Advanced Computer Graphics Report (CM30075)

Louie Middle
lm2166@bath.ac.uk

September 11, 2023

Contents

Chapter 1

Global Lighting

1.1 Reflection

If a ray is to be reflected, it bounces back in a direction which is symmetrical to the incident direction about the surface normal at the hit point i.e. if the angle between the incident direction and the surface normal is denoted θi and the angle between the reflected direction and the surface normal is θr, then $\theta i = \theta r$ as shown in 1.1 Scratchapixel (2022).

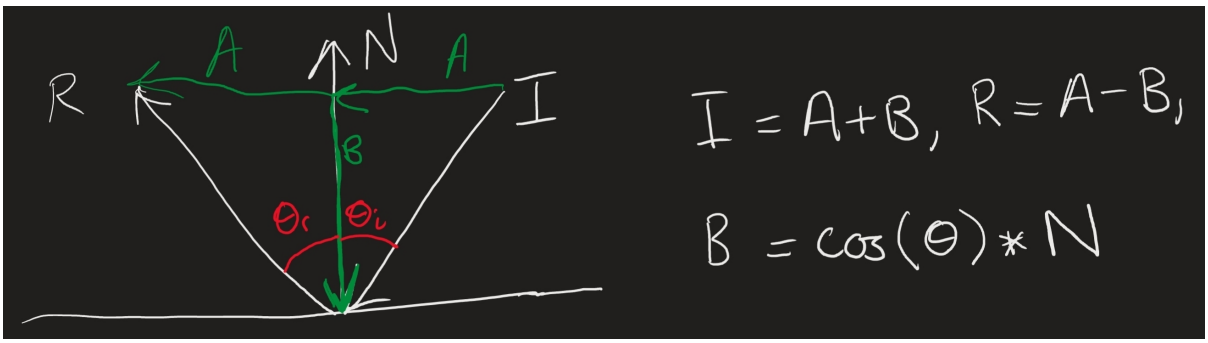

Figure 1.1: Reflection Diagram

We know $cos\theta = N \cdot I$. We can say $I = A + cos\theta * N$ and $R = A - cos\theta * N$. Rewrite the first equation to get $A = I - cos\theta * N$. Substitute A into R to get 1.1 Scratchapixel (2022).

$$R = I - cos\theta * N - cos\theta * N = I - 2cos\theta * N = I - 2(N \cdot I) * N \tag{1.1}$$

To simulate reflection in a ray-tracer, if the object hit is reflective, compute a reflection ray using 1.1 and recursively ray trace again. Note that this will only produce perfect reflections, not blurry or glossy reflections Scratchapixel (2022). Using a recursive depth of five prevents infinite reflections and wasted compute, but it maintains a good quality reflection. See A.1 for improved reflection over many recursions. It is important to shift the hit point by some small ϵ in the direction or opposite direction of the surface normal. Which direction depends on if the ray is inside or outside the object. This is because limited floating point precision can cause some hits to be mistakenly treated as being beneath the surface of the object, thus leading to self intersection (similarly to shadows in the labs).

1.2 Refraction

Refraction is when a ray bends at the boundary between two different mediums (for example air to glass or air to water). The new direction depends on the refractive indexes of the two mediums Scratchapixel (2022).

When light travels through a vacuum it travels at the speed of light, c, but when it travels through another medium its speed decreases to v. The index of refraction for the medium, η is the ratio $\eta = c/v$ Scratchapixel (2022). The refractive index of air is 1.0003 and in graphics this is usually treated as a vacuum with $\eta = 1$. This is how air is treated in the implementation.

Refraction is described by Snell's law $sin\theta_1/sin\theta_2 = \eta_2/\eta_1$. Figure 1.2 shows various geometric deductions that can be used to get T, the refracted ray direction. Eventually, T derives to $T = \eta I + (\eta cos\theta_1 - cos\theta_2) * N$ where $cos\theta_1 = I \cdot N$ and $cos\theta_2 = \sqrt{1 - (n_1/n_2)^2(1 - cos^2\theta_1)}$. By default the implementation assumes the starting medium is air. There is a possibility of a ray's starting medium being an object. Knowing if the ray starts inside an object can be done by checking the sign of the dot product between the hit normal and the incident ray. If it is positive the ray is already inside the object and the hit normal needs to be negated and the refraction indexes swapped (so the exiting medium is air). If the dot product is negative, negate the value so it is positive. Total internal reflection has occurred if the value in the square root for $cos\theta_2$ is negative and the method can return early Scratchapixel (2022).

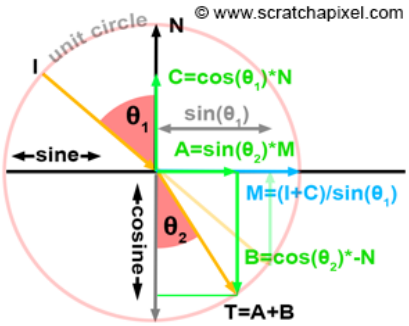

Figure 1.2: Compute refraction ray using geometry Scratchapixel (2022)

1.2.1 Fresnel Terms

Transparent objects such as glass will reflect as well as refract light. The amount of light objects reflect (FR) or transmit (FT) depends on the angle of incidence and is given by the Fresnel terms $FR_{parallel} = (\frac{\eta_2 cos\theta_1 - \eta_1 cos\theta_2}{\eta_2 cos\theta_1 + \eta_1 cos\theta_2})^2$ and $FR_{perpendicular} = (\frac{\eta_1 cos\theta_2 - \eta_2 cos\theta_1}{\eta_1 cos\theta_2 + \eta_2 cos\theta_1})^2$ where $FR = \frac{FR_{parallel} + FR_{perpendicular}}{2}$, or more simply the ratio of the parallel and perpendicular waves that compose light. Due to the conservation of energy, $FT = 1 - FR$. In the case of total internal reflection $FR = 1$ Scratchapixel (2022). See A.2 for refraction example.

Like reflection, the above can generate refraction rays and recursively ray tracing enables the generation of refractive objects. A recursive depth of five was once again used.

Chapter 2

Modelling

2.1 Quadratic Surfaces

A quadratic surface is an implicit surface is given by 2.1 Cameron (2022b). Examples of a quadratic surface include ellipsoids, cones and hyperboloids.

$$ax^2 + 2bxy + 2cxz + 2dx + ey^2 + 2fyz + 2gy + hz^2 + 2iz + j = 0 \tag{2.1}$$

2.1.1 Intersection

Given a ray defined by its three axes in parametric form $x = P_x + tD_x$, $y = P_y + tD_x$ and $z = P_z + tD_z$, the intersection $I = [x_i, y_i, z_i]$ of a ray and a quadratic surface can be found by substituting these equations into 2.1 giving 2.2, where A_q, B_q and C_q are equations composing the constants in 2.1, P and D Cameron (2022b).

$$A_q t^2 + B_q t + C_q = 0. \tag{2.2}$$

2.2 is a quadratic and can be solved using the quadratic formula to get values for t. If $A_q = 0$ then there is only one intersection point. If the discriminant is negative there is no intersection Cameron (2022b).

2.1.2 Normals

The normal $N = [x_n, y_n, z_n]$ can be calculated using $x_n = ax_i + by_i + cz_i + d$, $y_n = bx_i + ey_i + fz_i + g$ and $z_n = cx_i + fy_i + hz_i + i$. It is possible the normal will be pointing away from the surface the camera is looking at. To check this, compute the dot product of the normal and the ray direction and check if this is greater than zero, which means the angle is less than 90° (as $cos\theta > 0$ when $\theta < 90°$). This means the normal is pointing in the same direction as the ray. Negating the normal fixes this Cameron (2022b).

An implementation to transform the object was also completed as described in Cameron (2022b). See A.3 for render.

2.2 Constructive Solid Geometry

Constructive solid geometry (CSG) allows the creation of complex objects by using Boolean operators to combine simpler objects. The operations implemented are union, intersection and difference. A CSG object can be defined by two child objects (left and right) and the operation to be performed on them. The child objects can themselves be CSG objects Cameron (2022a).

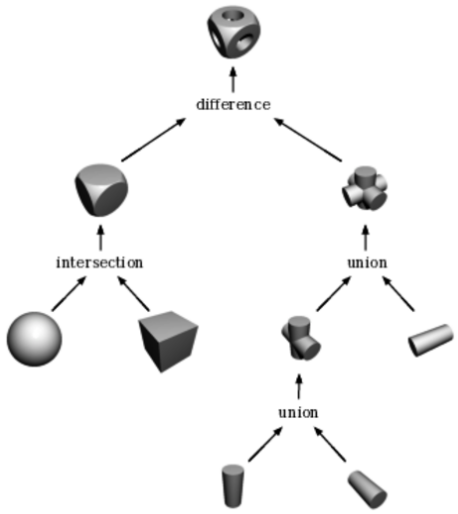

Figure 2.1: CSG building complex objects

2.2.1 Filtering Intersections

Intersecting a ray with a CSG shape begins with intersecting the ray with the shape's children. Then iterate over the resulting intersections, tracking which ones are inside which child and filtering out those that don't conform to the current operation. The resulting list of intersections is then returned Buck (2019).

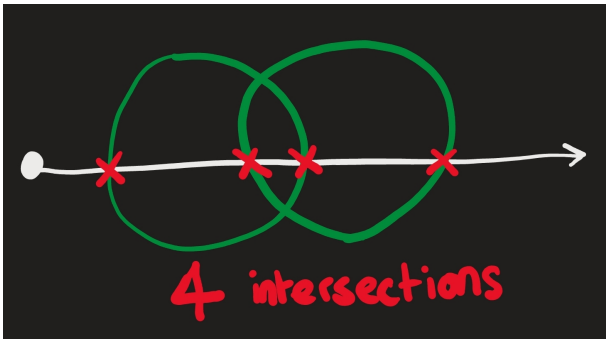

Figure 2.2: Ray intersections with two spheres

To achieve this, the goal is to implement a function that will return a Boolean if the current CSG Boolean operation is allowed. This function would be passed in whether the intersection intersects with the left or right child object, and whether it is currently inside the left or right objects. For example, the first intersection for 2.2 would hit the left object, and is not in either the left or right object. The second intersection hits the right object and is currently in the left object but not the right object (and this would continue for all intersections). See 2.1 as an example truth table for the union operator and whether an intersection is allowed.

More simply, consider what the union operation should actually return. That is the intersections that are not inside another object. If the hit is on the left object, it must not also be inside the right, and if the hit is on the right, it must not be inside the left. This method can be extended for all CSG operations and their truth tables Buck (2019).

Operation	Left Hit	In Left	In Right	Result
Union	True	True	True	False
Union	True	True	False	True
Union	True	False	True	False
Union	True	False	False	True
Union	False	True	True	False
Union	False	True	False	False
Union	False	False	True	True
Union	False	False	False	True

Table 2.1: Example intersection allowed for union operator

To determine what intersections to return for the parent CSG, get all the intersections for all the child objects. Then sort this list by distance and filter the intersections for those allowed given the current CSG operation using the rules described above. See A.4 for renders.

Chapter 3

Photon Mapping

Photon mapping is technique used to model the behaviour of photons emitted from light sources within a scene Jensen (1996). It is a two pass algorithm. The first pass distributes photons through a scene and a second pass renders the scene using ray tracing with the use of the photons for radiance estimations. Photons are objects with intensity, direction and type. The type can be normal photons, shadow photons, or caustic photons.

3.1 Constructing the Photon Maps

During the first pass, photons are emitted from all the light sources in a scene. The photon is then traced through the scene, similarly to a ray, and at each intersection the photon is stored. The subsequent direction of the photon is determined through Russian roulette as described by Jensen (1996), with it either being reflected or absorbed.

The photons are stored using a KD-tree. Although KD-trees may be used with any number of points (N) and dimensions (NX), they are only more efficient than brute-force methods when $N > 4^N X$. So they are most useful in low-dimensional tasks. $NX = 1$ is another inefficient case, because simple binary search (without additional structures) is much more efficient in such tasks than KD-trees ALGLIB (2010). As a photon map KD-tree is 3 dimensional a KD-tree is an appropriate data structure to store the photon map. This is implemented using ALGLIB, a popular numerical analysis library. Additionally, as mentioned by Jensen (1996), searching for photons using a KD-tree is a good choice as the photons are located in the same parts of the tree.

3.2 Rendering

The final image is rendered using a Monte Carlo ray trace in which the radiance of a pixel is estimated by averaging many sample estimates Jensen (1996). The radiance returned by each ray is computed at the first surface intersected by the ray and it equals the surface radiance computed using the rendering equation Kajiya (1986).

The BRDF can be estimated by the sum of the diffuse and specular reflections. The incoming radiance is the sum of direct lighting from light sources, indirect caustic illumination via specular reflection or transmission and indirect lighting from diffuse reflections Jensen and Christensen (2001).

Nearby photons to a point are gathered by querying the KD-tree using a K Nearest Neighbours approach, which is a built in method of ALGLIB's KD-tree implementation.

A Gaussian filter as described in Jensen and Christensen (2001) was used to scale the intensity of the photon the further away it was from the point. This places greater value on the photons closer to the point. Furthermore, the radiance estimate was scaled based on the volume of all the photons used for the estimate.

See A.5 for an example render without caustics.

3.3 Caustics

As suggested in Jensen (1996), a second photon map was implemented for caustics. The caustic photon map stores photons that are emitted directly towards the transmissive objects in the scene and they are then stored once they hit a diffuse surface. As suggested in Jensen and Christensen (2001) and described in Jensen and Christensen (1995), a projection map was used to focus photons towards the transmissive objects. Currently this implementation will work for objects with a centre point and radius, which includes spheres and polymeshes which had bounding spheres added (see 4.1). A triangle can be created between the light position, the objects centre, and the radius which can be used to construct a cone that encapsulates all the possible directions for a photon to be fired at that object.

Unfortunately caustic effects were not fully implemented. See A.6 for the best caustic results achieved.

3.4 Shadow Photons

To reduce the number of shadow rays required, shadow photons were implemented as described in Jensen (1996). Whenever a photon intersects with an object additional shadow photons are sent from that point. Any additional intersections with objects are stored as shadow photons. When querying the KD-tree for photons, only if over half the photons are shadow photons will a shadow intersection test be carried out.

Unfortunately shadow photons are currently disabled, as they were causing segmentation faults.

Chapter 4

Additional features

4.1 Bounding Spheres

Bounding spheres were added to the polymesh object to decrease render times and for use in the caustic photon map (see 3.3). A polymesh intersection test requires a ray to be intersection tested against every single triangle that composes the polymesh. The Utah teapot used has 2256 triangles, meaning every ray has to compute 2256 intersections tests. A sphere on the other hand, can have its intersections calculated relatively cheaply using arithmetic. By placing polymeshes inside a bounding sphere and checking for intersections with the sphere first, this reduces the number of rays that will check for intersections with all the triangles in the polymesh. Whilst adding initial compute time in the creation of the polymesh, overall this reduces compute time drastically.

The bounding sphere algorithm implemented is that described by Ritter and Glassner (1990). According to Ritter the algorithm is 5-20% larger than the optimum, but has the benefits of being simple to implement.

Bibliography

ALGLIB, 2010. *Alglib manual* [Online]. Available from: `https://www.alglib.net/translator/man/manual.cpp.html#struct_kdtree` [Accessed 2022-12-07].

Buck, J., 2019. *The ray tracer challenge : a test-driven guide to your first 3d renderer*. 1st ed. Pragmatic Bookshelf.

Cameron, K., 2022a. Cm30075: Advanced graphics short note constructive solid geometry. Accessed: 2022-12-04.

Cameron, K., 2022b. Cm30075: Advanced graphics short note raytracing quadratic surfaces. Accessed: 2022-12-03.

Jensen, H. and Christensen, N., 2001. A practical guide to global illumination using photon maps.

Jensen, H.W., 1996. Global illumination using photon maps. *Rendering techniques '96*. Springer Vienna, Eurographics, pp.21–30.

Jensen, H.W. and Christensen, N.J., 1995. Photon maps in bidirectional monte carlo ray tracing of complex objects. *Computers and graphics*, 19(2), pp.215–224.

Kajiya, J.T., 1986. The rendering equation. *Computer graphics (new york, n.y.)*, 20(4), pp.143–150.

Ritter, J. and Glassner, A., 1990. *Graphics gems i*. Academic Press.

Scratchapixel, 2022. *Reflection, refraction (transmission) and fresnel* [Online]. Available from: `https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/reflection-refraction-fresnel` [Accessed 2022-11-26].

Appendix A

Renders

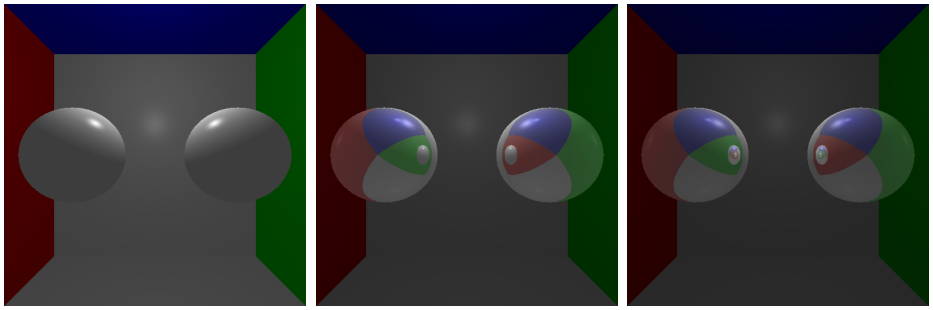

Figure A.1: Reflective spheres with recursion depth 0, then 1, then 5

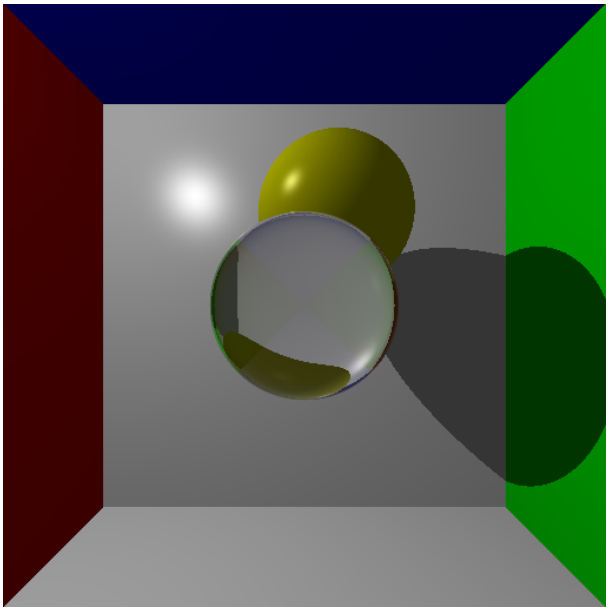

Figure A.2: Refractive and reflective sphere using Fresnel terms

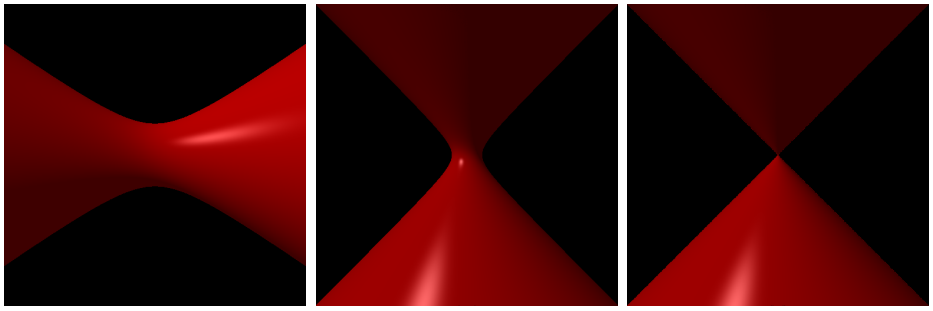

Figure A.3: Quadratic surfaces

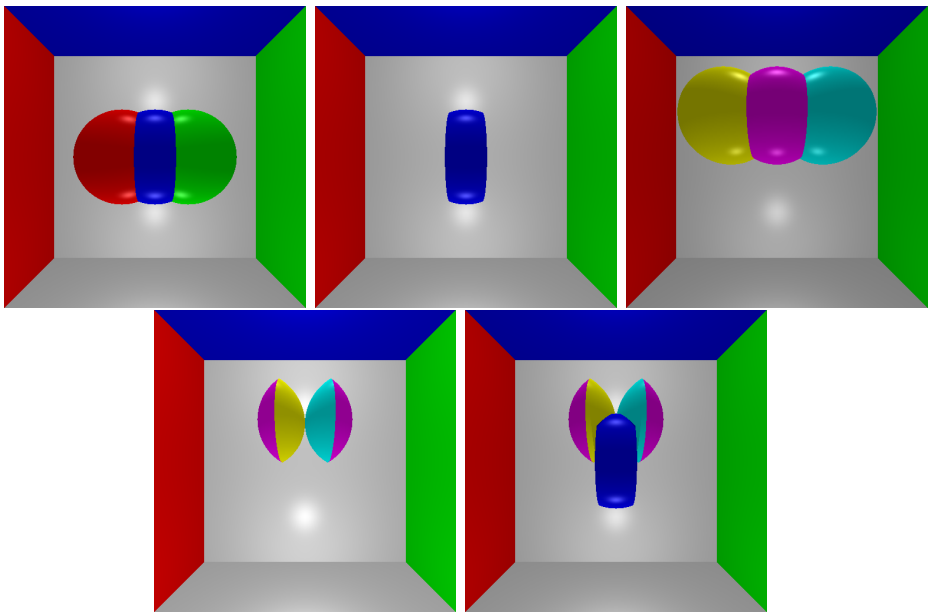

Figure A.4: Constructive solid geometry

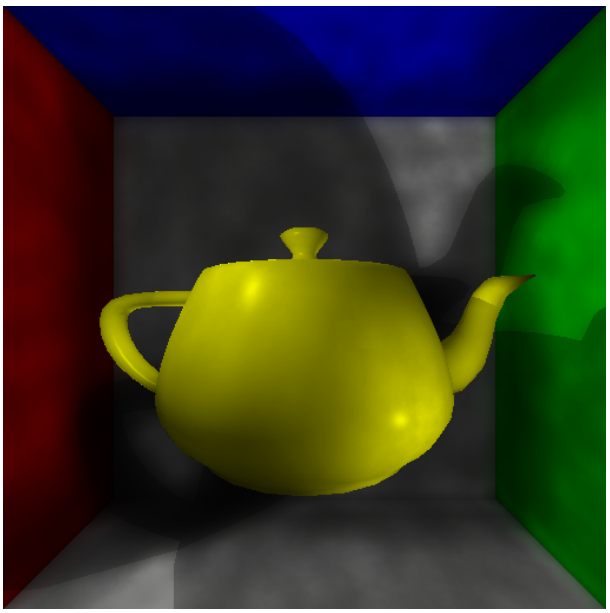

Figure A.5: An early implementation image rendered with photon maps

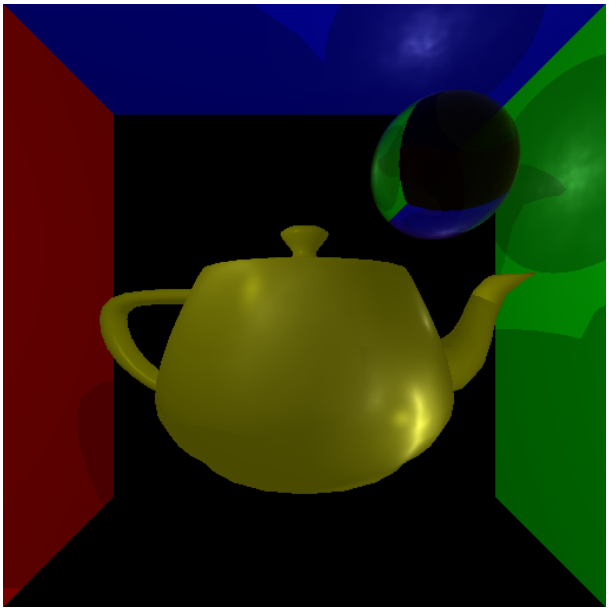

Figure A.6: An image rendered with attempted caustics