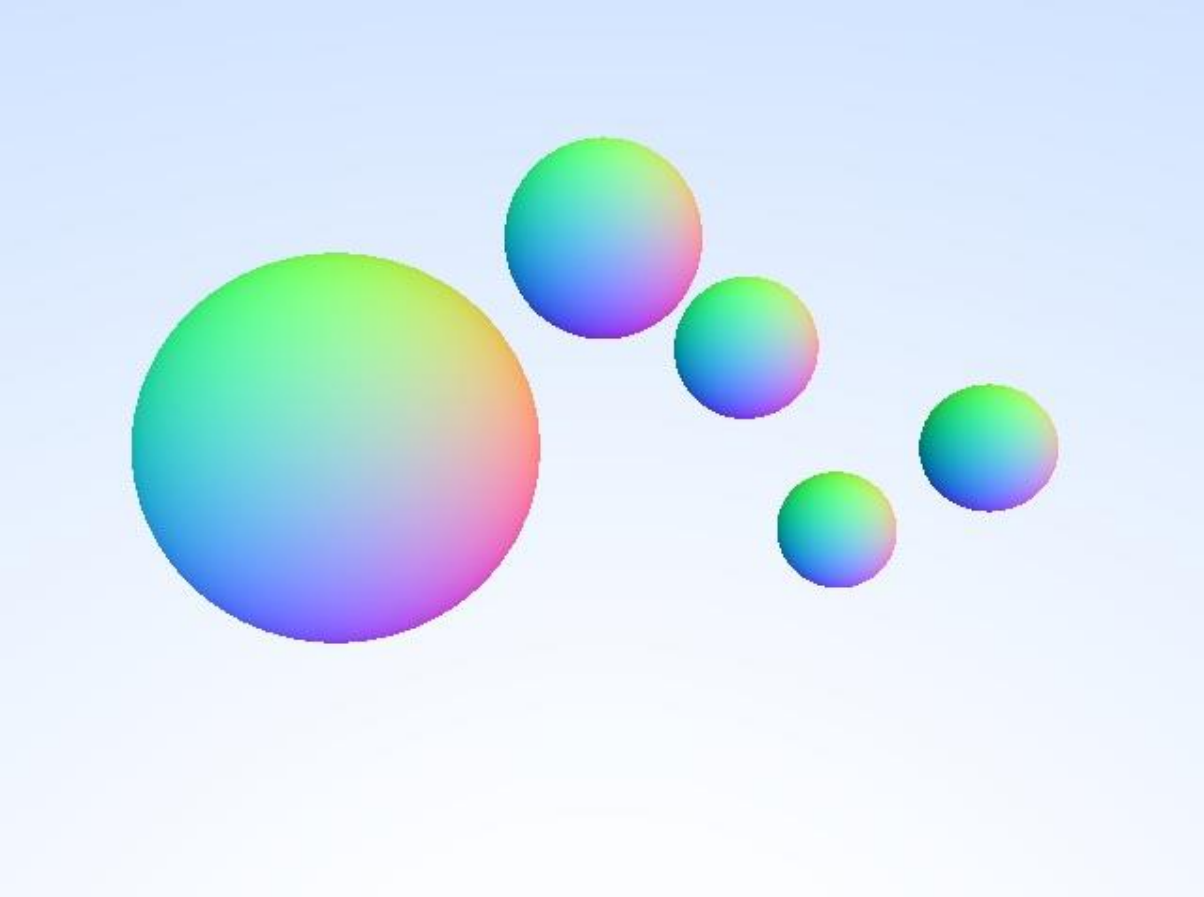
Ece Alptekin
24156

Ray Tracing Project

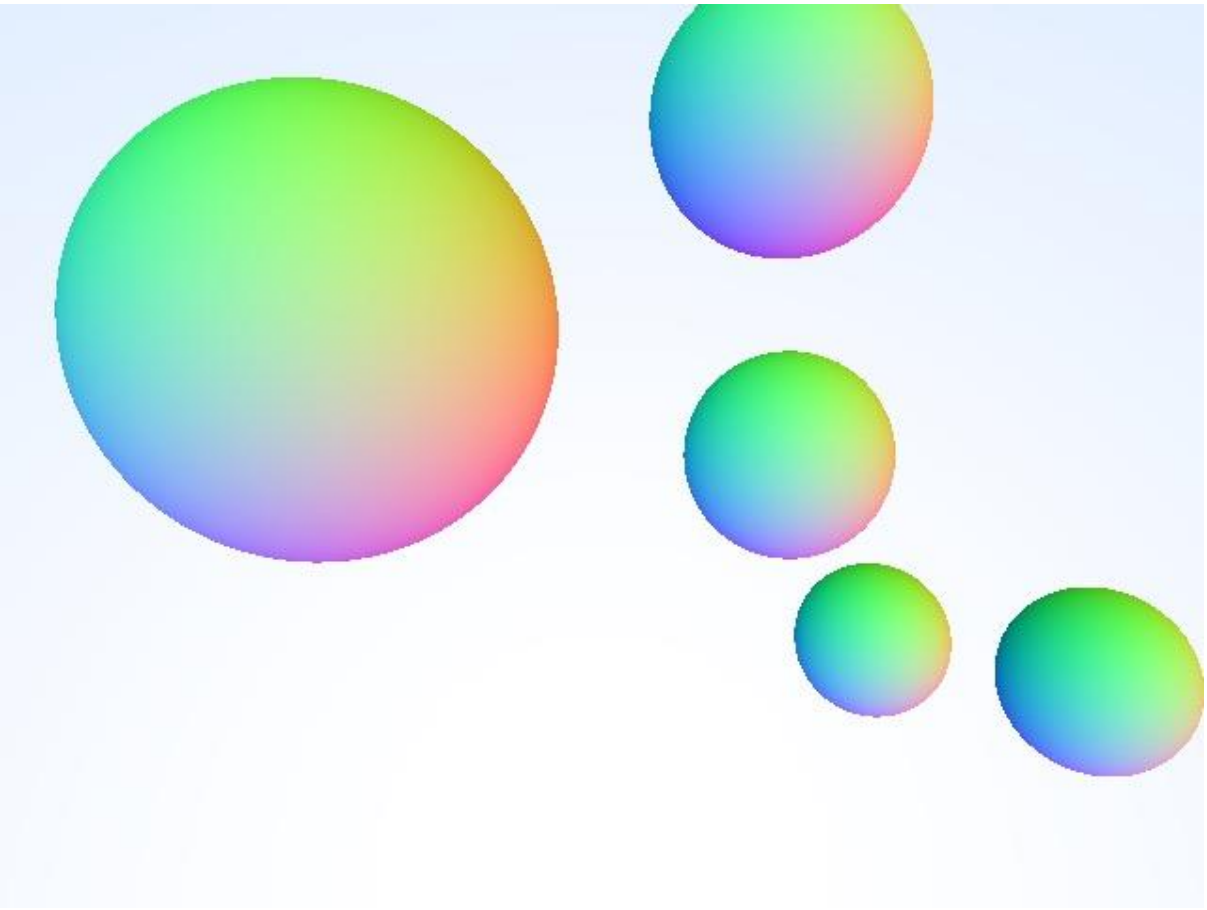
Task 1 Basic Scene



```
//position,target,pixel_buffer
Camera camera(glm::dvec3(0, 5, 5), glm::dvec3(0, 0, 0), pixel_buffer);

hittable_list world;

world.add(make_shared<sphere>(glm::dvec3(-2,0,0), 1.5));
world.add(make_shared<sphere>(glm::dvec3(0,0,-3), 1.0));
world.add(make_shared<sphere>(glm::dvec3(1,1,0), 0.5));
world.add(make_shared<sphere>(glm::dvec3(2,-1,0), 0.5));
world.add(make_shared<sphere>(glm::dvec3(3,0,0), 0.5));
```

Ece Alptekin
24156

The camera's position: (0,5,5)
Sphere's position: (-2,0,0)
Sphere's radius: 1.5
Sphere's position: (0,0,-3)
Sphere's radius: 1.0
Sphere's position: (1,1,0)
Sphere's radius: 0.5
Sphere's position: (2,-1,0)
Sphere's radius: 0.5
Sphere's position: (3,0,0)
Sphere's radius: 0.5



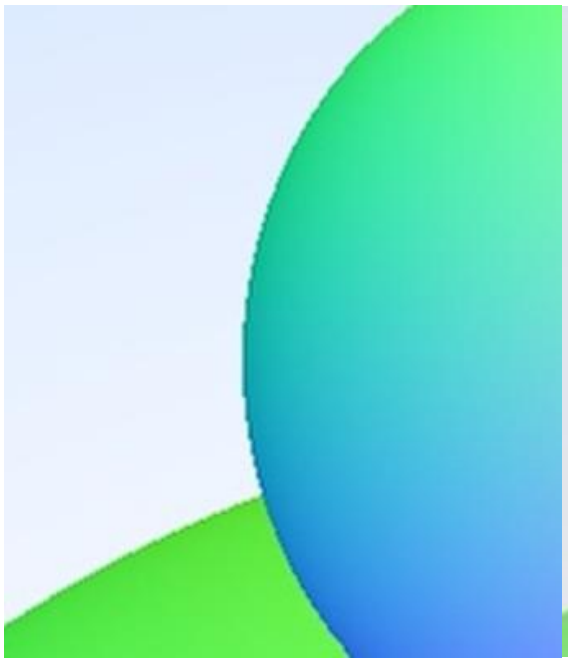
```
//position,target,pixel_buffer
Camera camera(glm::dvec3(1, 5, 2), glm::dvec3(0, 0, 0), pixel_buffer);

hittable_list world;

world.add(make_shared<sphere>(glm::dvec3(-2,0,0), 1.5));
world.add(make_shared<sphere>(glm::dvec3(0,0,-3), 1.0));
world.add(make_shared<sphere>(glm::dvec3(1,1,0), 0.5));
world.add(make_shared<sphere>(glm::dvec3(2,-1,0), 0.5));
world.add(make_shared<sphere>(glm::dvec3(3,0,0), 0.5));
```

Ece Alptekin
24156

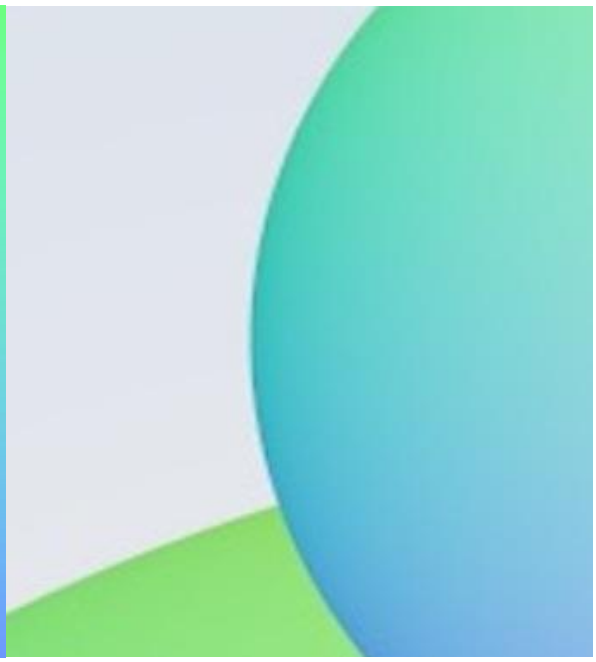
The camera’s position: (1,5,2)
Sphere’s position: (-2,0,0)
Sphere’s radius: 1.5
Sphere’s position: (0,0,-3)
Sphere’s radius: 1.0
Sphere’s position: (1,1,0)
Sphere’s radius: 0.5
Sphere’s position: (2,-1,0)
Sphere’s radius: 0.5
Sphere’s position: (3,0,0)
Sphere’s radius: 0.5

Ece Alptekin
24156

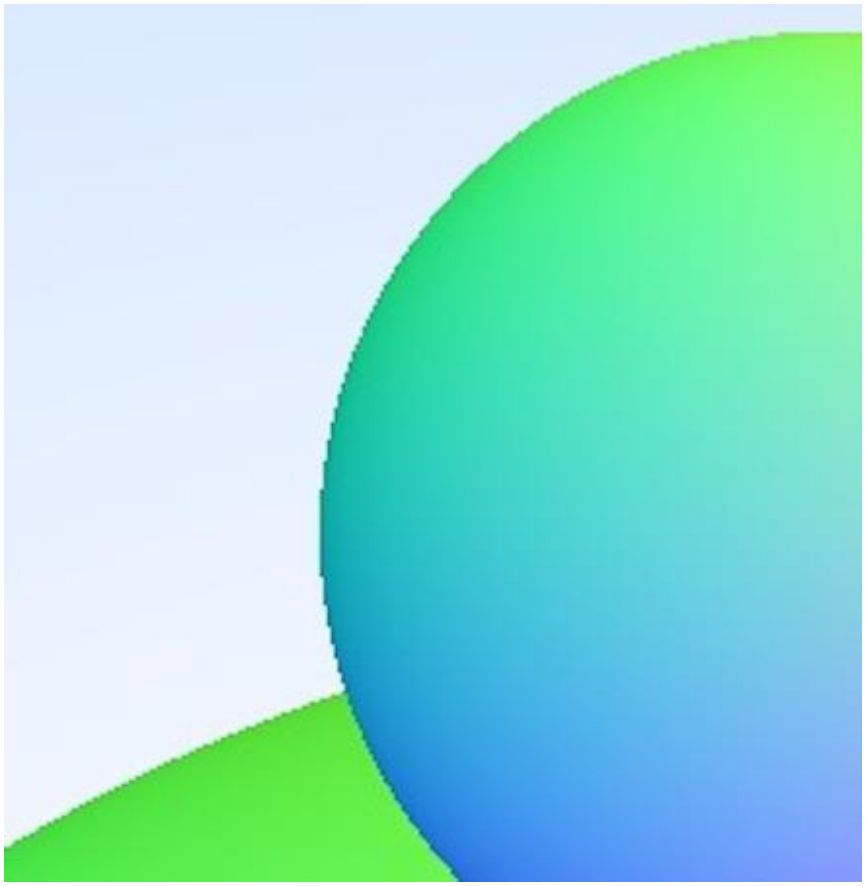
Task 2 Anti-Aliasing



No Anti-Aliasing

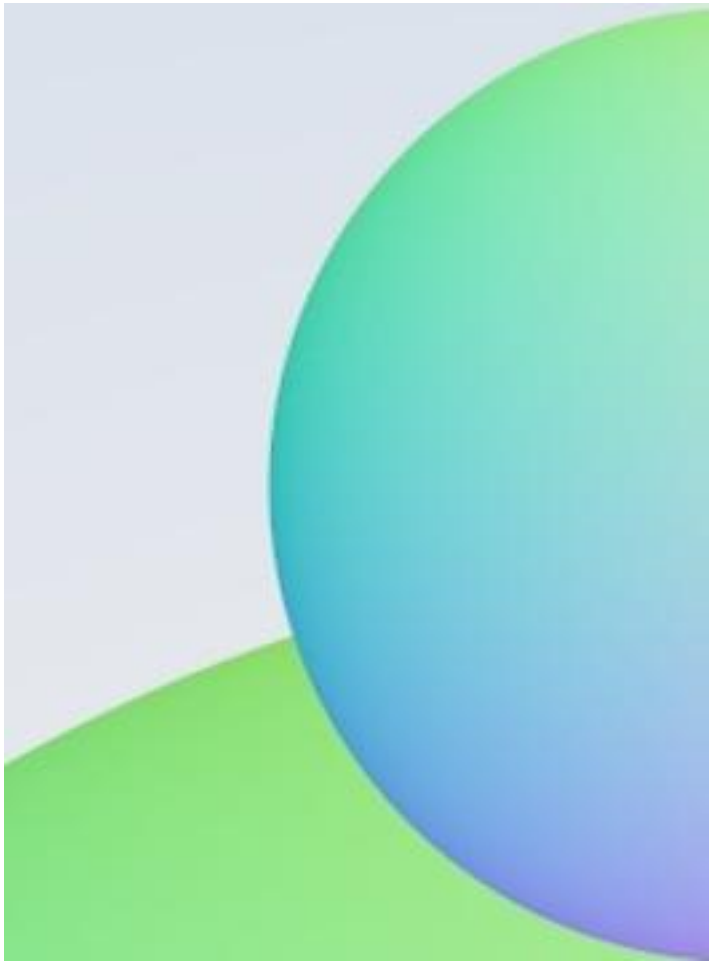


with Anti-Aliasing



No Anti-Aliasing (without supersampling)

Ece Alptekin
24156



with Anti-Aliasing (with supersampling)

I would prefer the render with supersampling. For a given pixel we have several samples within that pixel and send rays through each of the samples. The colors of these rays are then averaged. Anti-aliasing contributes to remove aliasing (jagged and pixelated edges). With supersampling, a line, which is a collection of pixels, does not appear jagged.

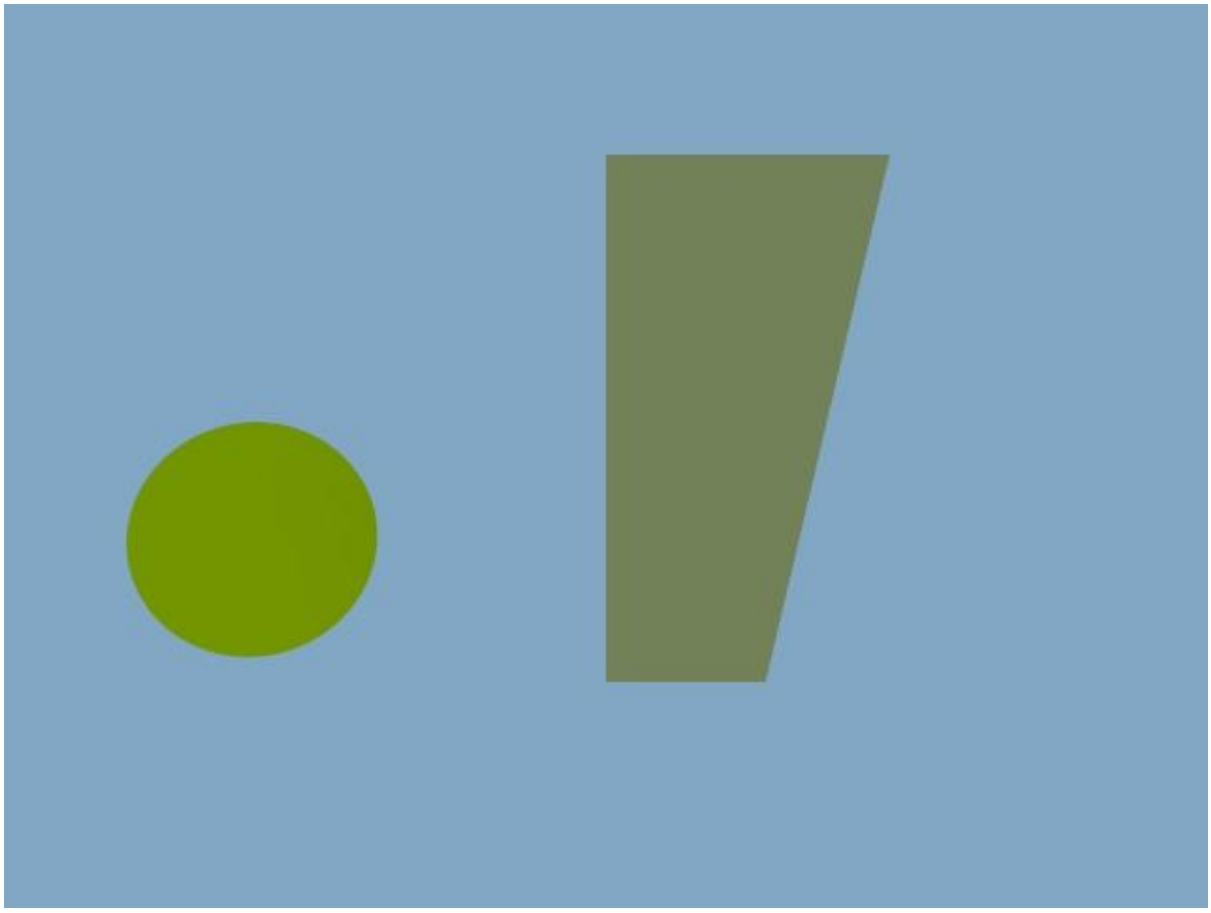
Ece Alptekin
24156

The implementation of supersampling is as follows:

```
for (int y = 0; y < pixel_buffer.dimensions.y; ++y)
{
    for (int x = 0; x < pixel_buffer.dimensions.x; ++x)
    {

        glm::dvec3 color(0,0,0);
        glm::dvec3 color2(0,0,0);

        for (int s = 0; s < samples_per_pixel; ++s)
        {

          auto u = x + random_double();
          auto v = y + random_double();
          glm::dvec2 uv(u, v);
          uv /= pixel_buffer.dimensions; //Normalization

          //origin, direction
          Ray ray(camera.position, glm::normalize(camera.raster_to_world(uv) - camera.position));

          color2 += ray_color(ray, background, world, max_depth);

        }

        color.x = double(color2.x) / samples_per_pixel;
        color.y = double(color2.y) / samples_per_pixel;
        color.z = double(color2.z) / samples_per_pixel;

        //for flipping the image
        //y'nin en ustunden baslar b rengine
        pixel_buffer.set(x, pixel_buffer.dimensions.y - y - 1, color);

    }

}
```

Ece Alptekin
24156

Task 3 More Shapes



The position’s of the shapes are as follows:

```
auto pertext = make_shared<metal>(glm::dvec3(0.8, 0.6, 0.2), 0.5);
//auto mat = make_shared<dielectric>(1.5);
auto material = make_shared<lambertian>(glm::dvec3(0.8, 0.8, 0.0));
auto material1 = make_shared<lambertian>(glm::dvec3(0.6, 0.8, 0.2));
//objects.add(make_shared<sphere>(glm::dvec3(0,-150,-1), 100, material1));
objects.add(make_shared<sphere>(glm::dvec3(-3,-1,0), 1.0, material));
objects.add(make_shared<xy_rect>(0, 2, -5, 2, -1, pertext));
```

Ece Alptekin
24156

The implementation of the rectangle is as follows:

```
bool xy_rect::hit(const Ray& r, double t_min, double t_max, hit_record& rec) const {
    auto t = (k-r.origin.z) / r.direction.z;
    if (t < t_min || t > t_max)
        return false;
    auto x = r.origin.x + t*r.direction.x;
    auto y = r.origin.y + t*r.direction.y;
    if (x < x0 || x > x1 || y < y0 || y > y1)
        return false;
    rec.u = (x-x0)/(x1-x0);
    rec.v = (y-y0)/(y1-y0);
    rec.t = t;
    auto outward_normal = glm::dvec3(0, 0, 1);
    rec.set_face_normal(r, outward_normal);
    rec.mat_ptr = mp;
    rec.p = r.at(t);
    return true;
}
```

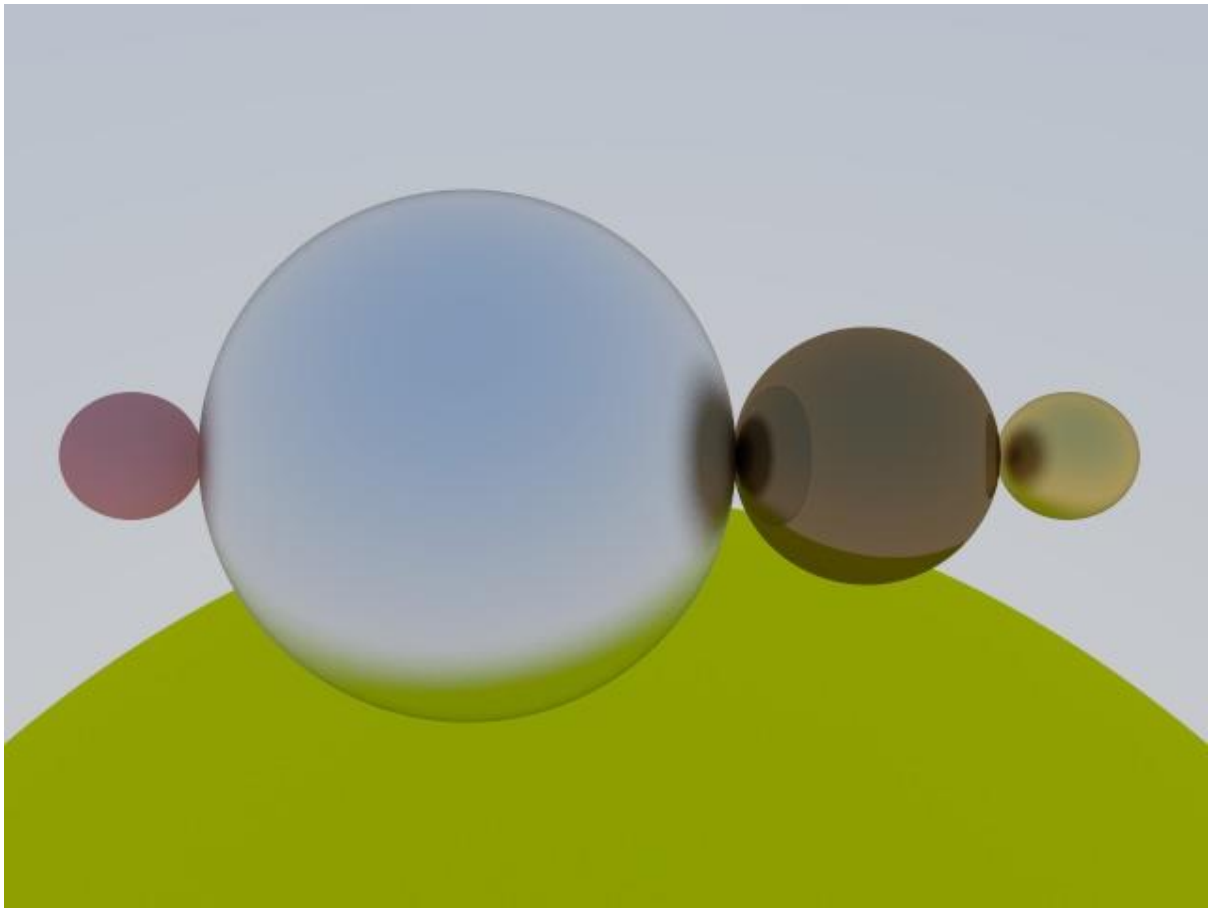
```
class xy_rect : public hittable {
    public:
        xy_rect() {}

        xy_rect(double _x0, double _x1, double _y0, double _y1, double _k,
            shared_ptr<material> mat)
            : x0(_x0), x1(_x1), y0(_y0), y1(_y1), k(_k), mp(mat) {};

        virtual bool hit(const Ray& r, double t_min, double t_max, hit_record& rec) const override;

        virtual bool bounding_box(double time0, double time1, aabb& output_box) const override {
            // The bounding box must have non-zero width in each dimension, so pad the Z
            // dimension a small amount.
            output_box = aabb(glm::dvec3(x0,y0, k-0.0001), glm::dvec3(x1, y1, k+0.0001));
            return true;
        }

    public:
        shared_ptr<material> mp;
        double x0, x1, y0, y1, k;
};
```

Reference:

P. Shirley, "Ray Tracing The Next Week," [Online]. Available: https://raytracing.github.io/books/RayTracingTheNextWeek.html. [Accessed October 2020].

Ece Alptekin
24156

Task 4 Diffuse and Metal Materials



Material properties are as follows:

```
hittable_list world;

auto material_ground = make_shared<lambertian>(glm::dvec3(0.8, 0.8, 0.0));
auto material_center = make_shared<lambertian>(glm::dvec3(0.7, 0.3, 0.3));
auto material_left   = make_shared<metal>(glm::dvec3(0.8, 0.8, 0.8), 0.3);
auto material_right  = make_shared<metal>(glm::dvec3(0.8, 0.6, 0.2), 0.5);
auto material_right2 = make_shared<metal>(glm::dvec3(0.3, 0.2, 0.1), 0.0);

world.add(make_shared<sphere>(glm::dvec3( 0.0, -150.0, -1.0), 100.0, material_ground));
world.add(make_shared<sphere>(glm::dvec3(-3.7,    0.0,  0.0),   0.5, material_center));
world.add(make_shared<sphere>(glm::dvec3(-1.0,    0.0,  0.0),   2.0, material_left));
world.add(make_shared<sphere>(glm::dvec3( 3.6,    0.0,  0.0),   0.5, material_right));
world.add(make_shared<sphere>(glm::dvec3( 2.0,    0.0,  0.0),   1.0, material_right2));
```

Ece Alptekin
24156

Gamma correction:

```
// Divide the color by the number of samples and gamma-correct for gamma=2.0.
auto scale = 1.0 / samples_per_pixel;
r = sqrt(scale * r);
g = sqrt(scale * g);
b = sqrt(scale * b);
```

Shadow acne:

```
if(!world.hit(r, 0.001, infinity, rec)) //shadow acne
{
```

The implemention of the Lambertian Reflectance model to simulate the diffuse materials:

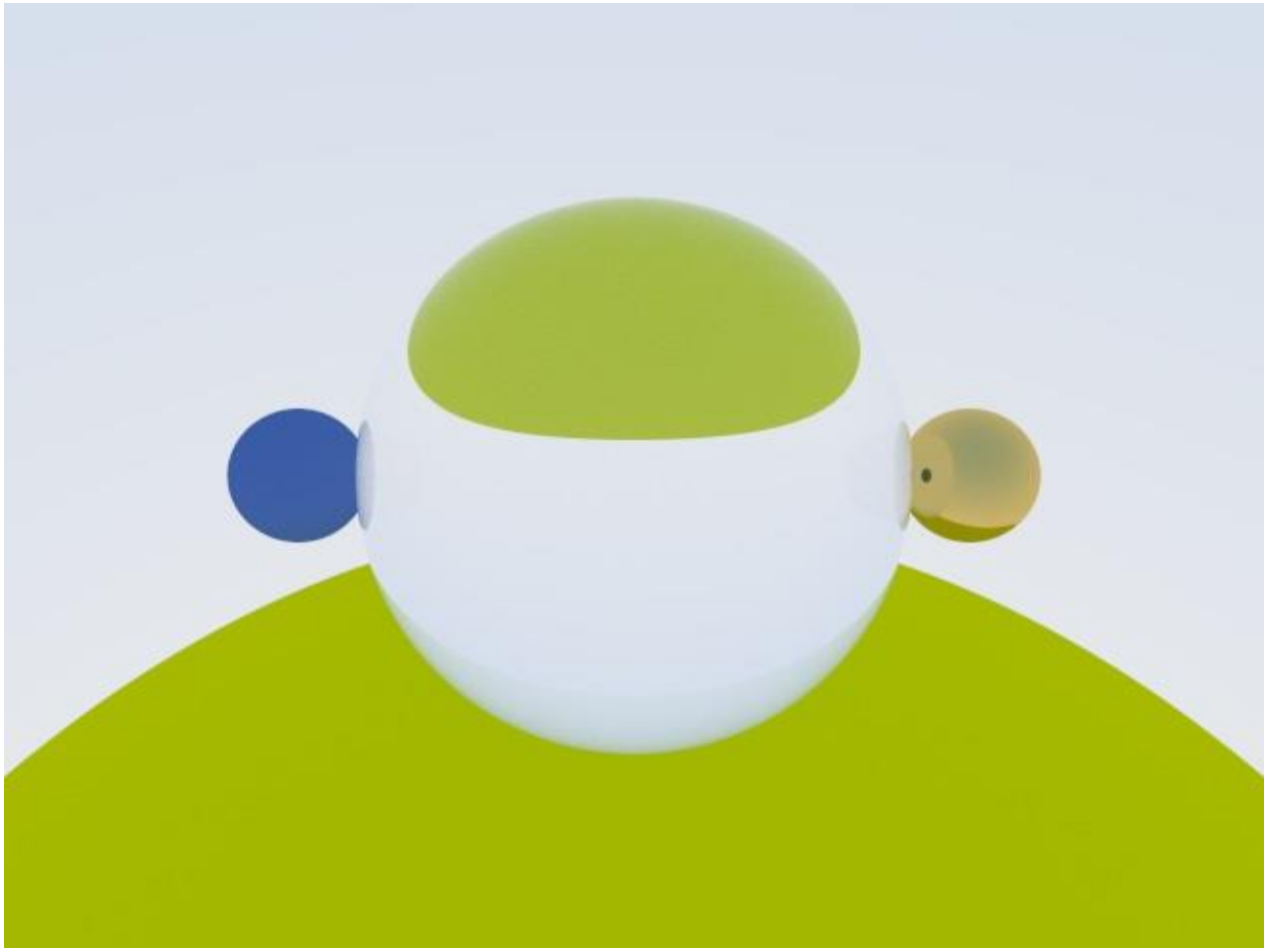
```
glm::dvec3 ray_color(const Ray& r, const hittable& world, int depth)
{
    hit_record rec;

    if (depth <= 0)
    {
        return glm::dvec3(0,0,0);
    }

    if(world.hit(r, 0.001, infinity, rec)) //shadow acne
    {

        //update the ray_color() function to use the new random direction generator
        glm::dvec3 target = rec.p + rec.normal + random_unit_vector();
        return 0.5 * ray_color(Ray(rec.p, target - rec.p), world, depth-1);

    }

    glm::dvec3 unit = glm::normalize(r.direction);
    auto t = 0.5 * (unit.y + 1.0);
    return (1.0-t) * glm::dvec3(1.0, 1.0, 1.0) + t * glm::dvec3(0.0, 0.0, 0.0);
}
```

```
inline double random_double()
{
    // Returns a random real in [0,1).
    return rand() / (RAND_MAX + 1.0);
}

inline double random_double(double min, double max)
{
    // Returns a random real in [min,max).
    return min + (max-min) * random_double();
}

glm::dvec3 random_in_unit_sphere()
{
    while (true) {
        auto p = glm::dvec3(random_double(-1,1), random_double(-1,1), random_double(-1,1));
        if (length_squared(p) >= 1) continue;
        return p;
    }
}

//True Lambertian Reflection
glm::dvec3 random_unit_vector() {
    return glm::normalize(random_in_unit_sphere());
}
```

Ece Alptekin
24156

Task 5 Refraction

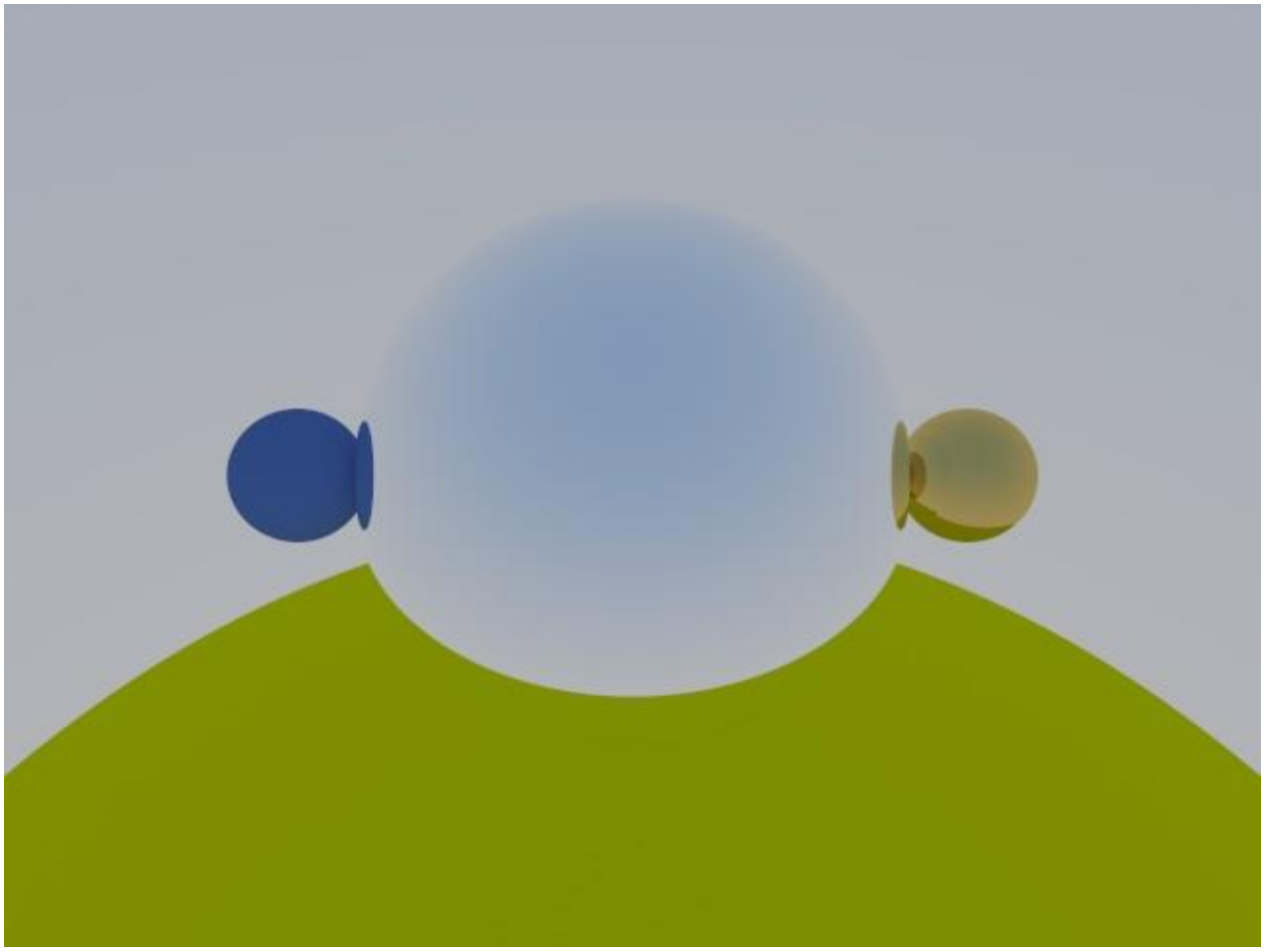
the index of refraction=1.5



```
hittable_list world;

auto material_ground = make_shared<lambertian>(glm::dvec3(0.8, 0.8, 0.0));
auto material_center = make_shared<lambertian>(glm::dvec3(0.1, 0.2, 0.5));
auto material_left   = make_shared<dielectric>(1.5);
auto material_right  = make_shared<metal>(glm::dvec3(0.8, 0.6, 0.2), 0.0);
//auto material_right2 = make_shared<metal>(glm::dvec3(0.3, 0.2, 0.1), 0.5);

world.add(make_shared<sphere>(glm::dvec3( 0.0, -150.0, -1.0), 100.0, material_ground));
world.add(make_shared<sphere>(glm::dvec3(-2.5,    0.0,  0.0),   0.5, material_center));
world.add(make_shared<sphere>(glm::dvec3( 0.0,    0.0,  0.0),   2.0, material_left));
world.add(make_shared<sphere>(glm::dvec3( 2.5,    0.0,  0.0),   0.5, material_right));
//world.add(make_shared<sphere>(glm::dvec3( 2.0,    0.0,  0.0),   1.0, material_right2));
```

Ece Alptekin
24156

the index of refraction=0.0



```
hittable_list world;

auto material_ground = make_shared<lambertian>(glm::dvec3(0.8, 0.8, 0.0));
auto material_center = make_shared<lambertian>(glm::dvec3(0.1, 0.2, 0.5));
auto material_left   = make_shared<dielectric>(0.0);
auto material_right  = make_shared<metal>(glm::dvec3(0.8, 0.6, 0.2), 0.0);
//auto material_right2 = make_shared<metal>(glm::dvec3(0.3, 0.2, 0.1), 0.0);

world.add(make_shared<sphere>(glm::dvec3( 0.0, -150.0, -1.0), 100.0, material_ground));
world.add(make_shared<sphere>(glm::dvec3(-2.5,    0.0,  0.0),   0.5, material_center));
world.add(make_shared<sphere>(glm::dvec3( 0.0,    0.0,  0.0),   2.0, material_left));
world.add(make_shared<sphere>(glm::dvec3( 2.5,    0.0,  0.0),   0.5, material_right));
//world.add(make_shared<sphere>(glm::dvec3( 2.0,    0.0,  0.0),   1.0, material_right2));
```

The first render has the index of refraction=1.5 and the second render has the index of refraction=0.0. The index of refraction describes how fast light travels through the material. Increasing refractive index corresponds to decreasing speed of light in the material. The material with the index of refraction=0.0 has lower density than the material with the index of refraction=1.5, which is a sphere that sometimes refracts.

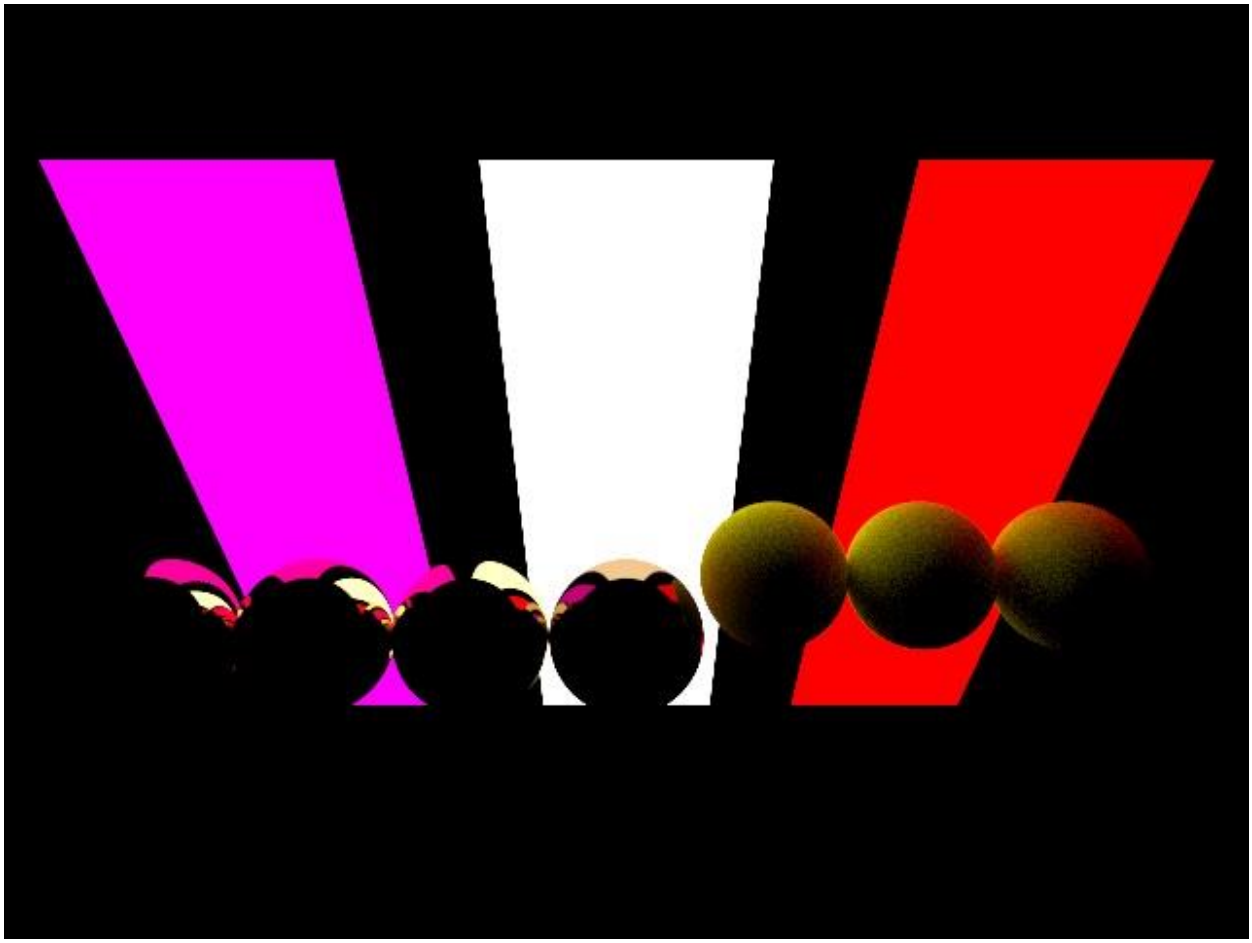
Ece Alptekin
24156

The implementation is as follows;

```
class dielectric : public material {
    public:
        dielectric(double index_of_refraction) : ir(index_of_refraction) {}

        virtual bool scatter(
            const Ray& r_in, const hit_record& rec, glm::dvec3& attenuation, Ray& scattered
        ) const override {
            attenuation = glm::dvec3(1.0, 1.0, 1.0);
            double refraction_ratio = rec.front_face ? (1.0/ir) : ir;

            glm::dvec3 unit_direction = glm::normalize(r_in.direction);

            double cos_theta = fmin(dot(-unit_direction, rec.normal), 1.0);
            double sin_theta = sqrt(1.0 - cos_theta*cos_theta);

            bool cannot_refract = refraction_ratio * sin_theta > 1.0;
            glm::dvec3 direction;

            if (cannot_refract || reflectance(cos_theta, refraction_ratio) > random_double())
            {
              direction = reflect(unit_direction, rec.normal);
            }

            else
            {
              direction = refract1(unit_direction, rec.normal, refraction_ratio);
            }

            scattered = Ray(rec.p, direction);

            return true;
        }

    public:
        double ir; // Index of Refraction

    private:
            static double reflectance(double cosine, double ref_idx)
    {
            // Use Schlick's approximation for reflectance.
            auto r0 = (1-ref_idx) / (1+ref_idx);
            r0 = r0*r0;
            return r0 + (1-r0)*pow((1 - cosine),5);
    }
};
```

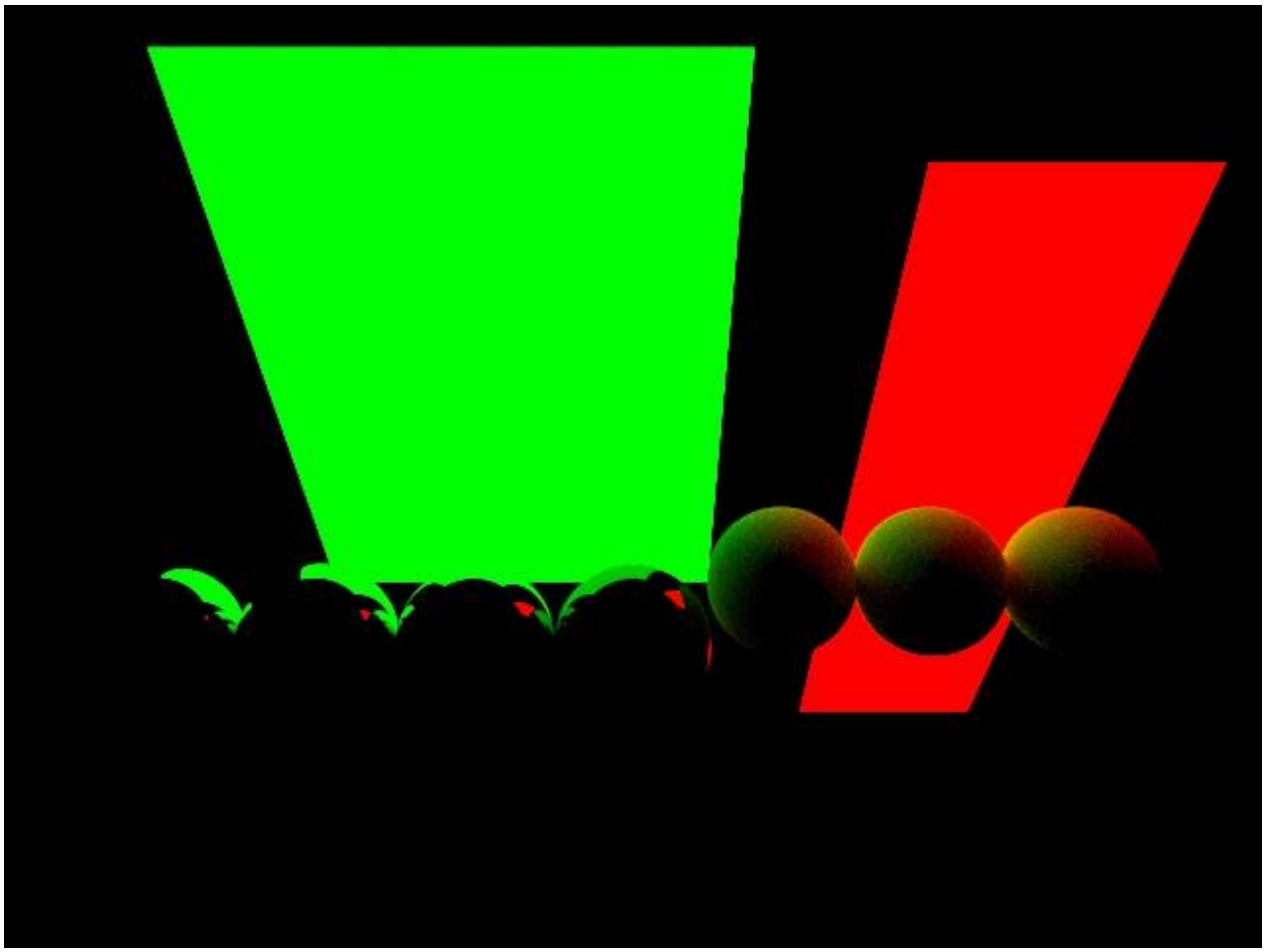
Ece Alptekin
24156

Task 6 Lights



```
auto pertext = make_shared<metal>(glm::dvec3(0.8, 0.6, 0.2), 0.0);
objects.add(make_shared<sphere>(glm::dvec3(-1,0,1.5), 0.5, pertext));
objects.add(make_shared<sphere>(glm::dvec3(-2,0,1.5), 0.5, pertext));
objects.add(make_shared<sphere>(glm::dvec3(-3,0,1.5), 0.5, pertext));

auto mat = make_shared<metal>(glm::dvec3(0.3, 0.2, 0.1), 0.0);
objects.add(make_shared<sphere>(glm::dvec3(0,0,1.5), 0.5, mat));

auto material = make_shared<lambertian>(glm::dvec3(0.8, 0.8, 0.0));
objects.add(make_shared<sphere>(glm::dvec3(1,0,1), 0.5, material));
objects.add(make_shared<sphere>(glm::dvec3(2,0,1), 0.5, material));
objects.add(make_shared<sphere>(glm::dvec3(3,0,1), 0.5, material));

auto difflight = make_shared<diffuse_light>(glm::dvec3(4,0,4));
auto difflight1 = make_shared<diffuse_light>(glm::dvec3(6,6,6));
auto difflight2 = make_shared<diffuse_light>(glm::dvec3(5,0,0));
objects.add(make_shared<xy_rect>(-4, -2, -5, 2, -1, difflight));
objects.add(make_shared<xy_rect>(-1, 1, -5, 2, -1, difflight1));
objects.add(make_shared<xy_rect>(2, 4, -5, 2, -1, difflight2));
```

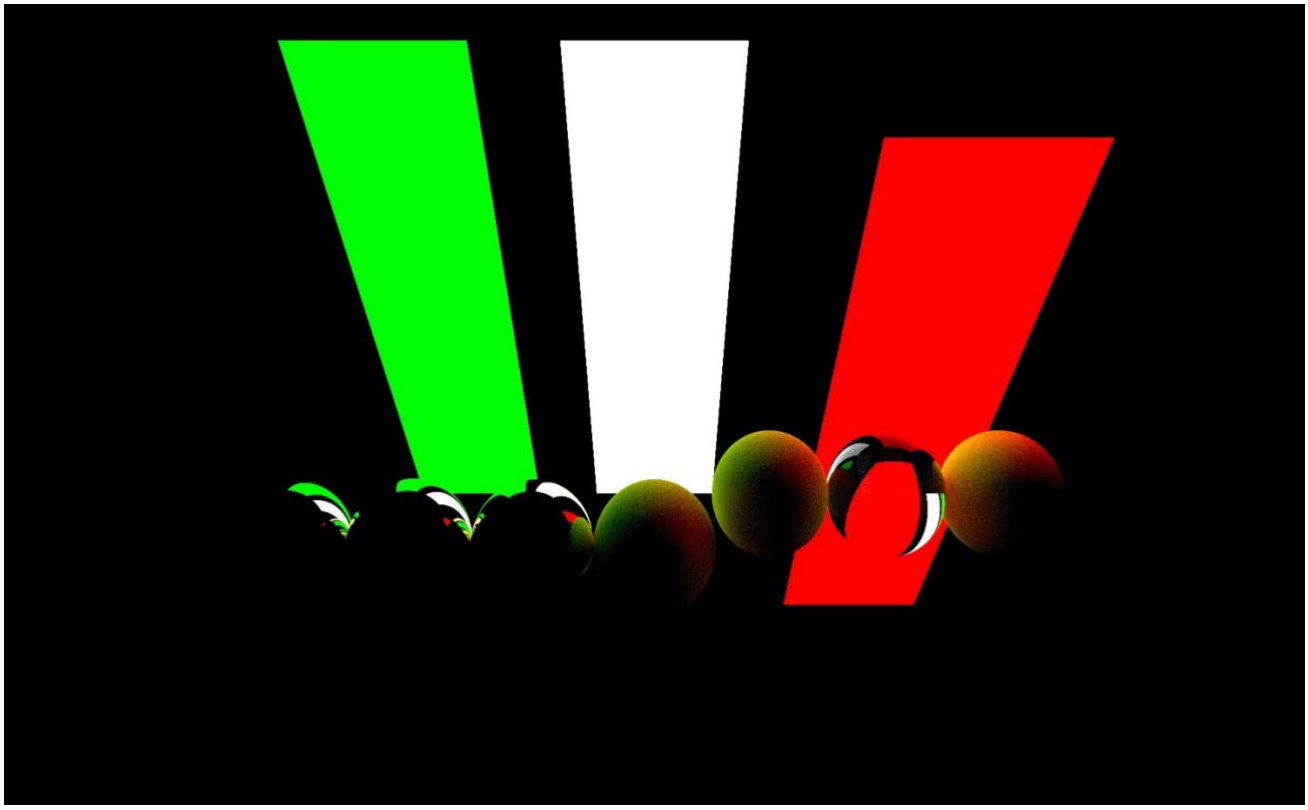
Ece Alptekin
24156



```
auto pertext = make_shared<metal>(glm::dvec3(0.8, 0.6, 0.2), 0.0);
objects.add(make_shared<sphere>(glm::dvec3(-1,0,1.5), 0.5, pertext));
objects.add(make_shared<sphere>(glm::dvec3(-2,0,1.5), 0.5, pertext));
objects.add(make_shared<sphere>(glm::dvec3(-3,0,1.5), 0.5, pertext));

auto mat = make_shared<metal>(glm::dvec3(0.3, 0.2, 0.1), 0.0);
objects.add(make_shared<sphere>(glm::dvec3(0,0,1.5), 0.5, mat));

auto material = make_shared<lambertian>(glm::dvec3(0.8, 0.8, 0.0));
objects.add(make_shared<sphere>(glm::dvec3(1,0,1), 0.5, material));
objects.add(make_shared<sphere>(glm::dvec3(2,0,1), 0.5, material));
objects.add(make_shared<sphere>(glm::dvec3(3,0,1), 0.5, material));

auto difflight = make_shared<diffuse_light>(glm::dvec3(0,6,0));
auto difflight1 = make_shared<diffuse_light>(glm::dvec3(0,6,0));
auto difflight2 = make_shared<diffuse_light>(glm::dvec3(8,0,0));
objects.add(make_shared<xy_rect>(-4, -2, -5, 2, -3, difflight));
objects.add(make_shared<xy_rect>(-2, 1, -5, 2, -3, difflight1));
objects.add(make_shared<xy_rect>(2, 4, -5, 2, -1, difflight2));
```

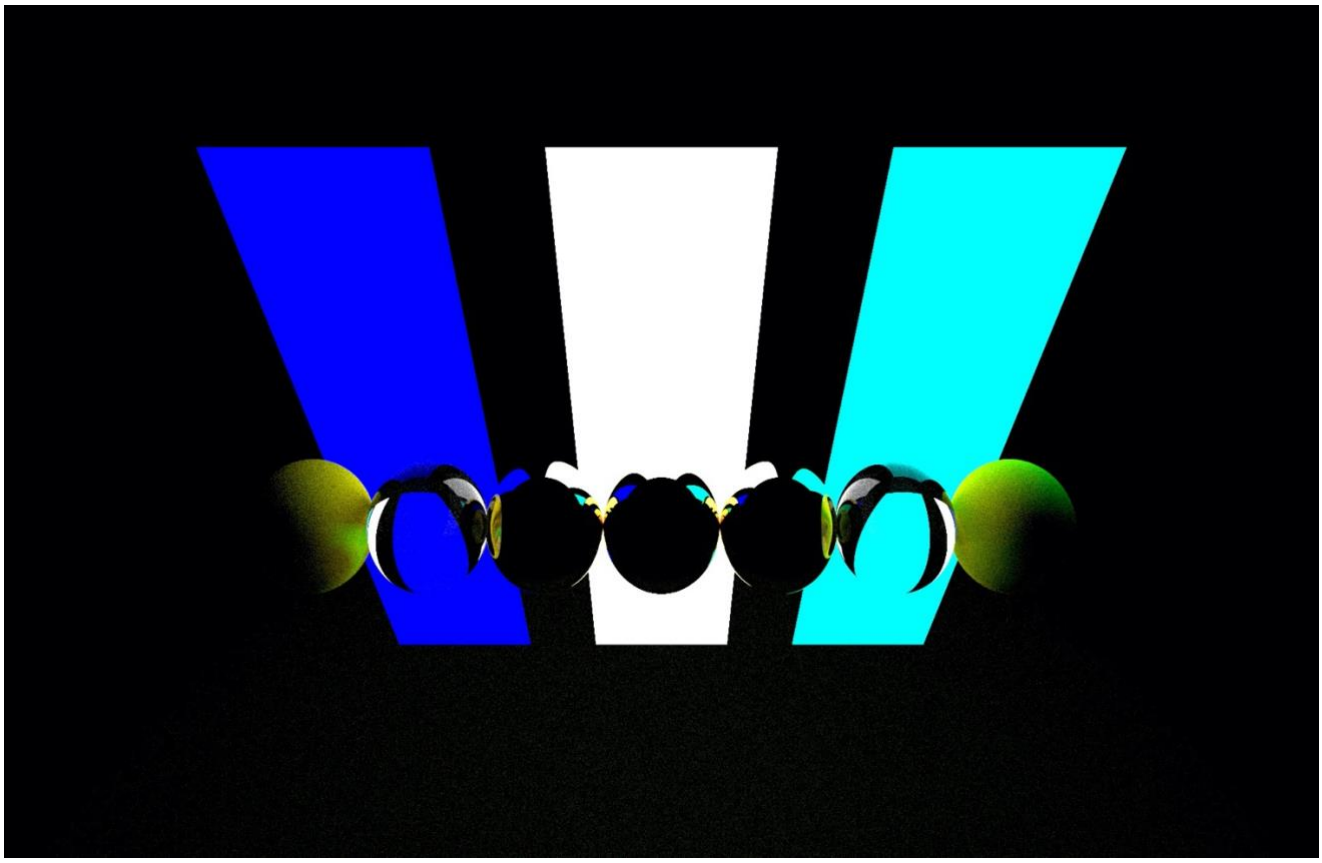
Ece Alptekin
24156

Task 7 Let's get creative!

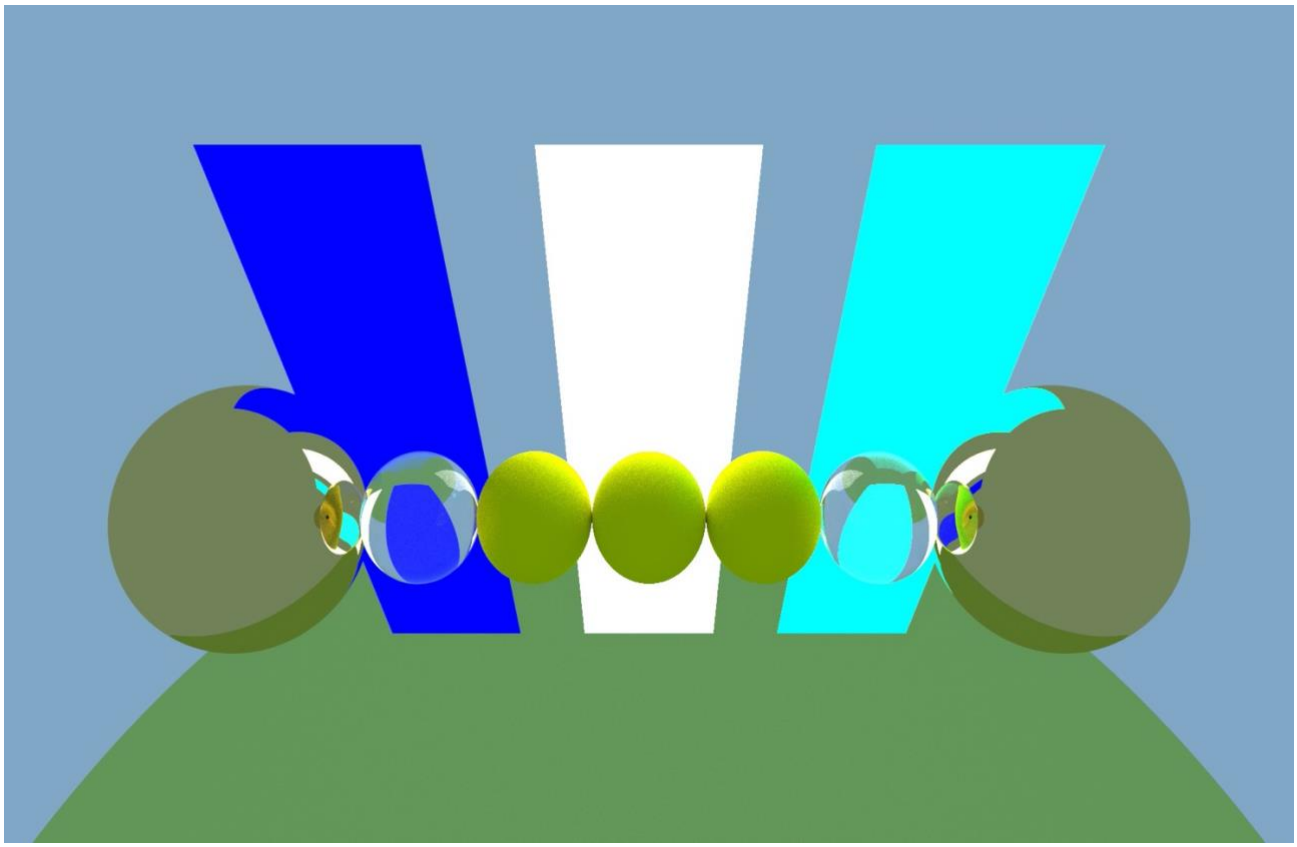


```
auto pertext = make_shared<metal>(glm::dvec3(0.8, 0.6, 0.2), 0.0);
auto mat = make_shared<dielectric>(1.5);
auto material = make_shared<lambertian>(glm::dvec3(0.8, 0.8, 0.0));

objects.add(make_shared<sphere>(glm::dvec3(-1,0,1), 0.5, pertext));
objects.add(make_shared<sphere>(glm::dvec3(-2,0,1), 0.5, mat));
objects.add(make_shared<sphere>(glm::dvec3(-3,0,1), 0.5, material));

objects.add(make_shared<sphere>(glm::dvec3(0,0,1), 1.0, pertext));

objects.add(make_shared<sphere>(glm::dvec3(1,0,1), 0.5, pertext));
objects.add(make_shared<sphere>(glm::dvec3(2,0,1), 0.5, mat));
objects.add(make_shared<sphere>(glm::dvec3(3,0,1), 0.5, material));

auto difflight = make_shared<diffuse_light>(glm::dvec3(0,0,6));  //sol
auto difflight1 = make_shared<diffuse_light>(glm::dvec3(6,6,6)); //orta
auto difflight2 = make_shared<diffuse_light>(glm::dvec3(0,6,6)); //sağ
objects.add(make_shared<xy_rect>(-4, -2, -5, 2, -1, difflight));
objects.add(make_shared<xy_rect>(-1, 1, -5, 2, -1, difflight1));
objects.add(make_shared<xy_rect>(2, 4, -5, 2, -1, difflight2));
```

Ece Alptekin
24156



```
auto pertext = make_shared<metal>(glm::dvec3(0.8, 0.6, 0.2), 0.0);
auto mat = make_shared<dielectric>(1.5);
auto material = make_shared<lambertian>(glm::dvec3(0.8, 0.8, 0.0));

objects.add(make_shared<sphere>(glm::dvec3(-1,0,1), 0.5, pertext));
objects.add(make_shared<sphere>(glm::dvec3(-2,0,1), 0.5, mat));
objects.add(make_shared<sphere>(glm::dvec3(-3,0,1), 0.5, material));

objects.add(make_shared<sphere>(glm::dvec3(0,-150,-1), 100, material));

objects.add(make_shared<sphere>(glm::dvec3(0,0,1), 0.5, pertext));

objects.add(make_shared<sphere>(glm::dvec3(1,0,1), 0.5, pertext));
objects.add(make_shared<sphere>(glm::dvec3(2,0,1), 0.5, mat));
objects.add(make_shared<sphere>(glm::dvec3(3,0,1), 0.5, material));

auto difflight = make_shared<diffuse_light>(glm::dvec3(0,0,6));  //sol
auto difflight1 = make_shared<diffuse_light>(glm::dvec3(8,8,8)); //orta
auto difflight2 = make_shared<diffuse_light>(glm::dvec3(0,6,6)); //sağ
objects.add(make_shared<xy_rect>(-4, -2, -5, 2, -1, difflight));
objects.add(make_shared<xy_rect>(-1, 1, -5, 2, -1, difflight1));
objects.add(make_shared<xy_rect>(2, 4, -5, 2, -1, difflight2));
```



```
auto pertext = make_shared<metal>(glm::dvec3(0.8, 0.6, 0.2), 0.0);
auto mat = make_shared<dielectric>(1.5);
auto material = make_shared<lambertian>(glm::dvec3(0.8, 0.8, 0.0));

objects.add(make_shared<sphere>(glm::dvec3(-1,0,1), 0.5, material));
objects.add(make_shared<sphere>(glm::dvec3(-2,0,1), 0.5, mat));
objects.add(make_shared<sphere>(glm::dvec3(-3.5,0,1), 1.0, pertext));

auto material1 = make_shared<lambertian>(glm::dvec3(0.6, 0.8, 0.2));
objects.add(make_shared<sphere>(glm::dvec3(0,-150,-1), 100, material1));

objects.add(make_shared<sphere>(glm::dvec3(0,0,1), 0.5, material));

objects.add(make_shared<sphere>(glm::dvec3(1,0,1), 0.5, material));
objects.add(make_shared<sphere>(glm::dvec3(2,0,1), 0.5, mat));
objects.add(make_shared<sphere>(glm::dvec3(3.5,0,1), 1.0, pertext));

auto difflight = make_shared<diffuse_light>(glm::dvec3(0,0,6));  //sol
auto difflight1 = make_shared<diffuse_light>(glm::dvec3(8,8,8)); //orta
auto difflight2 = make_shared<diffuse_light>(glm::dvec3(0,6,6)); //sağ
objects.add(make_shared<xy_rect>(-4, -2, -5, 2, -1, difflight));
objects.add(make_shared<xy_rect>(-1, 1, -5, 2, -1, difflight1));
objects.add(make_shared<xy_rect>(2, 4, -5, 2, -1, difflight2));
```

Ece Alptekin
24156

Task 8 Questions

The approximate time complexity of ray tracing on models with triangles.

The asymptotic time complexity of ray tracing is O(b.p). (b= # of objects in scene, p= # of pixels on screen). The asymptotic time complexity of accelerated ray tracing is O(logb.p).

What is the difference between preprocessing and computing the image? Why?

The purpose of the computing the image is to increase the aesthetic of the image. It fits an image to the human visual system. Computing the image consists of set of algorithms. Image preprocessing aims to improve the image data by suppressing unwanted distortions. (the images do not fit those criteria, the geometry can be distorted, the lighting can be irregular, too strong or insufficient, there can be noise in the images). Image preprocessing prepares images to be used in algorithms. An image, which is coming from a camera or a sensor, goes through the image preprocessing. The output of this process is an input for the image computing. The output of the image computing is an image to be viewed by a human eye.

What are the critical parameters for the ray tracer algorithm's performance?

Critical parameters are number of objects, number of lights, shading, texture, width and height. The properties of the material can have a huge impact on the performance. For instance, reflectivity can change the ray depth of the scene. Additionally, the performance can be changed by total number of rays, resolution and sampling rate.

Imagine you are a systems engineer at Pixar. There is a new super-resolution in 6000×6000 pixels and you have to estimate the maximum rendering time per frame. Assume that the scenes are static, so you are not going to spend any CPU on animation, scene hierarchy, etc. The average scene has 500 objects with a total of 5.000.000 triangles to check intersection with.

The expect time to be:

Xresolution * Yresolution * (number of objects + log(triangles))

6000 * 6000 * (500 + log(5.000.000))