SI100B, Spring 2018
Homework 3: Draw 3D Line Segments

Deadline: April 30, 2018

April 10, 2018

1 Introduction

In this assignment, you are required to draw 3D lines on an image, and this document helps you go through necessary details in order to help you finish this task. In general, there are several major steps to do this. First, you need to know what your inputs are. In order to draw 3D line segments, you need to specify each 3D line segment with two end points $\mathbf{P}_1$ and $\mathbf{P}_2$. You can specify multiple pairs of them $\mathbf{P}_1^{(i)}$ and $\mathbf{P}_2^{(i)}$ in order to draw multiple 3D line segments, where (i) indicates each pair for one particular 3D line segment. Then, you also need to specify where your virtual camera is, where it looks at, how large its imaging plane is, and where the near and far clipping planes locate, all with respect to the world coordinate system. The simplest case is that you align the camera coordinate system with the world coordinate system, and the camera looks at the negative z direction. Figure. 1 gives an illustration of such a setup.

To draw 3D line segments, we first need to construct a projection matrix (fixed once the camera is fixed in space), which will be used to project 3D points in space onto the camera imaging plane (2D). The projection is done using homogeneous transformation. During the projection, both 3D and 2D clippings are performed to remove lines or part of the line segments that are outside the viewing range. Once the projection and clipping are finished, we are left with only 2D points on the imaging plane and within the viewing range. Since the viewing plane is discretized into pixels within the viewing range, we then rasterize the 2D line segments and give particular colors for the pixels in order to give the final resulted image for display. You can specify multiple 3D line segments to represent complex scenes.

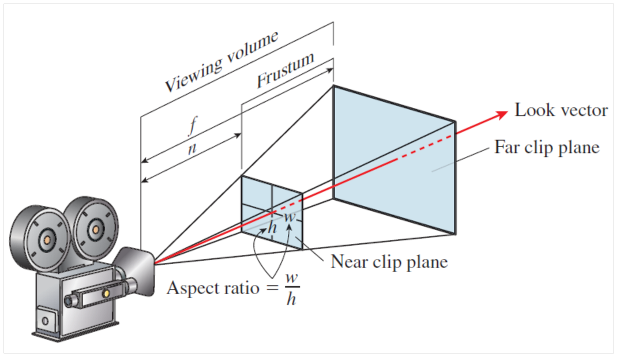

Figure 1: An illustration of camera and 3D scene setup.

2 Implementation Details

To help you get started with your assignment, in this section, we describe every step in more detail so that you can follow to implement the related algorithm and get your final result.

2.1 Camera and 3D scene setup

To set up your own 3D scene for drawing 3D line segments on an image, you need to define your world coordinate system, which is a Cartesian coordinate system. As stated before, for simplicity, you can set your camera coordinate system to be aligned with the world coordinate system (they are exactly the same), otherwise, you will need to construct a viewing matrix that transforms points in world coordinate system into the camera coordinate system. Once you have done this, we need to define near and far planes, and only points or line segments between the near and far planes will be displayed. The near and far planes are both in negative z direction and parallel to the x-y plane of the camera coordinate system. The distances of the near and far planes to the x-y plane are n and f respectively, as shown in Fig. 1. We also need to specify the camera size with respect to the world coordinate system, where w and h represent the width and height of the camera, respectively, which defines the viewing range. Any points or line segments outside this range will be removed. The process of removing points or line segments or part of the line segments outside the viewing range, both in 3D and 2D, is called view clipping.

After you setup your camera, you also need to specify your 3D scene. In this assignment, the 3D scene is simple, which only consists of multiple 3D line segments. Thus, you only need to specify a set of pairs of the end points $\mathbf{P}_1^{(i)}$ and $\mathbf{P}_2^{(i)}$ for the line segments so that the later algorithm will help you draw these line segments onto an image.

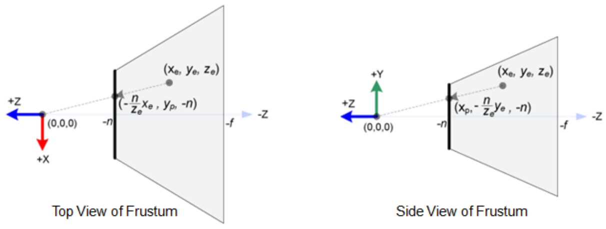

Figure 2: An illustration of 3D projection.

2.2 Constructing projection matrix

Once you have set up your camera and 3D scene, you need to project the 3D points onto the imaging plane. For simplicity, the near plane is selected to be the imaging plane. Figure 2 shows such a case where only side views are shown for clarity. It is clear that given any 3D point in camera coordinate space (x_e, y_e, z_e), the projected point is $(-nx_e/z_e, -ny_e/z_e, -n)$. Ignoring the z-component of the projected point since they are always the same, we get the projected 2D point coordinate $(-nx_e/z_e, -ny_e/z_e)$. Such a projection can be represented as a homogeneous projection matrix $\mathbf{P}$:

$$\mathbf{P} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix},$$

where we can verify that given a 3D point in homogeneous coordinates $(x_e, y_e, z_e, 1)$, multiplying the projection matrix in front of this point gives $(nx_e, ny_e, nz_e, -z_e)$. This equals the homogeneous representation $(-nx_e/z_e, -ny_e/z_e, -n, 1)$, which is exactly the projection result we obtained previously.

2.3 Clipping

As we stated in the introduction, two kinds of clippings should be performed before and after the projection. Before the projection, 3D clipping should be performed to remove points and line segments that are not between the near and far clipping planes(described in 2.3.1). After the projection, 2D clipping should be performed to remove those that are outside the viewing range(described in 2.3.2). We can also use the methods discussed in the discussion class(described in 2.3.3).

2.3.1 3D clipping

To perform 3D clipping, we first identify the z-component of each 3D point. We denote the z-component of the near and far planes by z_{near} and z_{far} which should be negative since they all locate in negative z-axis, see Figures 1 & 2. Given any point (x,y,z), if $z > z_{near}$ or $z < z_{far}$, that point is located outside the 3D clipping plane and should be discarded. Given any two points $\mathbf{P}_1$ and $\mathbf{P}_2$ which form a line segment, if $\mathbf{P}_1$ and $\mathbf{P}_2$ are both outside the near and far clipping planes, the whole line segment should be removed and not considered for drawing anymore. If $\mathbf{P}_1$ and $\mathbf{P}_2$ are both between the near and far clipping planes, the whole line segment must be kept unchanged. The complicated case is when one of the two points is outside the near and far clipping planes and the other is between them. In this case, we postpone the clipping as a 2D clipping problem later.

2.3.2 2D clipping

After the 3D clipping, we project all the end points of the line segment using the homogeneous projection matrix and obtain the 3D projected point, and we denote such a point as $\mathbf{p} = (-nx_e/z_e, -ny_e/z)$. The two end points on a 3D line segment will be projected to form two 2D end points $\mathbf{p}_1$ and $\mathbf{p}_2$ on the imaging plane and then a 2D line segment. This 2D line segment will need further clipping before finally displaying it on an image.

To perform the 2D clipping, we use Cohen-Sutherland line clipping algorithm, which can be achieved by first subdividing the whole 2D space into 9 regions, and encoding them with binary codes as shown in Figure 3. For any 2D point, we can identify which region it belongs to and thus obtain a binary code for that point. For two end points $\mathbf{p}_1$ and $\mathbf{p}_2$ of the 2D line segment, we can obtain two codes, c_1 and c_2, respectively. We define $'\&'$ and $'|'$ as the binary "or" and "and" operators respectively. Cohen-Sutherland line clipping algorithm determines whether a line is inside the viewing range as follows: if $c_1|c_2 = 0$, the whole line segment is within the viewing range, and no clipping is needed; if $c_1|c_2 \neq 0$, we further check $c_1\&c_2$; if $c_1\&c_2 \neq 0$, the whole line segment is completely outside the viewing range and should be discarded entirely; otherwise, the line segment is partially within the viewing range and should be further clipped.

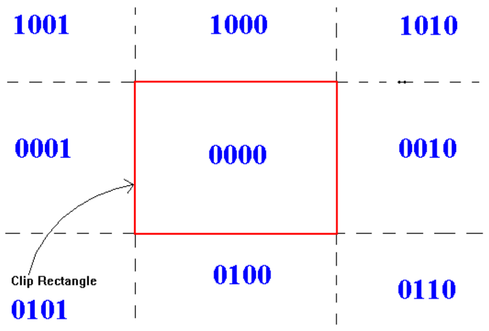

Figure 3: Cohen-Sutherland 2D line clipping algorithm: encoding

To perform such a clipping, we use the implicit form of the 2D line segment, which is written as:

$$\Delta y x - \Delta x y + (y_1 \Delta x - x_1 \Delta y) = 0, \qquad (1)$$

where $\Delta x = x_2 - x_1$ and $\Delta y = y_2 - y_1$. Note that the line segment will intersect with the boundary of the viewing range and we need to calculate the intersection point with the boundary line. By checking the code, we can identify the boundary line, which can be represented by either a fixed x-value or a fixed y-value. Figure 4 shows such a case where the top boundary line is intersected with the projected line segment and we need to determine the intersection point. Since such a boundary line can be represented as $y = h/2$, we insert such value into the implicit line equation and

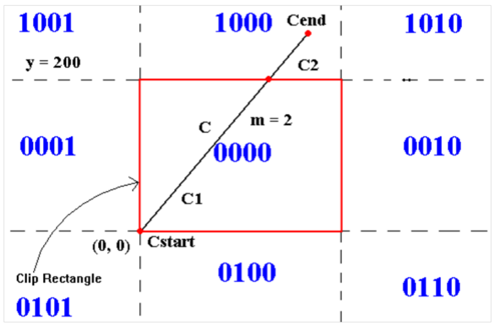

Figure 4: Cohen-Sutherland 2D line clipping algorithm: finding the intersection point.

obtain:

$$x = \frac{(x_2 - x_1)h/2 + (x_2 - x_1)y_1 - (y_2 - y_1)x_1}{y_2 - y_1}. \tag{2}$$

This defines a new end point for the 2D line segment. After the 2D clipping, all the line segments are constrained within the viewing range. Thus, we can draw them with a set of pixels into an image for display.

2.3.3 Clipping Using Perspective Matrix

Figure 5 shows that in perspective projection, a 3D point in a truncated pyramid frustum (eye coordinates) is mapped to a cube (NDC); the range of x-coordinate from $[l,r]$ to $[-1,1]$, the y-coordinate from $[b,t]$ to $[-1,1]$ and the z-coordinate from $[n,f]$ to $[-1,1]$.

We can first map the points in the frustum to the points in the cube and then clip them. The perspective matrix mapping the points is:

$$\mathbf{PM} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix},$$

Then, we map the clipped points in the cube back into the frustum and get the projected 2D point coordinate of the points. By now, all the line segments are constrained within the viewing range. Thus, we can draw them with a set of pixels into an image for display.

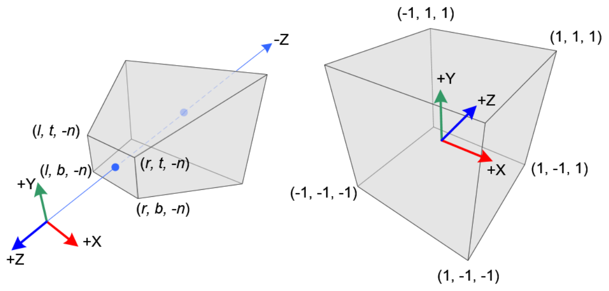

Figure 5: Perspective Frustum and Normalized Device Coordinates (NDC).

2.4 Rasterization

To convert continuous representation of 2D line segments into pixels, we perform rasterization. One of the commonly used rasterization algorithm is the Bresenham's line rasterization algorithm based on the mid-point rule. The other method is DDA algorithm which mainly takes unit steps along one coordinate and compute the corresponding values along the other coordinate.

2.4.1 Mid-point rule

To do this, we first need to represent the 2D line segment using implicit formulation as in Eq. 1, where $x \in [x_1, x_2]$ and $y \in [y_1, y_2]$, and we define a function as:

$$f(x,y) = \Delta y x - \Delta x y + (y_1 \Delta x - x_1 \Delta y), \tag{3}$$

which can be an indicator whether a point is below the line ($f(x,y) < 0$), on the line ($f(x,y) = 0$), or above the line ($f(x,y) > 0$).

More specifically, at any point (x_i, y_i), we want to determine the next point, which is either (x_{i+1}, y_i) or (x_{i+1}, y_{i+1}) (suppose the tangent is positive; otherwise the next point should be either (x_{i+1}, y_i) or (x_{i+1}, y_{i-1})), see Figure 6 for an example. To determine which point is the next point, we examine the function value

$f(x_{i+1}, y_{i+1/2})$. If $f(x_{i+1}, y_{i+1/2}) > 0$, the line is close to (x_{i+1}, y_i) and we select (x_{i+1}, y_i) as the next point; otherwise we select (x_{i+1}, y_{i+1}) as the next point (the decision rule can be similar for negative tangent). The above rule can be applied independently to every line segment given two end points.

2.4.2 Digital Differential Analyzer (DDA)

DDA algorithm is an incremental scan conversion method. Here we perform calculations at each step using the results from the preceding step. The characteristic of the DDA algorithm is to take unit steps along one coordinate and compute the corresponding values along the other coordinate. The unit steps are always along the coordinate of greatest change, e.g. if $dx = 10$ and $dy = 5$, then we would take unit steps along x and compute the steps along y.

- Step1: Get the input of two end points (X_0, Y_0) and (X_1, Y_1).
- Step2: Calculate the difference between two end points.

```
dx = X1 - X0
dy = Y1 - Y0
```

- Step 3: Based on the calculated difference in step-2, you need to identify the number of steps to put pixel. If dx > dy, then you need more steps in x coordinate; otherwise in y coordinate.

```
if (absolute(dx) > absolute(
    dy)):
Steps = absolute(dx)
else:
Steps = absolute(dy)
```

- Step 4:Calculate the increment in x coordinate and y coordinate.

```
Xincrement = dx / (float)
    steps
Yincrement = dy / (float)
    steps
```

- Step 5:: Put the pixel by successfully incrementing x and y coordinates accordingly and complete the drawing of the line.

```
for v in range(Steps):
x = x + Xincrement
y = y + Yincrement
putpixel(Round(x), Round(y))
```

2.4.3 Bresenham's Line Generation

The Bresenham algorithm is another incremental scan conversion algorithm. The big advantage of this algorithm is that, it uses only integer calculations. Moving across the x axis in unit intervals and at each step choose between two different y coordinates. You can use this method to optimize your program and make your program become faster.

2.4.4 Line color

Once you determine the pixels of the line segment, you will need to set a color (an RGB value) to the pixel so that it can be stored in an image and such an image can be opened by some software tools for display. The simplest method for giving the line color is to set every pixel on a line segment with a constant color, for example, you can set a pixel with red color by giving R=255, G=0 and B=0.

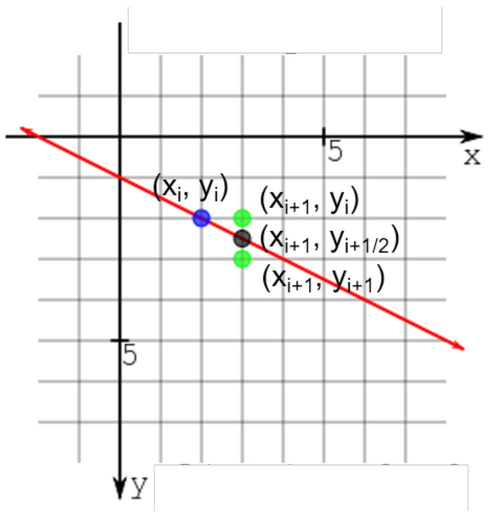

Figure 6: The 2D line rasterization based on mid-point rule.

3 Submission guideline

3.1 Assignment requirement

We will not require you to implement all of the above described algorithms and codes from scratch. We will provide you with a skeleton code where we leave out some portion for you to fill in. The tasks that you need to finish are:

- Projection: you need to create your own projection matrix and project the two end points of each line segment onto the imaging plane.
- Clipping: Both 3D and 2D clippings are required for you to implement independently.
- Rasterization: you will also need to implement the rasterization algorithm based on the mid-point rule.

After the implementation, we have provided you with some test data which include a set of line segments with each line segment a constant color defined. Your program should at least pass these test cases.

3.2 On submission

You should submit your code and a report describing what methods you used for each step. Your should include your result (a image) in the report too. Please send your homework with Email Subject ”SI100B HW3 Name” to SI100B_03@163.com.

Due: 23:59:59, April 30, 2018