CUDA Lab 6. A simple CUDA ray caster

1. Understand how to draw simple image in CUDA
2. Understand how to draw Mandelbrot and Julia Sets.

Exercise 1. Set up a virtual canvas and draw on it an image in CUDA

1. The lab descriptions shown below are based on the start programs “Drawing_cuda.cpp” and “Drawing _cuda.cu”, simple to the framework provided in Lab 4, which are created directly from CUDA SDK sample “bicubicTexture”. Start from your work on Lab 4 and replace “ImageProcess_cuda.cpp” and “ImageProcess_cuda.cu” with “Drawing_cuda.cpp” and “Drawing _cuda.cu”.

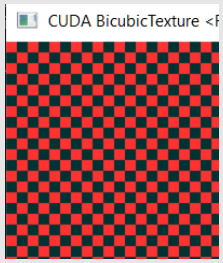
2. Compile your program, you should see a red image.

3. Modify the first three values shown in make_uchar4( ) in the following line of code to draw an image of different colours, say, a green image, a grey image.

```
d_output[i] = make_uchar4(0, 0, 0xff, 0);
```

Exercise 2. Drawing a checkboard in CUDA

1. Edit the d_render( ) method to draw an checkboard image shown below.



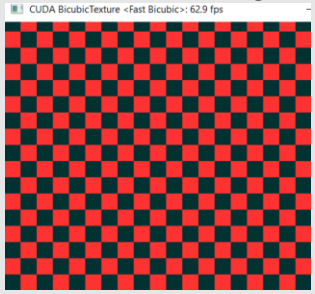
You can get an idea of how to draw it by refer to the *void makeCheckImage(void) method* provided from the link:

https://www.glprogramming.com/red/chapter09.html

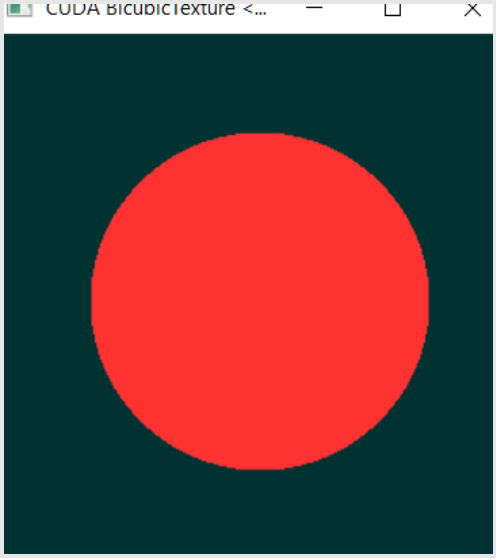
```
void makeCheckImage(void)
{
   int i, j, c;

   for (i = 0; i < checkImageHeight; i++) {
      for (j = 0; j < checkImageWidth; j++) {
         c = ((((i&0x8)==0)^((j&0x8))==0))*255;
         checkImage[i][j][0] = (GLubyte) c;
         checkImage[i][j][1] = (GLubyte) c;
         checkImage[i][j][2] = (GLubyte) c;
         checkImage[i][j][3] = (GLubyte) 255;
      }
   }
}
```

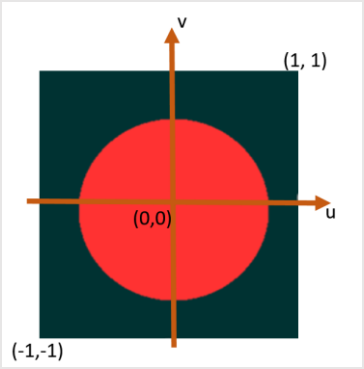
2. Modify you code to draw a checkboard with much larger red-blocks, as shown below



3. Further modify your code to draw a red disc in the middle of the image of a red disc:



4. Redraw the above image based on pixel coordinates defined in float type variables in [-1, 1]x[-1, 1], as is shown below.



This can be done by performing transformation to (x, y) coordinates (screen space pixel location) in the following way.

```
float u = x / (float)width;
float v = y / (float)height;

u =2.0*u- 1.0;
v = -(2.0*v - 1.0);
```

To avoid distortion to the image when resizing, it is better to make the image has the same aspect ratio as the window. This can be achieved by scaling u-coordinate using the window aspect ratio:

```
u *= width / (float)height;
```

Now draw the red disc using the (u,v) coordinates.

Exercise 3. Drawing the Mandelbrot and Julia Sets.

a) Both the Mandelbrot set and the Julia set are famous 2D fractal objects initially defined based on complex number z=x+yi, by considering iteratively calculating a sequence of complex numbers from a given complex number Z_0 in the following way:

$$Z_0=x+y*i,$$

$$Z_{n+1}=z_n^2+C, \qquad n=1, 2, 3, \dots \dots,$$

where C is a constant complex number. Mandelbrot and Julia sets are defined as sets of those complex numbers such that the sequence of complex numbers from each of these complex numbers never goes to infinity.

For z= x+yi, if we regard it as a point in an image with coordinates(x, y), then z^2 corresponds to a point with coordinates (x^2-y^2, 2*x*y). Thus, we can directly visualize these fractals in CUDA.

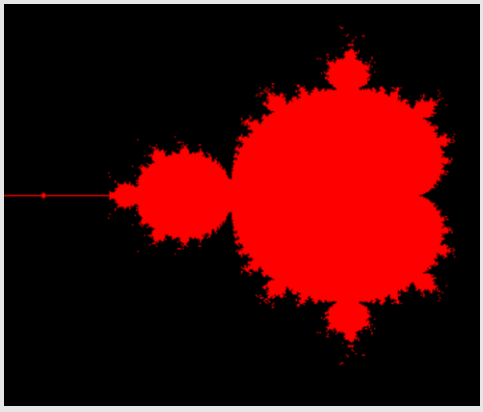
Scale the image size you used in Exercise 2 to a relatively bigger size, say, [-4, 4]x[-4, 4]. You can easily achieve this by multiplying u and v with 4:

```
u *=4.0; v *=4.0;
```

Now regard each (u, v) as a complex number u+v*i, we can visualize what Mandelbrot set by modifying the code you achieved in exercise 2 in the following way:

```
float2 z = { u, v };
float2 T = z;
float r = 0;
float c = 1.0;
for (int i = 0; i < 30; i++) {

    z = { z.x * z.x - z.y * z.y, 2.0 * z.x * z.y };
    z += T;
    r = sqrt(z.x*z.x+z.y*z.y);
    if (r > 5.0) {
        c = 0.0;
        break;
    }
}
```



b) In a), for each point (u, v), the 2D vector T is defined with the start coordinates (u, v). If you replace it with vector independent of (u,v), say, T = {0.25, 0.5}, we can get a Julia set shown below. Different Ts give different Julia sets. Thus, in general, there are infinite different Julia sets.