Problem 1. A simple image formation model

The goal of this first exercise is to take images with different settings of a camera to create pictures with perspective projection and with orthographic projection. Both pictures should cover the same piece of the scene. You can take pictures of real places (e.g., the street, a living room, . . .) or you can also create your own simple world (e.g., you can print simpleWorld.pdf and create your own scenes. I recommend take a picture on plane background). To create pictures with orthographic projection you can do two things:

(1) use the zoom of the Digital camera

(2) crop the central part of a picture

You will have to play with the distance between the camera and the scene, and with the zoom (or amount of cropping) so that both images look as similar as possible only differing in the type of projection (similar to figure 1.4, in the lecture 1 notes). Submit the two pictures and label out clearly which parts of the images reveal their projection types.

1.1 Perspective projection

Figure 1.1 Perspective Projection

A – Vanishing point

B – Not Parallel

1.2 Orthographic Projection

Figure 1.2. Orthographic Projection

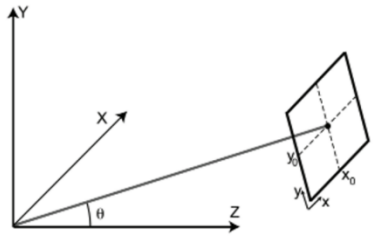
Camera can zoom in parallel and keeps the object of some size. Both the red lines on side says that object sides are parallel.

Problem 2. Orthographic projection

Prove the projection equations (eq. 1.2 and 1.3 in chapter_01_simplesystem.pdf) that relate the coordinates of one point in 3D world and the image coordinates of the projection of the point in the camera plane. You can use drawings or sketches if necessary.

$x = \alpha X + x0$ (1.2)

$y = \alpha\,(cos(\theta)Y - sin(\theta)Z) + y0$ (1.3)



There are three hypotheses in this simple vision system.

1. The camera center (x0, y0) is inside the plane X=0

2. The horizontal axis of camera (x) is parallel to the ground plane (Y=0)

3. The camera is tilted so that the line connecting the origin of the world coordinates system and the

image center is perpendicular to the image plane

- To prove equation 1.2, we know that x is parallel to the ground plane, therefore x-axis in the camera plane is parallel to the X-axis in the 3D world. We can make any shift on the X-axis and the shift will be carried over to the x-axis with a resolution factor α which is a constant, but without any change in magnitude. However, since (0, 0, 0) in the 3D world is mapped to (x0, y0), we will need to add x0 to any shift made on the X-axis to reflect the actual coordinates in the camera plane. Therefore, we have $x = \alpha X + x_0$.
- To prove equation 1.3, we need to be reminded that the Z-axis does not look like is pointing towards the observer. Instead, the Z-axis is identical to the Y-axis up to a sign change and a scaling. A point moving parallel to the Z-axis will be indistinguishable from a point moving down parallel to the Y-axis. Therefore, y is a linear combination of Y and Z. Using the same method as above, we first make a shift on Y and calculate the corresponding shift on the camera plane and then make a shift on Z and calculate the distance moved from (x0, y0). Using trigonometry, we can easily know that the shift on Y projecting onto the camera plane is equal to cos(θ)Y and the shift on Z projecting onto the camera plane is equal to sin(θ)Z. Taking the difference between Y and Z, multiplying by the resolution factor and add y0 because (0, 0, 0) is mapped to (x0, y0). Therefore, we have $y = \alpha (\cos(\theta)Y - \sin(\theta)Z) + y_0$

Problem 3. Constraints

In the Lecture slide, we have written all the derivative constraints for Y(x,y). Write the constraints for Z(x,y).

According to the projection equations, we have

$$x = \partial X + xo$$

$$y = \partial\,(cos(\theta)Y - sin(\theta)Z) + yo$$

Since we are interested in the constraints for Z(x, y), we are going to only focus on the second equation.

• We first solve for Z

$$y = \partial\, cos(\theta)Y - \partial\, sin(\theta)Z + yo$$

$$\partial\, sin(\theta)Z = \partial\, cos(\theta)Y - y + yo$$

If (x, y) belongs to a vertical edge.

• We then take derivative of Z with respect to y.

$$\partial Z \,/\, \partial Y \;=\; -1 \,/\, sin(\theta)$$

If (x, y) belongs to a horizontal edge.

• t is a vector parallel to the edge.

$$t \;=\; (-ny, nx)$$

• And then we take derivative of Z with respect to t.

$$\frac{\partial Z}{\partial t} = -ny\frac{\partial Z}{\partial x} + \; ny\frac{\partial Z}{\partial x} = \; -ny\frac{\partial Z}{\partial x} - nx\frac{1}{sin(\theta)}$$

If (x, y) does not belong to an edge

• Lastly, we take second derivatives to account for the cases where there are no edges

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

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

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

Problem 4. Approximation of derivatives

Fill the missing kernels (lines 51 and 65 in Build Constraints) in the script *SimpleWorld.ipynb*. Please include your answers also in the report.

1. The problem 4.1 (lines 51) is from Sobel operators.

$$\frac{\partial Y}{\partial t} = -ny^*\frac{\partial Y}{\partial x} + nx^*\frac{\partial Y}{\partial y}$$

2. The problem 4.2 (lines 65) is from Laplacian operators.

$$\frac{\partial^2 y}{\partial y^2}$$

```
    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 = np.sum(dmdx[i-1:i+2,j-1:j+2]*edges[i-1:i+2,j-1:j+2])/edgesum
    ny = np.sum(dmdy[i-1:i+2,j-1:j+2]*edges[i-1:i+2,j-1:j+2])/edgesum


    if contact_edges[i, j]:
      # dY/dy = 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 verticalsum > 0 and groundsum == 0:
      # dV/Dy = 1/cos a
      Aij[:,:,c] = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])/8;
      ii[c] = i
      jj[c] = j
      b[c] = 1/np.cos(alpha)
      c += 1 # increment constraint counter
    if horizontalsum > 0 and groundsum == 0 and verticalsum == 0:
      # dV/dt = 0
      Aij[:,:,c] = -ny*np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])/8 + nx*np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])/8;# Fill ou
t the kernel
      ii[c] = i
      jj[c] = j
      b[c] = 0
      c += 1 # increment constraint counter
    if groundsum == 0:
      # laplacian = 0
      # 0.1 is a weight to reduce the strength of this constraint
      Aij[:,:,c] = 0.1*np.array([[0, 0, 0], [-1, 2, -1], [0, 0, 0]]);
      ii[c] = i
      jj[c] = j
      b[c] = 0
      c += 1 # increment constraint counter

      Aij[:,:,c] = 0.1*np.array([[0, 0, 0], [-1, 2, -1], [0, 0, 0]]);# Fill out the kernel
      ii[c] = i;
      jj[c] = j;
      b[c] = 0;
      c = c+1; # increment constraint counter

      Aij[:,:,c] = 0.1*np.array([[0, -1, 1], [0, 1, -1], [0, 0, 0]]);
      ii[c] = i;
      jj[c] = j;
      b[c] = 0;
      c = c+1; # increment constraint counter
```

Problem 5. Explain the algorithm in your own words

In your own words, please explain the goal of the SimpleVision model. First describe the hypothesis of the problem. List all the steps needed to recover the world coordinates (X, Y, Z) from a single image of the blocks. You can find most of the information from Chapter01_simplesystem.pdf and the lecture notes. It will also be helpful for you to understand the Python notebook code you will be working on. Feel free to use equations and drawings. Please do not write a long paragraph. A few bullet points and equations will be sufficient.

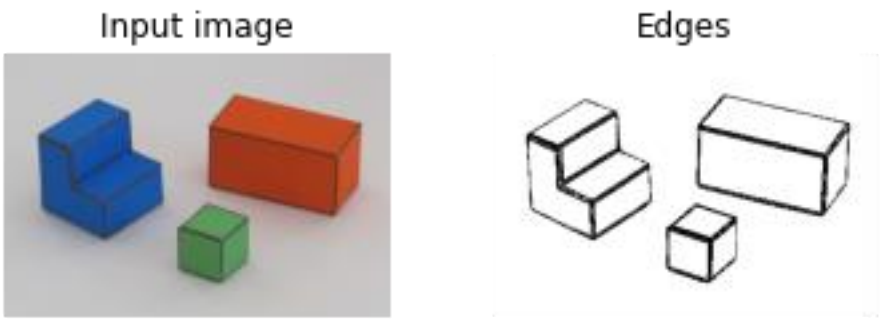
- Once image loaded in the algorithm, it detects the images vertical edges and horizontal edges separately and adds them in single image. Occlusion edges helps to detect the horizontal edges and contact edges helps to detect vertical edges.
- Making the constraints, which all touches the edges on the ground and allocates the histogram on a matrix array so they can be calculated.
- SparseMatrix function helps to find the size of the 2D image by reconstructing the scenario.
- Finally, rendering the image and viewing the image from a certain angle at a certain point of angle.

Problem 6. Run the code.

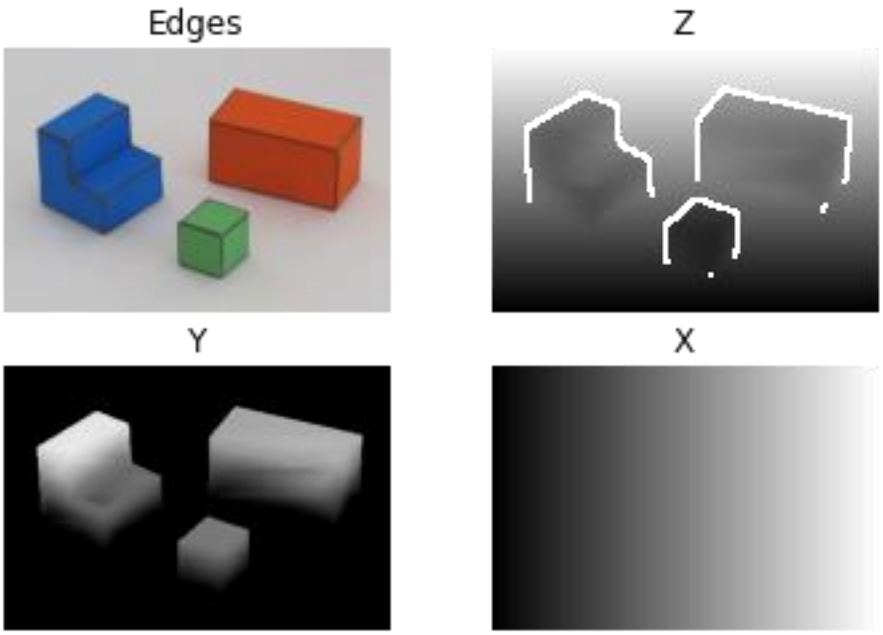
Select some of the images included with the code and show some new viewpoints on them. You can also try with new images taken by you if you decide to create your own simple world.

6.1 Choose “img1.png” for the example

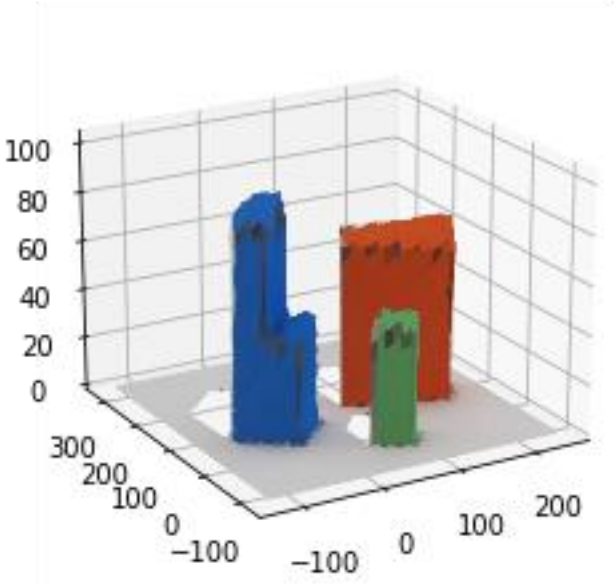
Classify in to 4 categories:
- 3D orientation: vertical
- 3D orientation: horizontal
- Occlusion edges
- Contact edges



6.2 Render Image



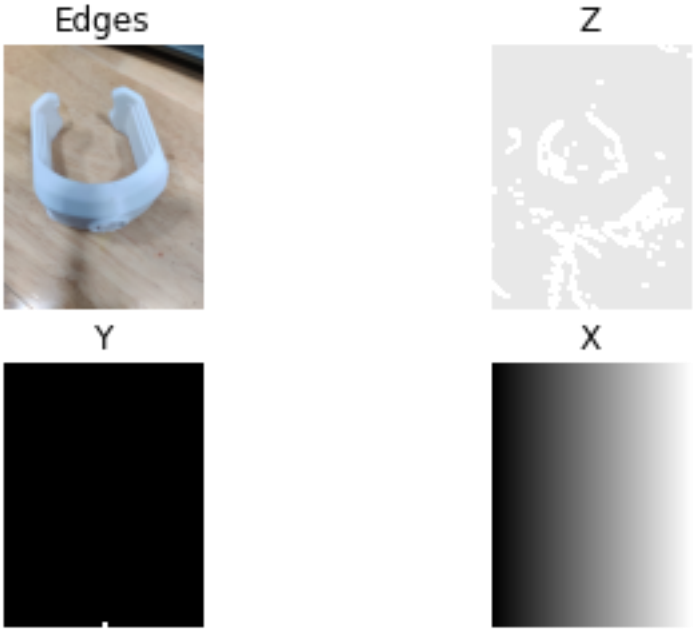
6.3 3D reconstruction



Problem 7. Violating simple world assumptions.

From the real-life image, when the

recovery of 3D information fails. Include the image and the reconstruction in your writeup, and explain why it fails.
I found image is failed to reconstruct a 3D object. According to Generic view assumption, the view that the observer sees from different angles might not be the same as that in real world. That is, the observer might see the same 2D image, but it does not mean the observer see the same object in 3D world.
In image, the white part in the algorithm cannot precisely distinguish horizontal and vertical edges on objects. Thus, the output of simple world would be failed.



Problem 8. The real world.

A research problem is a question for which we do not know the answer. In fact, there might not even be an answer. This question is optional and could be extended into a larger course project.
The goal of this problem is to test the 3D reconstruction code with real images. Several the assumptions we have made will not work when the input are real images of more complex scene. For instance, the simple strategy of differentiating between foreground and background segmentation will not work with other scenes. Try taking pictures of real-world scenes (not the cubes) and propose modifications to the scheme proposed in this lecture so that you can get some better 3D reconstructions. The goal is not to build a general system, but to be able to handle a few more situations.

- It's hard to recover 3D objects because compared to cube images, the real-world image has more complicated edges and colours. The image file called *problem7 test.jpeg* that I took in real-world scenes. Based on the screenshot below, after running the 3D reconstruction model, it cannot produce any output, which means that the XYZ edges of this image have not been successfully detected by this model. Our solution is to rework the XYZ surface constraints, solve equations that deviate from non-linear images to 3D space.

Input image

Edges

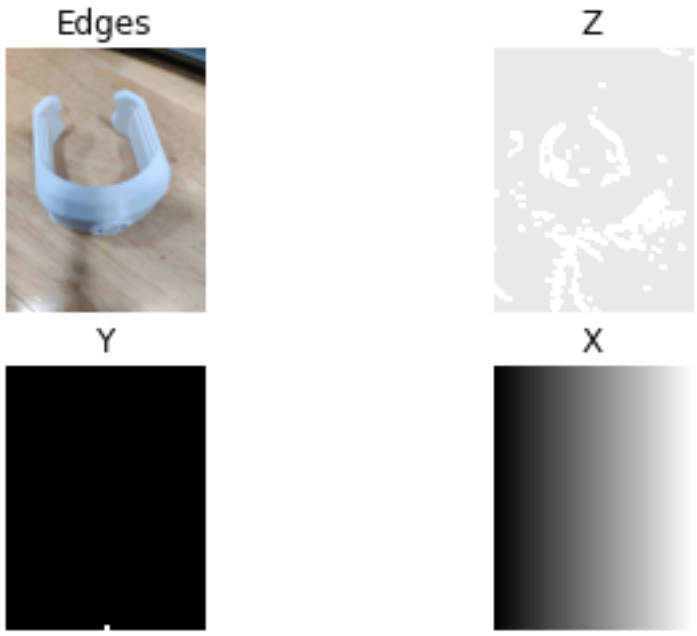
Normals

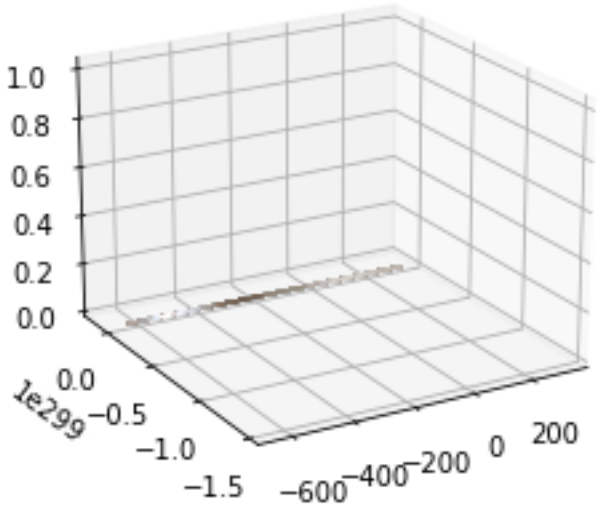
Input image

Edges

Occlusion boundaries

Contact boundaries


Edges
Z
Y
X


1.0
0.8
0.6
0.4
0.2
0.0
0.0
−0.5
−1.0
−1.5
1e299
−600
−400
−200
0
200