Advanced Computer Graphics Coursework

Timothy Coulter

December 13, 2019

1 Introduction

Raytracing is a rendering process which works by tracing a ray, for each pixel, from an eye into a virtual scene. Colour is then calculated for that pixel based on the properties found at the intersection point.

In this report I will give some context for photon mapping by talking about how indirect and global illumination are achieved. I will then demonstrate how photon mapping is done and finally I will talk about caustics.

2 Basic Raytracing

In order to make a raytracer I needed to create a camera. A camera is defined by one vertex and two vectors: a vertex called eye ($\boldsymbol{e}$), a vector called up ($\boldsymbol{u}$) and a vector called look ($\boldsymbol{l}$). From these vectors we create a new orthogonal basis of camera vectors ($\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$). This is done by the following (note we treat vertices as vectors for these calculations):

$$\boldsymbol{w} = \frac{\boldsymbol{e} - \boldsymbol{l}}{||\boldsymbol{e} - \boldsymbol{l}||}$$

$$\boldsymbol{u} = \frac{\boldsymbol{u} \times \boldsymbol{w}}{||\boldsymbol{u} \times \boldsymbol{w}||}$$

$$\boldsymbol{v} = \boldsymbol{w} \times \boldsymbol{u}$$

It should be noted that all these vector have length 1 (normal) and all the vectors are orthogonal to each other ($\boldsymbol{u}.\boldsymbol{v} = \boldsymbol{w}.\boldsymbol{u} = \boldsymbol{w}.\boldsymbol{v} = 0$). So we now have an orthonormal basis of vectors. To create sample rays we do the following (scale is field changes the field of view):

```
for(each pixel (x,y))
{
    x_sample = scale * (x - image_width/2);
    y_sample = scale * (y - image_height/2);

    ray_direction = x_sample * u + y_sample * v - distance * w;
}
```

After making the camera I then looked to create a triangle intersection method which finds a hit if the ray intersects that triangle. The hit contains useful information such as the surface normal, the object we hit, the distance travelled along the ray. The triangle intersection method I used uses Barycentric coordinates. This coordinate system was also used to interpolate the normals of my mesh in order to get a smooth teapot.

Barycentric coordinates define where a point lies in a triangle. Each coordinate is worked out using areas of subtriangles. Suppose we have a triangle ABC with corresponding edges: $\vec{\boldsymbol{AB}}, \vec{\boldsymbol{AC}}, \vec{\boldsymbol{BC}}$ and a point lying in the triangle called $\boldsymbol{P}$. Now we can calculate the area of the subtriangles $\boldsymbol{APB}, \boldsymbol{BPC}$ and $\boldsymbol{CPA}$. Then we calculate the area of the triangle $\boldsymbol{ABC}$. I will represent the area of a triangle with $A()$. We can now define our Barycentric coordinates for $\boldsymbol{P}$ as follows:

$$Bary(\boldsymbol{P}) = \left(\frac{A(\boldsymbol{APB})}{A(\boldsymbol{ABC})}, \frac{A(\boldsymbol{BPC})}{A(\boldsymbol{ABC})}, \frac{A(\boldsymbol{CPA})}{A(\boldsymbol{ABC})}\right)$$

Where:

$$A(\boldsymbol{ABC}) := \frac{||\vec{\boldsymbol{AB}} \times \vec{\boldsymbol{AC}}||}{2}$$

These coordinates : (t_1, t_2, t_3) have the nice property that $t_1 + t_2 + t_3 = 1$ and $t_1, t_2, t_3 \in [0, 1]$. So they are ideal for smoothing normals and detecting if a point is in a triangle.
Essentially, if $t_1 + t_2 + t_3 \neq 1$ or $t_1, t_2, t_3 \notin [0, 1]$ we know $\boldsymbol{P}$ is not in the triangle.
In order to interpolate normals we need to firstly find the normals for every vertex in our mesh. These normals are defined to be the average of the normals of the 3 triangles which connect to that vertex. Now we consider a point in a triangle: $\boldsymbol{ABC}$ with corresponding Barycentric coordinates (t_1, t_2, t_3). We have vertex normals: $\hat{\boldsymbol{N}}_{\boldsymbol{A}}, \hat{\boldsymbol{N}}_{\boldsymbol{B}}, \hat{\boldsymbol{N}}_{\boldsymbol{C}}$. The larger t_1 is the more we want $\hat{\boldsymbol{N}}_{\boldsymbol{C}}$ to affect our points normal – when $t_1 = 1$ our point is at $\boldsymbol{C}$. So we end up with an interpolated normal: $\boldsymbol{N} = t_1\hat{\boldsymbol{N}}_{\boldsymbol{C}} + t_2\hat{\boldsymbol{N}}_{\boldsymbol{A}} + t_3\hat{\boldsymbol{N}}_{\boldsymbol{B}}$ This vector is the normalised to give our interpolated unit-normal: $\hat{\boldsymbol{N}} = \frac{\boldsymbol{N}}{||\boldsymbol{N}||}$. This normal is then used in colour calculations and as it is now interpolated the colour changes are smoother and more realistic.

3 Local illumination

Now we have hit information for a certain pixel we can calculate colour. At first I will only consider local lighting. We will use the phong model:

$$I_A k_A + v \sum_{i=1}^{M} I_i (k_D(\hat{\boldsymbol{N}}.\boldsymbol{L}_i) + k_S(\boldsymbol{R}.\boldsymbol{V})^p)$$

Where M is the number of lights, I_A is the ambient light intensity and I_i is the intensity of the i^{th} light, k_D is the diffuse coefficient of the material and k_S is the specular coefficient. $\hat{\boldsymbol{N}}$ refers to the normal of at our hit, $\boldsymbol{L}_i$ is the vector pointing toward the light from the hit. $\boldsymbol{R}$ is the perfect reflection vector and $\boldsymbol{V}$ is the vector pointing towards our eye from our hit. Also v is a visibility flag, if our hit has an object between the hit position and the light then $v = 0$ so we would only consider ambient light. Finally p refers to power and essentially determines how small the specular contribution will be. It should be noted this should be done for red, green and blue separately. This model uses the assumption that ambient light is constant – we will ignore this term when we do photon mapping.

4 Global illumination

Now we have done local illumination we shall consider global illumination. Global illumination takes into consideration light which bounces off other objects instead of just out initial light. Global illumination adds to the local illumination. We essentially make our raytracer recursive in the following way:

```
raytracer(Ray ray, Object *objects, Light *light, int level, Colour &colour)
{
    colour = Colour(0,0,0,0);
    level = level - 1;
    if(level < 0)
    {
        return; // We have recursed too far.
    }

    trace(ray,objects);
    if(intersection == false)
    {
        return; // If we have no intersection return.
    }

    // Compute base colour and add it to colour

    if(kr != 0) // kr is our coefficient of reflection.
    {
        // Find reflection ray
        raytracer(reflection_ray,..., reflection_colour);
        colour.add(kr*reflection_colour);
    }
    if(kt != 0) // kt is our coefficient of refraction.
    {
        // Find refraction ray
        raytracter(refraction_ray,...,refraction_colour);
        colour.add(kt*refraction_colour);
    }
    return;
}
```

To add a little more realism we can implement Fresnel equations. These calculate k_r and k_t based on the index of refraction of the object and the viewing angle. Using this we can also account for total internal reflection. Total internal reflection is caused by viewing a refractive surface from a very shallow angle which causes it to appear reflective.

5 Reflection, Refraction and Fresnel

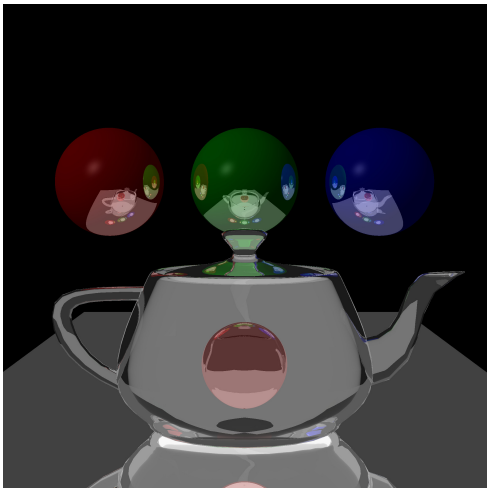


Figure 1: Refractive teapot with total internal reflection.

Above is an image generated using the teapot data set with global illumination using the Fresnel equations. You may note the total internal reflection at the top and sides of the teapot.

The Fresnel equations essentially increase the coefficient of refraction (k_t) of an object if the viewing angle is orthogonal to the surface for example: looking down at a glass of water, the water will appear completely refractive. However, if the viewing angle is almost parallel to the surface (the viewing ray grazes the sides and top of the teapot) then we have total internal reflection – so $k_r = 1$.

Firstly, however we need to find the directions of reflection and refraction. For reflection this is reasonably simple - note we use the reflection law (angle of incidence = angle of reflection):

$$\boldsymbol{R} = \boldsymbol{I} - 2(\hat{\boldsymbol{N}}.\boldsymbol{I})\hat{\boldsymbol{N}}$$

Essentially $(\hat{\boldsymbol{N}}.\boldsymbol{I})\hat{\boldsymbol{N}}$ is the projection of $\boldsymbol{I}$ in the normal direction. So $\boldsymbol{I} - (\hat{\boldsymbol{N}}.\boldsymbol{I})\hat{\boldsymbol{N}}$ is pointing in the normal direction but has a different size based on $\boldsymbol{I}$. So taking away another $(\hat{\boldsymbol{N}}.\boldsymbol{I})\hat{\boldsymbol{N}}$ gives a vector pointing in the reflection direction.

Refraction is slightly more complex and I will just show the equations used:

Refraction relies on Snell's law which states:

$$\frac{sin(\theta_t)}{sin(\theta_i)} = \frac{\eta_2}{\eta_1}$$

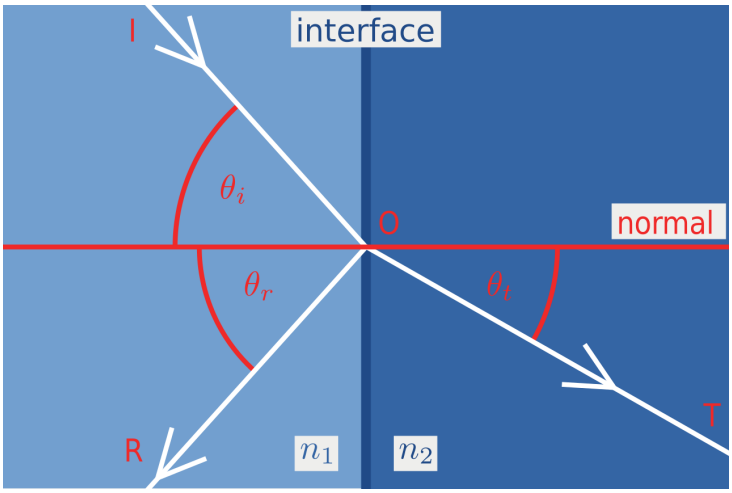


Figure 2: Vectors used for Fresnel calculation and refraction direction [2]

Where η_1 is the index of refraction of the material you start in and η_2 is the index of refraction of the material you end up in. Using this knowledge we find that our transmission ray is:

$$\boldsymbol{T} = \boldsymbol{M} sin(\theta_t) - \hat{\boldsymbol{N}} cos(\theta_t)$$

$$\boldsymbol{M} = \frac{\boldsymbol{I} + cos(\theta_i)\hat{\boldsymbol{N}}}{sin(\theta_i)}$$

Simplifying this using Snell's law and the fact that $cos^2(\theta) + sin^2(\theta) = 1$ we find:

$$\boldsymbol{T} = \eta \boldsymbol{I} + (\eta cos(\theta_i) - cos(\theta_t))\hat{\boldsymbol{N}}$$

Where $\eta = \frac{\eta_1}{\eta_2}$. Now using the fact that $cos(\theta_i) = \hat{\boldsymbol{N}}.\boldsymbol{I}$ and $cos(\theta_t) = \sqrt{1 - \eta^2(1 - cos^2(\theta_i))}$ we have:

$$\boldsymbol{T} = \eta \boldsymbol{I} + \left(\eta \hat{\boldsymbol{N}}.\boldsymbol{I} - \sqrt{1 - \eta^2(1 - (\hat{\boldsymbol{N}}.I)^2)}\right)\hat{\boldsymbol{N}}$$

Now we have both our reflection ray direction and our refraction ray direction we can look at Fresnel equations. We have a general concept of why the Fresnel equations are used so I will just show how to calculate to coefficients k_r and k_t. It should be noted that light has two waves which are called parallel and polarised and we calculate the amount of reflected light for each wave as shown below:

$$k_{r_\parallel} = \left(\frac{\eta_2 cos(\theta_i) - \eta_1 cos(\theta_t)}{\eta_2 cos(\theta_i) + \eta_1 cos(\theta_t)}\right)^2$$

$$k_{r_\perp} = \left(\frac{\eta_1 cos(\theta_t) - \eta_2 cos(\theta_i)}{\eta_1 cos(\theta_t) + \eta_2 cos(\theta_i)}\right)^2$$

So finally we have a coefficient of reflection to be the average of these two waves:

$$k_r = \frac{1}{2}(k_{r_\parallel} + k_{r_\perp})$$

Due to the conservation of energy: $k_t = 1 - k_r$
[3]

6 Photon mapping

In this section I will talk about global photon mapping. It should be noted that I use the following library for k-d tree building and manipulation: `https://github.com/jlblancoc/nanoflann`. Firstly, what is photon mapping?

Photon mapping is generally comprised of two steps:

1. Photon emission and storage.
2. Raytracing and radiance estimation.

It is essentially a process in which we approximate ambient light and caustics by sending out photons from a light.

6.1 Photon emission and storage

In my case I used a diffuse light which outputs photons in every direction. It is possible to send photons from the light in certain directions e.g. straight down. These photons are traced in a similar way to our raytracer. When a photon hits an object we decide what to do with it using Russian Roulette. Russian Roulette means instead of producing multiple photons whenever a photon interacts with a surface we instead pick what we do with a random number and probabilities. Essentially, a material will have specular colour and a diffuse colour these colours determine the probability of a photon having a diffuse interaction, a specular interaction or absorption. These probabilities are calculated as follows (p_r is the probability of reflection):

$$p_r = max\left(k_d^r + k_s^r, k_d^g + k_s^g, k_d^b + k_s^b\right)$$

Note: the d stands for diffuse the s stands for specular and r, g, b represent red, green and blue. Now let $D = k_d^r + k_d^g + k_d^b$ and $S = k_s^r + k_s^g + k_s^b$. We can now compute our probabilities of diffusion and specular transmission (p_d, p_s):

$$p_d = p_r \frac{D}{D+S}$$

$$p_s = p_r \frac{S}{D+S} = p_r - p_d$$

We then, using a uniform distribution, obtain a random number (ξ) between 0 and 1 and use that number to determine the type of interaction. The table below shows some useful information about each interaction. Note that photons are reflected until they are either absorbed or we have done a certain number of reflections.

Type of interaction	Diffusion	Specular	Absorbtion
Russian Roulette	$0 \leq \xi \leq p_d$	$0 < \xi \leq p_d + p_s$	$p_d + p_s < \xi \leq 1$
Is photon stored?	Yes	No	Yes
New photon transmitted?	Yes	Yes	No
New photon power is:	Lambertian	Unchanged	N/A
New photon direction:	Random	Reflection or refraction	N/A

We store the position, incident direction and power for each photon in an array. This array is then turned into a k-d tree. A k-d tree is essentially a tree which takes k-dimensional vectors (in our case we use 3 dimensions). The top of the tree will be a point which splits $\mathbb{R}^3$ into 2 subsets $\left(U, V \subset \mathbb{R}^3\right)$. If a point lies is U then it will go into the left sub-tree and if it lies in V it will go into the right sub-tree. These new points divide U and V into more subsets and this process continues. Now we have a global photon map and tree we can begin the 2nd part of photon mapping. [1]

6.2 Raytracing and radiance estimation

It should be noted that photon mapping also requires local illumination and global illumination to make an image. We compute the local (direct) and global colours the same as before but we add the photon mapping contribution to our colour. For each hit position we find the nearest N photons and work out the BRDF for each photon (I used phong without an ambient term). These photon contributions are summed and then divided by the radius of the smallest circle enclosing all the photons. This means if the photons are spread out their contribution to the overall intensity will be smaller than if the photons were really close together. Below is some pseudocode to show how we raytrace with photon mapping:

```
raytrace(ray)
{
    trace(ray,hit);
    // As shown before compute direct and global light.
    // Note that we neglect any ambient terms.
    direct_colour = ...
    global_colour = ...
    // Finds N nearest neighbours for our hit position.
    tree.find_nearest_neighbours(hit.position,N);

    for(each photon found)
    {
        // Use information stored to compute Phong for each photon.
        photon_colour += Phong(photon);
    }
    photon_colour = photon_colour/circle_radius;
    pixel_colour = direct_colour + global_colour + photon_colour;
}
```

The expression for reflected radiance at surface point $\boldsymbol{x}$ in direction $\vec{\boldsymbol{\omega}}$ if given as follows:

$$L_r(\boldsymbol{x}, \vec{\boldsymbol{\omega}}) = \int_{\Omega_x} f_r\left(\boldsymbol{x}, \vec{\boldsymbol{\omega}}', \vec{\boldsymbol{\omega}}\right) L_i\left(\boldsymbol{x}, \vec{\boldsymbol{\omega}}'\right) |\hat{\boldsymbol{N}}_{\boldsymbol{x}} \cdot \vec{\boldsymbol{\omega}}| d\vec{\boldsymbol{\omega}}'$$

Note that: f_r is the BRDF, L_i is the incident radiance at x and $\vec{\boldsymbol{\omega}}'$ is the direction of incoming radiance. Also, Ω_x is the hemisphere orientated in the normal direction.

We can simplify this term to get to an expression that we can compute:

$$L_r\left(\boldsymbol{x}, \vec{\boldsymbol{\omega}}\right) = \sum_{i=1}^{n} f_r\left(\boldsymbol{x}, \vec{\boldsymbol{\omega}}_i, \vec{\boldsymbol{\omega}}\right) \frac{\Delta\Phi_i}{\pi r^2}$$

So this equation is used for each pixel. Every time we trace a ray for a pixel we compare the hit position to our k-d tree and find n photons. We compute the BRDF for each photon and at their contributions up. Each photon has power: $\Delta\Phi_i$ we divide that power by the area of the smallest circle containing all the photons (πr^2). [1]

7 Caustics

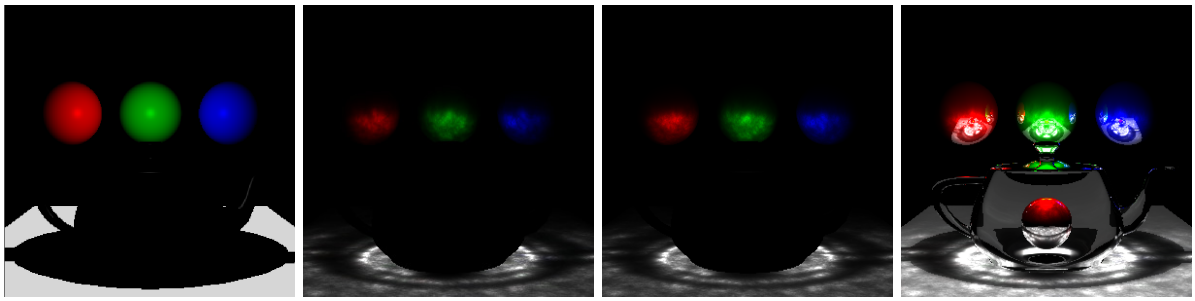
Caustics are the projection of specular reflections or refractions onto a diffuse surface. For example when you shine a light into a mug you should see a cardioid (love heart) appear at the bottom. Generating caustics involves the introduction of a caustic map. For our global map photons are emitted from a light and can take any arbitrary number of reflections before being absorbed. So our global map is: $L\{D|S\}^*D$ where the * means we can have any number of diffuse or specular reflections before absorption. For our caustic map however we only consider LS^+D where the + means 1 or more specular reflections. The important thing to note here is that in our caustic map we are ignoring a lot of photon interactions which means we can more efficiently get sharper caustics. [1]

To add caustics I created a new emit photon function. This function still emits photons in random directions, however it calls my photon tracer with two counters photon_trace(...,int diffuse_counter, int specular_counter). The diffuse_counter is set to 1 and the specular counter is set to 0. If I have a specular reflection then the specular counter is increased by 1. If I have a diffuse reflection (after a specular reflection) the diffuse counter is set to 0 and we stop tracing that photon. The way I determine that I have at least one specular reflection is by increasing the specular counter by 1 and checking the counter is greater than 0 if I have a diffuse reflection before storing it in the caustic map.

One problem with caustics is that they are very bright. This means when we go to plot using framebuffer, all the light in the scene is scaled down so the brightest caustic has intensity of 1. This issue is solved by choosing the correct photon power when emitting - it took a lot of time to get this right. I found most success setting the photon power to be: 1/(num_of_photons*0.005).

8 Image

In the images below we see how we form a final image. Direct lighting and global illumination are added to the global and caustic photon maps. It should be noted the the caustic and global photon map look very similar - this is due to the fact that one caustic map adds a very bright caustic under the red sphere when global illumination is added. Also, the teapot is not illuminated in the direct lighting because the teapot diffuse coefficients are all 0 and my light is above the teapot so $\boldsymbol{R} \cdot \boldsymbol{V} \approx 0$.



Direct lighting | Global photon map | Caustic photon map | Final image

References

[1] Henrik Wann Jensen. A practical guide to global illumination using photon maps, July 2000.

[2] `http://en.wikipedia.org/wiki/File:Fresnel.svg`. Vectors used in fresnel equations, 2012.

[3] `https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/reflection-refraction-fresnel`. Introduction to shading, 2019.