Homework 2: Ray Tracing

REPORT

EECS 598 Graphics and Generative Models

Written By: Xun Yang

Date: October 11, 2024

Contents

1 Task 5. Tracing the Rays

1.1 Implementation

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

    // TODO
    Intersection inter = getIntersection(ray); // find the intersection
    if (!inter.happened)
        return Vec3(0.0f, 0.0f, 0.0f);
    Vec3 diffuseColor = inter.object->kd; // return its diffuse color
    return diffuseColor;
}
```

1.2 Results and Evaluation

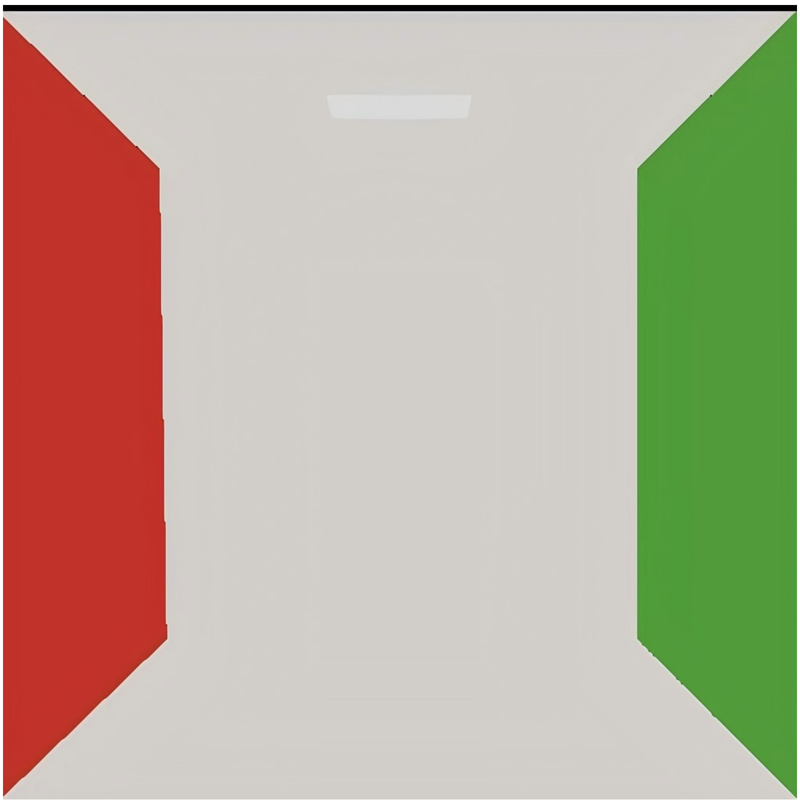

Figure 1: Tracing the Rays

2 Task 6. Direct Illumination

2.1 Implementation

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

    // TODO
    Intersection inter = getIntersection(ray);
    if (!inter.happened) return Vec3(0.0f, 0.0f, 0.0f);

    Vec3 R_out = inter.object->ke; // emission
    // randomly shoot a ray in the hemisphere.
    Vec3 sample_dir =
        Random::randomHemisphereDirection(inter.getNormal()); // w_i
    Ray nextRay(inter.pos, sample_dir); // construct the secondRay

    // if the sampling ray intersects
    if (getIntersection(nextRay).happened) {
        // Task 6 Direct Illumination
        Vec3 L_i = getIntersection(nextRay).object->ke; // emission
            radiance
        Vec3 brdf = inter.calcBRDF(-sample_dir, -ray.dir); // fr
        float cosineTerm = nextRay.dir.dot(inter.getNormal());
        R_out += (2 * PI) * L_i * brdf * cosineTerm; // Monte Carlo
            sampling
    }

    return R_out;
}
```

```
Vec3 Random::randomHemisphereDirection(const Vec3 &normal) {
    float p = Random::randUniformFloat();
    float q = Random::randUniformFloat();
    float azimuth = 2 * PI * p;
    float elevation = acos(q);

    // Convert spherical coordinates to Cartesian
    float x = cos(azimuth) * sin(elevation);
    float y = sin(azimuth) * sin(elevation);
    float z = cos(elevation);

    return localDirToWorld({x, y, z}, normal);
}
```

2.2 Results and Evaluation

The result of rendering the image with 32 sample per pixel(SPP) is shown in Fig2

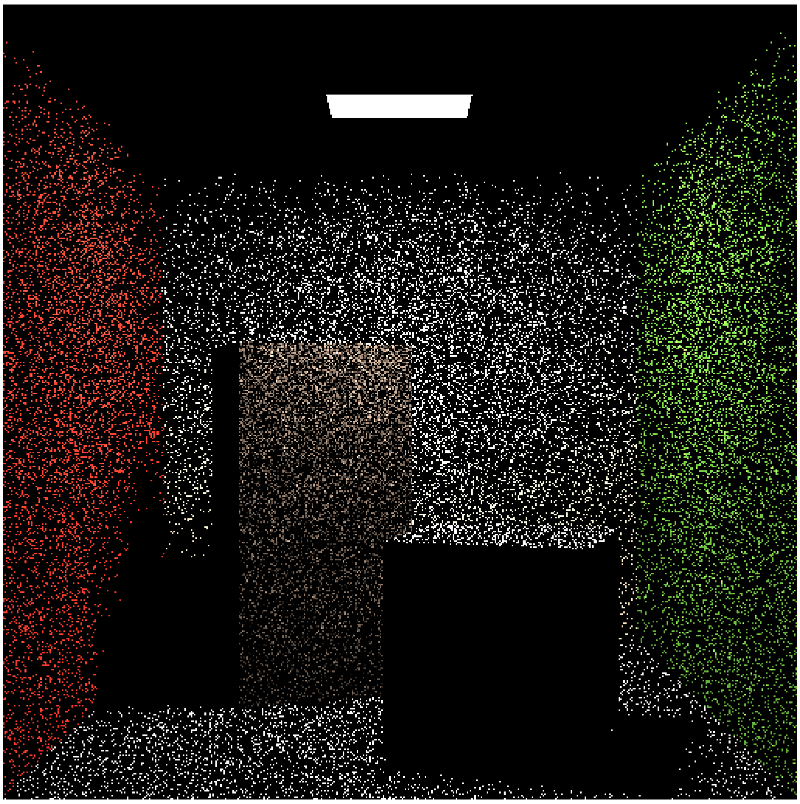

Figure 2: Direct Illumination with 32 SPP

2.3 Thought Question

Q: Why does the image of 16 SPP look darker than 128 SPP?

During ray tracing, we implement random sampling (e.g., Monte Carlo sample). Fewer SPP can result in under-sampling, where some pixels do not adequately capture indirect light, shadows, or other lighting effects. This results in darker image because the average values of several areas are not as accurate. Increasing the SPP reduces noise, captures more light, and produces a more accurate image, but costs more time and spaces on the other hand.

3 Task 7. Global Illumination

3.1 Implementation

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

    // TODO
```

```
    Intersection inter = getIntersection(ray);
    if (!inter.happened)
        return Vec3(0.0f, 0.0f, 0.0f);
    Vec3 R_out = inter.object->ke; // emission
    Vec3 sample_dir =
        Random::randomHemisphereDirection(inter.getNormal()); // w_i
    Ray nextRay(inter.pos, sample_dir); // construct the secondRay
    // if the sampling ray intersects
    if (getIntersection(nextRay).happened) {
        // Global Illumination
        Vec3 L_i = trace(nextRay, bouncesLeft - 1, false); // recursion
        Vec3 brdf = inter.calcBRDF(-sample_dir, -ray.dir); // fr
        float cosineTerm = nextRay.dir.dot(inter.getNormal());
        R_out += (2 * PI) * L_i * brdf * cosineTerm; // Monte Carlo
    }

    return R_out;
}
```

3.2 Results and Evaluation

By setting SPP = 32, MAX_DEPTH = 8, the rendering result is shown in Fig3. Rendering time in seconds: 196.

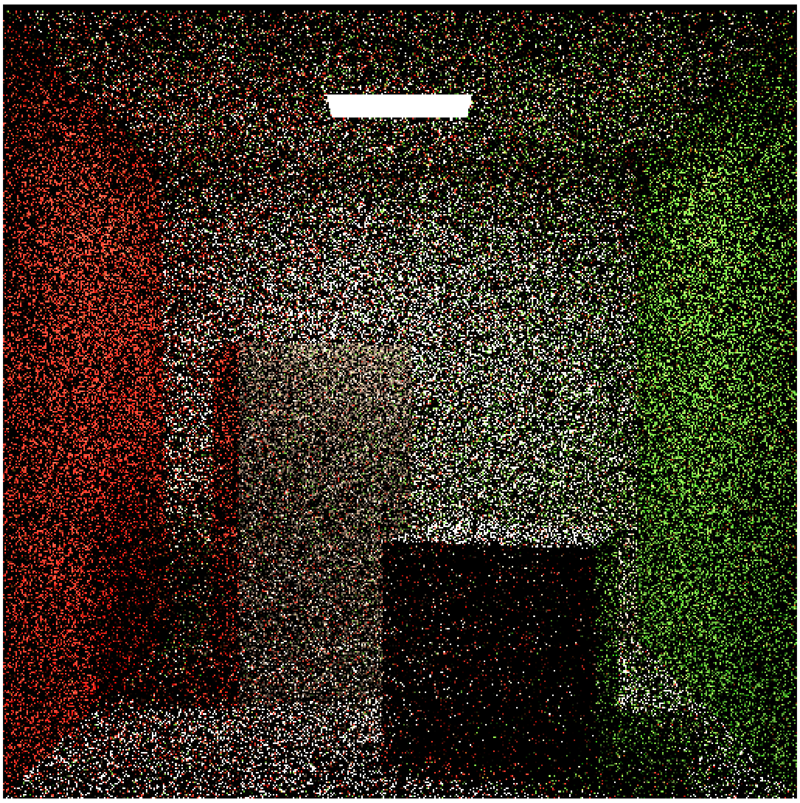

Figure 3: Global Illumination with 32 SPP

3.3 Thought Question

Q: compare the results to direct illumination only, what new features can you identify, and why do we have them?

Some dark areas are now brighter, such as the ceiling and the left and right sides of the cube. This is because we recursively calculate the indirect lights which bounce for multiple times instead of ignoring them. This process makes the image closer to the reality because objects can absorb and reflect radiance.

4 Task 8. Acceleration

4.1 Implementation

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

    // TODO
    Intersection inter = getIntersection(ray);
    if (!inter.happened)
        return Vec3(0.0f, 0.0f, 0.0f);
    Vec3 R_out = discardEmission? Vec3(0.0f, 0.0f, 0.0f) :
        inter.object->ke; // discard emission

    // ———————— direct radiance (L from the light source) ————————
    Intersection lightSample = sampleLight(); // a sample on the
        light, not an actual intersection
    Vec3 light_dir = lightSample.pos - inter.pos;
    float d = light_dir.getLength();             // distanceToLight
    light_dir.normalize();
    Ray rayToLight(inter.pos, light_dir);        // second ray as
        described in HW
    if (getIntersection(rayToLight).mesh == lightSample.mesh) {
        Vec3 L_di = getIntersection(rayToLight).object->ke; // direct
            instance radiance
        Vec3 brdf = inter.calcBRDF(-light_dir, -ray.dir); // fr
        float cosineTerm1 = light_dir.dot(inter.getNormal());
        float cosineTerm2 = -light_dir.dot(lightSample.getNormal());
        float pdfLightSample = 1 / lightArea;
        R_out += L_di * brdf * cosineTerm1 * cosineTerm2 /
            (pdfLightSample * d * d); // Monte Carlo
    }

    // ———————— indirect radiance (L from bouncing rays) ————————
    Vec3 sample_dir =
        Random::cosWeightedHemisphere(inter.getNormal()); //
        importance sampling
    sample_dir.normalize();
```

```
    Ray rayCosWeight(inter.pos, sample_dir);    // first ray as described in HW
    // if the sampling ray intersects
    if (getIntersection(rayCosWeight).happened) {
        Vec3 L_i = trace(rayCosWeight, bouncesLeft - 1, true); // discard the direct radiance
        Vec3 brdf = inter.calcBRDF(-sample_dir, -ray.dir);      // fr
        float cosineTerm = sample_dir.dot(inter.getNormal());
        float pdfWeightSample = sample_dir.dot(inter.getNormal()) / PI;
        R_out += L_i * brdf * cosineTerm / pdfWeightSample; // Monte Carlo
    }

    return R_out;
}
```

```
Vec3 Random::cosWeightedHemisphere(const Vec3 &normal) {
    // Task 8.2
    float p = Random::randUniformFloat();
    float q = Random::randUniformFloat();
    float azimuth = 2 * PI * p;
    float elevation = acos(sqrt(q));

    // Convert spherical coordinates to Cartesian
    float x = cos(azimuth) * sin(elevation);
    float y = sin(azimuth) * sin(elevation);
    float z = cos(elevation);

    return localDirToWorld({x, y, z}, normal);
}
```

4.2 Results and Evaluation

By setting SPP = 32, MAX_DEPTH = 8, the rendering result is shown in Fig4. Rendering time in seconds: 53.
By setting SPP = 256, MAX_DEPTH = 8, the rendering result is shown in Fig5. Rendering time in seconds: 422.
Comparing with the time consumed during Task 7, it is suggested that the **acceleration** does speed up the rendering process.

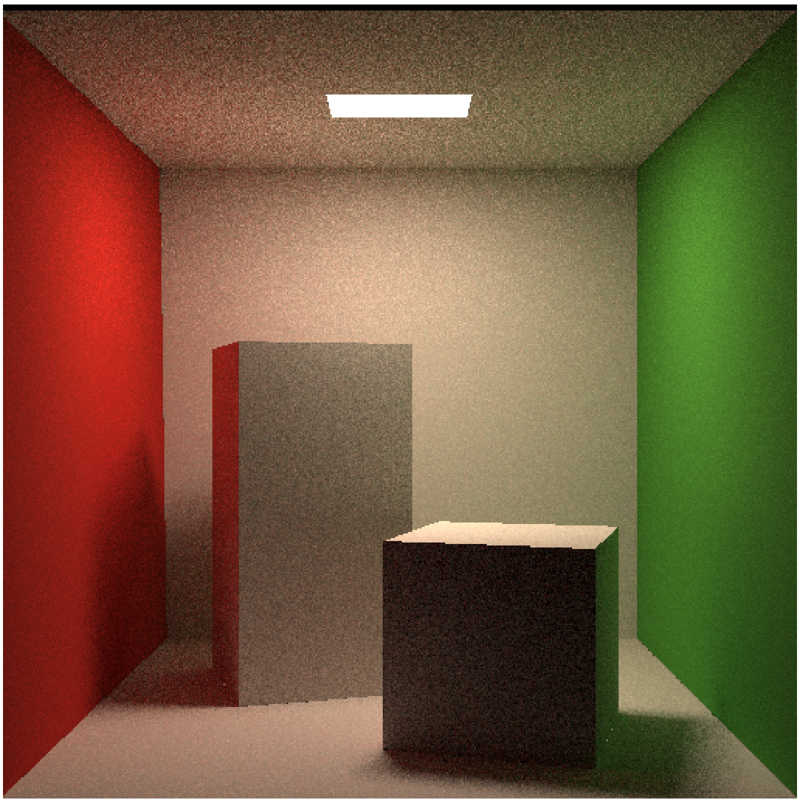

Figure 4: Accelerated result with 32 SPP

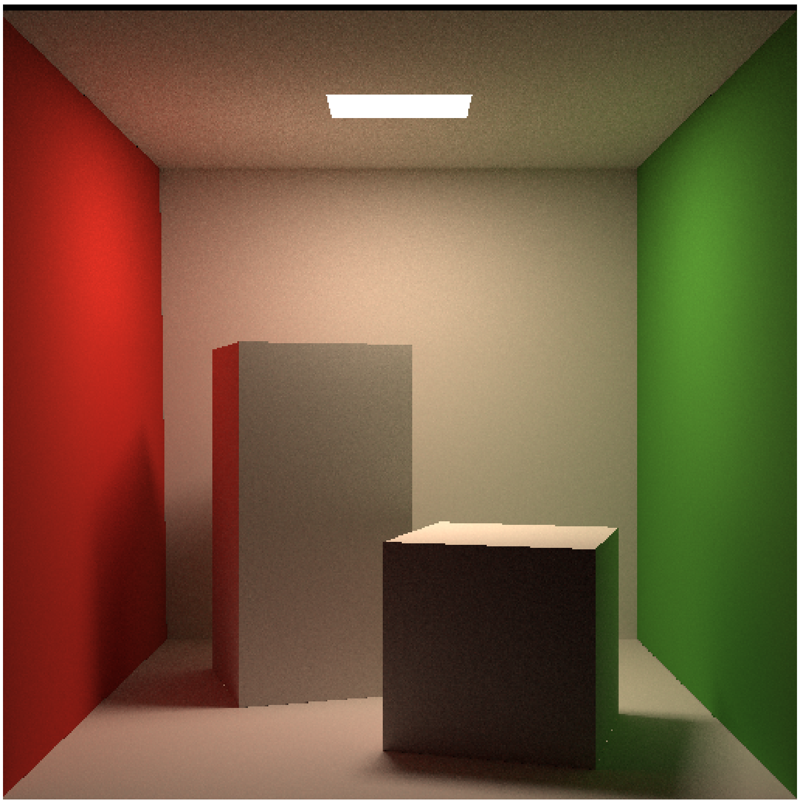

Figure 5: Accelerated result with 256 SPP