EECS 498 HW2: Ray Tracing

Author: Minxuan Chen

October 10, 2024

1 Task 1 (5.3)

1.1 Code

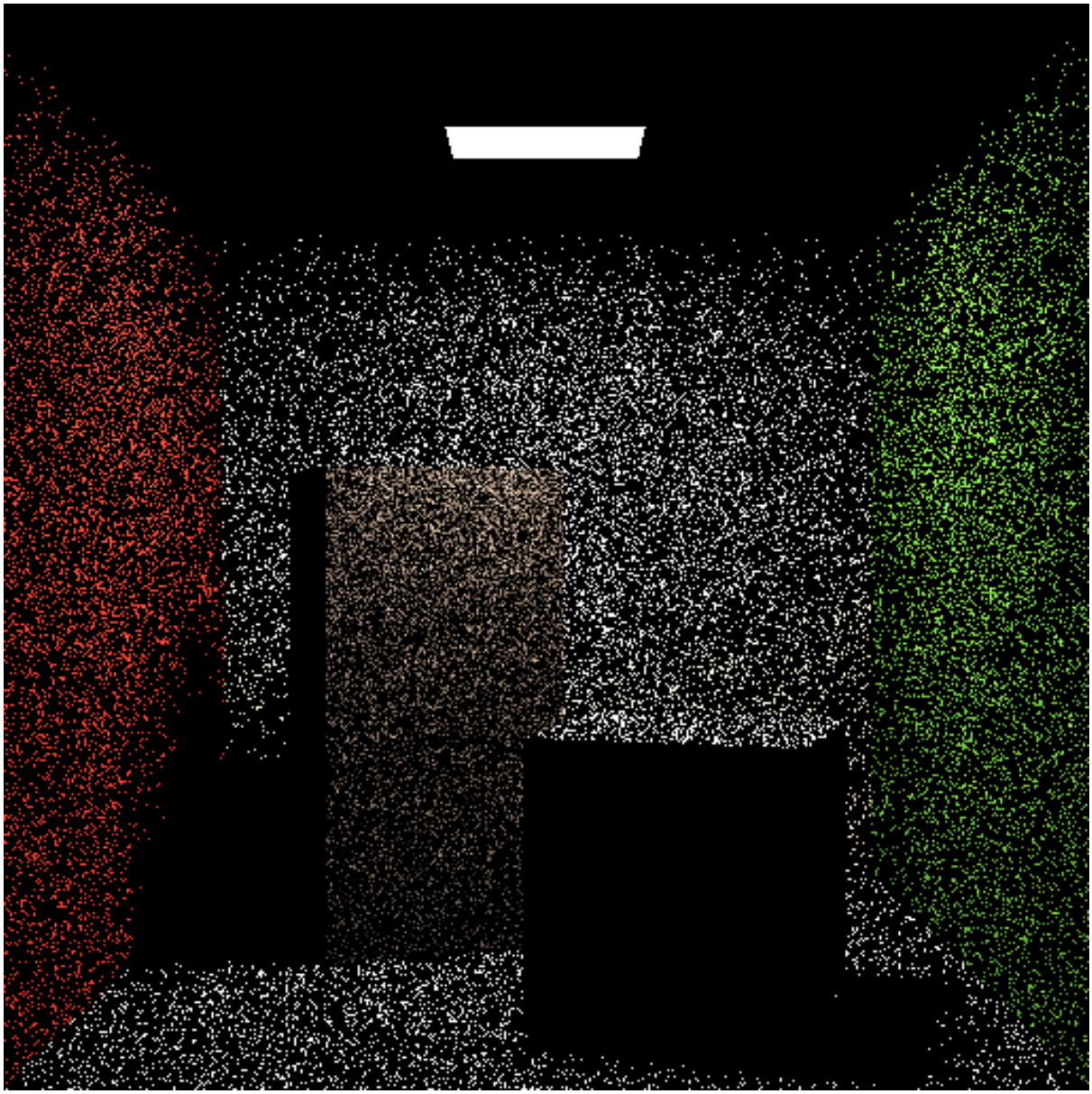
```
Vec3 Scene::trace(const Ray &ray, int bouncesLeft, bool discardEmission) {
    if constexpr(DEBUG) {
        assert (ray.isNormalized());
    }
    if (bouncesLeft < 0) return {};

    //task 1
    Intersection inter = getIntersection(ray);

    if (!inter.happened) {
        return {};
    }
    else {
        Vec3 diffuseColor = inter.getDiffuseColor();
        return diffuseColor;
    }
}
```

2 Task 2 (6.4)

2.1 Code

```
Vec3 Scene::trace(const Ray &ray, int bouncesLeft, bool discardEmission) {
    if constexpr(DEBUG) {
        assert (ray.isNormalized());
    }

    //task 2
    // ray is camera ray
    if (bouncesLeft < 0) return {};

    Intersection inter = getIntersection(ray);
    if (!inter.happened) {
        return {};
    }
    else {
        Vec3 wi_dir = Random::randomHemisphereDirection(inter.getNormal());
        Ray secondRay = Ray{inter.pos, wi_dir};
        Vec3 brdf = inter.calcBRDF(-secondRay.dir, -ray.dir);

        float cosineTerm = secondRay.dir.dot(inter.getNormal());

        // shoot a single ray from interaction point on the surface;
        Intersection inter_Li = getIntersection(secondRay);

        if (inter_Li.happened && inter_Li.object->hasEmission){
        //hasemission means it is light source
            Vec3 Li = inter_Li.getEmission();
            Vec3 Lo = inter.getEmission() + (2*PI*cosineTerm)*Li*brdf;
            return Lo;
        }
        else{
            // no Li term
            return inter.getEmission();
        }
    }
}
```

2.2 Image

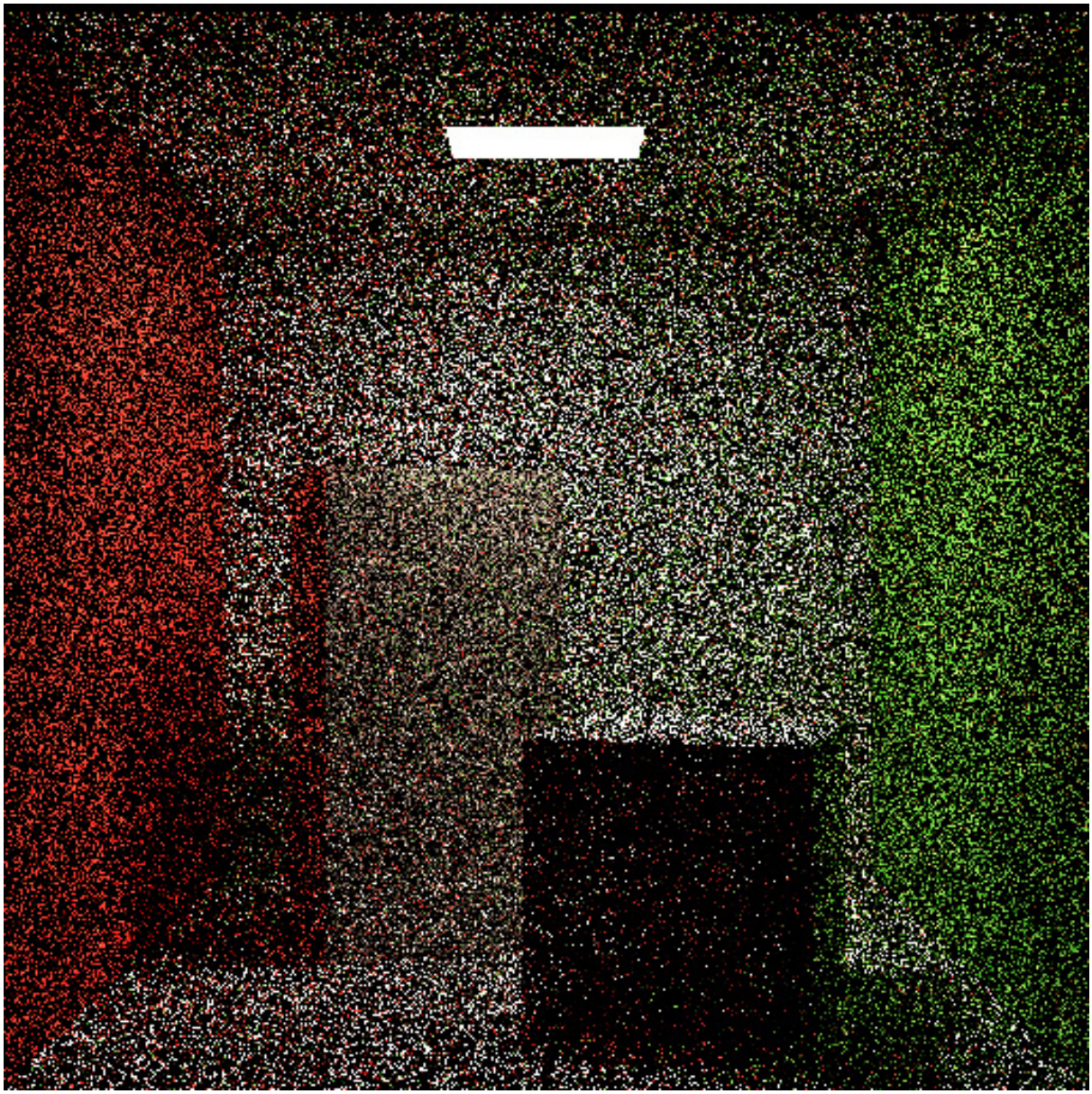


3 Task 3 (7.1)

3.1 Code

```
Vec3 Scene::trace(const Ray &ray, int bouncesLeft, bool discardEmission) {
    if constexpr(DEBUG) {
        assert (ray.isNormalized());
    }
    //task 2/3
    // ray is camera ray
    if (bouncesLeft < 0) return {};

    Intersection inter = getIntersection(ray);
    if (!inter.happened) {
        return {};
    }
    else {
        Vec3 wi_dir = Random::randomHemisphereDirection(inter.getNormal());
        Ray secondRay = Ray{inter.pos, wi_dir};
        Vec3 brdf = inter.calcBRDF(-secondRay.dir, -ray.dir);

        float cosineTerm = secondRay.dir.dot(inter.getNormal());

        // shoot a single ray from interaction point on the surface;
        Intersection inter_Li = getIntersection(secondRay);

        if (inter_Li.happened){
            Vec3 Li = trace(secondRay, bouncesLeft-1, false);
            Vec3 Lo = inter.getEmission() + (2*PI*cosineTerm)*Li*brdf;
            return Lo;
        }
        else{
            // no Li term
            return inter.getEmission();
        }
    }
}
```

3.2 Image



4 Task 4 (8.3)

4.1 Code

```
Vec3 Scene::trace(const Ray &ray, int bouncesLeft, bool discardEmission) {
    if constexpr(DEBUG) {
        assert (ray.isNormalized());
    }
    //task 4
    if (bouncesLeft < 0) return {};
    Intersection inter = getIntersection(ray);
    if (!inter.happened){
        return {};
    }
    else{
        Vec3 L_indirect = {0.0f, 0.0f, 0.0f};
        Vec3 L_direct = {0.0f, 0.0f, 0.0f};

        // indirect radiance
        Vec3 wi_dir = Random::cosWeightedHemisphere(inter.getNormal());
        Ray secondRay = Ray{inter.pos, wi_dir};
        Vec3 brdf = inter.calcBRDF(-secondRay.dir, -ray.dir);
        Intersection inter_Li = getIntersection(secondRay);
        if (inter_Li.happened){
            L_indirect = PI*brdf*trace(secondRay, bouncesLeft-1, true);
        }

        //direct radiance
        float pdfLightSample = 1 / lightArea;
        Intersection lightSample = sampleLight();
        Vec3 lightDir = lightSample.pos - inter.pos;
        float distanceToLight = lightDir.getLength();
        lightDir.normalize();

        //shawdow ray
        Ray rayToLight = {inter.pos, lightDir};
        Intersection shawdow_inter = getIntersection(rayToLight);

        if ((shawdow_inter.pos-lightSample.pos).getLength()<1e-3){
            //not block
            Vec3 brdf_direct = inter.calcBRDF(-lightDir, -ray.dir);
            float cos_n_wi = lightDir.dot(inter.getNormal());
            float cos_np_wi = -lightSample.getNormal().dot(lightDir);
            L_direct = 1/(pdfLightSample*distanceToLight*distanceToLight) *
            (lightSample.getEmission() * brdf_direct * cos_n_wi * cos_np_wi);
        }

        //total Li
        Vec3 Li = L_indirect + L_direct;
        Vec3 Lo =  {0.0f, 0.0f, 0.0f};
        if (discardEmission){
            Lo = Li;
        }
        else{
            Lo = inter.getEmission() + Li;
        }
        return Lo;
    }
}
```

4.2 Image