# 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
   1. The general format of this file is

500 500

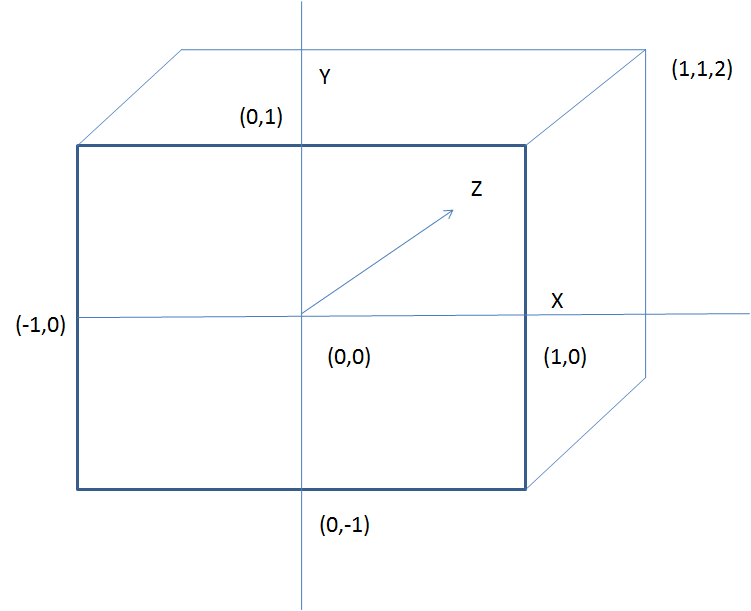
-1.00

-1.00

0.00 2.00

* 1. First Line of file represents [Screen\_Width X Screen\_Height]
  2. Second line specify the left limit of X. [x\_right\_limit=-x\_left\_limit]
  3. Third line specify the bottom limits of Y. [y\_top\_limit=-y\_bottom\_limit]
  4. Fourth line specify the front and rear limits of Z

1. **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:

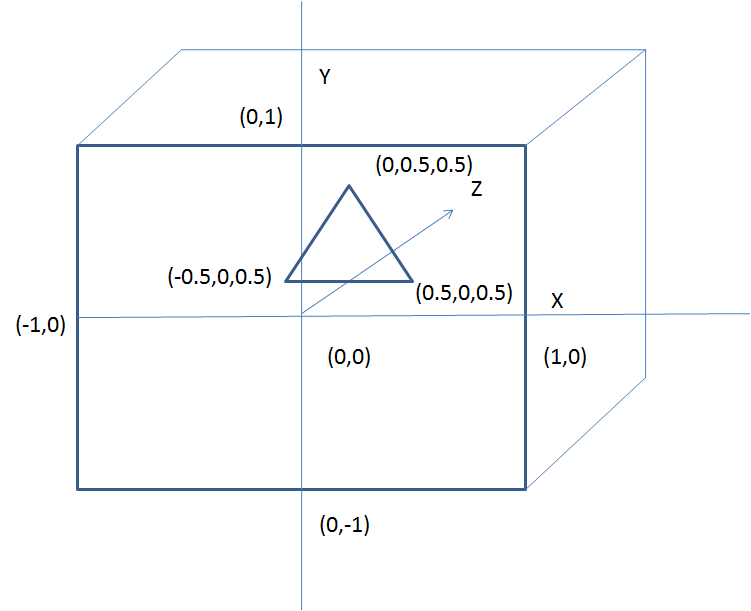
Suppose the stage3.txt contains only

0.50 0.00 0.50

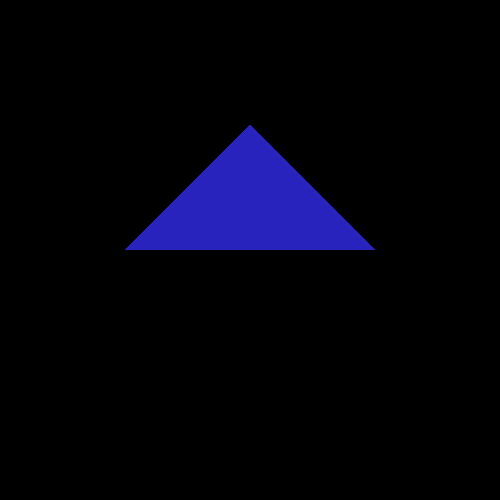
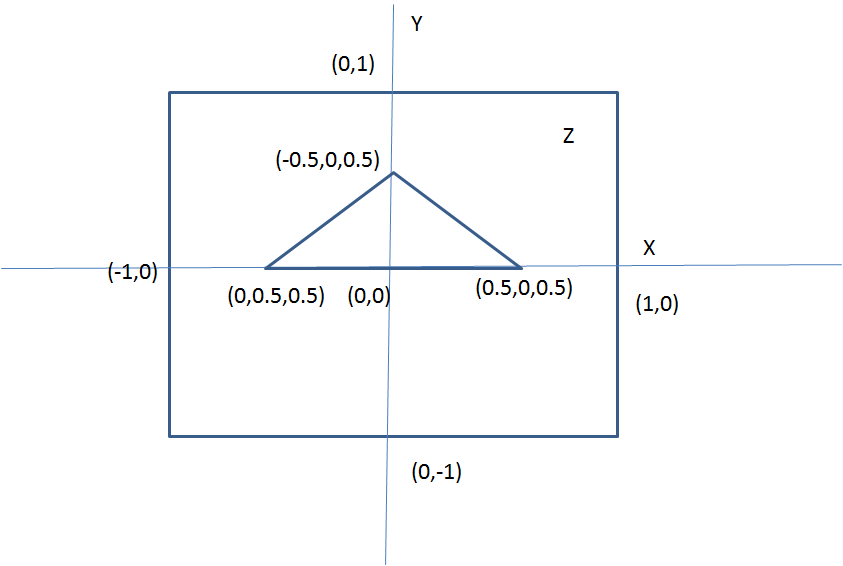
-0.50 0.00 0.50

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

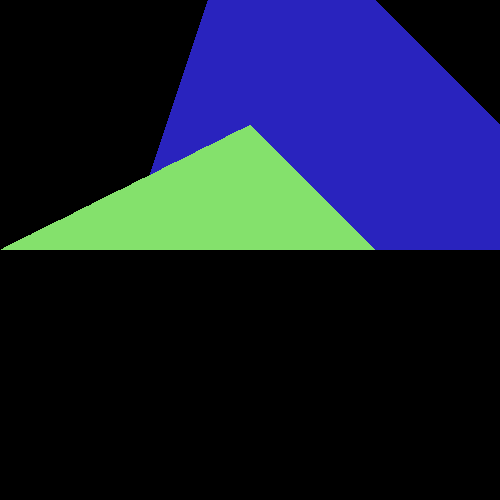
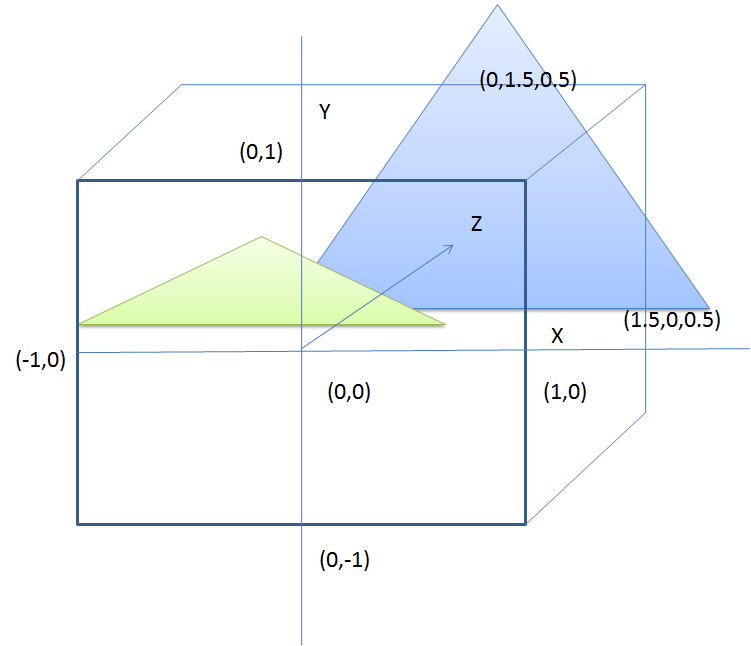
0 1.5 0.5

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
   1. read\_data()
   2. initialize\_z\_buffer\_and\_frame\_buffer()
   3. apply\_procedure()
   4. save()
   5. free\_memory()

2. read\_data() :

* 1. Read config.txt file and store the values as Screen\_Width, Screen\_Height, x\_limit, y\_limit, z\_limit accordingly.
  2. 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.
  3. 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.
  4. 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():

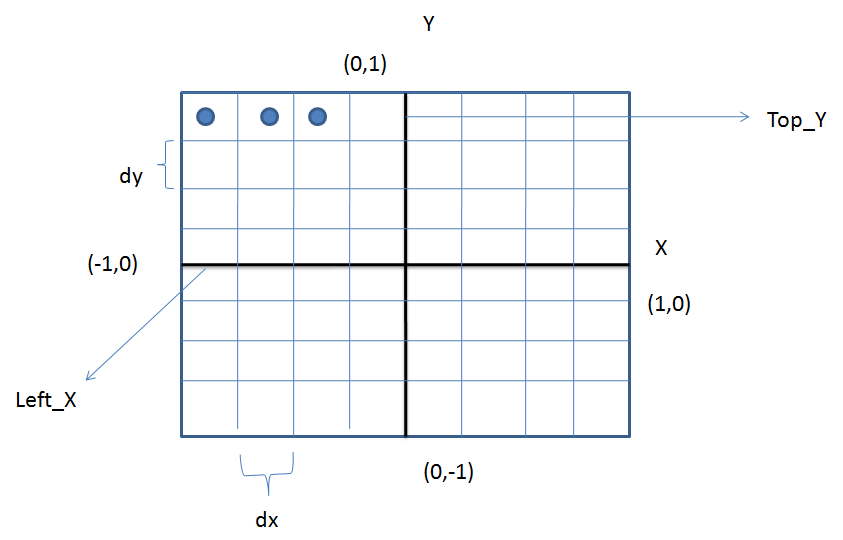
* 1. You have to create a pixel mapping between the x-y range values and the Screen\_Width X Screen\_height range.
     1. To do this we first need to find the values of dx and dy.

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 “output.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

Procedure:

1. read\_data()
2. initialize\_edge\_table\_and\_polygon\_table()
3. apply\_procedure()
4. save()
5. free\_memory()

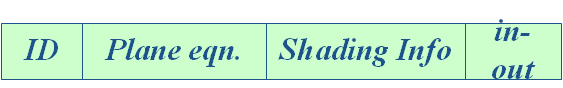
Details:

1. Step 1, 4 & 5 is same as before
2. initialize\_edge\_table\_and\_polygon\_table():
   1. 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.

***Δx=Δy/m ; Δy=1***



* 1. Create polygon table:



1. apply\_procedure():-
   1. Maintain active edge table
   2. Maintain active polygon table

## Important:

1. **Mark Distribution:** <https://docs.google.com/spreadsheets/d/19POfJOtJyDPPPoEaZrT06iCjFrfcVLMATPveQ9hDhKM/edit?usp=sharing>

## 