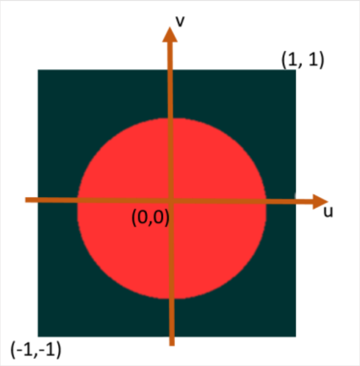
CUDA Lab 7. A simple CUDA ray caster

1. Understand how ray casting works
2. Learn how to implement a simple ray-caster

Exercise 1. Drawing based on a canvas of size [-1, 1]x[-1, 1]

This lab is built upon Lab 6 Exercise 2 task 4 and make sure you understand how to draw a disc based on pixel coordinates defined in float type variables in [-1, 1]x[-1, 1], as is shown below.



Exercise 2. Write a simple ray caster

Parts of the sample is directly based on Ray Tracing in One Weekend in CUDA
https://github.com/rogerallen/raytracinginoneweekendincuda

1. Go to chapter 5: "Chapter 5 - Nomals" and add to the project the following header files *vec3.h*, *sphere.h*, *ray.h*, *hittable_list.h*, *hittable.h*, which can be downloaded from GitHub - rogerallen/raytracinginoneweekendincuda at ch05_normals_cuda
2. To make the elements' names used in ray.h more meaningful, rename the variable names A, B used in ray.h as O and Dir, corresponding to ray origin and ray direction.
3. Add the following CUDA code to the CUDA program

```
#define checkCudaErrors(val) check_cuda( (val), #val, __FILE__, __LINE__ )

void check_cuda(cudaError_t result, char const *const func, const char *const file, int const line) {
    if (result) {
            std::cerr << "CUDA error = " << static_cast<unsigned int>(result) << " at " <<
                    file << ":" << line << " '" << func << "' \n";
            // Make sure we call CUDA Device Reset before exiting
            cudaDeviceReset();
            exit(99);
    }
}


__device__ vec3 castRay (const ray& r, hitable **world) {
    hit_record rec;
    if ((*world)->hit(r, 0.0, FLT_MAX, rec)) {
            return 0.5f*vec3(rec.normal.x() + 1.0f, rec.normal.y() + 1.0f, rec.normal.z() + 1.0f);
    }
    else {
            vec3 unit_direction = unit_vector(r.direction());
            float t = 0.5f*(unit_direction.y() + 1.0f);
            return (1.0f - t)*vec3(1.0, 1.0, 1.0) + t * vec3(0.5, 0.7, 1.0);
    }
}
```

```
__global__ void create_world(hitable **d_list, hitable **d_world) {
    if (threadIdx.x == 0 && blockIdx.x == 0) {
            *(d_list) = new sphere(vec3(0, 0, -1), 0.5);
            *(d_list + 1) = new sphere(vec3(0, -100.5, -1), 100);
            *d_world = new hitable_list(d_list, 2);
    }
}


__global__ void free_world(hitable **d_list, hitable **d_world) {
    delete *(d_list);
    delete *(d_list + 1);
    delete *d_world;
}
```

4. Go to __global__ void d_render( ) method

   a) add an additional argument “`hitable **d_world)`” to the method and perform ray casting in themethod:

```
    __global__ void
    d_render(uchar4 *d_output, uint width, uint height, hitable **d_world)
    {
            uint x = blockIdx.x * blockDim.x + threadIdx.x;
            uint y = blockIdx.y * blockDim.y + threadIdx.y;
            uint i = y * width + x;


            float u = x / (float)width;  //----> [0, 1]x[0, 1]
            float v = y / (float)height;

            u = 2.0*u - 1.0; //---> [-1, 1]x[-1, 1]
            v = -(2.0*v - 1.0);
            u *= width / (float)height;

            u *= 2.0;
            v *= 2.0;

            vec3 eye = vec3(0, 0.5, 1.5);
            float distFrEye2Img = 1.0;;
            if ((x < width) && (y < height))
            {
                    //for each pixel
                    vec3 pixelPos = vec3(u, v, eye.z() - distFrEye2Img);
                    //fire a ray:
                    ray r;
                    r.O = eye;

                    r.Dir = pixelPos - eye;    //view direction along negtive z-axis!


            vec3 col = castRay(r, d_world);
            float red = col.x();
            float green = col.y();
            float blue = col.z();
            d_output[i] = make_uchar4(red * 255, green * 255, blue * 255, 0);

    }
}
```

   b) Modify render() method to create some spheres and pass them to the d_render( )

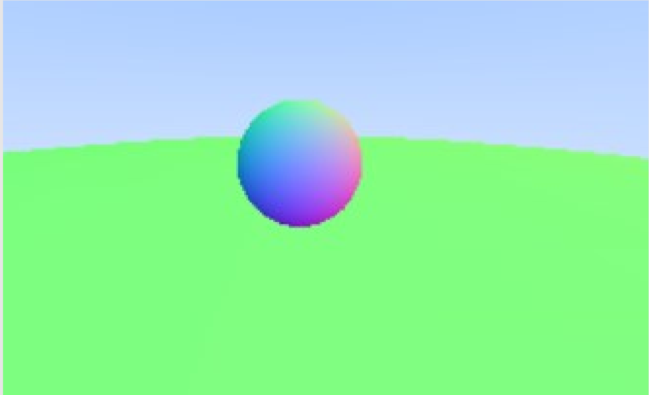
```
    extern "C"
            void render(int width, int height, dim3 blockSize, dim3 gridSize, uchar4 *output)
    {

            // make our world of hitables
            hitable **d_list;
            checkCudaErrors(cudaMalloc((void **)&d_list, 2 * sizeof(hitable *)));
            hitable **d_world;
            checkCudaErrors(cudaMalloc((void **)&d_world, sizeof(hitable *)));
            create_world << <1, 1 >> > (d_list, d_world);
            checkCudaErrors(cudaGetLastError());
```

```
        checkCudaErrors(cudaDeviceSynchronize());


        d_render << <gridSize, blockSize >> > (output, width, height, d_world);
        getLastCudaError("kernel failed");
    }
```

5. Run your program you should see the following image:



6. Edit `create_world( ) to` add 10 more spheres to the scene.