Assignment 3

I implemented perspective camera, specular highlights, shadow, reflections, refractions and depth of field in this homework.

Ex.1

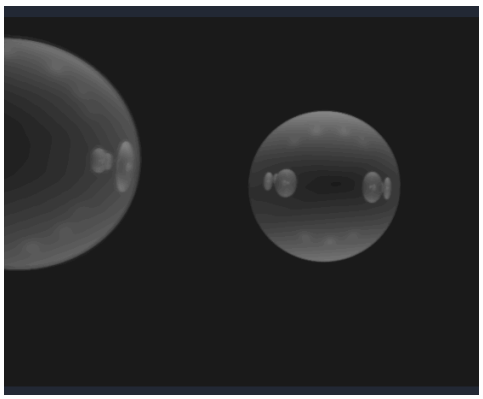
I stretch the pixel in this way, so no matter how we change the focal_length, the field_of_view is fixed and we can see the exactly same scene(don't consider the different caused by depth of field). And scale_x is proportionable with scale_y.

```
// The camera always points in the direction -z
// The sensor grid is at a distance 'focal_length' from the camera center,
// and covers an viewing angle given by 'field_of_view'.
double aspect_ratio = double(w) / double(h);
double scale_y = scene.camera.focal_length * tan(scene.camera.field_of_view / 2);
double scale_x = scale_y * aspect_ratio; //
```

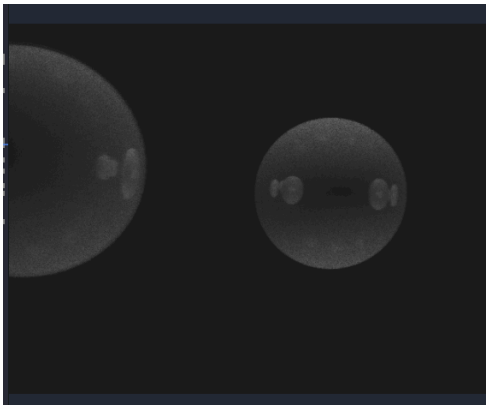
Ex.2

I implemented shadow rays, and set offset so the shadow ray won't be blocked by itself.

```
// TODO: Shoot a shadow ray to determine if the light should affect the intersection point
Ray shadow_ray(hit.position + epsilon * Li, Li); // set some offset so won't intersect with itself
if (!is_light_visible(scene, shadow_ray, light)) continue;
```



If there is no offset, then many pixels may be dark. Just like following:



Ex.3

Implemented refract function to compute the refraction ray direction.

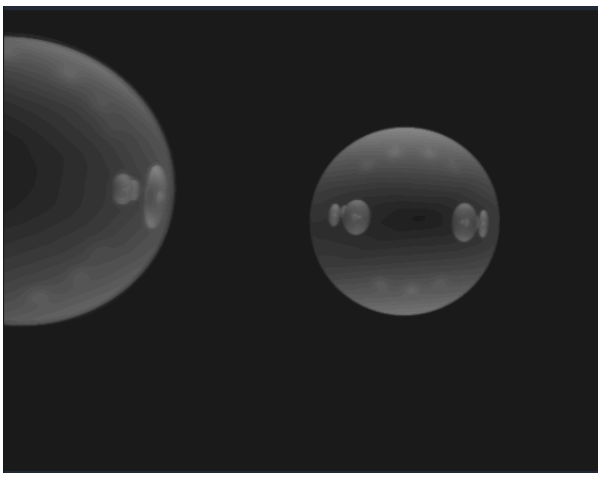
```
bool refract(const Vector3d &d, const Vector3d &N, double n0, double n1, Vector3d &t) {
    double cos_fi_square = 1 - (n0 * n0 * (1 - d.dot(N) * d.dot(N)))/(n1 * n1);
    if (cos_fi_square < 0) return false;
    t = (n0 * (d - N * d.dot(N)) / n1 - N * sqrt(cos_fi_square)).normalized();
    return true;
}
```

Using recursion to compute the reflection color and refraction color.

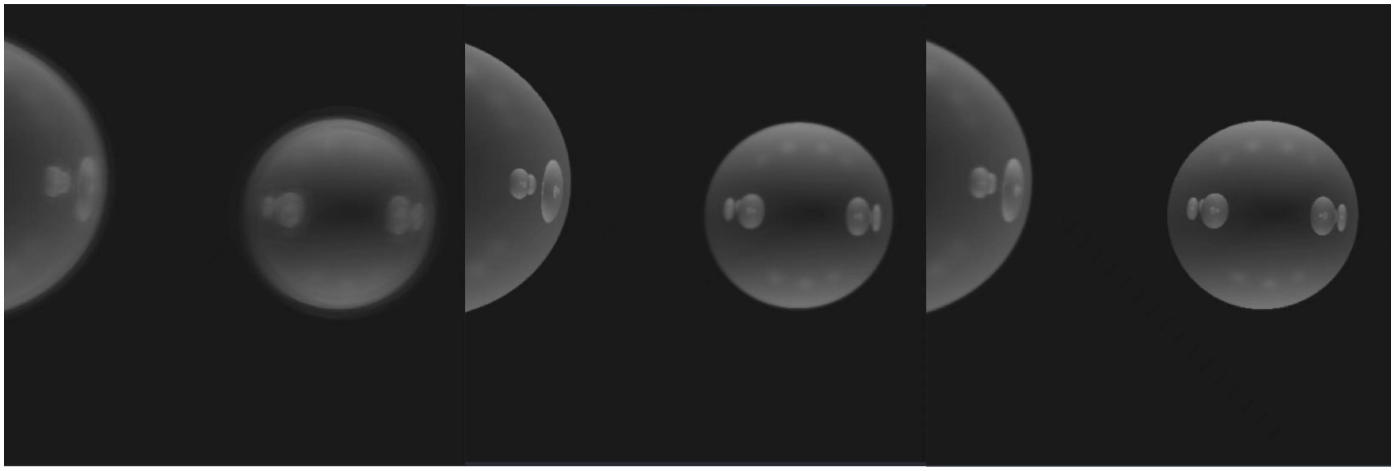
```
// TODO: Compute the color of the reflected ray and add its contribution to the current point color.
Vector3d reflection_color(0, 0, 0);
//if km is not zero(black)
if (mat.reflection_color.norm() > 0 && max_bounce > 0) {
    Vector3d r = ray.direction - 2 * ray.direction.dot(hit.normal) * hit.normal;
    Ray reflection_ray(hit.position + epsilon * r, r);
    reflection_color += mat.reflection_color.cwiseProduct(shoot_ray(scene, reflection_ray, max_bounce - 1));
}

// TODO: Compute the color of the refracted ray and add its contribution to the current point color.
//       Make sure to check for total internal reflection before shooting a new ray.
Vector3d refraction_color(0, 0, 0);
Vector3d t;
if (mat.refraction_color.norm() > 0 && max_bounce > 0) {
    if (ray.direction.dot(hit.normal) < 0) { // air to object
        if (refract(ray.direction, hit.normal, 1.0, mat.refraction_index, t)) {
            Ray refraction_ray(hit.position + epsilon * t, t);
            refraction_color += mat.refraction_color.cwiseProduct(shoot_ray(scene, refraction_ray, max_bounce - 1));
        }
    }
    else { // object to air
        if (refract(ray.direction, hit.normal, mat.refraction_index, 1.0, t)) {
            Ray refraction_ray(hit.position + epsilon * t, t);
            refraction_color += mat.refraction_color.cwiseProduct(shoot_ray(scene, refraction_ray, max_bounce - 1));
        }
    }
}
```

Ex.4

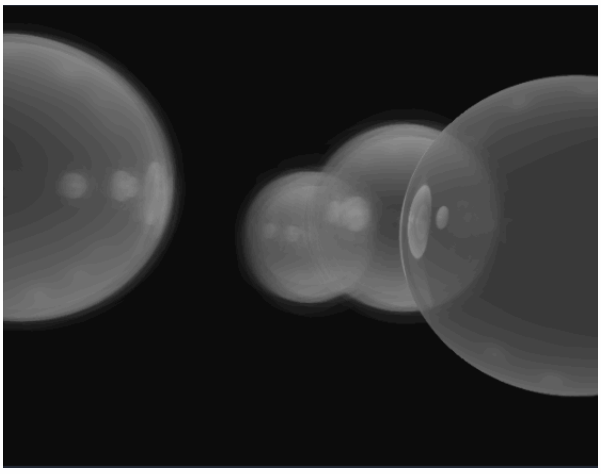
Randomly sampled ten rays from camera, and averaged their result to get pixel color. What I get is identical with the professor's sample.



If I change the focal length from 2.0 to 6.0, the rightmost sphere is much more sharp.



There is a scene in wich more object are pictured. The rightmost sphere is much more clear than others. And refraction is working.



Ex.5

I implemented animation, you can check the gif in my submission.

```
int main(int argc, char *argv[]) {
    if (argc < 2) {
        std::cerr << "Usage: " << argv[0] << " scene.json" << std::endl;
        return 1;
    }
    Scene scene = load_scene(argv[1]);
    render_scene(scene); // simple ray tracing one picture
    // animation(scene); // sample an animation gif
    return 0;
}
```

Uncomment the animation function, you can reproduce the result. It takes some time because there are recursions in both reflection and refraction. Or you can set the refraction color to zero so it will be much faster but without refraction pattern.