CSE 167 (WI 2021) Raytracer with global illumination – Bonus Credit Guide

This is a guide for the bonus credit for the raytracer final project. The main idea is given in the slides of Wednesday Week 7.

3.1 Global Illumination

Here, we expand the ray tracer into a full global illumination rendering. In particular, not only that the specular reflection is computed through a recursive ray tracing, but also that the diffuse reflection is computed recursively.

3.1.1 Simulating photons' paths from the camera

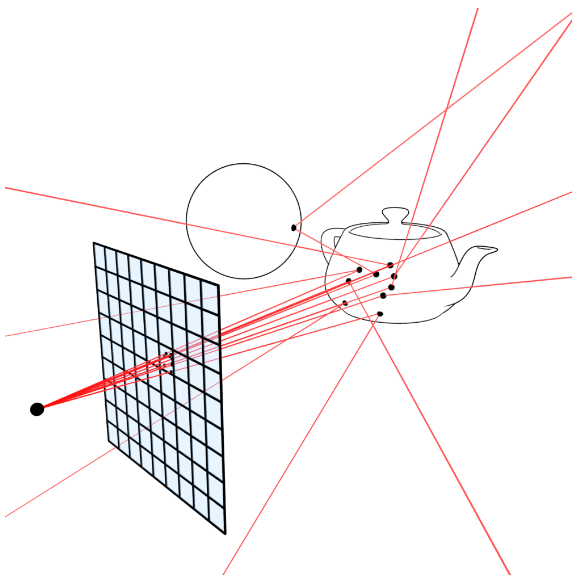

Figure 1 The color of each pixel is the average of all colors of rays (paths of photons) through the pixel that connect the vantage point to the light source. A diffuse material scatters the ray at random direction, while a specular material mirror reflects the ray. A path of n bounces randomly picks the diffuse mode or specular mode of reflecting at every bounce; at the last bounce, shade the final hit by the diffuse lighting model; this amounts to directly connecting the ray to every light sources (with visibility check). The ray color at the pixel (the color to be summed to the pixel) is the final hit color multiplied by the diffuse colors (with Lambert cosine law) and/or the specular colors along the journey.

The idea is that we paint each pixel of the image by the total colored light (in RGB) contributed by all the photons that would travel from every light source to the perspective center through the pixel (see Figure 1).

How would a photon travel in the scene? Instead of simulating their physical trajectory from the light sources to the eye, we trace the trajectory backward from the pixel back to the lights. This latter strategy is more convenient if we only want to collect those rays that pass through the pixel. It is also reminiscent to the recursive mirror reflection in the main part of your ray tracer.

The light path for a concrete number of bounces

Let us first consider paths with a known number of bounces n. To simulate/sample one of these paths, first shoot a random ray from the eye through the pixel. Unlike the main part of your project where the ray pass the center of the pixel, here we take the ray through a uniformly distributed random point in the pixel. This is illustrated in Figure 1.

Next, as it intersects with a surface, we toss a coin (50%-50%) to decide whether we want the ray to reflect like a diffuse reflection or a specular reflection. If we go about the diffuse mode, then we shoot the next ray at a random direction (described more concretely below). If we go about the specular reflection, then we shoot the next ray along the mirror reflection direction.

If we have reached the nth bounce, then we terminate the random process. The color of this last hit is computed using the direct diffuse lighting like for local illumination:

$$\mathrm{Color}(\mathrm{hit}_{\mathrm{last}}) = \sum_{\ell \in \mathrm{Lights}} \begin{pmatrix}\text{visibility}\\ \text{to light}_\ell\end{pmatrix} \frac{L_\ell D}{\mathrm{attenuation}_\ell} \max(\langle \mathbf{l}_\ell, \mathbf{n}\rangle, 0) \tag{1}$$

where D is the diffuse color, and L_ℓ is the light color of the ℓ-th light. Effectively, the last bounce forces the light ray to connect to each visible light source, contributing the color at this last bounce with a Lambert cosine weight.

Effectively, the color of this photon path through the pixel (*i.e.* beginning of the path) is the last color multiplied by the material diffuse/specular color at all the intermediate bounces:

$$\mathrm{Color}(\mathrm{RayAtPixel}) = W_1 W_2 \cdots W_{n-1} \mathrm{Color}(\mathrm{hit}_{\mathrm{last}}), \tag{2}$$

where the weight W_i is the diffuse or the specular color of the material at the i-th bounce depending on whether that bounce is of a diffuse mode or a specular mode.

Emissive Surface

If the light is bouncing off from a surface with emission, then we want to add that emission color E_i to the path color. In a recursive formulation,

$$\mathrm{Color}(\mathrm{RayAtPixel}) = \mathrm{Color}(\mathrm{RayBetweenHit}_0\mathrm{AndHit}_1) \tag{3}$$

$$\begin{aligned}&\mathrm{Color}(\mathrm{RayBetweenHit}_{i-1}\mathrm{AndHit}_i)\\ &\qquad = E_i + W_i \cdot \mathrm{Color}(\mathrm{RayBetweenHit}_i\mathrm{AndHit}_{i+1}), \quad i = 1, 2, \ldots, n-1\end{aligned} \tag{4}$$

$$\mathrm{Color}(\mathrm{RayBetweenHit}_{n-1}\mathrm{AndHit}_n) = E_n + \mathrm{Color}(\mathrm{hit}_{\mathrm{last}}). \tag{5}$$

R Note that we don't add ambient color like in the shading model for HW2 and the main part of HW4. Our full global illumination with recursive lighting replaces the ambient lighting.

Diffuse bounce

Suppose we are bouncing the ray using the diffuse mode; that is, we want to shoot the next ray at a random direction. What should the distribution of the random direction be?

One direct approach is to sample this next ray direction uniformly on the hemisphere (above the surface), like Figure 2 (left). Let $\mathbf{d}$ be this new random ray direction. If one does so, the weight at the bounce is

$$W_i = D_i \underbrace{\langle \mathbf{d}, \mathbf{n}\rangle}_{\cos \angle_{\mathbf{d}}^{\mathbf{n}}}, \quad \mathbf{d} \text{ is sampled uniformly on the hemisphere.} \tag{6}$$

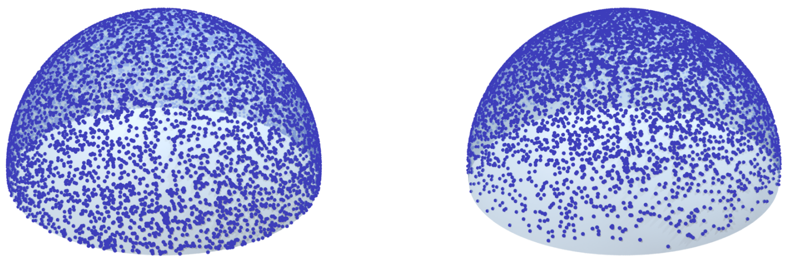

Figure 2 Left: Uniform hemisphere. Right: Cosine-weighted hemisphere.

where the inner product is the Lambert cosine term.

Another approach is to sample the next ray direction **d** with a distribution on the hemisphere that is already weighted by the Lambert cosine factor, shown in Figure 2 right. That is, there is a higher probability to shoot a ray near the normal direction than the directions nearly tangent to the surface. In that case, the weight is just the diffuse color

$$W_i = D_i, \quad \mathbf{d} \text{ is sampled according to the cosine-weighted distribution.} \tag{7}$$

The latter approach will let the final image converge a bit faster than the former uniform sampling.

These hemisphere samplings are obtained by the following mapping formulas. Let s, t be two independent uniform random numbers in the interval $[0, 1]$. Then a uniform hemisphere sampling of **d** is given by

$$u = 2\pi s, \quad v = \sqrt{1 - t^2}, \quad \mathbf{d} = \begin{bmatrix} v\cos(u) \\ t \\ v\sin(u) \end{bmatrix}. \tag{8}$$

A cosine weighted hemisphere sampling of **d** is given by

$$u = 2\pi s, \quad v = \sqrt{1 - t}, \quad \mathbf{d} = \begin{bmatrix} v\cos(u) \\ \sqrt{t} \\ v\sin(u) \end{bmatrix}. \tag{9}$$

Remember to rotate the frame so that the hemisphere is pointing in the normal direction.

3.1.2 Path summation

In the earlier discussion about sampling paths (and their colors) with a fixed number of bounces, we can compute the color contributed by all paths by n bounces. For each pixel (i, j), we take a large number N of random paths and take the average of their resulting colors

$$\text{TotalColorFromPhotonsWith}n\text{Bounces}_{ij} = \frac{1}{N}\sum_{k=1}^{N} \text{Color}(\text{RayAtPixel}_{ij,n,k}). \tag{10}$$

The final pixel color is the sum of the color from all path lengths:

$$\begin{aligned}\text{PixelColor}_{ij} &= \text{TotalColorFromPhotonsWithOneBounce}_{ij}\\ &\quad+ \text{TotalColorFromPhotonsWithTwoBounces}_{ij}\\ &\quad+ \text{TotalColorFromPhotonsWithThreeBounces}_{ij}\\ &\quad+ \cdots \qquad (11)\end{aligned}$$

In practice, you may truncate this infinite series so that you only compute up to a certain number of bounces (like a maximal recursion depth). In your implementation, you may use this direct truncation method, or the Russian Roulette method.

3.1.3 Russian Roulette

Instead of truncating the path by setting an artificial choice of maximal recursion depth, the Russian Roulette method is a practical method that sums all the terms in the infinite series (11).

First of all, introduce a parameter $0 < \lambda < 1$. Now modify the expression tautologically at first

$$\begin{aligned}\text{PixelColor}_{ij} &= \frac{(1-\lambda)\lambda}{(1-\lambda)\lambda}\text{TotalColorFromPhotonsWithOneBounce}_{ij}\\ &\quad+ \frac{(1-\lambda)^2\lambda}{(1-\lambda)^2\lambda}\text{TotalColorFromPhotonsWithTwoBounces}_{ij}\\ &\quad+ \frac{(1-\lambda)^3\lambda}{(1-\lambda)^3\lambda}\text{TotalColorFromPhotonsWithThreeBounces}_{ij}\\ &\quad+ \cdots \qquad (12)\end{aligned}$$

Then, instead of picking a fixed path length n and sampling paths with that length, we will let the photon run indefinitely. But, at every bounce, we toss a coin with probability λ to decide whether we want to terminate the path tracing. In that way, we have a probability $(1-\lambda)^{n-1}\lambda$ to obtain a path of n bounces (to survive $(n-1)$ bounce and to be killed at the nth bounce). By looking at each term of (12), we learn that we should re-weight the resulting color from an n-bounce path by a factor of $\frac{1}{(1-\lambda)^{n-1}\lambda}$. By doing so, the expectation value of the resulting color will unfold into the formula (12), which is the same as what we want (11).

```
 1: $\text{ColorSum}_{ij} = 0$;
 2: for $(i,j) \in \text{Image}, k = 1,2,\ldots,N$ do
 3:     $\text{Color}_{ij,k} = (0,0,0)$; $\text{TotalWeight}_{ij,k} = (1,1,1)$; $\text{Factor} = 1$;
 4:     $n_k = 1$;                                              ▷ Path length
 5:     $\text{Ray}_{ij,k} = \text{RandomRayThruPixel}_{ij}()$;
 6:     while True do
 7:         $\text{hit} = \text{Intersect}(\text{Ray}_{ij,k}, \text{Scene})$;
 8:         if $\text{hit} = \varnothing$ then break;
 9:         $\text{terminate} = \text{random}\begin{cases}1 & \text{probability} = \lambda\\ 0 & \text{probability} = (1-\lambda)\end{cases}$
10:         if terminate then
11:             $\text{Factor} *= \lambda$;
12:             $\text{Color}_{ij,k} += \text{TotalWeight}_{ij,k}(E_{\text{hit}} + \text{FinalDiffuseLighting}_{\text{hit}})$;
13:             break;
```

```
        else
            $n_k$ + +;
            Factor $* = (1 - \lambda)$;
            $\text{Color}_{ij,k} + = \text{TotalWeight}_{ij,k} E_{\text{hit}}$;
            $\text{TotalWeight}_{ij,k} * = W_{\text{hit}}$;        ▹ Specular/Diffuse decision
            $\text{Ray}_{ij,k}$ = bounce into next ray;        ▹ Specular/Diffuse decision
        end if
    end while
    $\text{ColorSum}_{ij} + = \frac{1}{\text{Factor}} \text{Color}_{ij,k}$;
    $\text{PixelColor}_{ij} = \frac{1}{N} \text{ColorSum}_{ij}$.
end for
```

3.2 Specification

This extra credit is open-ended. The minimal requirement is to implement the global illumination, and design your own artistic scene (for example a *Cornell box* (search this keyword online and you will see what that is)). The objects and their materials should be placed to best display the nontrivial results from your global illumination.