600086 Lab Book

Week 2 – Lab 6 A simple CUDA ray caster

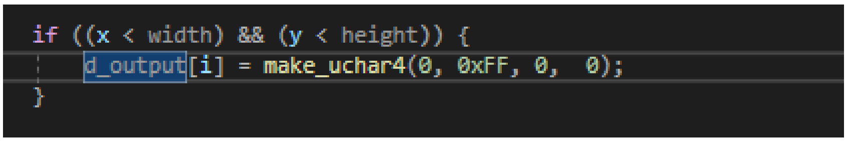
Date: 10th Mar 2022

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

Question1 :

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.

Solution:

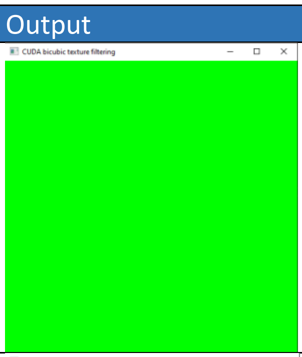
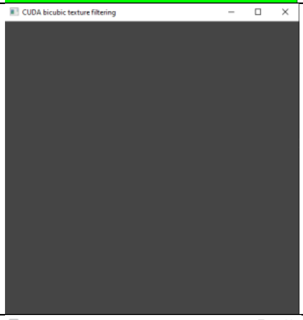
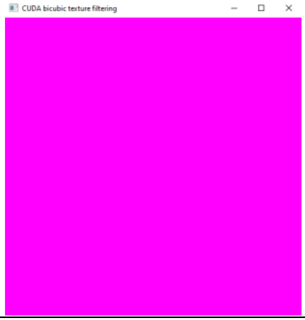


```
if ((x < width) && (y < height)) {
    d_output[i] = make_uchar4(0, 0xFF, 0,  0);
}
```

Test data:

N/A

Sample output:

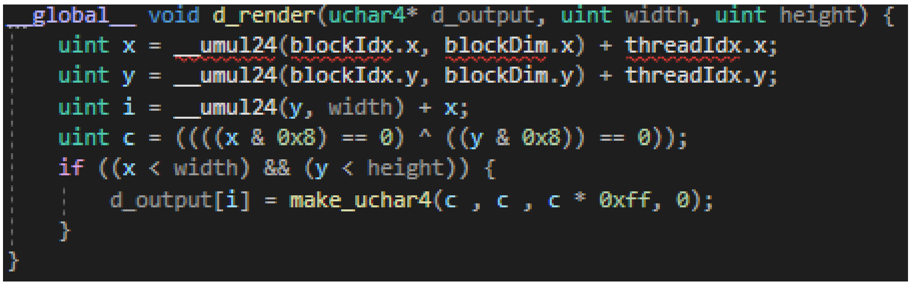
Input	expectation	Output
`(0, 0xFF, 0, 0)`	Green	
`(0x45, 0xF45, 0x45, 0)`	Grey	
`(0, 0xFF, 0, 0)`	Fuchsia	

Exercise 2. Drawing a checkboard in CUDA

Question2: implement a solution using GPU processing to solve the problem

Solution:

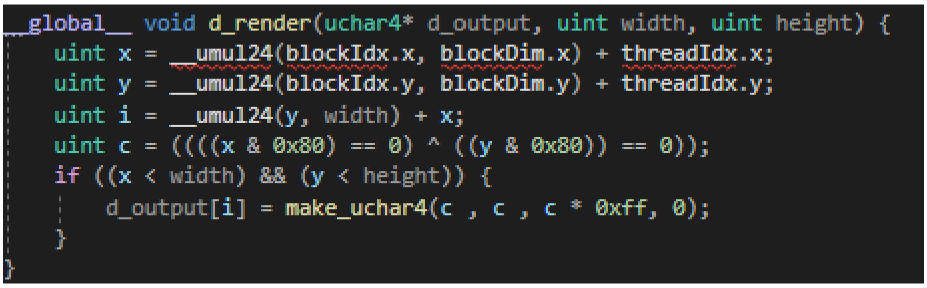
1. Edit the d_render( ) method to draw a checkboard



```
__global__ void d_render(uchar4* d_output, uint width, uint height) {
    uint x = __umul24(blockIdx.x, blockDim.x) + threadIdx.x;
    uint y = __umul24(blockIdx.y, blockDim.y) + threadIdx.y;
    uint i = __umul24(y, width) + x;
    uint c = ((((x & 0x8) == 0) ^ ((y & 0x8)) == 0));
    if ((x < width) && (y < height)) {
        d_output[i] = make_uchar4(c , c , c * 0xff, 0);
    }
}
```

Created a new variable C which is governed by the x and y position of the pixel and applied a colour mask to it in the make_uchar4() to make the odd segments red. See Sample output ref1 for result

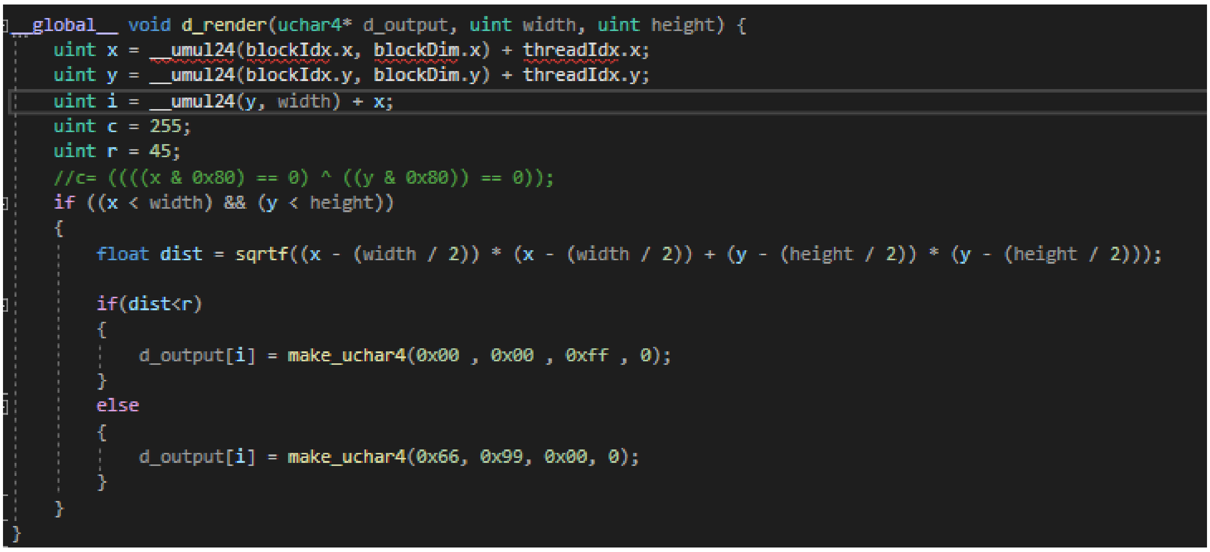
2. Modify the code to draw a checkboard with much larger red-blocks



```
__global__ void d_render(uchar4* d_output, uint width, uint height) {
    uint x = __umul24(blockIdx.x, blockDim.x) + threadIdx.x;
    uint y = __umul24(blockIdx.y, blockDim.y) + threadIdx.y;
    uint i = __umul24(y, width) + x;
    uint c = ((((x & 0x80) == 0) ^ ((y & 0x80)) == 0));
    if ((x < width) && (y < height)) {
        d_output[i] = make_uchar4(c , c , c * 0xff, 0);
    }
}
```

By increasing the value that x and y are multiplied by when calculating c the size of the squares in the grid can be modified I increased this to 0x80. see Sample output Ref 2 for result.

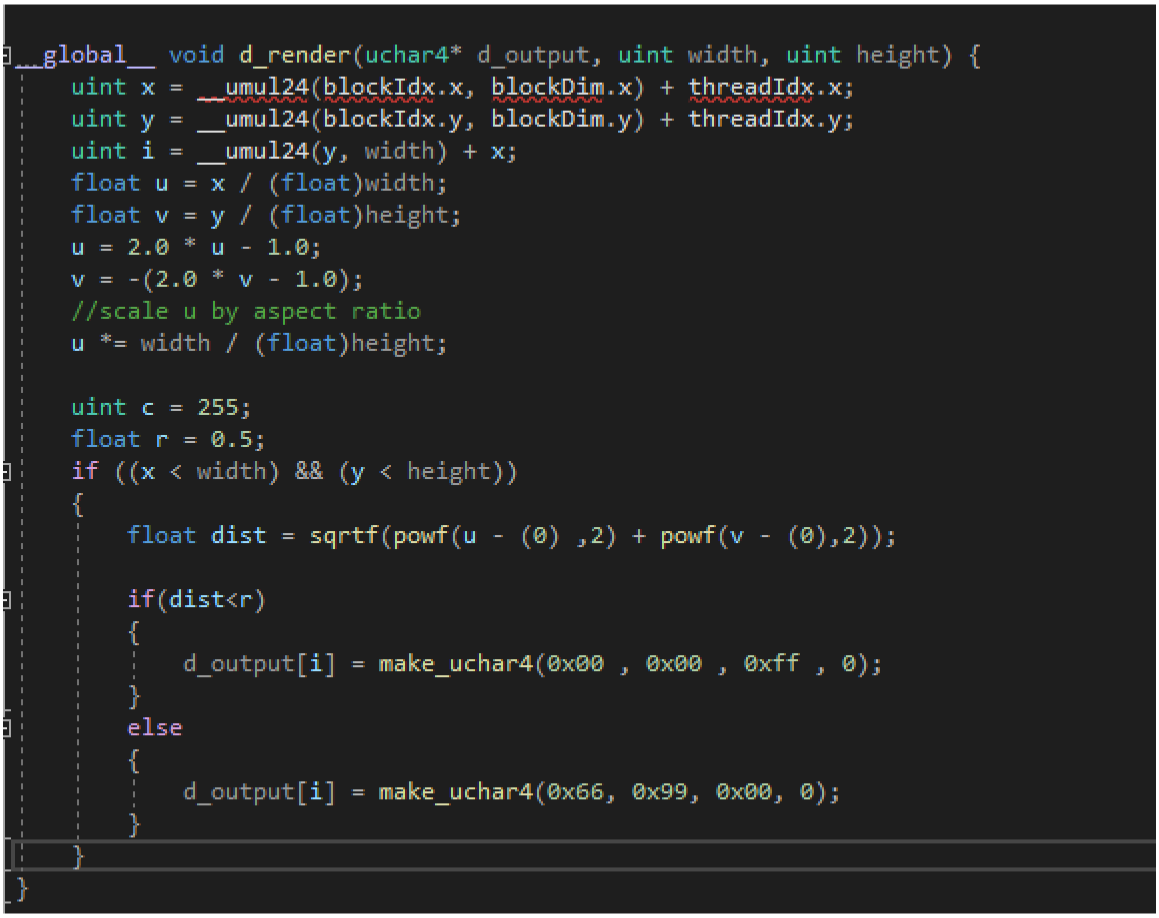
3. Further modify your code to draw a red disc in the middle of the image of a red disc:



```
__global__ void d_render(uchar4* d_output, uint width, uint height) {
    uint x = __umul24(blockIdx.x, blockDim.x) + threadIdx.x;
    uint y = __umul24(blockIdx.y, blockDim.y) + threadIdx.y;
    uint i = __umul24(y, width) + x;
    uint c = 255;
    uint r = 45;
    //c= ((((x & 0x80) == 0) ^ ((y & 0x80)) == 0));
    if ((x < width) && (y < height))
    {
        float dist = sqrtf((x - (width / 2)) * (x - (width / 2)) + (y - (height / 2)) * (y - (height / 2)));

        if(dist<r)
        {
            d_output[i] = make_uchar4(0x00 , 0x00 , 0xff , 0);
        }
        else
        {
            d_output[i] = make_uchar4(0x66, 0x99, 0x00, 0);
        }
    }
}
```

I added in a check to see if the coordinate distance of the pixel is within the range r and if so colour it red if not colour it teal. See Sample output Ref 3 for the result.

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



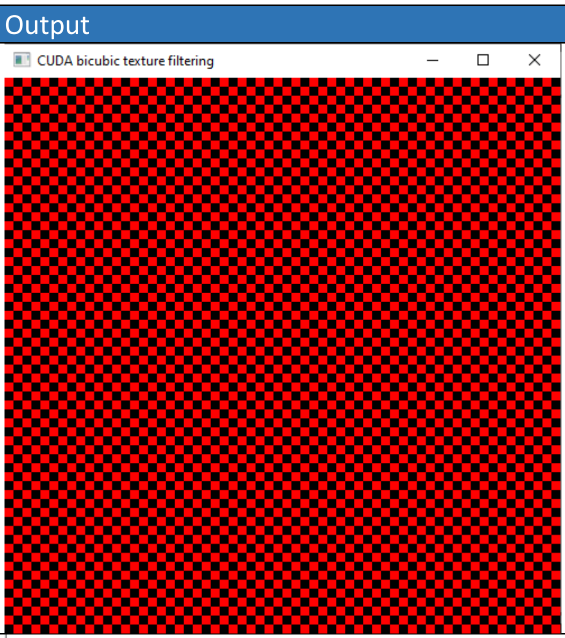
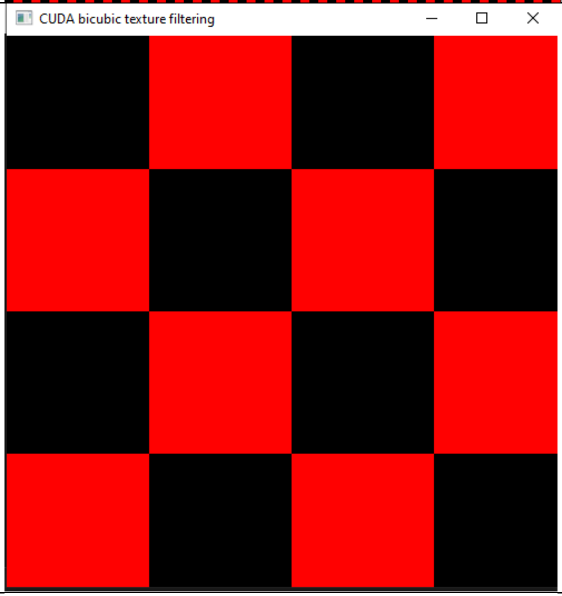
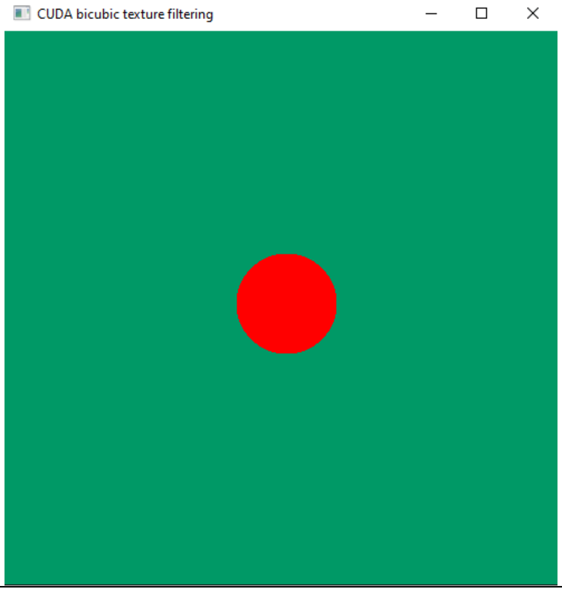
```
__global__ void d_render(uchar4* d_output, uint width, uint height) {
    uint x = __umul24(blockIdx.x, blockDim.x) + threadIdx.x;
    uint y = __umul24(blockIdx.y, blockDim.y) + threadIdx.y;
    uint i = __umul24(y, width) + x;
    float u = x / (float)width;
    float v = y / (float)height;
    u = 2.0 * u - 1.0;
    v = -(2.0 * v - 1.0);
    //scale u by aspect ratio
    u *= width / (float)height;

    uint c = 255;
    float r = 0.5;
    if ((x < width) && (y < height))
    {
        float dist = sqrtf(powf(u - (0) ,2) + powf(v - (0),2));

        if(dist<r)
        {
            d_output[i] = make_uchar4(0x00 , 0x00 , 0xff , 0);
        }
        else
        {
            d_output[i] = make_uchar4(0x66, 0x99, 0x00, 0);
        }
    }
}
```

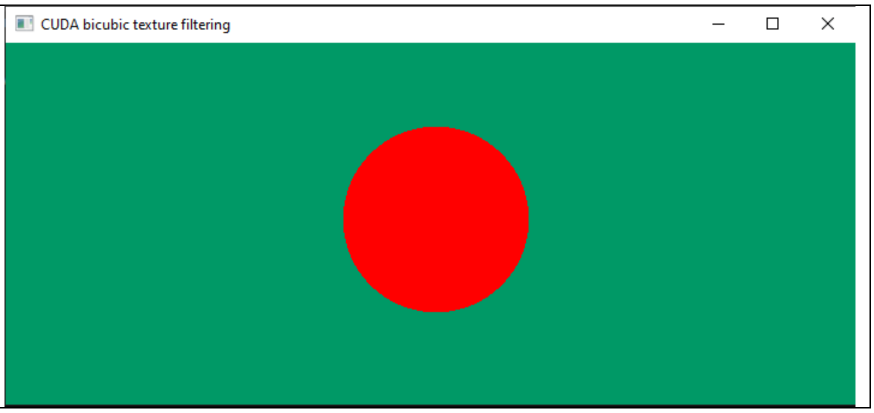
I Added in a translation for the pixel location represented by u and v and then added a acle translation to ensure the resultant image matched the aspect ratio of the window to prevent distortion. See Sample Output Ref 4 for the result.

Test data:

N/A

Sample output:

REF	Output
1	
2	
3	

4	

Reflection:

This task seemed fairly perfunctory, but was very interesting seeing how shapes can be drawn to the screen using vectors,

Metadata:

Further information:

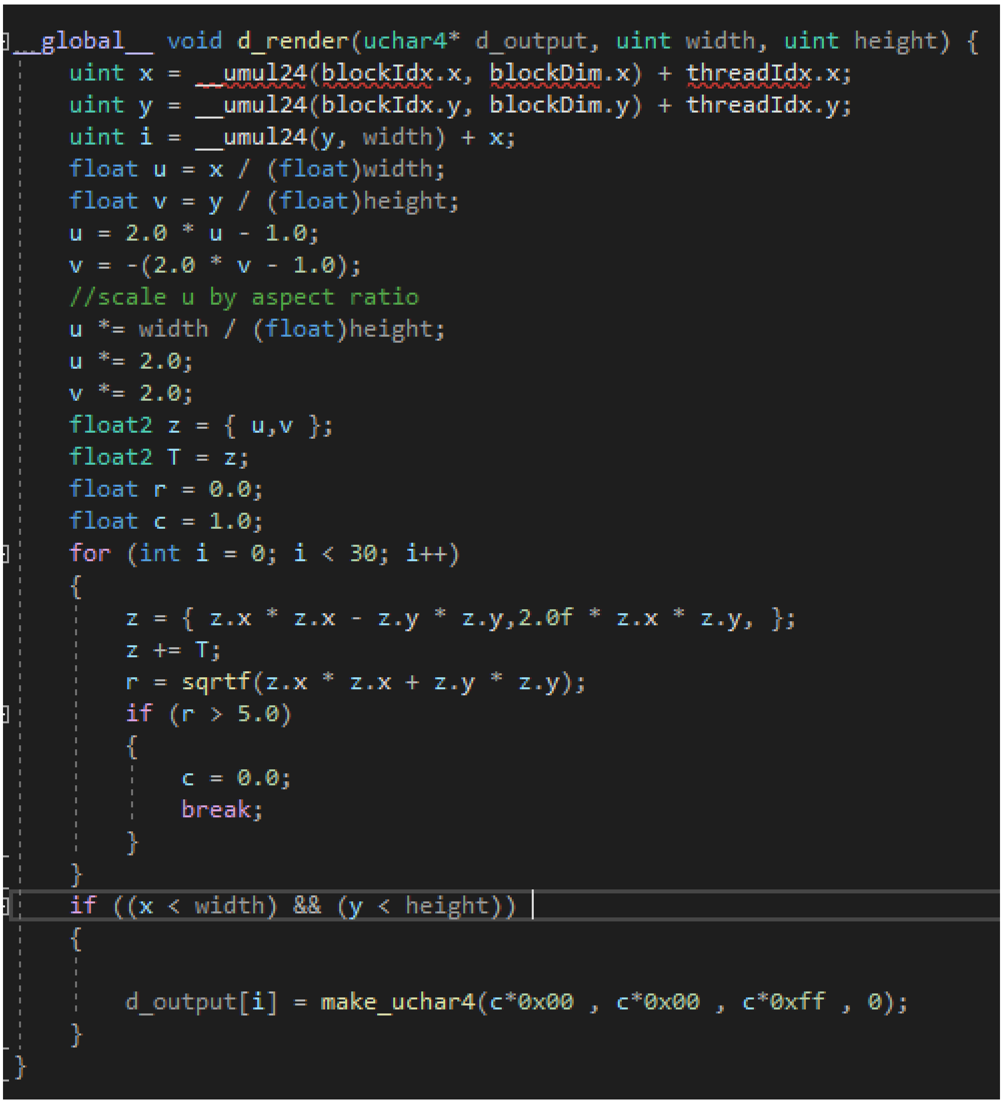
is this similar to how vector graphics are created?

Exercise 3. Drawing the Mandelbrot and Julia Sets.

Question1: modify the previous code in order to draw Mandelbrot and Julia sets.

Solution:

1. Modify the code to draw a Mandelbrot set



```
__global__ void d_render(uchar4* d_output, uint width, uint height) {
    uint x = __umul24(blockIdx.x, blockDim.x) + threadIdx.x;
    uint y = __umul24(blockIdx.y, blockDim.y) + threadIdx.y;
    uint i = __umul24(y, width) + x;
    float u = x / (float)width;
    float v = y / (float)height;
    u = 2.0 * u - 1.0;
    v = -(2.0 * v - 1.0);
    //scale u by aspect ratio
    u *= width / (float)height;
    u *= 2.0;
    v *= 2.0;
    float2 z = { u,v };
    float2 T = z;
    float r = 0.0;
    float c = 1.0;
    for (int i = 0; i < 30; i++)
    {
        z = { z.x * z.x - z.y * z.y,2.0f * z.x * z.y, };
        z += T;
        r = sqrtf(z.x * z.x + z.y * z.y);
        if (r > 5.0)
        {
            c = 0.0;
            break;
        }
    }
    if ((x < width) && (y < height))
    {

        d_output[i] = make_uchar4(c*0x00 , c*0x00 , c*0xff , 0);
    }
}
```

Added in a for loop that iteratively validates whether the current pixel is not within the Mandelbrot set and leaves the loop early if this is the case setting the c value to zero so the pixel will be black. See sample output ref 1 for resultant image.

2. Modify the code to draw a Julia set

By changing the Vector T so that it is not the start coordinates then we can create Julia sets as there are an infinite number of Julia sets I have created 3 using the Vector values for T in the Test Data section the results can be seen in Sample output 2,3 and 4 respectively

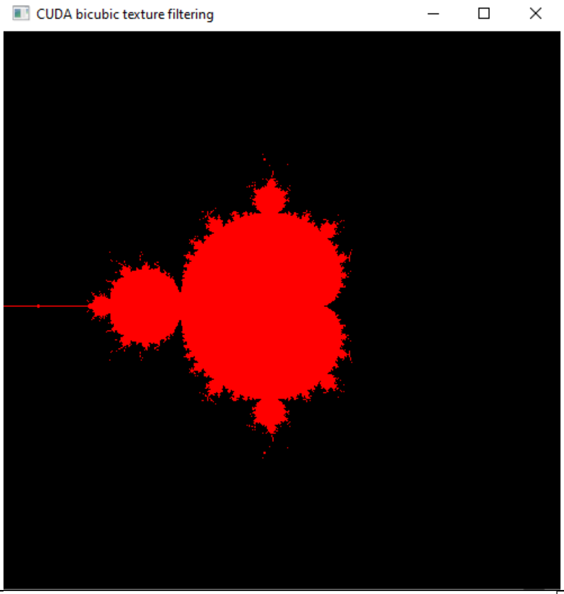
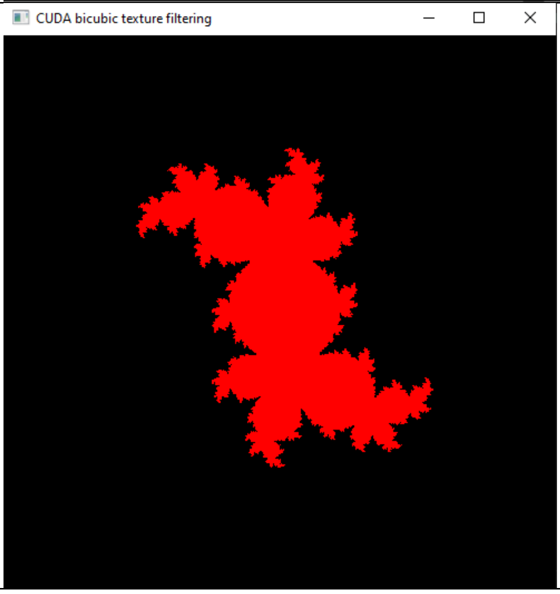
Test data:

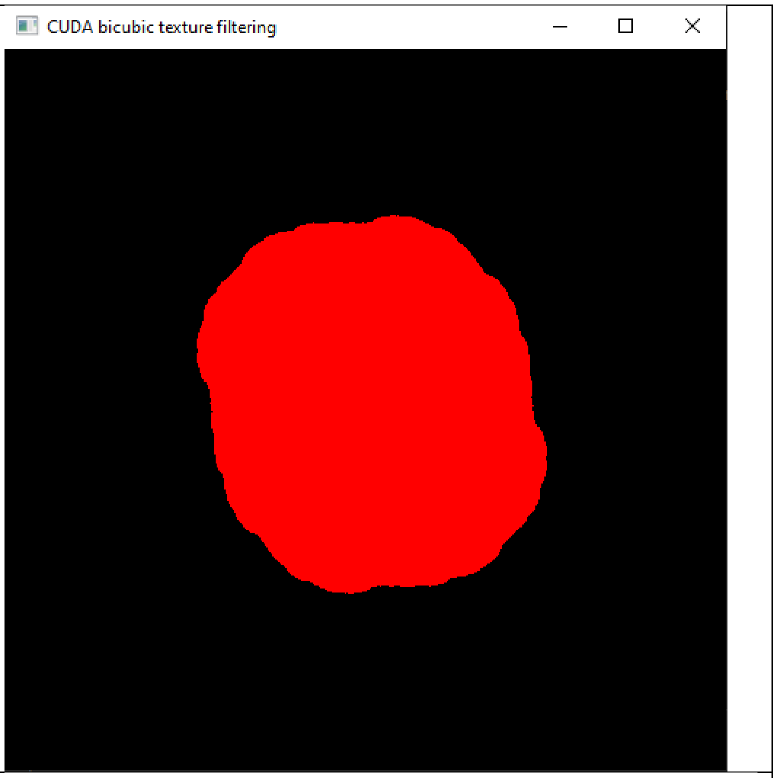
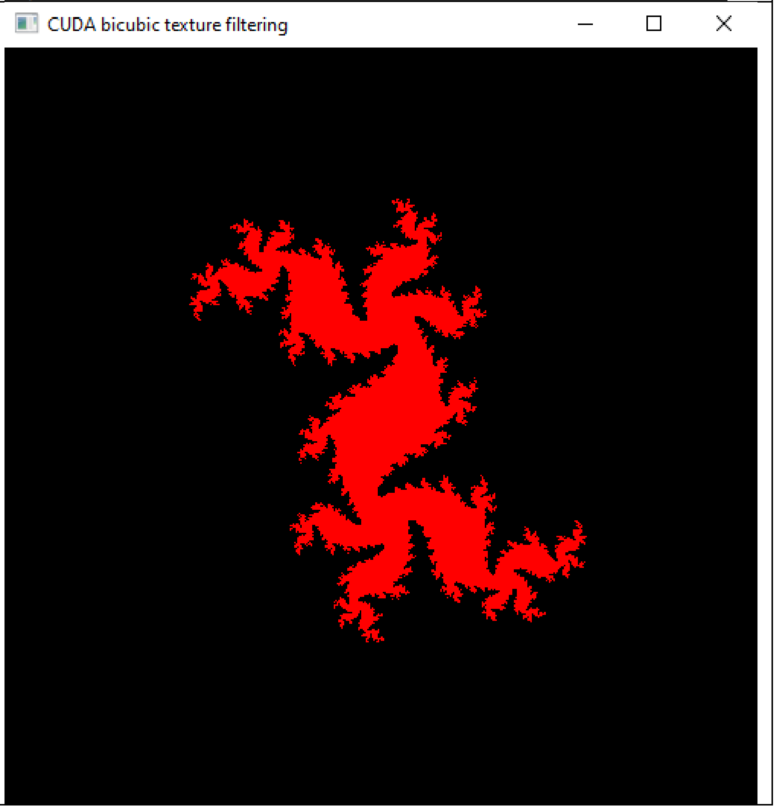
T = {0.25, 0.5}

T = {0.1, 0.1}

T = {0.3, 0.5}

Sample output:

ref	Output
1	 CUDA bicubic texture filtering
2	 CUDA bicubic texture filtering

3	
4	

Reflection:

Adjusting the x value of the T vector makes the Julia set pattern have deeper grooves whereas the y value seems to cause the pattern to have softer edges effectively smoothing the shape.

Metadata: