6.869 PSet 1

Nada Hussein

2 March 2021

Problem 1

Below you can see my orthographic and perspective projection images:

Figure 1: Orthographic View

Figure 2: Perspective View

I took the perspective projection photo up close, whereas the orthographic image was taken from far away with zoom. In the perspective projection, we can see that the lines of the tissue box are not parallel, and instead converge to a vanishing points. However, in the orthographic view, the tissue box edges all look approximately parallel when they are parallel in the real world.

Problem 2

We are given that our rotation was θ about the x-axis. We can denote this with a 3x3 rotation matrix:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Going off of this, we get the equations:

$$x = X$$

$$y = \cos(\theta)Y - \sin(\theta)Z$$

However, we need to consider that there is a translation and a scaling of the axes from world to 2D. Because the pixels are square, the scaling factor is the same for both x and y. Thus, we can show the scaling as a factor a in front of the equations we have from the rotation. This leads to:

$$x = aX$$

$$y = a(\cos(\theta)Y - \sin(\theta)Z)$$

Finally, to translate we must offset x by x_{offset} and y by y_{offset}, leading to our final equations:

$$x = aX + x_{offset}$$

$$y = a(\cos(\theta)Y - \sin(\theta)Z) + y_{offset}$$

When we project (0, 0, 0) onto (x_0, y_0), we get the following constraints:

$$x_0 = a(0) + x_{offset} => x_{offset} = x_0$$

$$y_0 = a(\cos(\theta)(0) - \sin(\theta)(0)) + y_{offset} => y_{offset} = y_0$$

We plug these new constraints in to get:

$$x = aX + x_0$$

$$y = a(\cos(\theta)Y - \sin(\theta)Z) + y_0$$

When we project (1, 0, 0) onto (x_1, y_0), we get the following equations:

$$x_1 = a(1) + x_0 => x_1 = a + x_0 => x_1 = a + x_0 => a = x_1 - x_0$$

$$y_0 = a(\cos(\theta)(0) - \sin(\theta)(0)) + y_0 => y_0 = y_0$$

This yields our final projection equations:

$$x = (x_1 - x_0)X + x_0$$

$$y = (x_1 - x_0)[\cos(\theta)Y - \sin(\theta)Z] + y_0$$

Problem 3

We can follow a similar structure to find the constraints for $Z(x, y)$. We start with the equation relating the 3D world to 2D coordinates:

$$y = \cos(\theta)Y - \sin(\theta)Z$$

Rearranging, we get:

$$Z = \frac{\cos(\theta)Y - y}{\sin(\theta)}$$

From here, we get the constraint of the derivative of Z along the edge:

$$\frac{\partial Z}{\partial y} = -\frac{1}{\sin(\theta)}$$

We also know that along the edge, Z will not change. Therefore:

$$\frac{\partial Z}{\partial t} = \nabla Y \cdot t = 0$$

We can write $t = (-n_y, n_x)$. Plugging this in, we get:

$$\frac{\partial Z}{\partial t} = \nabla Y \cdot t = -n_y \partial Z/\partial x + n_x \partial Z/\partial y$$

We also can propagate information from the boundaries to make inferences about the surface orientation. We can say that all of the object faces are planar, in which case all of the second derivatives of Z will be0:

$$\frac{\partial^2 Z}{\partial x^2} = 0$$

$$\frac{\partial^2 Z}{\partial y^2} = 0$$

$$\frac{\partial^2 Z}{\partial x \partial y} = 0$$

Problem 4

To classify a vertical edge, I wanted to take any edge that had an angle from vertical of less than 15 degrees. There were 2 different ranges of angles that this could happen: Between 90-15 degrees and 90+15 degrees, covering the positive sweep, and between -(90-15) and -(90+15) degrees, covering the negative sweep. To implement this, I decided to classify any edge with a theta in these two ranges as vertical. From here, the horizontal edges were the complement of vertical – that is, any edge that was not classified as vertical was classified as horizontal.

To implement the different constraints, I first checked for contact points. If (i, j) was a contact point, I knew that Y must be 0. Thus, for this case, I set Aij and b to the following:

$$Aij = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, b = 0$$

This system captured just the (i, j) point, and set it equal to 0.

After this case, I checked if groundsum was 0. If so, we had three cases to deal with: vertical, horizontal, or planar constraints.

I first considered whether we were dealing with a vertical edge. To do this, I checked if vertical sum $>$ 0. In this case, we know that $\partial Y/\partial y = 1/\cos(\alpha)$. Thus, I set Aij and b as follows:

$$Aij = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, b = 1/\cos(\alpha)$$

This Aij which was meant to approximate $\partial Y/\partial y$ as shown in class.

If we weren't at a vertical edge, I then checked if we were at a horizontal edge, by checking if horizontalsum > 0. If we were at a horizontal edge, we knew that $\partial Y/\partial t = 0$ – that is, the slope should be 0 along the edge. We can approximate $\partial Y/\partial t$ as $-n_y \partial Y/\partial x + n_x \partial Y/\partial y$. We can write $\partial Y/\partial x \approx [[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]$, and $\partial Y/\partial y \approx [[-1, -2, -1], [0, 0, 0], [1, 2, 1]]$. Therefore, in total, we set Aij and b as follows:

$$Aij = -n_y \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} + n_x \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, b = 0$$

Finally, I considered planar constraints. In this case, we simply had to set all second derivatives equal to zero. First, I used the Laplacian to approximate $\frac{\partial^2 Y}{\partial x^2} = [[0, 0, 0], [-1, 2, -1], [0, 0, 0]]$. I then knew that $\frac{\partial^2 Y}{\partial y^2}$ was simply the transpose of $\frac{\partial^2 Y}{\partial x^2}$. Finally, I approximated $\frac{\partial^2 Y}{\partial x \partial y} = [[-1, 0, 1], [0, 0, 0], [1, 0, -1]]$. Adding all these constraints, I ended up with:

$$Aij = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 2 & -1 \\ 0 & 0 & 0 \end{bmatrix}, b = 0$$

$$Aij = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 2 & 0 \\ 0 & -1 & 0 \end{bmatrix}, b = 0$$

$$Aij = \begin{bmatrix} -1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & -1 \end{bmatrix}, b = 0$$

My code for this is shown below:

```
Nconstraints = nrows*ncols*20
Aij = np.zeros((3, 3, Nconstraints))
ii = np.zeros((Nconstraints, 1));
jj = np.zeros((Nconstraints, 1));
```

```
b = np.zeros((Nconstraints, 1));

V = np.zeros((nrows, ncols))
# Create linear contraints
c = 0
for i in range(1, nrows-1):
  for j in range(1, ncols-1):
    if ground[i,j]:
      # Y = 0
      Aij[:,:,c] = np.array([[0, 0, 0], [0, 1, 0], [0, 0, 0]])
      ii[c] = i
      jj[c] = j
      b[c] = 0
      V[i,j] = 0
      c += 1 # increment constraint counter
    else:
      # Check if current neirborhood touches an edge
      edgesum = np.sum(edges[i-1:i+2,j-1:j+2])
      # Check if current neirborhood touches ground pixels
      groundsum = np.sum(ground[i-1:i+2,j-1:j+2])
      # Check if current neirborhood touches vertical pixels
      verticalsum = np.sum(vertical_edges[i-1:i+2,j-1:j+2])
      # Check if current neirborhood touches horizontal pixels
      horizontalsum = np.sum(horizontal_edges[i-1:i+2,j-1:j+2])
      # Orientation of edge (average over edge pixels in current
      # neirborhood)

      nx = dmdx[i,j]/mag[i,j]
      ny = dmdy[i,j]/mag[i,j]

      dYdx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])/8
      dYdy = dYdx.T
      dYdt = -ny*dYdx + nx*dYdy
      d2Ydx2 = np.array([[0, 0, 0], [-1, 2, -1], [0, 0, 0]])/4
      d2Ydy2 = d2Ydx2.T
      d2Ydxdy = np.array([[-1, 0, 1], [0, 0, 0], [1, 0, -1]])/4

      if contact_edges[i, j]:
        #Y = 0
        Aij[:,:,c] = np.array([[0, 0, 0], [0, 1, 0], [0, 0, 0]])
        ii[c] = i
        jj[c] = j
        b[c] = 0
        c += 1 # increment constraint counter

      if groundsum == 0:
        if verticalsum > 0:
          #vertical edge conditions
          #dY/dy = 1/cos(alpha)
          Aij[:,:,c] = dYdy
          ii[c] = i
          jj[c] = j
          b[c] = 1/np.cos(alpha)
          c += 1 # increment constraint counter
        elif horizontalsum > 0:
          #horizontal edge conditions
          #dY/dt = 0
          Aij[:,:,c] = dYdt
          ii[c] = i
```

```
  jj[c] = j
  b[c] = 0
  c += 1 # increment constraint counter
#planar conditions
#d2Y/dx2 = 0
Aij[:,:,c] = d2Ydx2
ii[c] = i
jj[c] = j
b[c] = 0
c += 1 # increment constraint counter

#d2Y/dy2 = 0
Aij[:,:,c] = d2Ydy2
ii[c] = i
jj[c] = j
b[c] = 0
c += 1 # increment constraint counter

#d2Y/dxdy = 0
Aij[:,:,c] = d2Ydxdy
ii[c] = i
jj[c] = j
b[c] = 0
c += 1 # increment constraint counter
```

Problem 5

Below you can see the results of image 1, 2, and 4, which worked out well:

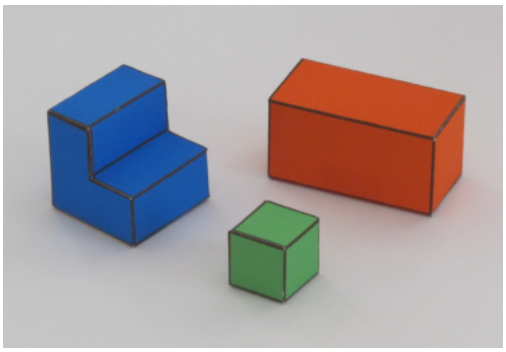

Figure 3: Image 1

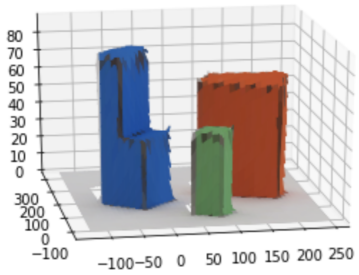

Figure 4: Image 1 3D Reconstruction

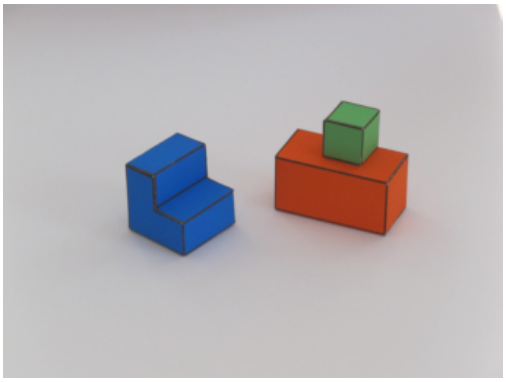

Figure 5: Image 2

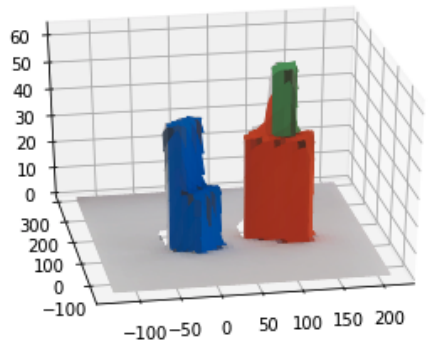

Figure 6: Image 2 3D Reconstruction

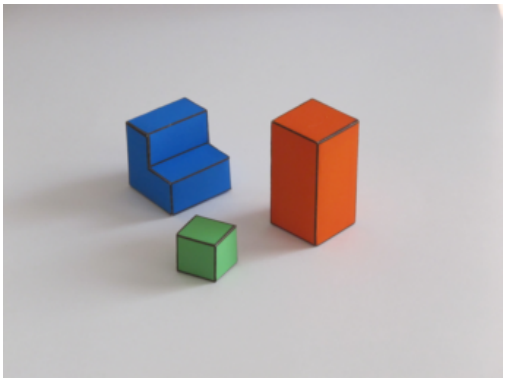

Figure 7: Image 4

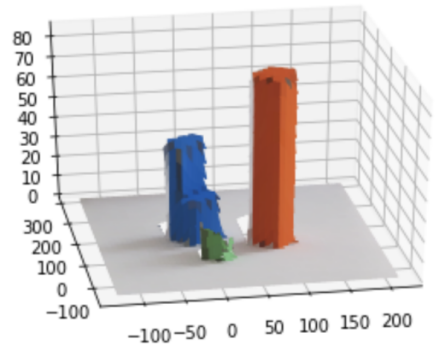

Figure 8: Image 4 3D Reconstruction

Problem 6

Image 3 fails the recovery of 3D information. Below you can find the initial image, as well as the 3D reconstruction:

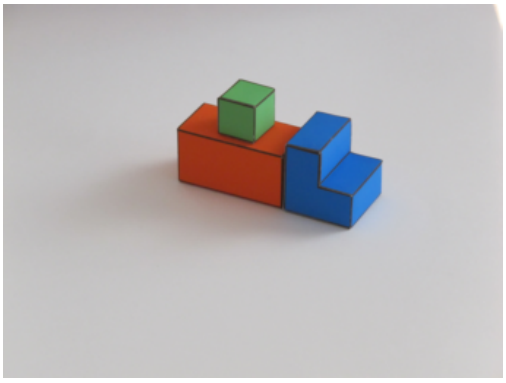

Figure 9: Image 3

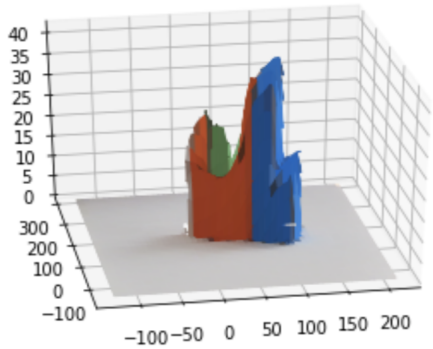

Figure 10: Image 3 3D Reconstruction

This case fails for a few reasons. First, the shadows and lighting present in the image make it such that the green cube's rightmost vertical edge is not detected as it does not make the magnitude cutoff. This results in a very incorrect reconstruction of the green cube. Second, this image violates the simple world

assumption that our objects do not have any touching edges. The edge between the orange and blue cube attempt to connect in the reconstruction because of this, which leads to the orange cube's tail extending to meet the blue block. This also happens with the green cube touching the orange cube, and this combined with the missing edge make the green cube reconstruct completely incorrectly.