Advanced Computer Graphics Report

John Steward

December 12, 2022

1 Reflection and Refraction

If the ray being emitted hits an object with a "global material", that object has reflective and refractive properties, and so the light behaves differently to that of a diffuse object. The reflective and refractive components of this object are calculated separately to each other, then added to the returned colour using the Fresnel term as a weight.[3]

1.1 Reflection

If a ray hits a reflective surface, A secondary reflected ray will be created using the formula:

$$R = E - 2(N.E)N$$

where R is the reflected ray direction, E is the incident ray direction, and N is the normal at the point of intersection between the incident ray and the object. If θ_i is the incident angle of the ray to the normal and θ_t is the reflected angle of the new ray to the normal, then $\theta_i = \theta_t$ (See figure 1).

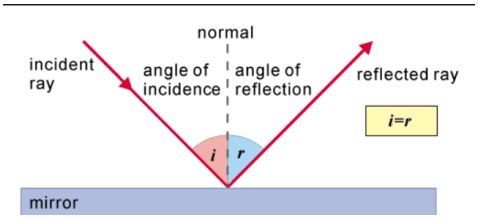

Figure 1: Diagram showing a reflected ray

A small bias is added to this new ray to avoid any noise from the ray intersecting with the same object again. This bias is calculated as a very small addition to the intersection position of the ray in the direction of R. This new ray is recursively traced until either it hits a diffuse surface, or it hits the recursion limit.
Once the ray hits a diffuse surface, the colour of the pixel that it hits is returned and therefore the colour of the diffuse object that is hit is now seen on the reflective surface. This means that any colour shown on a purely reflective surface is just a reflection of a diffuse object (See figure 7 in Appendix).

1.2 Refraction

If a ray hits an object that is refractive, a new secondary ray will be calculated using Snell's law:

$$T = 1/\eta * I - (\cos\theta_t - (1/\eta) * \cos\theta_i) * N$$

where

$$\cos\theta_t = \sqrt{1 - (1/\eta^2) * (1 - \cos^2\theta_i)}$$

and

$$\cos\theta_i = N.I$$

where T is the refracted ray direction, θ_i is the incident angle, θ_t is the refracted angle, η is the index of refraction($\eta = \frac{\eta_{in}}{\eta_{out}}$)(Snell's law), N is the normal at the point of intersection and I is the incident ray direction. If $\eta_{out} > \eta_{in}$, then the refracted angle θ_t will be less than the incident angle θ_i (See figure 2).

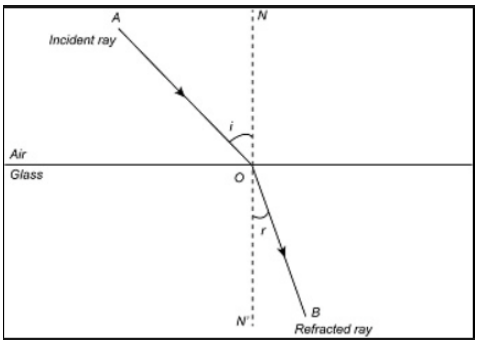

Figure 2: Diagram showing a refracted ray

For surfaces that are both reflective and refractive, a Fresnel term must be calculated as a weighting for both of these properties:

$$r_{per} = (\eta \cos\theta_i - \cos\theta_t)/(\eta \cos\theta_i + \cos\theta_t)$$

$$r_{par} = (\cos\theta_i - \eta \cos\theta_t)/(\cos\theta_i + \eta \cos\theta_t)$$

$$k_r = \frac{1}{2}(r_{per}^2 + r_{par}^2)$$

This k_r value (the Fresnel term)[3] is then used as a weighting for the colour that is returned by the ray, where the reflective component is multiplied by k_r, and the refractive component is multiplied by $(1 - k_r)$ (See figure 8 in appendix). The refractive component is ignored if $\sin\theta_t = 1$, as this means there is total internal reflection, and so the light cannot possibly refract. In the case of total internal reflection, $k_r = 1$ so the refractive component is multiplied by 0.
I faced some problems while implementing the refraction, as the center of the sphere I was testing on would not refract the light correctly. This was fixed by using a fresh colour variable to store and add to the resultant colour for each ray, and this meant that there was no additional colour accidentally leaked into the refraction method.

2 Quadratic Surfaces

A quadratic surface is an implicit surface that is defined by:

$$ax^2 + 2bxy + 2cxz + 2dx + ey^2 + 2fyz + 2gy + hz^2 + 2iz + j = 0$$

The equation for a ray is:

$$R = P + D(t)$$

Where R is the ray, P is the original position of the ray, D is the direction of the ray and t defines a point along the ray[2].
The ray's three components can be expressed by:

$$x = P_x + D_x(t)$$

$$y = P_y + D_y(t)$$

$$z = P_z + D_z(t)$$

Substituting these into the quadratic surface equation, we have:

$$A_q t^2 + B_q t + C_q = 0$$

where:

$$A_q = aD_x^2 + 2bD_xD_y + 2cD_xD_z + eD_y^2 + 2fD_yD_z + gD_z^2$$

$$B_q = 2(aP_xD_x + b(P_xD_y + D_xP_y) + c(P_xD_z + D_xP_z) + dD_x + eP_yD_y + f(P_yD_z + D_yP_z) + gD_y + hP_zD_z + iD_z)$$

$$C_q = aP_x^2 + 2bP_xP_y + 2cP_xP_z + 2dP_x + eP_y^2 + 2fP_yP_z + 2gP_y + hP_z^2 + 2iP_z + j$$

A quadratic surface is convex, so there is a maximum of two intersections per ray. If there are no intersections (the ray does not pass through the object), then the discriminant ($B_q^2 - 4A_qC_q$) will be negative[2].
If $A_q = 0$, then there is only one intersection between the ray and the object, meaning the ray is at a tangent to the object, and the intersection can either be ignored, or can be calculated by:

$$t = \frac{-C_q}{B_q}$$

If the ray intersects with the object twice, then the intersections can be calculated using the quadratic formula:

$$t_0 = \frac{-B_q - \sqrt{B_q^2 - (4A_qC_q)}}{2A_q}$$

$$t_1 = \frac{-B_q + \sqrt{B_q^2 - (4A_qC_q)}}{2A_q}$$

Where t_0 is the closest intersection[2].
If both t_0 and t_1 are negative, then we do not need to render the object, as it is behind the camera. If only t_0 is negative, then the camera is inside the object and we return the intersection using t_1. if both are positive, then we return the intersection using t_0, with the intersection using t_1 being stored as the next in the hit list so we can use the quadratic object in CSG operations (See figure 9 in Appendix).

3 Constructive Solid Geometry (CSG)

A CSG is a type of object that is the result of some operation on two sub-objects. There are three main operations used in CSGs: Union, Intersection and Difference[5].

3.0.1 Union

Union is the simplest of these operations. It takes the two sub-objects and simply returns the closest intersection with the ray[5]. As the intersections are stored in a linked list, when we return the closest intersection, we also return any subsequent intersection from the ray with that object. This means that the code will not crash when a CSG operation is being performed on a CSG. This is also the case with both intersection and difference. Since all you see is the closest hit, union is seemingly identical to placing the sub-objects within the scene separately, however they now act as one object (See figure 10 in appendix).

3.0.2 Intersection

This operation takes the two sub-objects and returns only the section where the sub-objects overlap[5]. Essentially, this means that it only returns a hit if the ray passes through both objects and, if it does, it returns the closest hit (See figure 11 in appendix). To check that the ray passes through both objects, we compare the intersections with each sub-object and, if the closest intersection with the first object is lower than the closest intersection with the second, and the furthest intersection the first object is greater than the closest intersection with the second, we return the closest intersection with the second sub-object and vice versa. If neither of these is the case, we return nothing. In order to compare these intersections, we compare the t values that correspond to them, as seen in figure 3:

```
if (leftHit->t < rightHit->t && leftMax->t > rightHit->t){
    return rightHit;
}
else if (rightHit->t < leftHit->t && rightMax->t > leftHit->t){
    return leftHit;
}else{
    return 0;
}
```

Figure 3: Code snippet to check that the sub-objects overlap

3.0.3 Difference

This operation takes the two sub-objects and returns the first without any section where they overlap[5]. Essentially, If the ray only intersects with the first sub-object, then we return the closest intersection, and if it only intersects with the second sub-object, we return nothing. It also returns the closest intersection with the first sub-object if it is closer than the closest intersection with the second, and returns the furthest intersection with the second sub-object if it is closer than the furthest intersection with the first sub-object (See figure 12 in appendix). To check this, we compare the t values of the closest and furthest intersections with each sub-object as seen in figure 4:

```
if (leftHit->t < rightHit->t){
    return leftHit;
}else if (rightMax->t < leftMax->t){
    return rightMax;
}else{
    return 0;
}
```

Figure 4: Code snippet for the difference operation

4 Photon Mapping

4.1 Point Lights

In order for us to implement a photon map, we must first implement a point light. Unlike directional lights, which only includes direction and intensity, point lights additionally have a position attribute, with light being sent out in all directions. The direction for the point light is calculated procedurally in the raytrace method when a hit is registered by finding the difference between the point of intersection between the ray and the object, and the position of the point light. This implementation of point lights then allows us to emit photons in random directions that originate from the position of the point light[1].

4.2 First Pass

During the first pass, n photons are emitted in random directions from the point light into the fully built scene. The direction of each photon is generated using the formula in Figure 5:

```
while(!valid){
    //Random floats between 0 and 1;
    a = static_cast <float> (rand()) / static_cast <float> (RAND_MAX);
    b = static_cast <float> (rand()) / static_cast <float> (RAND_MAX);
    c = static_cast <float> (rand()) / static_cast <float> (RAND_MAX);
    x = 2*a-1;
    y = 2*b-1;
    z = 2*c-1;
    if (x*x+y*y+z*z > 1){
        valid = true;
    }
}
direction.x = x;
direction.y = y;
direction.z = z;
```

Figure 5: Code snippet showing generation of the photon directions

These photons are given a flux (intensity) of 1/(number of photons emitted), then they are traced and, if a photon hits an object, one of three outcomes can occur: the photon can either be reflected, refracted or absorbed, and this occurs randomly through a process called "Russian Roulette"[4], where each material has a weighted chance to absorb, reflect or refract a photon. If the photon hits a diffuse object, it can only be absorbed or reflected, and if it hits a reflective/refractive object, it can only be reflected or refracted.
Photons are initially emitted as "direct" photons and, as they reflect and refract off objects, they become "indirect" photons. Once the photon is absorbed by a diffuse object, it is stored within a KD-tree to be accessed in the second pass as the "global photon map"[4]. When a photon is reflected off a diffuse object, it is sent in a random direction such that the dot product of this new direction and the normal at the point of intersection is positive. Once a direct photon comes into contact with an object, the path of that photon is further traced and we store "shadow" photons at any subsequent intersections with objects, to be accessed later in the second pass (See figure 6), and these shadow photons are stored with a flux of 0. These shadow photons allow us to reduce the number of times we need to send a shadow ray in the second pass. If the photon tracer hits its recursion limit before the photon is absorbed by an object, it is dissipated and therefore not stored (See figure 13 in appendix). When a photon reflects off a diffuse object, its flux takes on a portion of the colour from that object, introducing "colour bleeding" to the scene (See figure 14 in appendix).
Caustics are rendered in a similar way to the photons in the global photon map, however, in this case, any direct photons that hit a diffuse surface are immediately dissipated and any indirect photons that hit such a surface are immediately absorbed and stored within a separate KD-tree as the "caustic photon map"[4]. When these photons are generated, they are also given a stronger flux than the photons that are sent into the global photon map. In this case, the caustic photons are given a flux of 1. The intensity of these photons are adjusted in the second pass to give a more realistic image. Caustic photons can only be stored in the caustic KD-tree if they have first been reflected or refracted (See figure 15 in appendix).
The caustic photons are stored in a separate KD-tree as they need to be accessed separately to the global photon map.

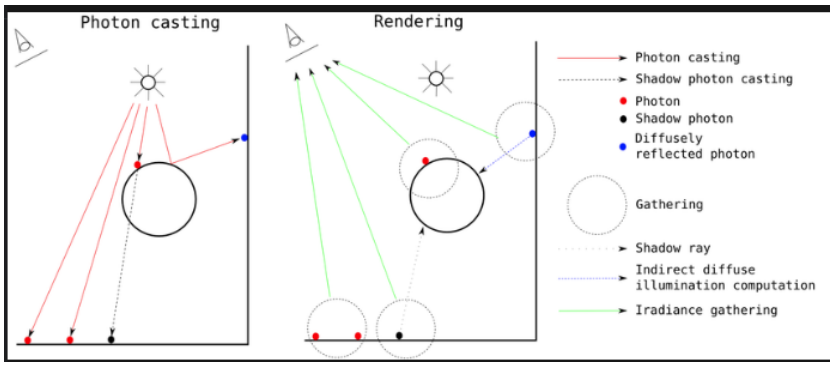

Figure 6: Diagram to show photon emission onto a diffuse surface

4.3 Second Pass

During the second pass, we use traditional ray tracing and, when the ray hits an object that is not reflective or refractive, it returns the colour of the object with a scaled flux of the photons in a predefined radius around the intersection point. If any of the photons included within this are shadow photons, then we use the "shadowtrace" method to check if there is an object blocking the light from that point and, if there is, we compute a shadow on that point[4]. The global photon map and the caustic photon map are searched separately, and the flux from the global and caustic photons are also added to the scene separately. These maps are searched separately so there is no interference between global and caustic photons, and we can achieve a more realistic image (See figure 16 in appendix). Photon mapping allows us to render much more complex lighting during the second pass, such as caustics and colour bleeding which, without this method, would be unachievable.

References

[1] Bacterius, *Shadows and point lights*, https://www.gamedev.net/blogs/entry/2260865-shadows-and-point-lights/, February 2015.

[2] K. Cameron, *Cm30075: Advanced graphics short note raytracing quadratic surfaces*, UK, Bath, November 2022.

[3] *Introduction to shading (reflection, refraction and fresnel)*, https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/reflection-refraction-fresnel.

[4] H. W. Jensen, *Global illumination using photon maps*, https://moodle.bath.ac.uk/pluginfile.php/1288264/mod_resource/content/0/Jensen_PhotonMaps.pdf, 1996.

[5] *Ray tracing csg tracing csg models*, http://web.cse.ohio-state.edu/~parent.1/classes/681/Lectures/19.RayTracingCSG.pdf, Ohio, USA.

5 Appendix

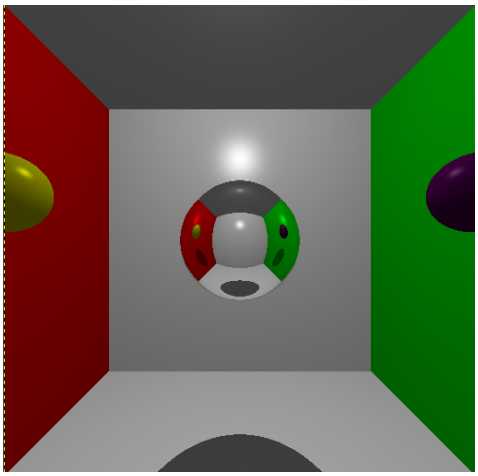

Figure 7: Purely reflective sphere

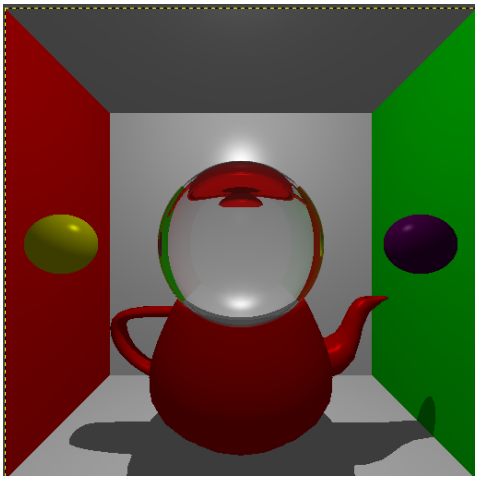

Figure 8: Sphere with both reflective and refractive properties

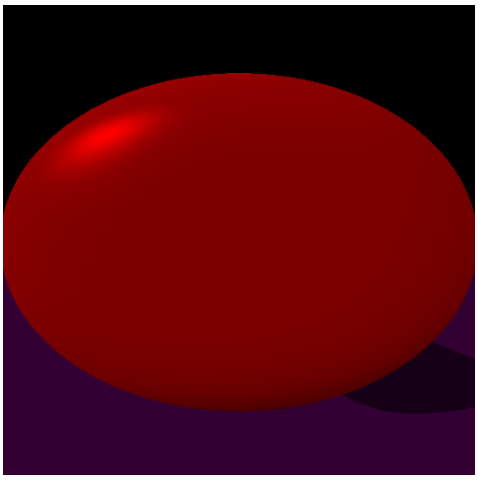

Figure 9: An ellipsoid made from a quadratic object

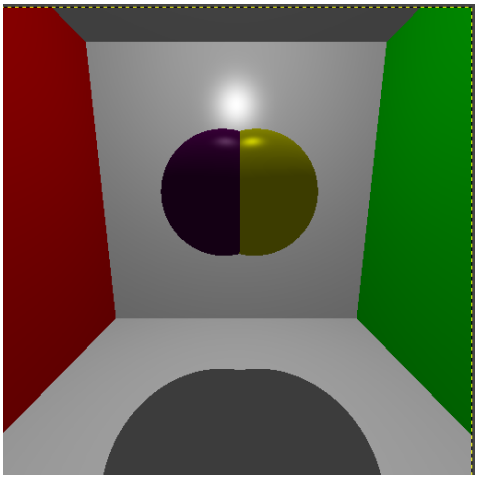

Figure 10: Two spheres used in a union operation

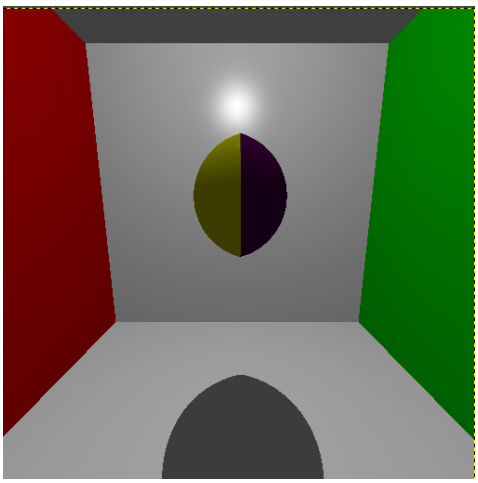

Figure 11: Two spheres used in an intersection operation

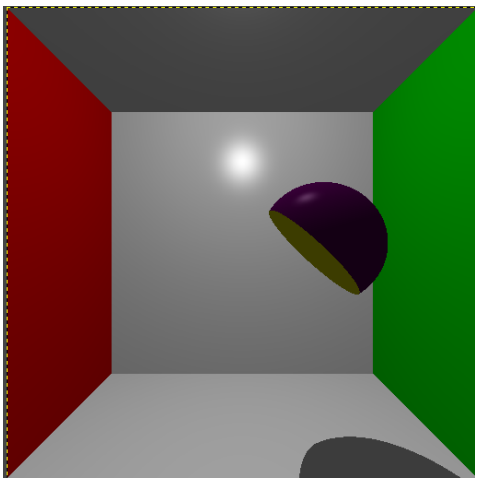

Figure 12: Two spheres used in a difference operation

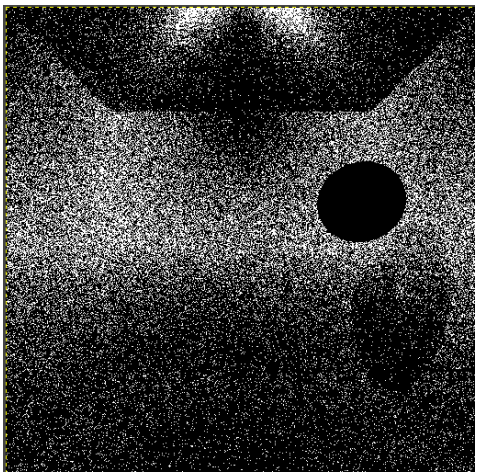

Figure 13: An image showing where the photons are absorbed into the global photon map

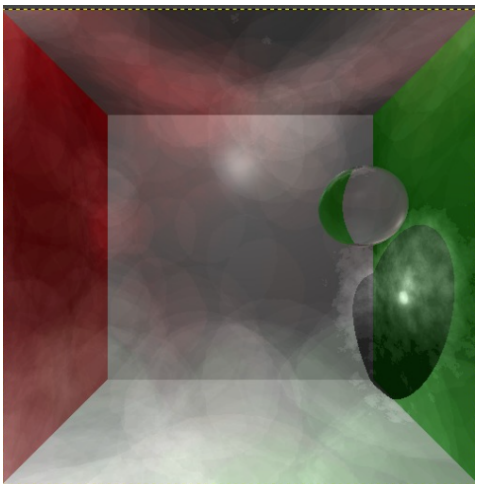

Figure 14: An image using photon mapping with colour bleeding exaggerated to easily show its functionality

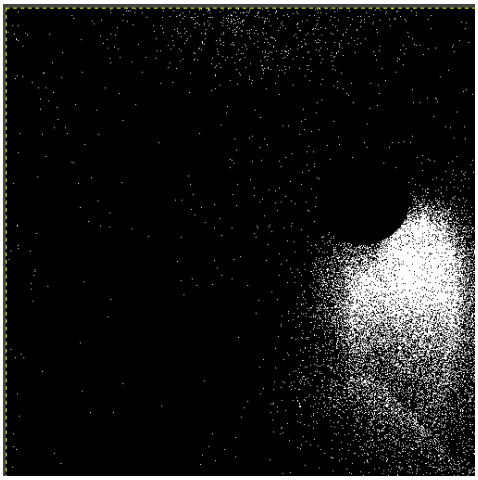

Figure 15: An image showing where the photons are absorbed into the caustic photon map

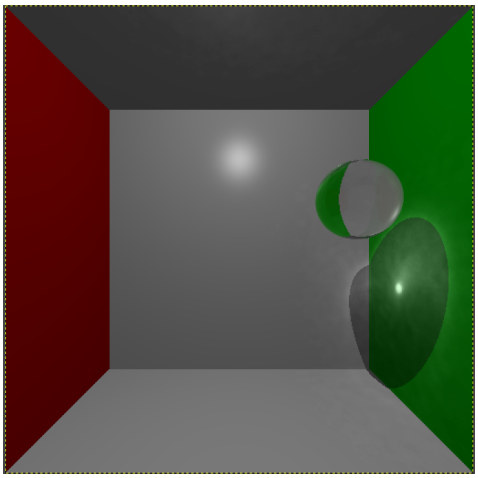

Figure 16: An image of a glass sphere using caustic photon mapping