심화전공실습

HW #11

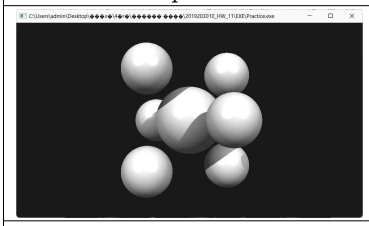
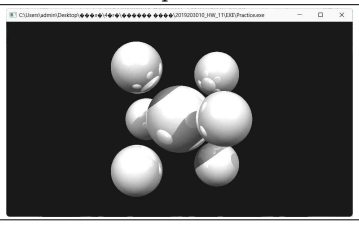
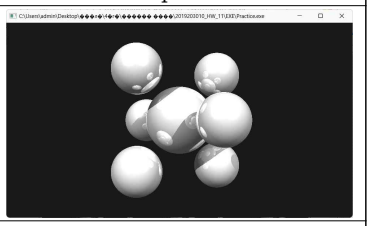
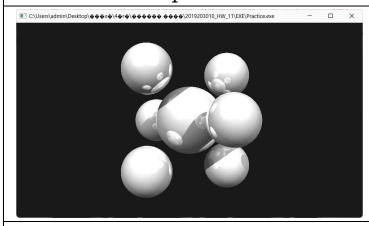
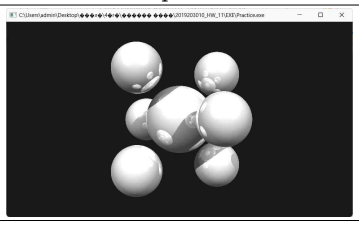
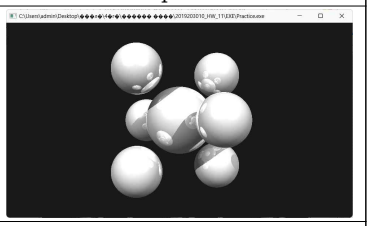
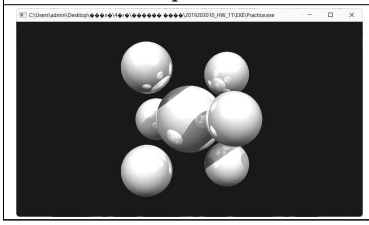
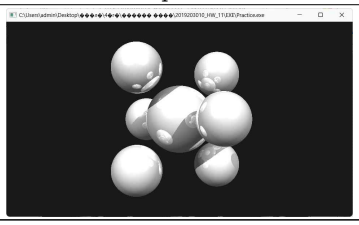
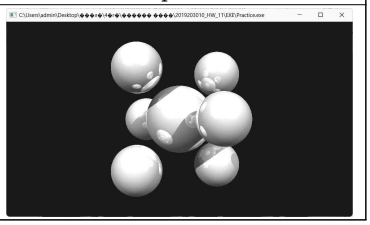
Self-scoring table

	P1	P2	P3	P4	Total
Score	1	1	1	1	4

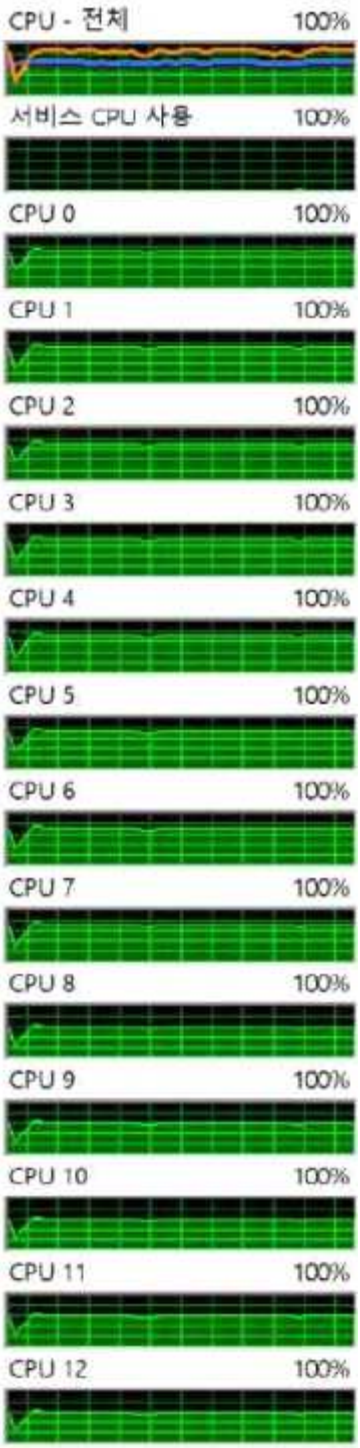
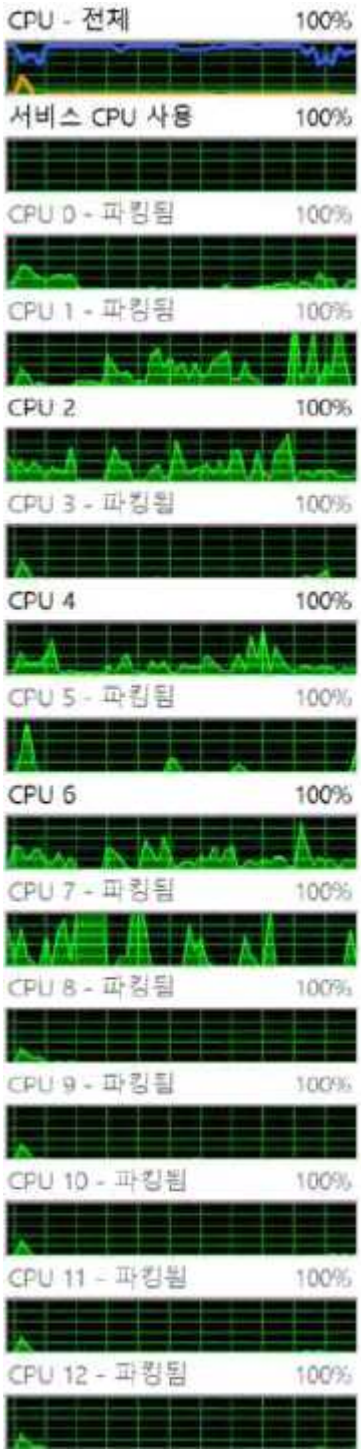
과목명	심화전공실습
학부	소프트웨어학부
학번	2019203010
이름	김민철
제출일자	2024년 11월 17일

I. Practive

Practice 01. Ray tracing of spheres with various ray tracing depth

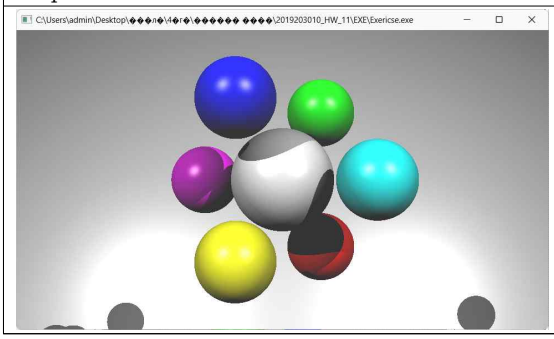
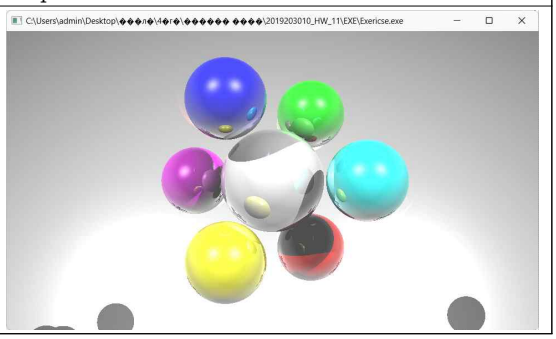
Depth 1	Depth 2	Depth 3
		
Depth 4	Depth 5	Depth 6
		
Depth 7	Depth 8	Depth 9
		

Practice 02. Parrel computing using OpenMP (show CPU usage

Parallel	Non-Parallel
	

II. Exercise

Exercise 01. Enclose the scene with a sphere

Depth 1	Depth 2
	

```
if (t0 > 0 && t1 > 0)
    n = normalize(p - center); // Normal at the point
else n = normalize(center - p);// enclosing sphere to center

return t;
```

findIntersection 함수의 목표:

ray와 sphere의 교점 p 찾기:주어진 광선(ray)과 구(sphere)가 만나는 점 p를 계산한다.

법선 벡터 n 계산 : 교점 p에서 구의 표면에 수직인 법선 벡터 n을 구한다.

Practice와 Exercise의 비교:

Practice:

ray가 sphere 외부에 존재할 수 있다.

두 교점 p0, p1이 모두 sphere 위에 있을 수 있다.

Exercise:

ray가 가장 큰 sphere 안쪽에 존재한다고 가정한다.

큰 sphere의 경우, 교점이 구의 바깥에 있으면 t0 또는 t1 값이 음수나 1보다 커진다.

법선 벡터 n의 방향:

일반적인 경우 (작은 sphere):n의 시작점은 중심(center), 끝점은 교점 p이다.

큰 sphere:

t0, t1이 모두 양수일 때: n의 시작점은 중심, 끝점은 교점 p이다.

t0 또는 t1이 음수나 1보다 클 때: n의 시작점은 교점 p, 끝점은 중심이다. (다른 구와 방향이 반대)

```
// Find the closest intersection with the spheres along the ray except E
int
findIntersection(const Ray& ray, vec3& p, vec3& n, int E)
{
    // Find the closest intersection within [ray.p0, ray.p1]
    int iSphere = 7;
    float T = 1.0; // The camera faces the negative z-axis.
    for (int i = 0; i < nSpheres; i++)
    {
        if (i == E) continue;

        vec3 center = vec3(viewModel * vec4(center_world[i], 1));
        vec3 p_i, n_i;
        float t;

        t = findIntersection(ray, center, radius[i], p_i, n_i);

        if (t < 0) continue;
        if (t ≤ T) { iSphere = i; T = t; p = p_i; n = n_i; }
    }
    return iSphere;
}
```

iSphere 초기값 차이

Practice:

iSphere 초기값: -1

의미: ray와 충돌하는 물체(sphere)가 없다는 것을 나타낸다.

이유: ray가 어떤 sphere와도 충돌하지 않을 수 있는 상황을 고려한다.

Exercise:

iSphere 초기값: 7

의미: ray가 가장 큰 sphere와 반드시 충돌한다고 가정한다.

이유: Exercise에서는 ray가 가장 큰 sphere에 의해 감싸져 있기 때문에, 다른 sphere와 충돌하지 않았다면 반드시 가장 큰 sphere와 충돌하게 된다. 따라서 코드는 다른 sphere와의 충돌을 먼저 검사하고, 충돌하지 않는 경우 자동으로 가장 큰 sphere와 충돌한 것으로 간주한다.

```
// Compute the intensity from ray using recursive ray casting.
// Exclude an intersection with the object E where the ray start from.
vec3 intensity(const Ray& ray, const Light l[], int nLights, int depth, int E = -1)
{
    vec3 I(0, 0, 0); // Final intensity

    // Find the closest intersection point and the normal
    vec3 p, n; // Position and normal in the eye coordinate system
    int iObject = findIntersection(ray, p, n, E);

    //hit object -> 필연적!!
    for (int i = 0; i < nLights; i++) {
        // Shadow ray
        vec3 p_shadow, n_shadow; // Not used

        vec3 pDistantLight = p + 1.0e10f * l[i].p_eye;
        Ray shadowRay(p, pDistantLight);

        int jObject = findIntersection(shadowRay, p_shadow, n_shadow, iObject);

        if (jObject == 7) // Not shadowed -> 그니까 마지막까지 안부닥치고 가장 큰 겉의 구에 닿는 거
        {
            // Phong reflection
            vec3 v = normalize(ray.p0 - ray.p1);        // Direction to the viewer
            vec3 r = normalize(reflect(l[i].p_eye, n)); // Reflection of light

            I += phong(n, v, l[i], r, iObject);
        }
        else I += ambient(l[i]);    // Shadowed
```

ray와 sphere의 충돌 여부에 대한 가정의 차이

Practice:

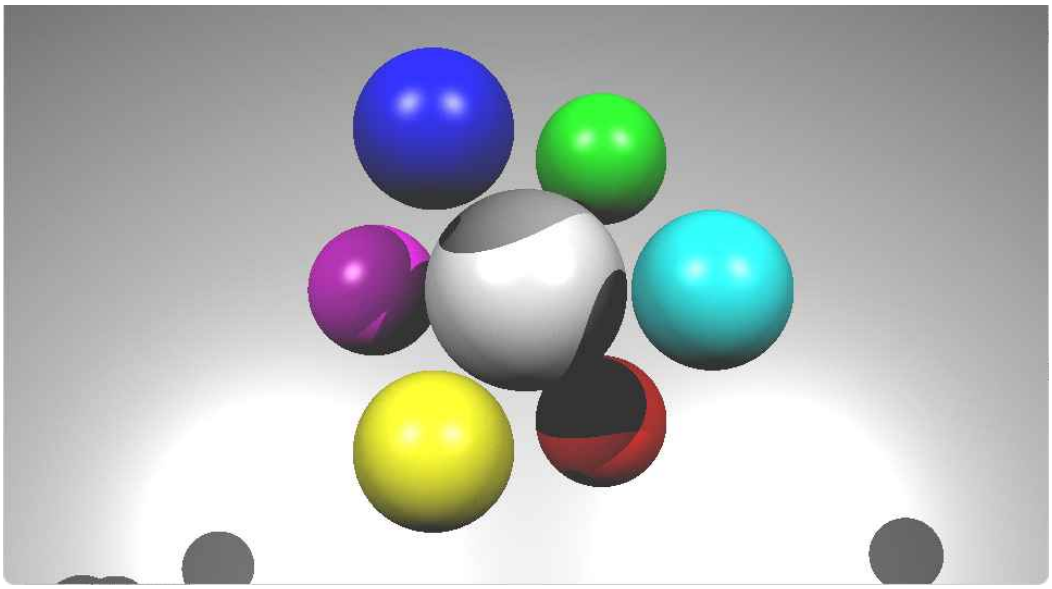
ray가 어떤 물체와도 충돌하지 않을 수 있다는 가능성을 고려하여 iObject != -1 조건을 사용한다. 즉, 충돌한 물체가 없으면 not shadowed 상태가 된다.

Exercise:

ray가 반드시 가장 큰 sphere와 충돌한다는 가정 하에 iObject != -1 조건을 생략한다. 즉, 다른 물체와의 충돌 여부에 상관없이 가장 큰 sphere와의 충돌만 고려한다.

not shadowed가 되는 경우는 오직 다른 모든 sphere와 충돌하지 않고 가장 큰 sphere와만 충돌하는 경우이다.

Exercise 02. Employ a different color for each sphere



```
// Material configuration
vec3 m_ambient(0.1, 0.1, 0.1);
vec3 m_diffuse[8] = { vec3(0.95, 0.95, 0.95),vec3(1, 0, 0),vec3(0, 1, 0),vec3(0, 0, 1),
                      vec3(1, 1, 0), vec3(1, 0, 1), vec3(0, 1, 1), vec3(0.8, 0.8, 0.8) };
vec3 m_specular(0.5, 0.5, 0.5);
float m_shininess = 25;
```

m_diffuse를 배열로 선언하고 8개의 색상을 지정한다.

추후 7개의 sphere들과 가장 큰 sphere 의 색상을 지정할 때 사용한다.

```
// Ambient, diffuse, specular
vec3
phong(const vec3& n, const vec3& v, const Light& l, const vec3& r, int E)
{
    vec3 I = ambient(l);

    float lambertian = std::max(dot(n, l.p_eye), 0.0f);

    if (lambertian > 0)
    {
        float specular = pow(std::max(dot(v, r), 0.0f), m_shininess);

        for (int i = 0; i < 3; i++)
        {
            I[i] += m_diffuse[E][i] * lambertian * l.diffuse[i];
            I[i] += m_specular[i] * specular * l.specular[i];
        }
    }
    return I;
}
```

Phong함수에서 앞서 m_diffuse의 값으로 지정했던 색상들로 sphere들의 색상을 설정한다. 기존의 phong 함수에서 인자 E를 추가적으로 입력 받아 해당 sphere에 따라서 다른 색상으로 나타나도록 한다.

III. 느낀점

이번 프로젝트를 진행하며 레이 트레이싱 알고리즘을 구현하면서 수학적인 개념을 코드로 옮기는 어려움을 겪었습니다. 하지만 끊임없이 자료를 찾아보고 코드를 수정하며 문제를 해결해나가는 과정에서 문제 해결 능력과 끈기가 향상되었음을 느꼈습니다. 특히, 행렬 연산을 이용한 변환에 대한 이해도가 높아졌고, 이를 통해 더욱 다양한 효과를 구현할 수 있게 되었습니다. 다음 프로젝트에서는 더욱 복잡한 장면을 구현하기 위해 병렬 처리 기술을 적용하여 렌더링 속도를 향상시키고 싶습니다.