MIT CSAIL

6.819/6.869 Advances in Computer Vision

Spring 2021

# Problem Set 1

Jiahui Tang
jiahuita@mit.edu

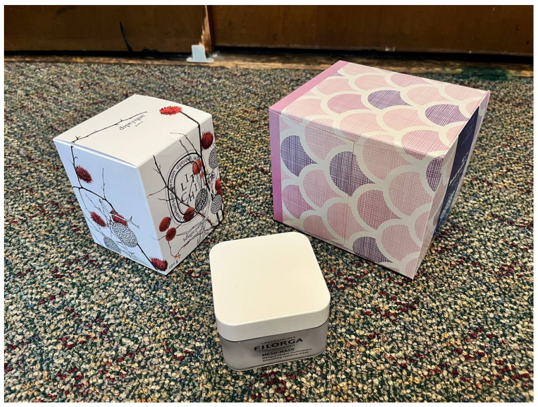
**Problem 1** *Perspective and orthographic projections* (1 point)

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 or objects (e.g. your furniture).

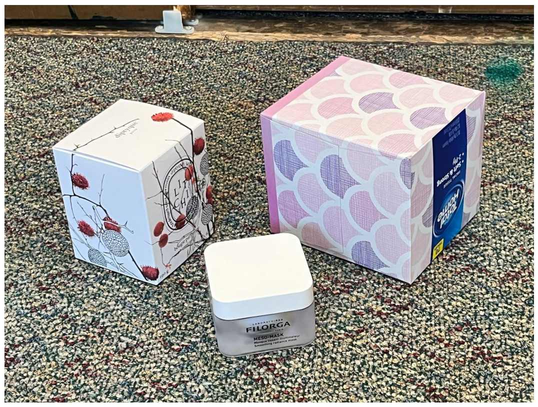
To create pictures with orthographic projection you can do two things: 1) use the zoom of the camera, 2) crop the central part of a picture. You will have to play with the distance between the camera and the scene, and with the zoom or cropping so that both images look as similar as possible, only differing in the type of projection (similar to figure 2.2 in the lecture 1 notes - section 1.2 of the book i.e. `simple_vision_system.pdf`).

Submit the two pictures and clearly label parts of the images that reveal their projection types.

**Answer**:



((a)) Perspective Projection



((b)) Orthographic Projection (taken far away using camera zoom)

Figure 1: Prespective and Orthographic projections

**Problem 2** *Orthographic projection equations* (1 point)

Recall the parallel projection equations stated in class:

$$x = \alpha X + x_0 \tag{1}$$
$$y = \alpha(\cos(\theta)Y - \sin(\theta)Z) + y_0 \tag{2}$$

which relate the coordinates of a point in the 3D world to the image coordinates of an orthogonal camera rotated by $\theta$ over the $X$-axis.

Show that the equations emerge naturally from a series of transformations applied to the 3D world coordinates $(X, Y, Z)$, of the form:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \alpha \cdot P \cdot R_x(\theta) \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{3}$$

Where $R_x(\theta)$ is a $3 \times 3$ matrix corresponding to a rotation over the $X$ axis, $P$ is a $2 \times 3$ matrix corresponding to the orthogonal projection and $\alpha$ is a scaling factor to account for the size of the camera sensor, which is a single scalar when the pixels are square (assumed in this case).

Then, find $\alpha$, $x_0$ and $y_0$ when the world point $(0, 0, 0)$ projects onto $(0, 0)$ (which corresponds to the center of the image) and the point $(1, 0, 0)$ projects onto $(3, 0)$.

**Answer:** In the series of transformations applied to the 3D world coordinates $(X, Y, Z)$, of the form:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \alpha \cdot P \cdot R_x(\theta) \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{4}$$

We know that

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{5}$$

and since it is an orthogonal camera rotated by $\theta$ over the X-axis, the rotation matrix $R_x(\theta)$ is

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) \\ 0 & sin(\alpha) & cos(\alpha) \end{bmatrix} \tag{6}$$

With the scaling factor $\alpha$ and adjust for translation of center of the coordination between 3D world coordinates and image coordinates with

$$\begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{7}$$

,

We could verified that the parallel projection equations are

$$x = \alpha X + x_0 \tag{8}$$
$$y = \alpha(\cos(\theta)Y - \sin(\theta)Z) + y_0 \tag{9}$$

Plugging in $[X, Y, Z] = (0, 0, 0), [x, y] = (0, 0)$, as well as $[X, Y, Z] = (1, 0, 0), [x, y] = (3, 0)$, we could obtain that:

$$x_0 = 0 \tag{10}$$
$$y_0 = 0 \tag{11}$$
$$\alpha = 3 \tag{12}$$

**Problem 3** *Edge and surface constraints* (1 point)

In the lecture 1 slides, we have written down the constraints for $Y(x, y)$. Briefly derive the constraints for $Z(x, y)$ along vertical edges, horizontal edges, and flat surfaces.

**Answer:**

Constraint for $Z(x, y)$ along vertical edges: Z should be constant along vertical edge, thus we have

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

Constraint for $Z(x, y)$ along horizontal edges: Y is horizontal along the horizontal edge, thus we have

$$\frac{\partial Y}{\partial t} = -n_y\frac{\partial Y}{\partial x} + n_x\frac{\partial Y}{\partial y} = 0$$

from formula $y = \alpha(\cos(\theta)Y - \sin(\theta)Z) + y_0$, we differentiate by t,

$$n_x = \alpha(-sin(\theta))\frac{\partial Z}{\partial t}$$

rearrange, we could get

$$\frac{\partial Z}{\partial t} = -\frac{n_x}{\alpha sin(\theta)}$$

Constraint on flat surface will be similar with second derivative being zero, when we differentiate formula (9) w.r.t to different variable, knowing their second derivative of $Y$ are all zero on the surface.

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

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

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

Thus, all constraints will be

$$\boxed{\begin{aligned} \frac{\partial Z}{\partial y} &= 0 \\ \frac{\partial Z}{\partial t} &= -\frac{n_x}{\alpha sin(\theta)} \\ \frac{\partial^2 Z}{\partial x^2} &= 0 \\ \frac{\partial^2 Z}{\partial y^2} &= 0 \\ \frac{\partial^2 Z}{\partial xy} &= 0 \end{aligned}}$$

**Problem 4** *Complete the code* (2 points)

Fill in the missing lines in the notebook: `Pset1.ipynb`, and include them in the report as screenshots. First, find a way to classify edges as vertical or horizontal edges. Next, fill in the rest of the conditions of the constraint matrix. The constraints for when the pixel is on the ground have already been done for you as an example. Put the kernel in `Aij` and the value you expect in `b` (the conversion to a linear system is done for you later so you don't need to worry about that part).

You only need to modify the locations marked with a `TODO` comment.

Please make sure to also include your answers for `vertical_edges`, `horizontal_edges`, and your formulations for `Aij` and `b` for the different constraints in the report.

**Answer:**

```
# TODO: edge orientation (average of edge pixels in current neighborhood)
nx = np.sum(edges[i-1:i+2,j-1:j+2] * dmdx[i-1:i+2,j-1:j+2] )/edgesum
ny = np.sum(edges[i-1:i+2,j-1:j+2] * dmdy[i-1:i+2,j-1:j+2] )/edgesum

# Populate Aij, ii, jj, b, and c using alpha, theta, and
# the constraint/transform matrices you derived in the written segment

###############################################################
### COPY YOUR CODE BELOW UNTIL THE LOOP INTO YOUR REPORT ###
###############################################################
```

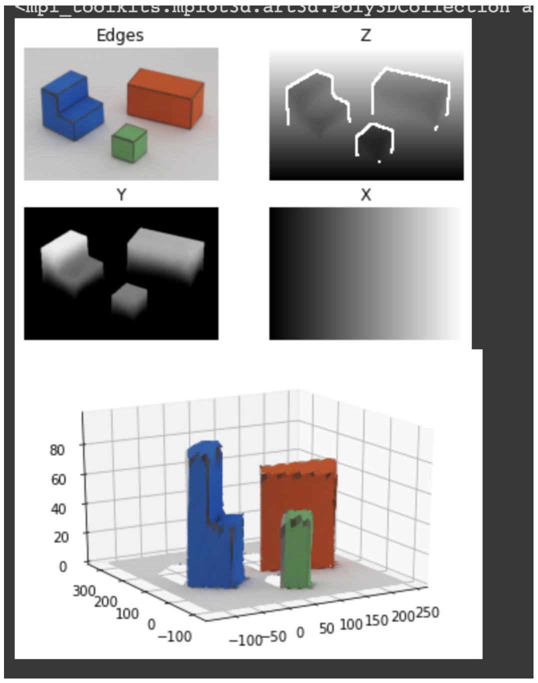
```
# Contact edge: dY/dy = 0,  Y(x,y) = 0
# Requires: a transform matrix
if contact_edges[i, j]:
  Aij[:,:,c] = np.array([[0, 0, 0], [0, 1, 0], [0, 0, 0]])
  b[c]       = 0
  update_indices()

# Vertical edge: dY/dy = 1/cos(alpha)
# 1/8: deriving the Sobel operator using finite differences.
# Requires: a transform matrix, alpha
if verticalsum > 0 and groundsum == 0:
  Aij[:,:,c] = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])/8;
  b[c]       = 1/np.cos(alpha) # alpha is theta
  update_indices()

# dY/dt = 0 (you'll have to express t using other variables)
# Requires: a transform matrix, i, j, theta
# dY/dt = -ny * dY/dx + nx* dY/dy
if horizontalsum > 0 and groundsum == 0 and verticalsum == 0:
  dYdx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])/8
  dYdy = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])/8
  Aij[:,:,c] = - ny * dYdx + nx * dYdy
  b[c]       = 0
  update_indices()

# laplacian = 0 (weighted by 0.1 to reduce constraint strength)
# Requires: multiple transform matrices
if groundsum == 0:
  # d^2 Y / dx^2
  Aij[:,:,c] = 0.1* np.array([[0, 0, 0], [-1, 2, -1], [0, 0, 0]]);
  b[c]       = 0
  update_indices()

  # d^2 Y / dy^2
  # = 2Y(x,y) - Y(x, y+1) - Y(x, y -1)?
  Aij[:,:,c] = 0.1 * np.array([[0, -1, 0], [0, 2, 0], [0, -1, 0]]);
  b[c]       = 0
  update_indices()

  # d^2 Y/dx dy
  Aij[:,:,c] = 0.1 * np.array([[0, -1, 1], [0, 1, -1], [0, 0, 0]]);
  b[c]       = 0
  update_indices()
```

**Problem 5** *Run the code* (1 point)

Select some of the images included with the code and show some new viewpoints for them.

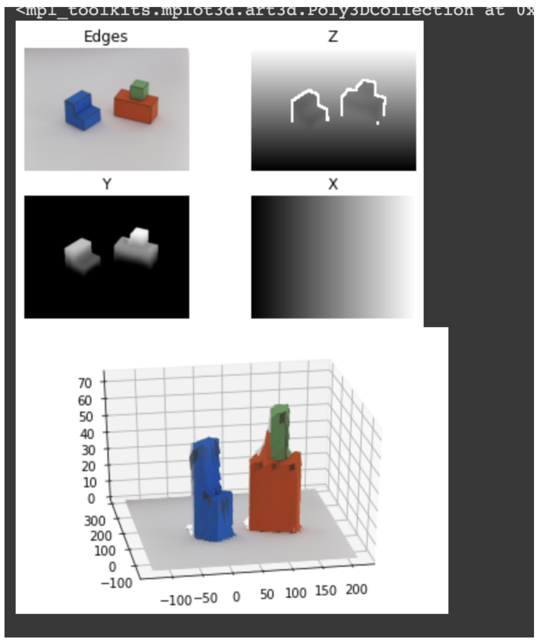
Optional: You can also try with new images taken by you if you decide to create your own simple world.

**Answer**:



```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection a
```

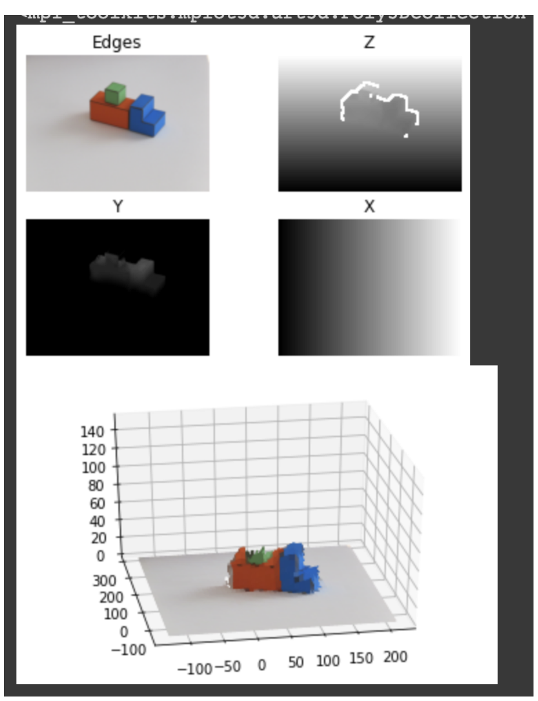
((a)) Image 1, from new viewpoints



```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x
```

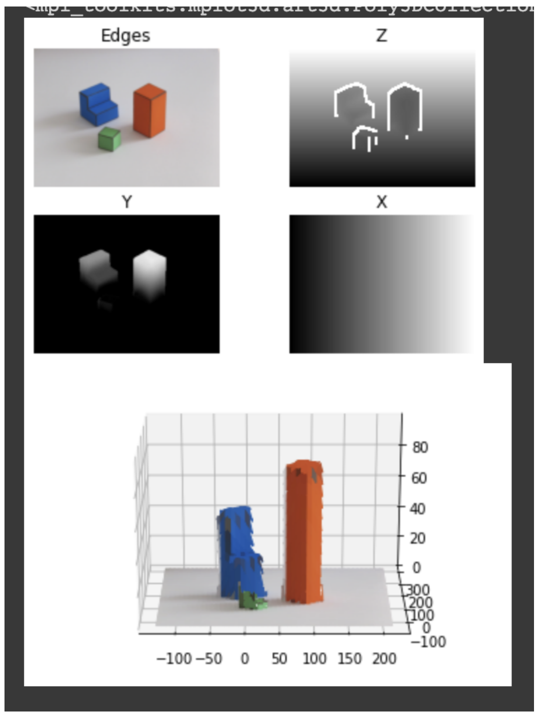
((b)) Image 2, from new viewpoints

Figure 2: new images from new viewpoints



```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection
```

((a)) Image 3, from new viewpoints



```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection
```

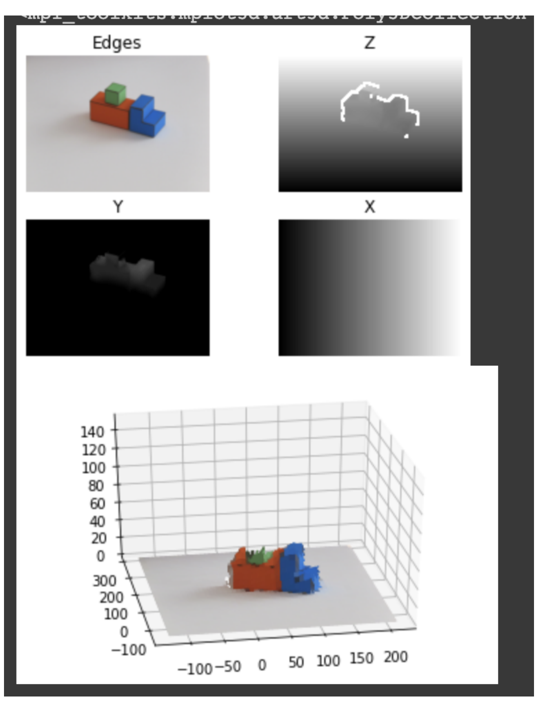
((b)) Image 4, from new viewpoints

Figure 3: new images from new viewpoints

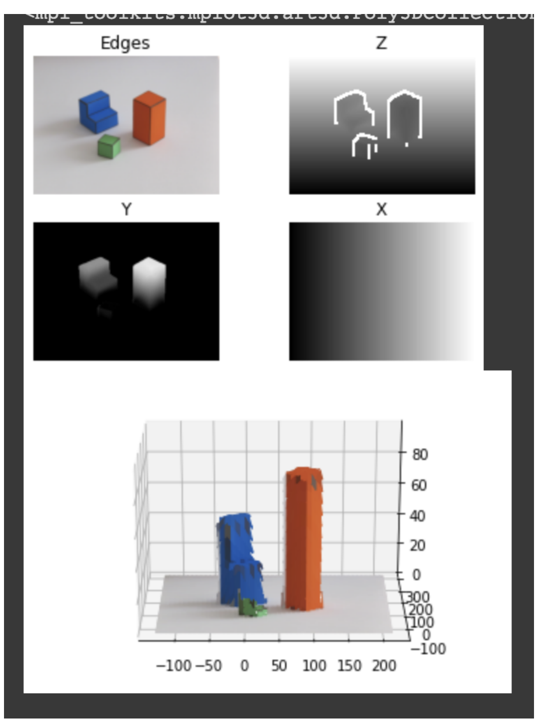
**Problem 6** *Violating simple world assumptions* (1 point)

Find one example from the four images provided with the problem set (`img1.jpg`, ..., `img4.jpg`) when the recovery of 3D information fails. Include the image and the reconstruction in your writeup, and explain why it fails.

**Answer:**



((a)) Image 3, from new viewpoints



((b)) Image 4, from new viewpoints

Figure 4: failed reconstruction

The recovery of 3D image 3 and image 4 from above example seem to be fail. The green box is broken and appear quite flat. There are also distortion of blue and orange box, especially when we change view points, the model is incapable of recover from angles that are unseen, and thus unable to infer the 3D structure there (From image 2, new view point with the orange box). This may make sense as the model do not hold general knowledge of the object in the world.

The model seems also not able to detect the edge of all box properly, especially the green box, it makes the green box disappear in the Y coordinate gradient (e.g. image 3 and 4, green box). This could be due to light, shadow, contact edge not bold enough, resulting the model unable to separate the object edge from the background. Maybe one side of the green cube is too bright making the gradient between box and background too low, thus it is considered as a flat part of the ground.