CSE 152 HW 2

1. $l: y = x + \lambda \quad \lambda \in (-\infty, \infty)$ at distance d from the origin

the perpendicular line that the same distance d from the origin has the equation $y = -x \pm \lambda$

since $|\lambda| = \sqrt{2}\, d$ $\quad y = -x \pm \sqrt{2}\, d$

2. A point x lies on the line l if and only if

$$l^T x = x^T l = 0$$

a) points $P_1 = (x_1, y_1, z_1)$

$P_2 = (x_2, y_2, z_2)$

$l = P_1 \times P_2 = (y_1 z_2 - z_1 y_2,\ z_1 x_2 - x_1 z_2,\ x_1 y_2 - y_1 x_2)$

$l^T P_1 = P_1^T l = x_1(y_1 z_2 - y_2 z_1) + y_1(x_2 z_1 - x_1 z_2) + z_1(x_1 y_2 - x_2 y_1)$

$= x_1 y_1 z_2 - x_1 y_2 z_1 + x_2 y_1 z_1 - x_1 y_1 z_2 + x_1 y_2 z_1 - x_2 y_1 z_1$

$= 0 + 0 = 0$

Similarly, $l^T P_2 = P_2^T l = x_2(y_1 z_2 - y_2 z_1) + y_2(x_2 z_1 - x_1 z_2) + z_2(x_1 y_2 - x_2 y_1)$

$= x_2 y_1 z_2 - x_2 y_2 z_1 + x_2 y_2 z_1 - x_1 y_2 z_2 + x_1 y_2 z_2 - x_2 y_1 z_2$

$= 0 + 0 = 0$

Therefore $P_1 \times P_2$ is the equation of the line connecting the two points.

b) lines $L_1 = (a_1, b_1, c_1)$

$L_2 = (a_2, b_2, c_2)$

$X = L_1 \times L_2 = (b_1 c_2 - c_1 b_2,\ c_1 a_2 - a_1 c_2,\ a_1 b_2 - b_1 a_2)$

$L_1^T x = x^T L_1 = 0$ so x lies on L_1

$L_2^T x = x^T L_2 = 0$ so x lies on L_2

Therefore $L_1 \times L_2 = x$ is the point of intersection of the two lines

3. $P_1 = (1, 4, 1) \quad P_2 = (4, 5, 7)$

$= (1, 4, 1) \quad = (4, 5, 1)$

$L = P_1 \times P_2 = (4 - 5,\ 5 - 4,\ 5 - 16)$

$= (-1, 3, -11)$

Yes $A'B'C'D'$ will remain a convex polygon because the pinhole perspective is essentially scaling the image, not warping it

b) $V_{P_1} = A'B' \times C'D' = (0, 30, 0) = (0, 3, 0)$

$= (4, 5, 1)$

$V_{P_2} = B'C' \times A'D' = (24, 24, 6) = (8, 8, 2)$

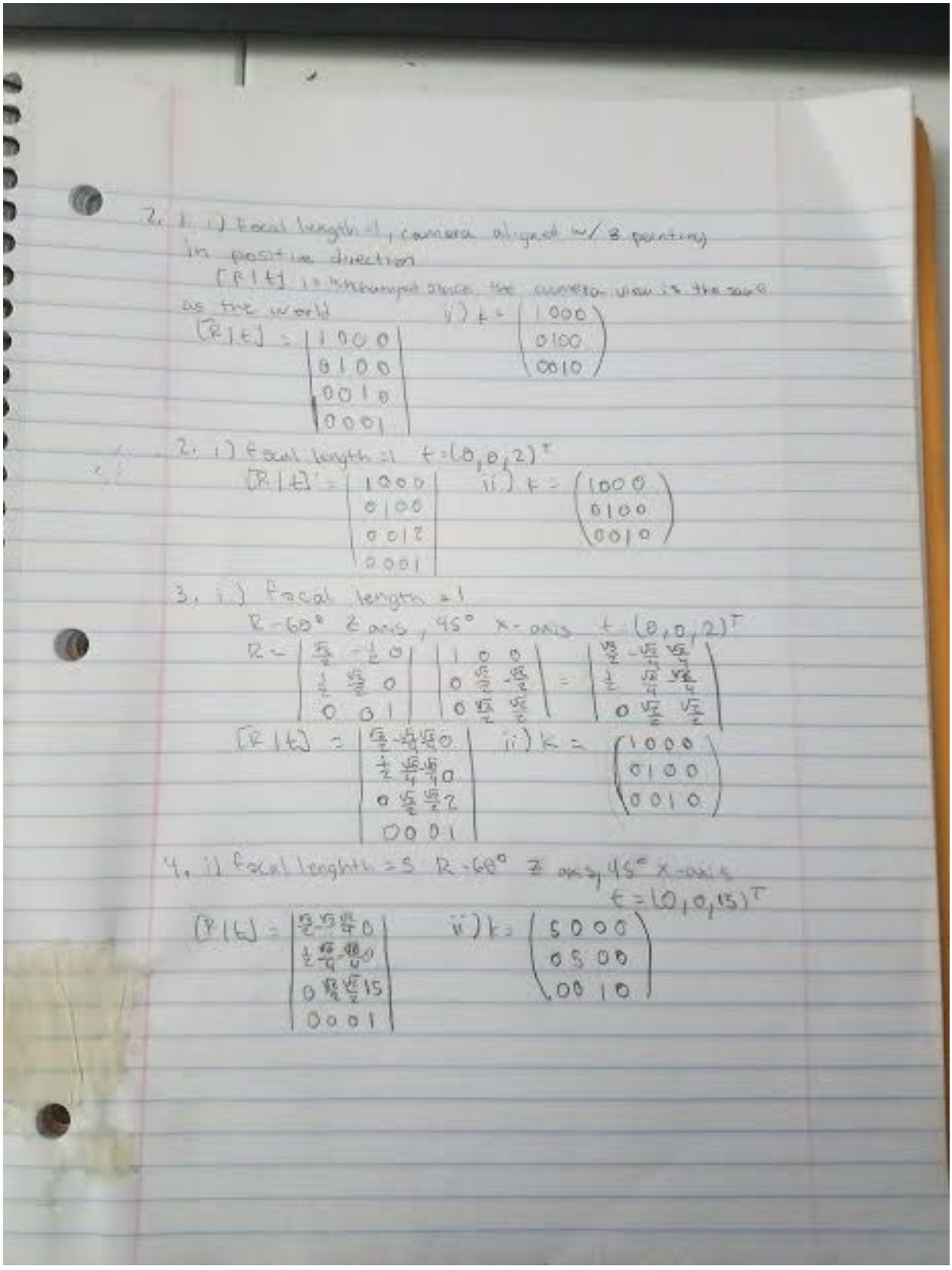
$A'B' = (1, 1, 1) \times (4, 6, 1) = (-5, 0, 5)$

$C'D' = (4, 6, 1) \times (7, 4, 1) = (2, -2, -4)$

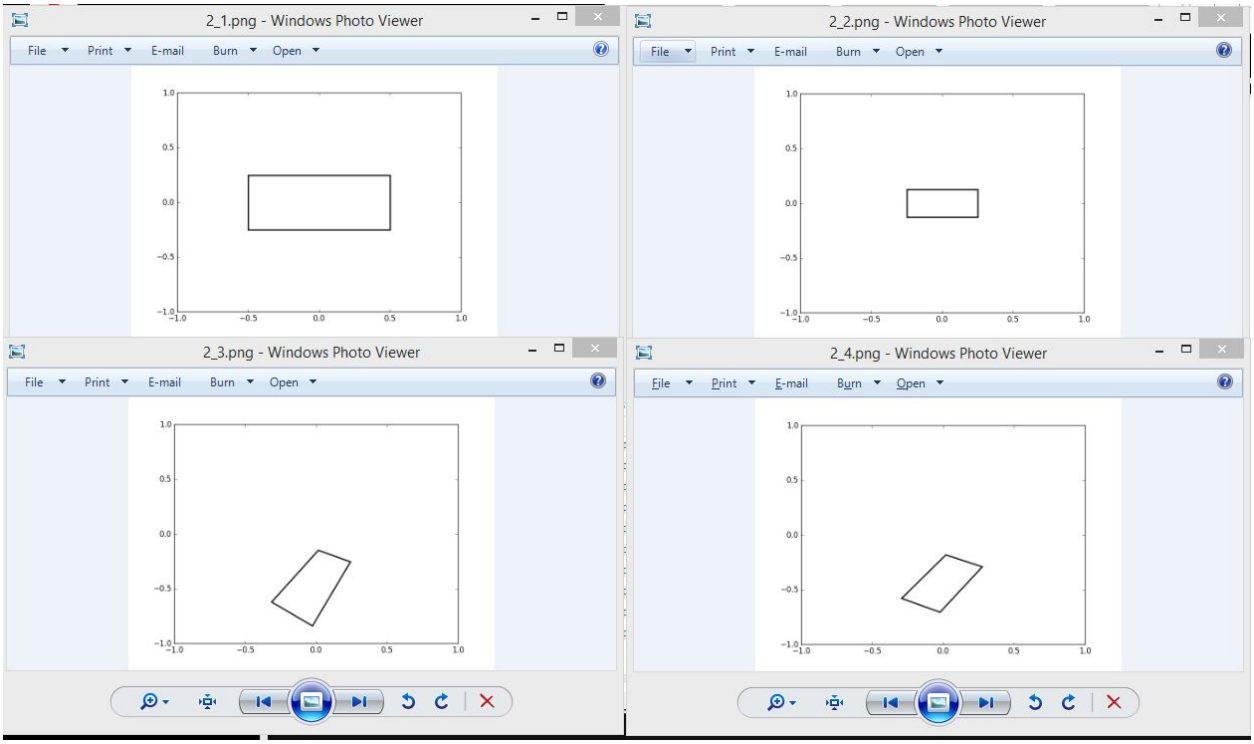
$B'C' = (4, 6, 1) \times (4, 6, 1) = (0, 3, -18)$

$A'D' = (1, 1, 1) \times (7, 4, 1) = (-3, 1, 2)$

R|t is the extrinsic matrix which changes the position and rotation of the camera while K is the intrinsic matrix which changes the focal length of the camera for this hw. The intrinsic matrix can also change the field of view of the camera.



iii) The 2d view of the world from the camera is obtained by the equation K*[R|t]*X.transpose
1)The first image appears smaller than the input coordinates because the camera is at the origin while the rectangle is 2 units away on the z axis
2)The second image is smaller because the the rectangle was moved 2 away on the z axis
3)This image was rotated about the z axis by 60 degrees, which appears as a counterclockwise rotation in 2d. This rotation occurs in place because the rectangle is centered on the axis it is being rotated on. The rectangle is then rotated 45 degrees about the x axis, which appears as the rectangle orbiting downwards. This is because the rectangle is 4 units away from the x axis of this left hand coordinate system.
4)The increase in focal length causes the camera to focus on a further plane of view, while the translation moves the image to that plane of view that is being focused on.This image looks less warped the 3rd image.

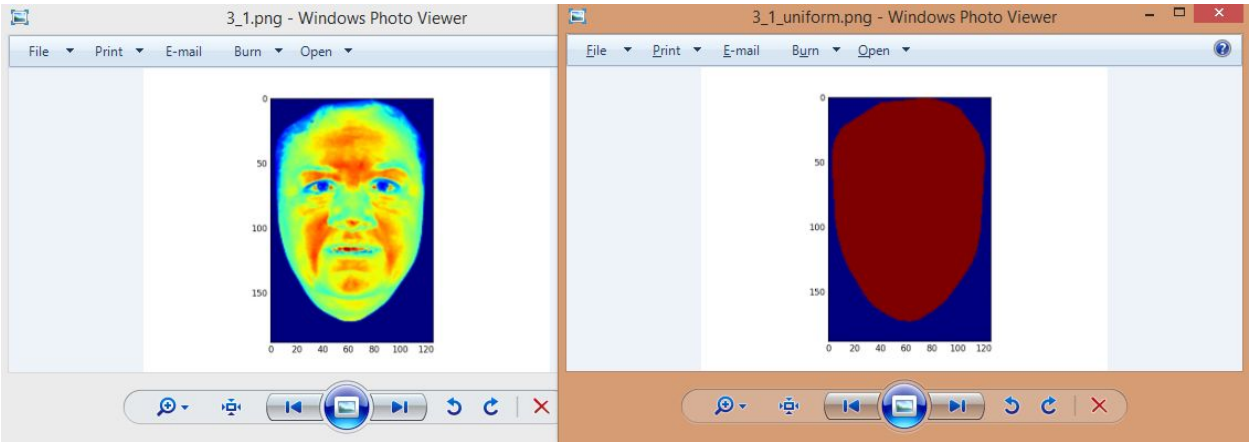


I changed the parameters of my rot_trans_mat in order to change the extrinsic matrix. The function Kmat was used to change the focal length.

```python
import os
import sys
import numpy
import math
import scipy
from plot_square import plot_square
#import access_facedata

def Rot_Trans_mat(xAng,yAng,zAng,trans):
    xAng = numpy.radians(xAng)
    yAng = numpy.radians(yAng)
    zAng = numpy.radians(zAng)
    zRot = numpy.array([[math.cos(zAng),-(math.sin(zAng)),0],
                        [math.sin(zAng),math.cos(zAng),0],
                        [0,0,1]]);
    yRot = numpy.array([[math.cos(yAng),0,math.sin(yAng)],
                        [0,1,0],
                        [-(math.sin(yAng)),0,math.cos(yAng)]]);
    xRot = numpy.array([[1,0,0],
                        [0,math.cos(xAng),-(math.sin(xAng))],
                        [0,math.sin(xAng),math.cos(xAng)]]);
    rotMat = numpy.matmul(xRot,numpy.matmul(yRot,zRot));
    RotTransMat = numpy.concatenate((rotMat,numpy.array([[0,0,0]])));
    RotTransMat = numpy.concatenate((RotTransMat,trans.T),axis = 1);
    return RotTransMat;

def Kmat(f):
    K = numpy.array([[f,0,0,0],[0,f,0,0],[0,0,1,0]]);
    return K;

def modelMat(coordinate):
    M = numpy.array([[1,0,0],[0,1,0],[0,0,1],[0,0,0]]);
    M = numpy.concatenate((M,coordinate.T),axis = 1);
    return M

Trans = numpy.array([[0,0,15,1]]);
X1 = numpy.array([[-1,-0.5,2,1]]);
X2 = numpy.array([[1,-0.5,2,1]]);
X3 = numpy.array([[1,0.5,2,1]]);
X4 = numpy.array([[-1,0.5,2,1]]);
p1 = numpy.matmul(Kmat(5),numpy.matmul(Rot_Trans_mat(45,0,60,Trans),X1.T));
p2 = numpy.matmul(Kmat(5),numpy.matmul(Rot_Trans_mat(45,0,60,Trans),X2.T));
p3 = numpy.matmul(Kmat(5),numpy.matmul(Rot_Trans_mat(45,0,60,Trans),X3.T));
p4 = numpy.matmul(Kmat(5),numpy.matmul(Rot_Trans_mat(45,0,60,Trans),X4.T));

verts = [
    (p1[0]/float(p1[2]),p1[1]/float(p1[2])), # left, bottom
    (p2[0]/float(p2[2]),p2[1]/float(p2[2])),  # left, top
    (p3[0]/float(p3[2]),p3[1]/float(p3[2])),   # right, top
    (p4[0]/float(p4[2]),p4[1]/float(p4[2])),  # right, bottom
    (0., 0.),      # ignored
    ]
x_min = -1
x_max = 1
y_min = -1
y_max = 1
plot_square(verts, x_min, x_max, y_min, y_max)
```

3)

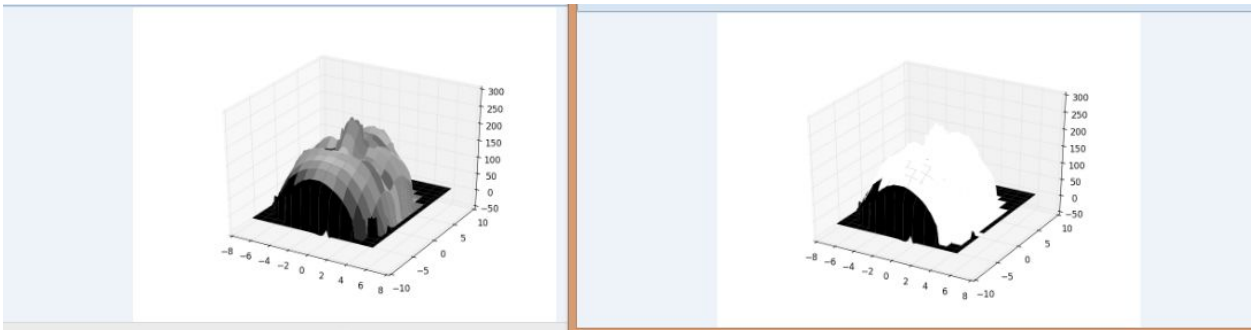
i) The left is a albedo texture map that has a variable reflectance properties depending on the location of the face. The right image uses a uniform albedo despite different depths/curvatures.




```
30   #3.1
31   #plt.imshow(albedo)
32   #plt.show()
33
```

This code is for the left image, albedo was changed to uniform_albedo for the right image

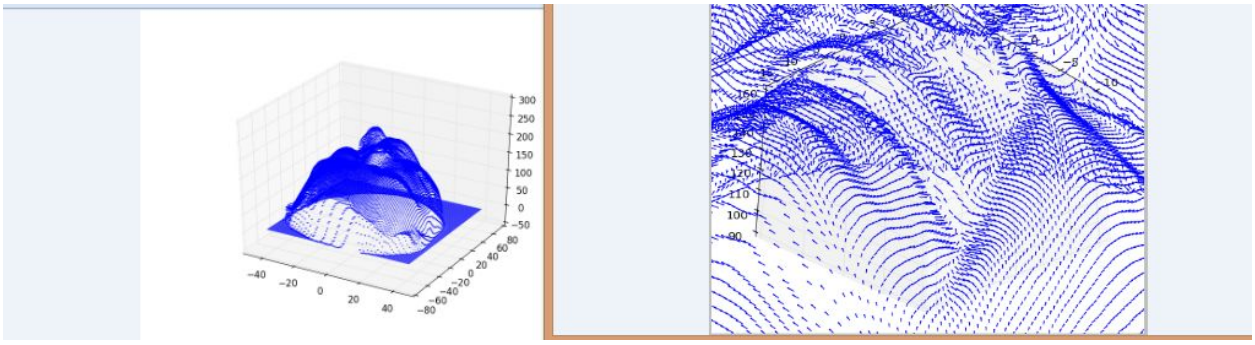
ii) The left heightmap has the albedo texture used in order to color it. This gives a variable grayscale shade depending on the part of the image. The right image uses the uniform albedo which has a single reflectance property, and is therefore all white.



```
#3.2
#fig = plt.figure()
#ax = fig.gca(projection='3d')
a = 7
X = np.arange(-6.3*a, 6.3*a, .1*a)
Y = np.arange(-9.4*a, 9.4*a, .1*a)
X, Y = np.meshgrid(X, Y)
"""surf = ax.plot_surface(X,Y,heightmap,cmap=cm.gray,
                          linewidth=0, antialiased=True,
                          facecolors=albedo.astype(str))
plt.show()"""
```

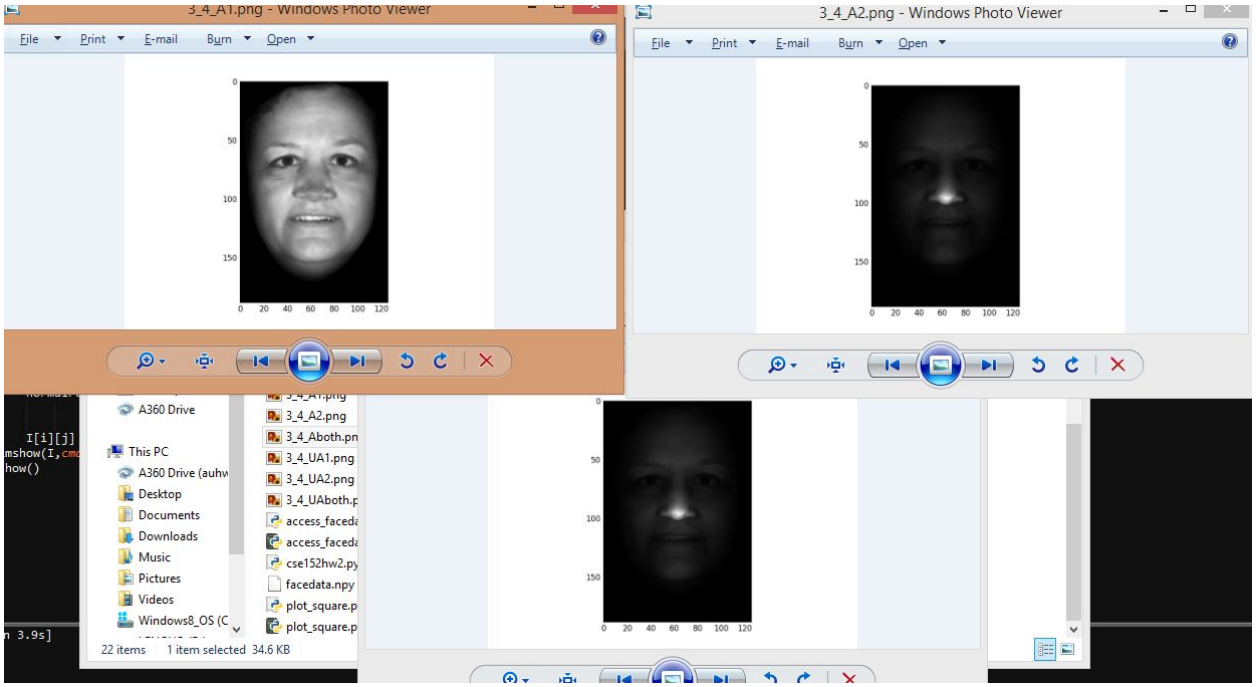
Two grids X and Y are made to be the same size as the heightmap, which gives the Z values.

iii) The normals were calculated by approximating (-df/dx,-df/dy,1). df/dx and df/dy were calculated by looping through the heightmap pixel by pixel. The current df/dx was the right - left pixel divided by 2 while df/dy was the bottom minus the top divided by 2. These values were then stored with a z of 1 and normalized. The flat plane below the face of the image on the right has normal vectors pointing upwards while the mouth/nose have variable normal vectors due to the curvature of the face.

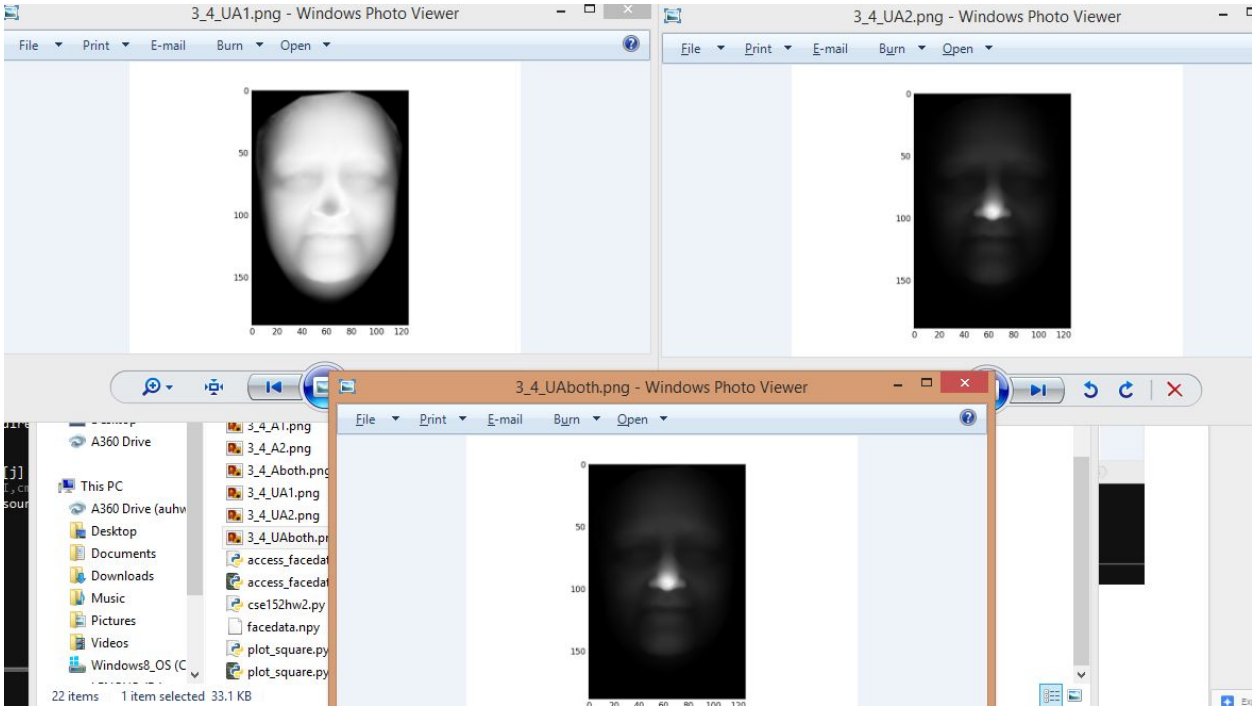


```
#3.3
normalX = np.zeros_like(X)
normalY = np.zeros_like(Y)
normalZ = np.zeros_like(heightmap)
for i in range(1,len(heightmap)-1):
    for j in range(1,len(heightmap[0])-1):
        dx = -(heightmap[i+1][j] - heightmap[i-1][j])/2.0
        dy = -(heightmap[i][j+1] - heightmap[i][j-1])/2.0
        normx = dx/np.sqrt(np.square(dx)+np.square(dy)+1)
        normy = dy/np.sqrt(np.square(dx)+np.square(dy)+1)
        normz = 1/np.sqrt(np.square(dx)+np.square(dy)+1)
        normalX[i][j] = normx
        normalY[i][j] = normy
        normalZ[i][j] = normz
#quiv = ax.quiver(X,Y,heightmap,normalX,normalY,normalZ)
#plt.show()
```

iv)
These image’s use different combination’s of 2 point light sources. The left image uses the [120,180,300], the right uses the [20 20 300], and bottom uses both. The reflectance varied based on the size,distance,direction of the light. These images use the albedo map which does a good job of giving variable reflectance properties based on the depth. I did not expect the image with both light sources to be dimmer since it was a combination of the 2 and may have made a mistake there.



The only difference between these and the above images is that these use the uniform albedo map. This turns out to make a huge difference in the amount of detail shown in the image. The left image reflected way too much light throughout the whole face since all the pixels had the same properties. The opposite was true for the right image, which did not have enough light to show the details of the face.




```
64  #3.4
65  ls1 = light_source[0]
66
67  ls2 = light_source[1]
68  I = np.zeros_like(heightmap)
69  for i in range(len(heightmap)):
70      for j in range(len(heightmap[0])):
71          surfnorm = np.array([normalX[i][j],
72                               normalY[i][j],heightmap[i][j]])
73          direc = ls1-surfnorm
74          distance = math.pow(direc[0],2)+math.pow(direc[1],2)+math.pow(direc[2],2)
75          normdirection = np.array([direc[0]/(math.sqrt(distance)),
76                    direc[1]/(math.sqrt(distance)),
77                    direc[2]/(math.sqrt(distance))])
78          direc2 = ls2-surfnorm
79          distance2 = math.pow(direc2[0],2)+math.pow(direc2[1],2)+math.pow(direc2[2],2)
80          normdirection2 = np.array([direc2[0]/(math.sqrt(distance2)),
81                    direc2[1]/(math.sqrt(distance2)),
82                    direc2[2]/(math.sqrt(distance2))])
83          I[i][j] = np.dot(np.dot(albedo[i][j],surfnorm.T),(normdirection/distance))
84  #plt.imshow(I,cmap = cm.gray)
85  print light_source
86  plt.show()
```

$$I = a(x,y)\hat{\mathbf{n}}(x,y)^{\top}\hat{\mathbf{s}}(x,y)\frac{s_0}{(d(x,y))^2}$$

This code implemented

Model matrix was not used
The Rot_Trans_mat function took in angles and a translation matrix and output a 4x4 matrix
Kmat output a 3x4 matrix with focal length as input
These matrices were multiplied by the transpose of the coordinate and that result was normalized and put into plot_square

```
import os
import sys
import numpy
import math
import scipy
from plot_square import plot_square
#import access_facedata

def Rot_Trans_mat(xAng,yAng,zAng,trans):
    xAng = numpy.radians(xAng)
    yAng = numpy.radians(yAng)
    zAng = numpy.radians(zAng)
    zRot = numpy.array([[math.cos(zAng),-(math.sin(zAng)),0],
                        [math.sin(zAng),math.cos(zAng),0],
                        [0,0,1]]);
    yRot = numpy.array([[math.cos(yAng),0,math.sin(yAng)],
                        [0,1,0],
                        [-(math.sin(yAng)),0,math.cos(yAng)]]);
    xRot = numpy.array([[1,0,0],
                        [0,math.cos(xAng),-(math.sin(xAng))],
                        [0,math.sin(xAng),math.cos(xAng)]]);
    rotMat = numpy.matmul(xRot,numpy.matmul(yRot,zRot));
    RotTransMat = numpy.concatenate((rotMat,numpy.array([[0,0,0]])));
    RotTransMat = numpy.concatenate((RotTransMat,trans.T),axis = 1);
    return RotTransMat;

def Kmat(f):
    K = numpy.array([[f,0,0,0],[0,f,0,0],[0,0,1,0]]);
    return K;

def modelMat(coordinate):
    M = numpy.array([[1,0,0],[0,1,0],[0,0,1],[0,0,0]]);
    M = numpy.concatenate((M,coordinate.T),axis = 1);
    return M
```

3.1 just output the albedo texture
3.2 creates the grids x and y and uses the heightmap for z values. These are then inputted into plot_surface which has the facecolors parameter set to albedo in order to texture the grid with different diffuse properties
3.3 calculates the surface normals by checking the top/bottom/left/right pixels and uses quiver to plot them
3.4 uses the albedo map and point lights to create the image. These are based on the direction of the lights and the surface normals provided

```
#3.1
#plt.imshow(albedo)
#plt.show()

#3.2
#fig = plt.figure()
#ax = fig.gca(projection='3d')
a = 7
X = np.arange(-6.3*a, 6.3*a, .1*a)
Y = np.arange(-9.4*a, 9.4*a, .1*a)
X, Y = np.meshgrid(X, Y)
"""surf = ax.plot_surface(X,Y,heightmap,cmap=cm.gray,
                          linewidth=0, antialiased=True,
                          facecolors=albedo.astype(str))
plt.show()"""


#3.3
normalX = np.zeros_like(X)
normalY = np.zeros_like(Y)
normalZ = np.zeros_like(heightmap)
for i in range(1,len(heightmap)-1):
    for j in range(1,len(heightmap[0])-1):
        dx = -(heightmap[i+1][j] - heightmap[i-1][j])/2.0
        dy = -(heightmap[i][j+1] - heightmap[i][j-1])/2.0
        normx = dx/np.sqrt(np.square(dx)+np.square(dy)+1)
        normy = dy/np.sqrt(np.square(dx)+np.square(dy)+1)
        normz = 1/np.sqrt(np.square(dx)+np.square(dy)+1)
        normalX[i][j] = normx
        normalY[i][j] = normy
        normalZ[i][j] = normz
#quiv = ax.quiver(X,Y,heightmap,normalX,normalY,normalZ)
#plt.show()
```

```
63
64  #3.4
65  ls1 = light_source[0]
66
67  ls2 = light_source[1]
68  I = np.zeros_like(heightmap)
69  for i in range(len(heightmap)):
70      for j in range(len(heightmap[0])):
71          surfnorm = np.array([normalX[i][j],
72                               normalY[i][j],heightmap[i][j]])
73          direc = ls1-surfnorm
74          distance = math.pow(direc[0],2)+math.pow(direc[1],2)+math.pow(direc[2],2)
75          normdirection = np.array([direc[0]/(math.sqrt(distance)),
76                      direc[1]/(math.sqrt(distance)),
77                      direc[2]/(math.sqrt(distance))])
78          direc2 = ls2-surfnorm
79          distance2 = math.pow(direc2[0],2)+math.pow(direc2[1],2)+math.pow(direc2[2],2)
80          normdirection2 = np.array([direc2[0]/(math.sqrt(distance2)),
81                      direc2[1]/(math.sqrt(distance2)),
82                      direc2[2]/(math.sqrt(distance2))])
83          I[i][j] = np.dot(np.dot(albedo[i][j],surfnorm.T),(normdirection/distance))
84  #plt.imshow(I,cmap = cm.gray)
85  print light_source
86  plt.show()
```