized Gradient in X Direction');
imwrite(mat2gray(nEx), 'normalized_gradient_x.jpg');

subplot(1,2,2);
imshow(mat2gray(nEy));
title('Normalized Gradient in Y Direction');
imwrite(mat2gray(nEy), 'normalized_gradient_y.jpg');



% read the image
E = imread('sphere.jpg');
% making sure that it is a grayscale image
if size(E,3) == 3
   E = rgb2gray(E);
end
E = double(E);
% normalizing the image to have maximum of one
E = E ./ max(E(:));
% first compute the surface albedo and illumination direction ...
[albedo,I,slant,tilt] = estimate_albedo_illumination (E);
```

```
% initializations ...
[M,N] = size(E);
% surface normals
p = zeros(M,N);
q = zeros(M,N);
% the surface
Z = zeros(M,N);
% surface derivatives in x and y directions
Z_x = zeros(M,N);
Z_y = zeros(M,N);
% maximum number of iterations
maxIter = 200;
% the normalized illumination direction
ix = cos(tilt) * tan(slant);
iy = sin(tilt) * tan(slant);
for k = 1 : maxIter
    % using the illumination direction and the currently estimate
    % surface normals, compute the corresponding reflectance map.
    % refer to (57) ...
    R =  (cos(slant) + p .* cos(tilt)sin(slant)+ q . sin(tilt)*sin(slant))./sqrt(1 + p.^2 + q.^2);

   % at each iteration, make sure that the reflectance map is positive at
   % each pixel, set negative values to zero.
   R = max(0,R);

   % compute our function f which is the deviation of the computed
   % reflectance map from the original image ...
   f = E - R;

   % compute the derivative of f with respect to our surface Z ... refer
   % to (62)
   df_dZ = (p+q).(ix*p + iy*q + 1)./(sqrt((1 + p.^2 + q.^2).^3) sqrt(1 + ix^2 + iy^2))-(ix+iy)./(sqrt(1 + p.^


   % update our surface ... refer to (61)
   Z = Z - f./(df_dZ + eps); % to avoid dividing by zero

   % compute the surface derivatives with respect to x and y
   Z_x(2:M,:) = Z(1:M-1,:);
   Z_y(:,2:N) = Z(:,1:N-1);

   % using the updated surface, compute new surface normals, refer to (58)
   % and (59)
   p = Z - Z_x;
   q = Z - Z_y;
end
% smoothing the recovered surface
Z = medfilt2(abs(Z),[21 21]);

% visualizing the result
figure;
surfl(Z);
shading interp;
colormap gray(256);
lighting phong;
```

```
function [albedo, I, slant, tilt] = estimate_albedo_illumination(E)
    % Compute brightness statistics
    Mu1 = mean(E(:));
    Mu2 = mean(mean(E.^2));

    % Compute estimated albedo
    gamma = sqrt((6 * (pi^2) * Mu2) - (48 * (Mu1^2)));
    albedo = gamma / pi;

    % Estimate slant
    slant = acos((4 * Mu1) / gamma);

    % Compute the image gradients
    [Ex, Ey] = gradient(E);

    % Compute averages of gradients
    avgEx = mean(Ex(:));
    avgEy = mean(Ey(:));

    % Estimate tilt
    tilt = atan(avgEy / avgEx);
    if tilt < 0
        tilt = tilt + pi;
    end
% Compute illumination direction
   I = [cos(tilt) * sin(slant), sin(tilt) * sin(slant), cos(slant)];
end
```