CSE 410

July 2016 Semester

Assignment 3

Prologue

- In this assignment you will implement *clipping* and *hidden surface removal* of the graphics pipeline.
- Assignment three will be depending on the proper functioning of assignment two. Consequently, it will also carry some marks on completion of functionalities implemented in assignment two. So, try to complete Assignment two if you have not done it already. Do not take codes from others.
- You will be using Z-buffer algorithm. Review it before starting.

Input

Input will be from a file named *scene.txt* . Format of the file will be almost similar to the one for assignment two except insertion of some new data. First few lines of the file will be as followed:

Line 1: `eyeX eyeY eyeZ`
Line 2: `lookX lookY lookZ`
Line 3: `upX upY upZ`
Line 4: `fovY aspectRatio near far`
Line 5: `screen_width screen_height`
Line 6: `r g b`

Input

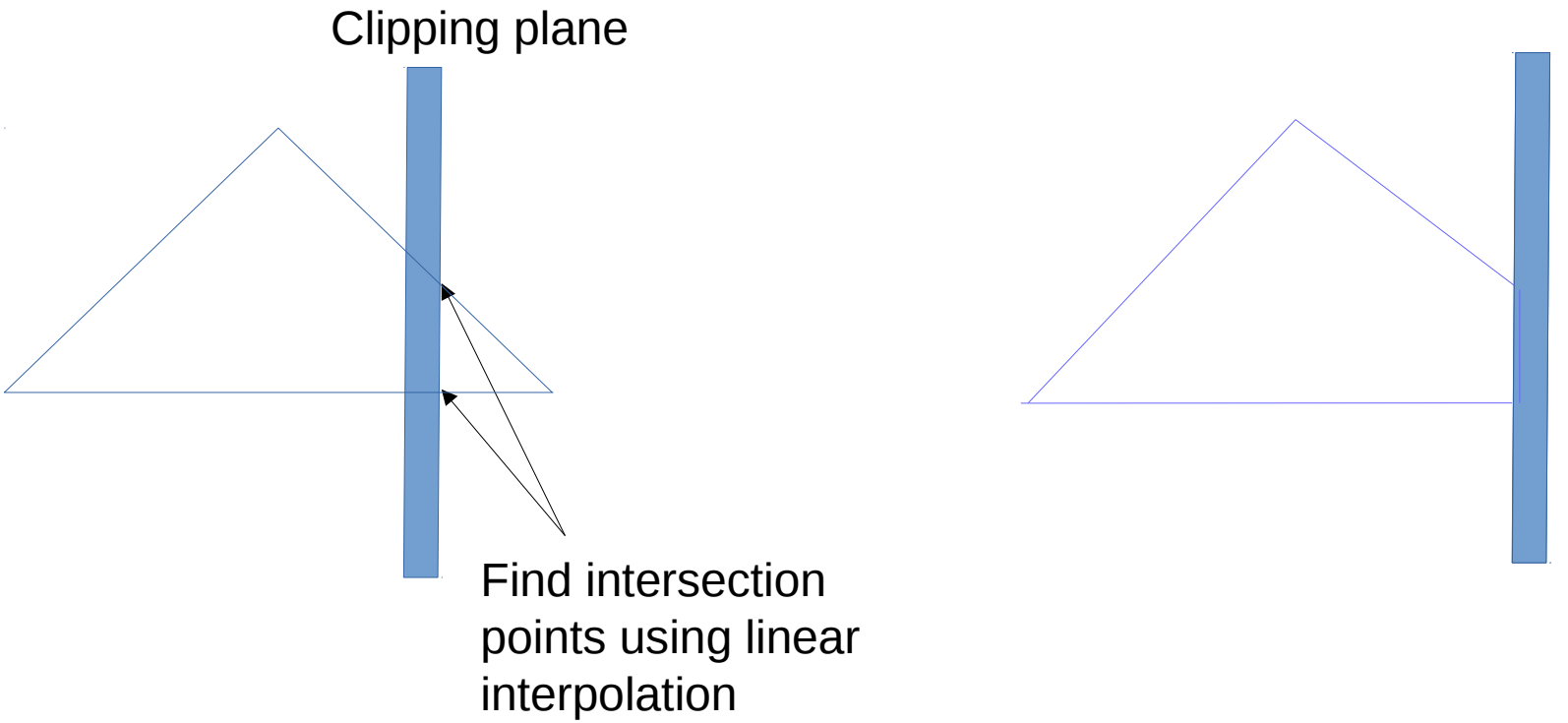
- `screen_width` and `screen_height` will be integers. `r g b` denote the components of background color and can have any integer value between 0 and 255.
- The `triangle` command will be followed by four lines. The first three will be specifying the co-ordinates of the vertices. The fourth line will contain three integers `r g b` (between 0 and 255) denoting the color of the triangle.
- Rest of the input file format will be similar to the one in assignment two.

Output

- You will output three text files : `stage1.txt, stage2.txt and stage3.txt` in the same way you did in assignment two.
- Remember, even for the same input scene, the content of `stage3.txt` may be different from the one generated in assignment two. This is going to be elaborated later on.
- You will also output a bmp image file named `out.bmp` having height and width equal to `screen_height` and `screen_width` respectively. This image will represent visually the input scene after passing through the implemented stages of graphics pipeline.

Step 1

- Clip the output of stage 2 with respect to near plane and far plane.
- Anything having z-value greater than `(-near)` or less than `(-far)` need to be clipped.

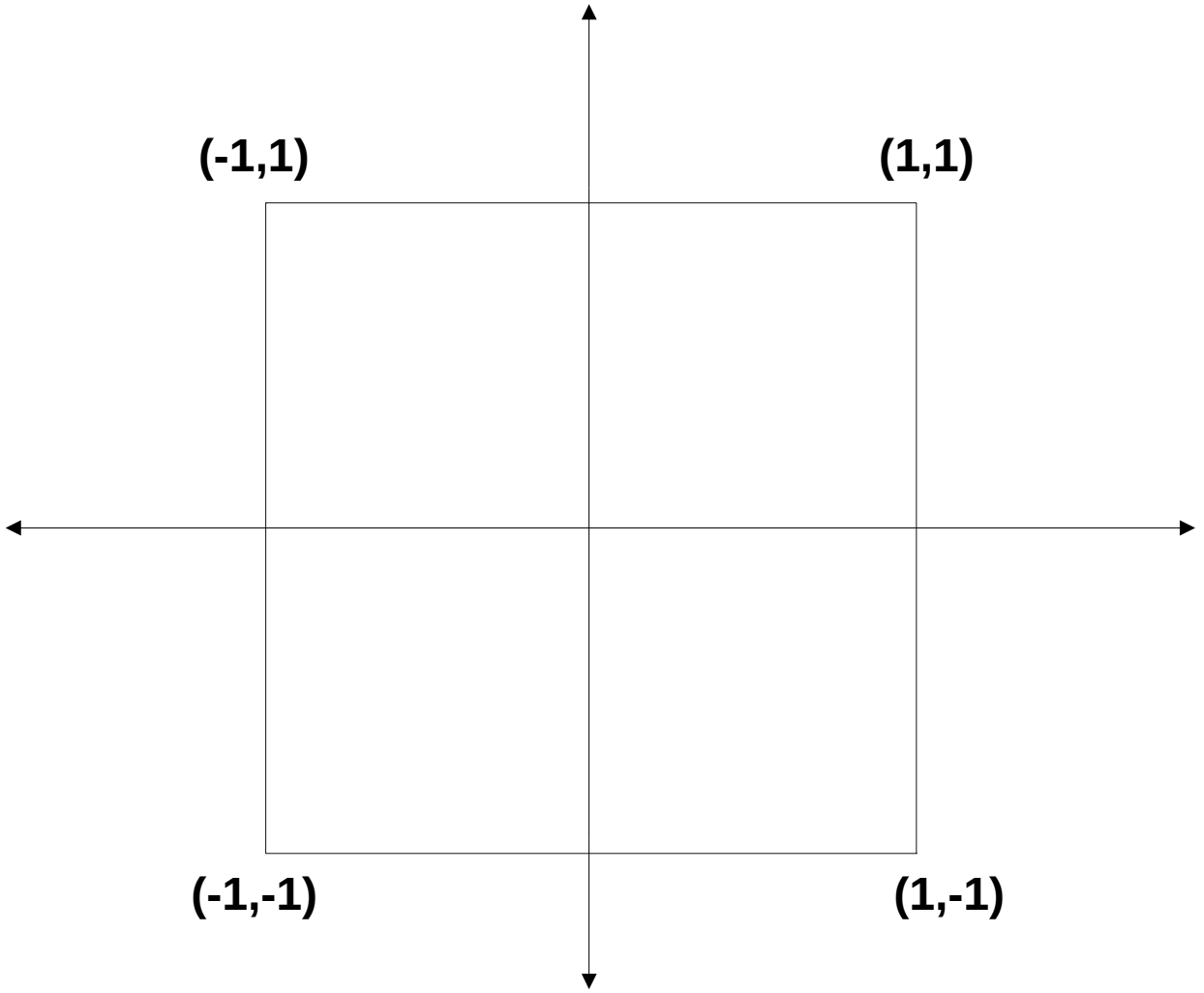


It will be convenient if you convert everything to triangles.

Step 2

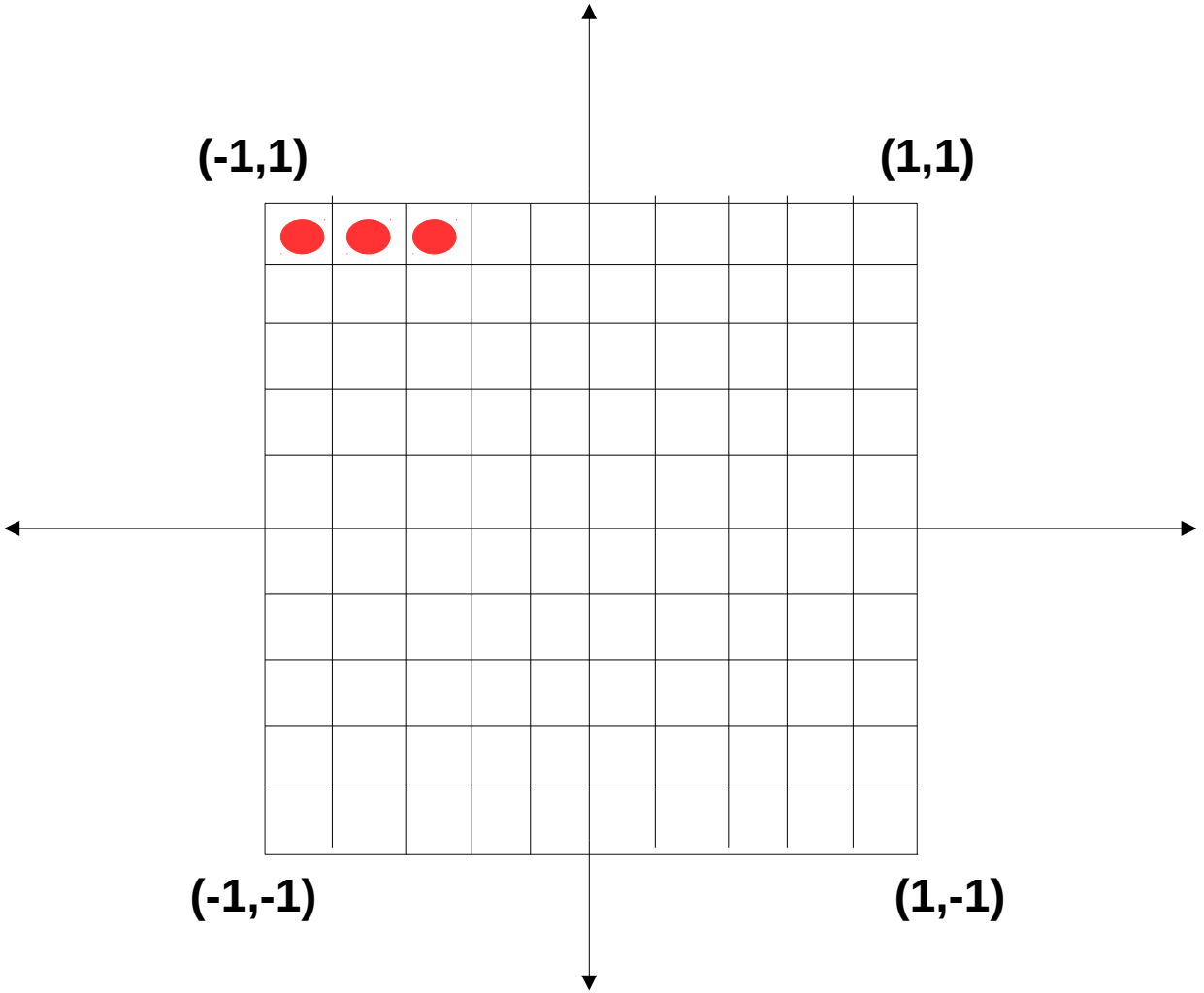
- If you are done with step 1, your next step will start from the output of stage 3 of assignment two.
- After projection transformation, everything that will be visible will have all of x,y and z co-ordinates in [-1,1].
- Since you have already performed clipping with respect to z-axis, z co-ordinates will be within this range by now.

Consider the projection plane as a square of length 2.



Step 2 (contd.)

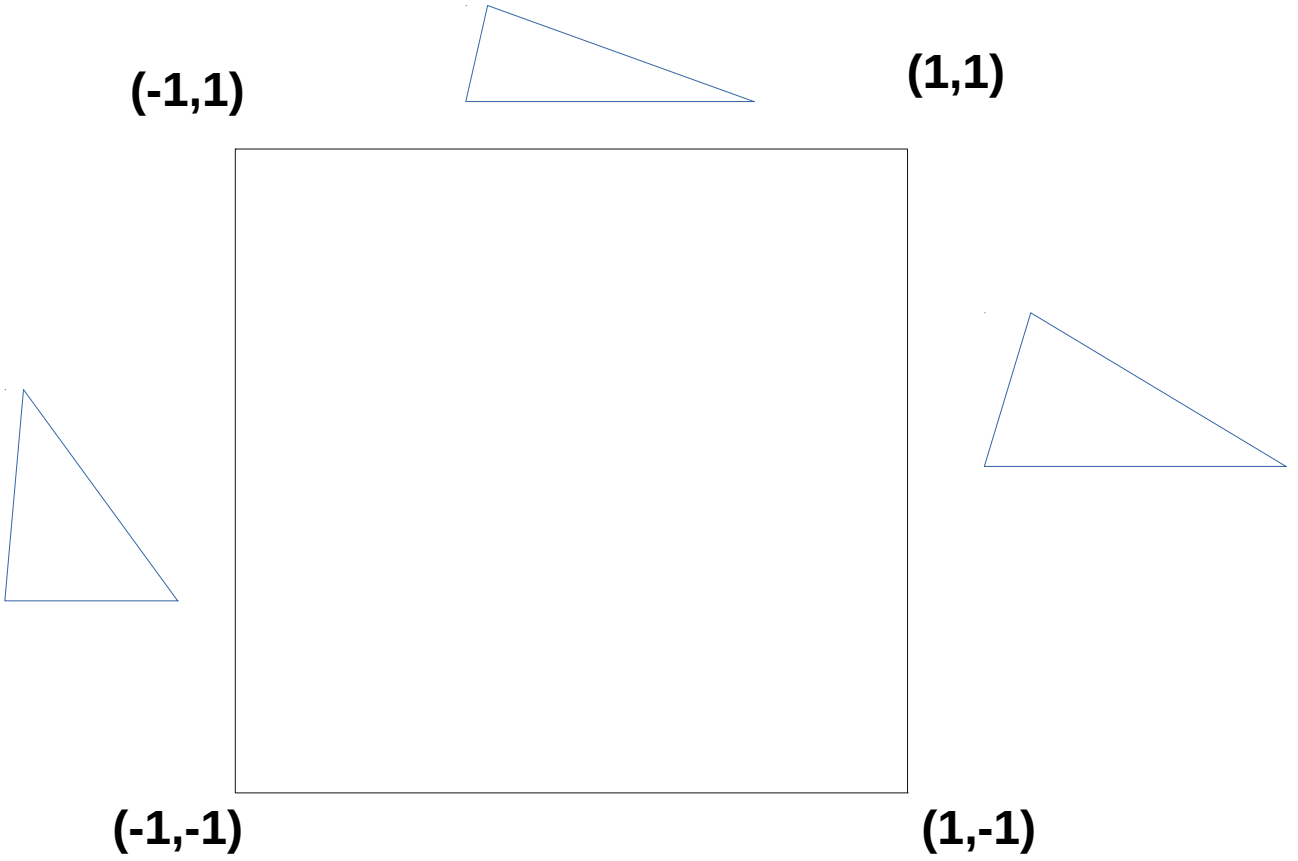
- Divide the projection plane into a two dimensional grid of *screen_width* x *screen_height*
- You have to calculate the center point of each small square to find out the color in the corresponding pixel.
- Construct a *pixel buffer* and a *z-buffer* of the same dimension and initialize them with appropriate values.

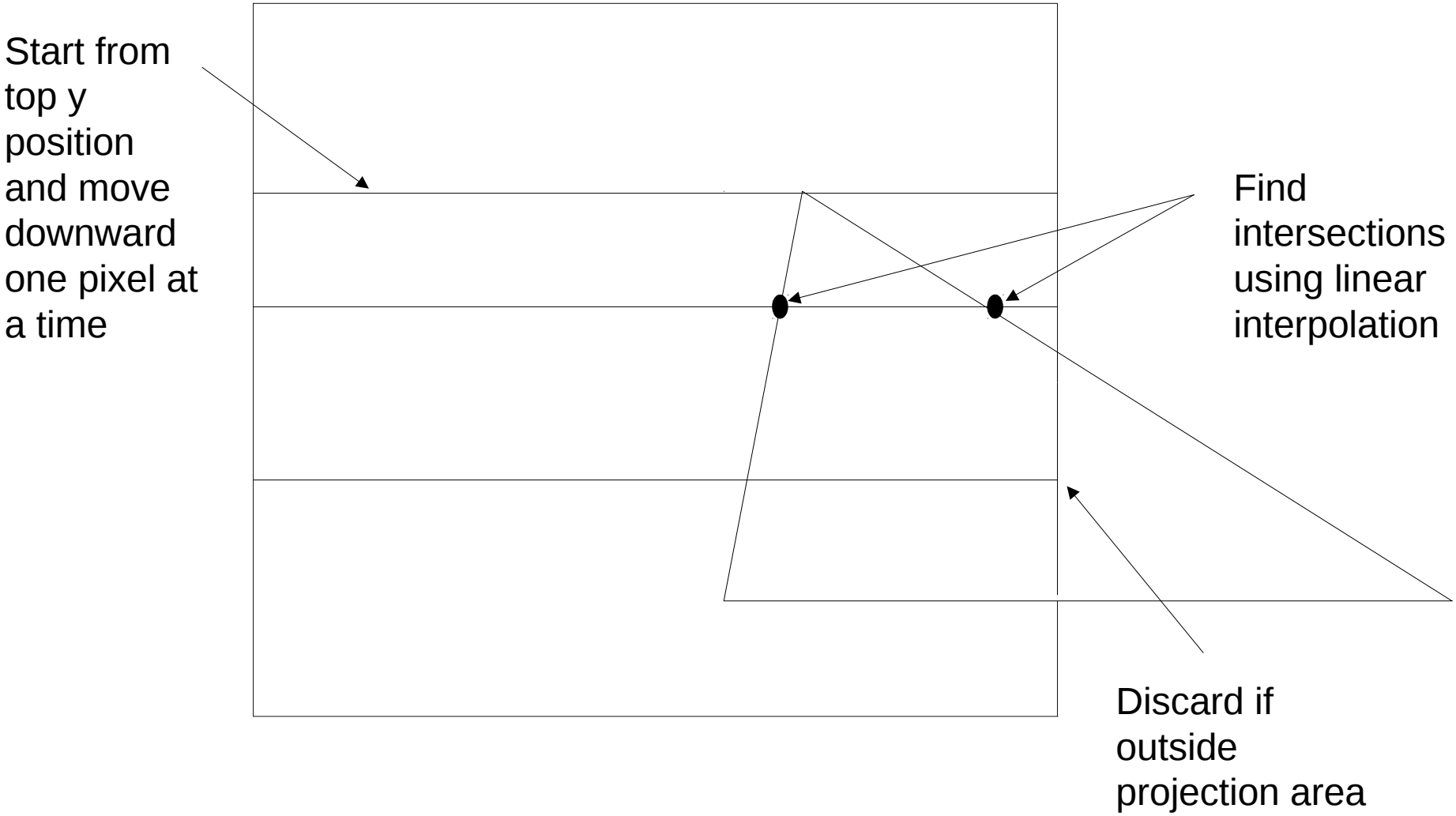

(-1,1)
(1,1)
(-1,-1)
(1,-1)

Step 3

- This is the most crucial step of the assignment.
- You have to scan convert each triangle to fill up the pixel buffer.
- You will be using the z-buffer for deciding whether to put a value in the pixel buffer and also update the z-buffer accordingly.

First, reject the triangles which are totally outside the projection area.




Start from top y position and move downward one pixel at a time
Find intersections using linear interpolation
Discard if outside projection area

Step 3 (contd.)

- Each scan-line will intersect a triangle in at most two points. Find the co-ordinates from the co-ordinates of the vertices using linear interpolation.
- Find out the pixel position of the left intersection point and compare the z-value from the Z-buffer. If required conditions are satisfied, put the color of the triangle in the corresponding position of the pixel buffer and update the Z-buffer.
- Move one pixel right and calculate the z-value at that position from linear interpolation of the two intersection points.

Step 3 (contd.)

- Repeat this step till you get to the right intersection point.
- Handle special cases of a single intersection point and intersecting a horizontal line.
- Take some precautions to avoid precision related issues. Consider a small margin for floating point comparisons.
- Finally save the pixel buffer as an image.

Creating Image

- Use the provided header file *bitmap_image.hpp*
- You will be required to use only a few functions. A sample code for creating an image and setting colors in specific pixel values is provided.
- Be careful about the indexing convention used here.
- In fact you can use the bitmap_image object as pixel buffer.

Happy Coding