CSE 410: Assignment 3, 2018

Submission deadline:

Hidden Surface Removal within the Bounding Box

Problem Definition:

1. In this assignment you have to implement Hidden Surface Removal algorithms for the objects within a bounding box.

2. For simplicity we will consider only Triangles.

3. Remember the output `stage3.txt` of your `assignment2`? You have to use that as input of your program. (But during implementation always test with smaller cases)

4. Another input of your program will be from, config.txt
    a. The general format of this file is

    ```
     500 500
    -1.00
    -1.00
     0.00 2.00
    ```

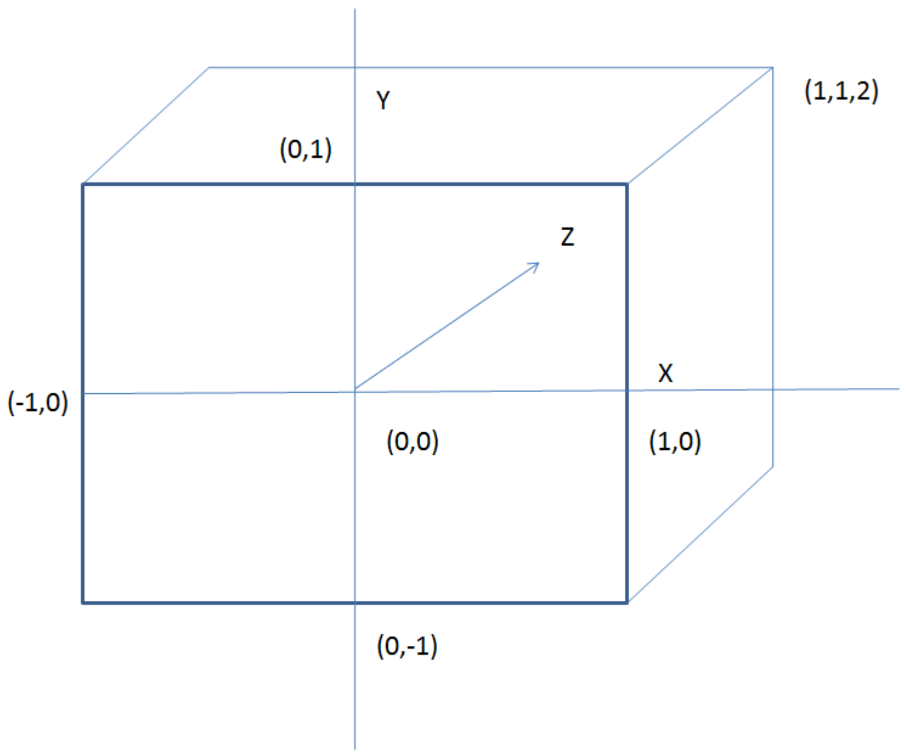
    b. First Line of file represents [Screen_Width X Screen_Height]

    c. Second line specify the left limit of X. [x_right_limit=-x_left_limit]

    d. Third line specify the bottom limits of Y. [y_top_limit=-y_bottom_limit]

    e. Fourth line specify the front and rear limits of Z

5. **Now check the figure below for the above configuration**: imagine all your triangles resides in X, Y, Z space and you only visible volume is bounded by



i) -1<=X<=1, -1<=Y<=1, 0<=Z<=2

ii) Everything outside this volume have to be clipped away and will be out of visibility.

iii) Now imagine yourself as a parallel-viewer from XY plane.

iv) You task is to generate the image that can be seen with respect to the XY plane within this bounding volume according to the depth information of triangles.

v) Also you need to print z_buffer value into a file named **z_buffer.txt. (only those values where z_buffer[row][col]<z_max)**

5. Your must take `stage3.txt` as your input file. As you are already familiar, the input file will contain each triangle information as three lines specifying the coordinates of the three points of the triangle. There will be no invalid cases so rest assured.

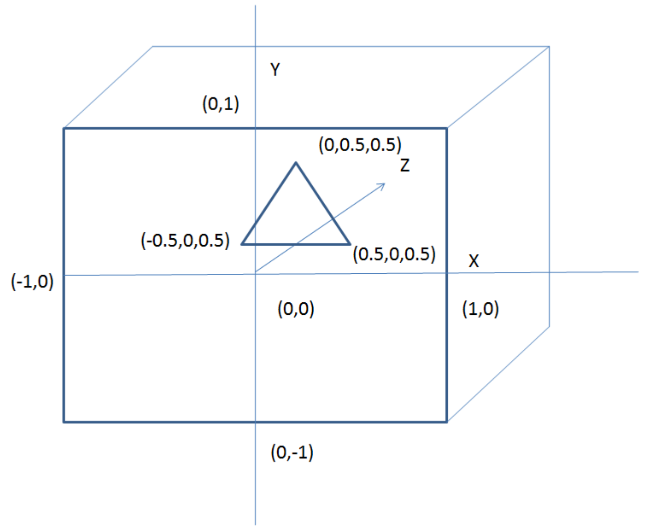
Your output will be an image defined by [Screen_Height X Screen_Width] as you view from parallel to XY plane.

For example:

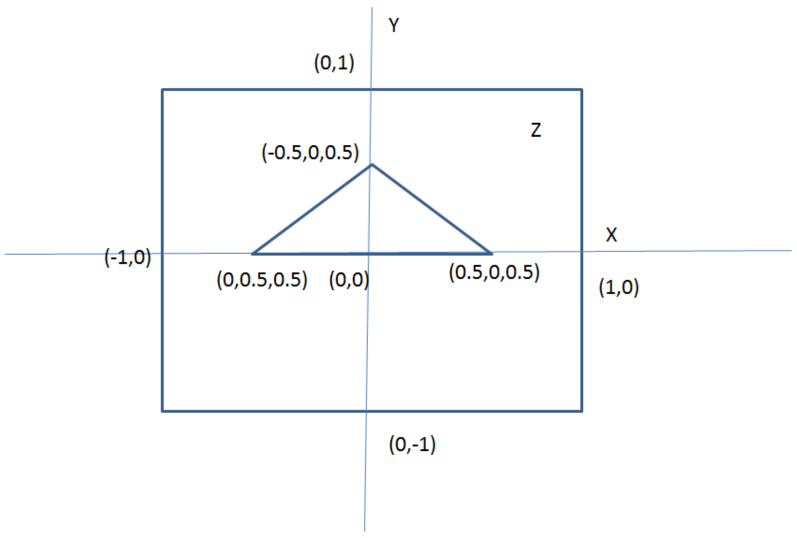
Suppose the `stage3.txt` contains only

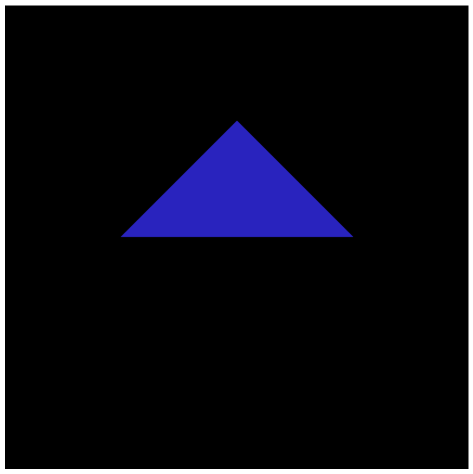
```
0.50  0.00  0.50
-0.50 0.00 0.50
0.00 0.50 0.50
```

That means there is only one triangle. Now imagine this within the viewing volume the triangle position can be shown by 1st figure. The actual output viewing figure from viewing plane is shown by 2nd figure.



But you have to draw this within the window defined by (Screen_Height, Screen_Width) so the output will be somewhat like,





Here during drawing of each triangle we will set its' color randomly.

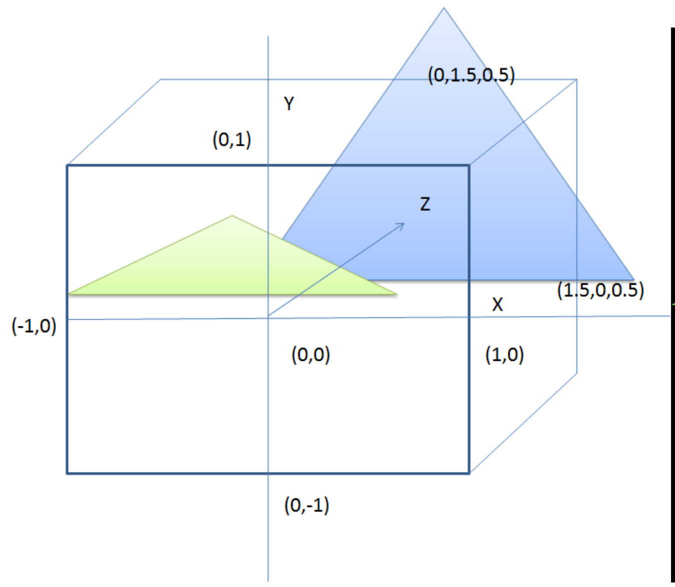
Another example, `stage3.txt` contains,

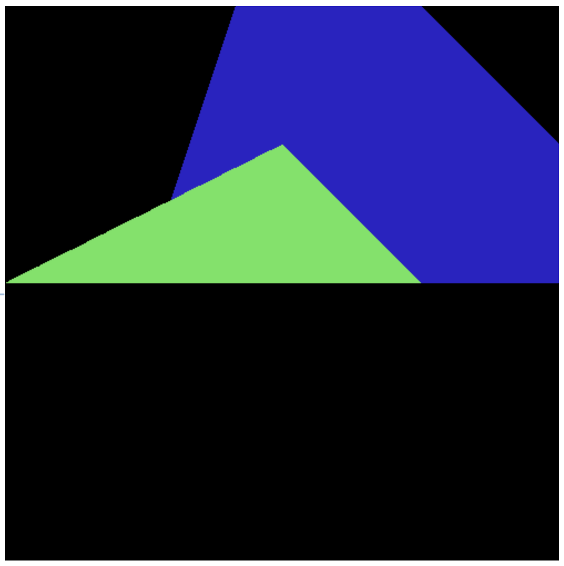
```
1.5 0 0.5
-0.5 0 0.5
0 1.5 0.5

0.5 0 0.25
-1.0 0 0.25
```

```
0 0.5 0.25
```

Output will be,





Algorithm 1: Z-Buffer

To do the above task please follow the guidelines described below.

1. inside main function
    a. read_data()
    b. initialize_z_buffer_and_frame_buffer()
    c. apply_procedure()
    d. save()
    e. free_memory()

2. read_data() :
    a. Read config.txt file and store the values as Screen_Width, Screen_Height, x_limit, y_limit, z_limit accordingly.

    b. Read input information from file named stage3.txt. In the file, each triangle information will be provided by consecutive three lines where each line will contain three coordinate values x, y, z as double.

    c. Use a suitable data structure to hold this information. Also associate a random color value(R, G, B) with each object. RGB values are bounded by 0-255.

    d. Example of a triangle object can be,
        Triangle{

```
                Point points[3];
                int color[3];
        }
```

d. Print and check whether you have correctly read the information from file.

3. initialize_z_buffer_and_frame_buffer():

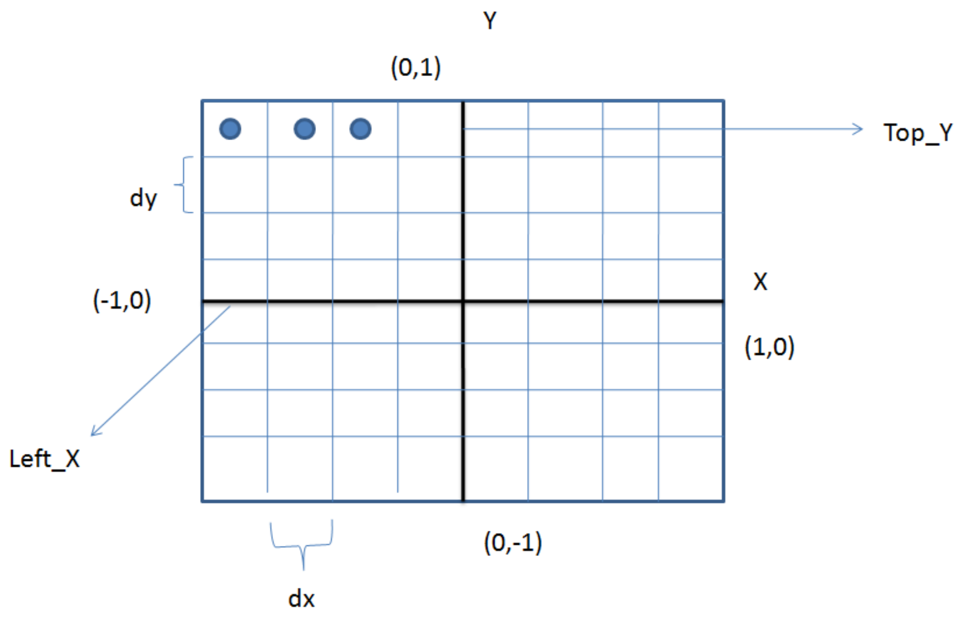
a. You have to create a pixel mapping between the x-y range values and the Screen_Width X Screen_height range.

   i. To do this we first need to find the values of dx and dy.
   For Screen_Width=8 and Screen_Height=8, dx= 2/8, dy= 2/8.

   ii. You also need to specify Top_Y, and Left_X values,

As during scanning from top to bottom and left to right, we will check for the middle values of each cell. Eg. Top_Y- r*dy, Left_X+c*dx.

Top_y= 1-dy/2, Left_x= -1+dx/2



c. Create a Z_buffer, a two dimensional array of Screen_Width X Screen_Height Dimension and Initialize its value with z_max for all positions. In example case, here z_max =2.0
(You must do this using dynamic memory allocation).

d. Create a bitmap_image object with Screen_Width X Screen_Height resolution and initialize its background color with black.

NB: Please follow my image_drawing.cpp code sample for checking how to utilize the image library.

4. apply_procedure():

```
foreach object:Triangles

        Find top_scanline and bottom_scanline after necessary clipping

        for row from top_scanline to bottom_scanline

                Find left_intersecting_column and right_intersecting_column
                after necessary clipping

                for col from left_column to right_column
                        Calculate z values
                        Compare with z_buffer and z_front_limit and update if
required
                        Update pixel information if required
                end
        end

end
```

****Additional details

1. **Find top_scanline & bottom scanline**
2. **and bottom_scanline after necessary clipping :** To do this, first find maximum y value and minimum y values. Then check whether these minimum and maximum values are within bounding y limit. Perform clipping if necessary. After that find the corresponding top_scan row from maximum y value and bottom_scan_row from minimum y value.

3. **Find left_intersecting_column and right_intersecting_column after necessary clipping:** To do this one, first for a scanline row find the two edges/one edge of triangle that intersect the row. After that find the intersecting points from edges and row (user linear interpolation or parametric equation). Let the intersecting point (x1, y, z1) (x2, y, z2). From these two points find maximum and minimum x which corresponds to left_scan_point and right_scan_point. Check whether these values are within x bounding limit or not. Perform clipping if necessary. After that find the corresponding Left_column and Right_column.
4. **Calculate z values:** Now that you have row number and col number, left x1,z1 and right x2,z2 for that row. For col find the corresponding x, and using x-z line equation (or incremental approach as shown in class) find the z value for a particular row, col.

5. save():

    Save the updated image as “1.bmp”.
    Save the z_buffer values as “z_buffer.txt”
        For each row, only save those values where z_buffer[row][col]<z_max.
Check the output files

6. free_memory():
    Free objects memory
    Free image memory,
    Free z_buffer memory

Algorithm 2: Scan-Line Algorithm

** Please refer to book chapter for details (chapter 15 and chapter 3)

Procedure:

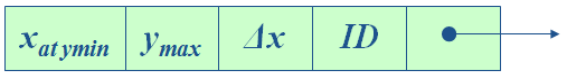
1. read_data()
2. initialize_edge_table_and_polygon_table()
3. apply_procedure()
4. save()
5. free_memory()

Details:

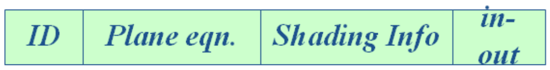
1. Step 1, 4 & 5 is same as before

2. initialize_edge_table_and_polygon_table():
    a. Create edge table :Edge Table (ET) (for non-horizontal edge)
       Sorted into buckets based on each edge’s smaller y-coordinate
       Within buckets are ordered by increasing x-coordinate of their lower end point. (refer chapter 3 for this)

       $\Delta x = \Delta y / m \; ; \; \Delta y = 1$



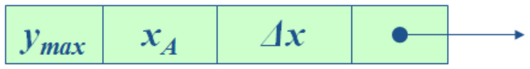
    b. Create polygon table:



ID	Plane eqn.	Shading Info	in-out

3. apply_procedure():-

a. Maintain active edge table: Active Edge Table (AET) Stores the list of edges intersecting current scanline in increasing order of current x-coordinate



b. Maintain active polygon table
Active Polygon Table (APT) At each x-scan value this table contains the list of polygons whose in-out flag is set to true

c. Initialization:
    Initialize the AEL to empty
    Initialize each screen pixel to bk-color
    Set y to the first nonempty cell value in edge table

d. Pseudo-code: (refer to chapter 15 for this)

```
For each scan line y do
    AEL ← AEL∪ Edges from ET that are in current scanline
    sort AEL in order of increasing xA

    For each edge e (except last one) in AEL do
        invert in-out flag of the polygon that contains e
        Update APL

        Determine polygon p in APL with smallest z value at (e.xA, y)

        The pixels from e upto next edge in AEL are set to the color of p

    AEL ← AEL– Edges from ET with ymax = y

    for each edge e in AEL do e.xA = e.xA + e.Δx
    Sort AEL on xA
```

e. Save image and free memory
Save the output image as “2.bmp”.

Important:

1. **Mark Distribution:**
https://docs.google.com/spreadsheets/d/19POfJOtJyDPPPoEaZrT06iCjFrfcVLMA

TPveQ9hDhKM/edit?usp=sharing

	Z-Buffer Algorithm					Scan Line Algorithm				
Submission (10)	File I/O (5)	color(5)	Clipping(10)	z_buffer (10) Relative Postion	Image (15)	File I/O (5)	color(5)	ET/PT/AET/APT (10)	Triangles Correct Depth(10)	output Image (15)