计算摄影学 project3

1900017872 乔思琦

- 编译：

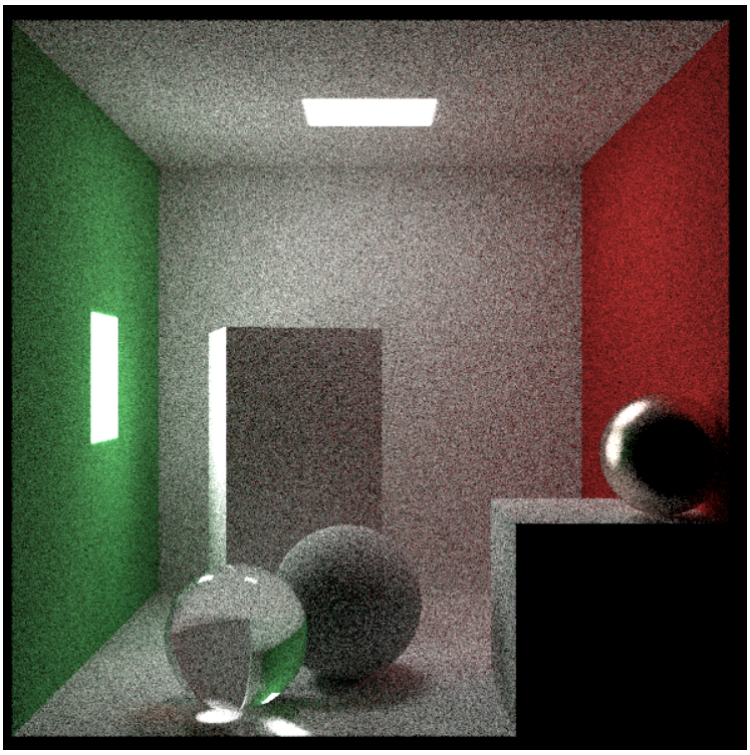
```
cmake -B build
cmake --build build
```

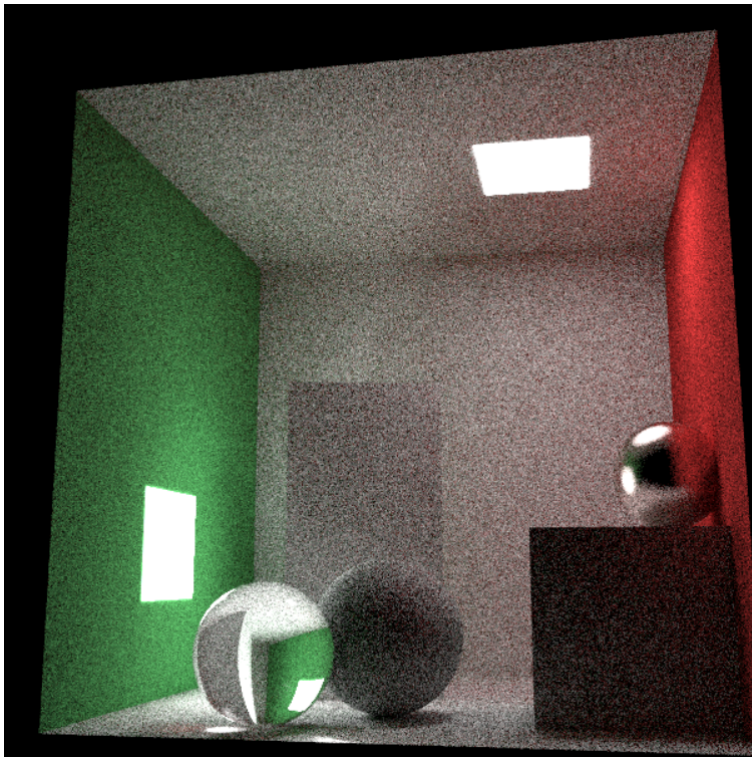
- 重定向：

```
build\debug\theNextWeek > image.ppm
```

- 查看方式：PPM Viewer (rhodes.edu)

实现效果：





- 间接光照:

  ray-tracing

```
color ray_color(
    const ray& r,
    const color& background,
    const hittable& world,
    shared_ptr<hittable> lights,
    int depth)
{
    hit_record rec;
    if (depth <= 0)
        return color(0,0,0);
    if (!world.hit(r, 0, infinity, rec))
        return background;
    if (!rec.mat_ptr->scatter(r, rec, srec))
        return emitted;

    if (srec.is_specular) {
        return srec.attenuation
             * ray_color(srec.specular_ray, background, world,
lights, depth-1);
    }

    auto light_ptr = make_shared<hittable_pdf>(lights, rec.p);
    mixture_pdf p(light_ptr, srec.pdf_ptr);
    ray scattered = ray(rec.p, p.generate(), r.time());
    auto pdf_val = p.value(scattered.direction());

    return emitted
```

```
            + srec.attenuation * rec.mat_ptr->scattering_pdf(r, rec,
scattered)
                                * ray_color(scattered, background,
world, lights,depth-1) / pdf_val;
}
```

- 折射：

$$\eta \cdot \sin\theta = \eta' \cdot \sin\theta'$$

On the refracted side of the surface there is a refracted ray $\mathbf{R}'$ and a normal $\mathbf{n}'$, and there exists an angle, θ', between them. We can split $\mathbf{R}'$ into the parts of the ray that are perpendicular to $\mathbf{n}'$ and parallel to $\mathbf{n}'$:

$$\mathbf{R}' = \mathbf{R}'_{\perp} + \mathbf{R}'_{\parallel}$$

If we solve for $\mathbf{R}'_{\perp}$ and $\mathbf{R}'_{\parallel}$ we get:

$$\mathbf{R}'_{\perp} = \frac{\eta}{\eta'}(\mathbf{R} + \cos\theta\mathbf{n})$$
$$\mathbf{R}'_{\parallel} = -\sqrt{1 - |\mathbf{R}'_{\perp}|^2}\mathbf{n}$$
$$\mathbf{R}'_{\parallel} = \frac{\eta}{\eta'}(\mathbf{R} + (-\mathbf{R}\cdot\mathbf{N})\mathbf{N})$$

```
vec3 refract(const vec3& uv, const vec3& n, double etai_over_etat) {
    auto cos_theta = dot(-uv, n);
    vec3 r_out_parallel =  etai_over_etat * (uv + cos_theta*n);
    vec3 r_out_perp = -sqrt(1.0 - r_out_parallel.length_squared()) *
n;
    return r_out_parallel + r_out_perp;
}
```

(对于从折射率高的物质射向折射率低的物质的光线，我们认为不发生折射)

Now real glass has reflectivity that varies with angle — look at a window at a steep angle and it becomes a mirror. There is a big ugly equation for that, but almost everybody uses a cheap and surprisingly accurate polynomial approximation by Christophe Schlick.

```
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);
 }
```

- 镜面反射：

此时反射的光线可由计算得出，而不是像漫反射那样随机sample

```
vec3 reflect(const vec3& v, const vec3& n) {
    return v - 2*dot(v,n)*n;
}
```

- 面光源：

  首先创建diffuse_light类，设置发光材质

```
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;

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

  同时构造rect类（后续用来生成面光源）

  First, here is a rectangle in an xy plane. Such a plane is defined by its z value. For example, $z = k$. An axis-aligned rectangle is defined by the lines $x = x_0, x = x_1$, $y = y_0$, and $y = y_1$

  To determine whether a ray hits such a rectangle, we first determine where the ray hits the plane. Recall that a ray $\mathbf{P}(t) = \mathbf{A} + t\mathbf{b}$ has its z component defined by $P_z(t) = A_z + tb_z$. Rearranging those terms we can solve for what the t is where $z = k$.

$$t = \frac{k - A_z}{b_z}$$

  Once we have t, we can plug that into the equations for x and y :

$$x = A_x + tb_x$$
$$y = A_y + tb_y$$

  It is a hit if $x_0 < x < x_1$ and $y_0 < y < y_1$.

```
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;

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

reference:

- RayTracingInOneWeekend
- RayTracingTheNextWeek
- RayTracingTheRestOfYourLife