1 A simple ray tracer

Ray tracing is a method of drawing images (of a number of geometric primitives; here: spheres and triangles) by following certain light paths from the viewpoint to the light sources and computing how much light travels along these paths. Here, we discuss a very simple variant of ray tracing that follows paths that bounce only once from some primitive. The input to the ray tracer consists of:

1. The coordinates of the viewpoint

2. The description of the screen that allows convenient calculation of locations of any pixel, e.g. coordinates of the lower left corner and two vectors running along the edges of the screen

3. The location and intensity of the light source (here, it's going to be a point light source) and the intensity of ambient light (representing light that bounced more than once, which we do not attempt to compute precisely here)

4. Locations of all of the primitives (three vertices for each of the triangles, center and radius for each of the spheres).

5. Description of material properties of each primitive: ambient, diffuse and specular coefficients and the specular exponents.

Roughly speaking, for each pixel P the ray tracer does the following:

1. Computes the closest intersection point p of a ray originating from the viewpoint through P with the primitives; this is what we see through the pixel

2. Decides whether p is in shadow or not; this can be done by testing the ray originating at p in the direction of the light source for intersections with the primitives – if there are intersections of that ray strictly between the light and p, then p is in shadow.

3. Compute the amount of light scattered at p toward the viewpoint; use the result to shade the pixel.

The first and second steps can easily be reduced to ray-primitive intersection calculations. To compute the closest intersection between a ray and primitive, one can intersect the ray with all primitives in some order, keeping track of what is the closest one (using the terminology introduced below, of the smallest t parameter for which the intersection happens). Similarly, to test whether p is in shadow we can compute the intersection of the ray from p through the light source with the primitives and report 'in shadow' if there is an intersection happening strictly between p and the lightsource (i.e. for t positive but less than 1 if the vector $\vec{pl}$, l=light source location, is used as the direction vector).

2 Rays

A ray in 3D is described by its *origin* o, a 3D point, and the *direction* $\vec{d}$, a 3D vector. A point belongs to the ray if and only if it has the form $o + t * \vec{d}$, where t is a nonnegative number.

3 Ray-Sphere Intersection

A sphere is described by its *center* c and *radius* r. A point p belongs to the sphere if its distance from the center is equal to r, i.e. $|\vec{cp}| = |p - c| = r$.

An intersection of a ray $o + t\vec{d}$ and a sphere centered at c and having radius r can be found from the above equations by substituting $p := o + t\vec{d}$:

$$r = |o + t\vec{d} - c| = |\vec{co} + t\vec{d}|.$$

By squaring both sides of the above equation we obtain

$$r^2 = |\vec{co}|^2 + 2(\vec{co} \cdot \vec{d}) * t + |\vec{d}|^2 * t^2.$$

We can write this as a quadratic equation

$$At^2 + Bt + C = 0,$$

where

$$A = |\vec{d}|^2,$$

$$B = 2(\vec{co} \cdot \vec{d}),$$

and

$$C = |\vec{co}|^2 - r^2.$$

Forgetting that we need $t >= 0$ for a while, we can say that the number of solutions depends on its discriminant $\Delta = B^2 - 4AC$:

1. if $\Delta < 0$, no solution exists (ray certainly misses the sphere)

2. if $\Delta = 0$, there is only one solution (geometrically, this means a tangency)

3. if $\Delta > 0$, there are two solutions.

In the second case, the solution is given by $-B/(2A)$ and in the third case, the two solutions are given by $(-B \pm \sqrt{\Delta})/(2A)$.

Since ray extends only in one direction (positive t), the solutions with $t < 0$ have to be rejected. Also, both for shadow and visibility calculations only the *closest* intersection matters. Therefore, what we really want is the least nonnegative t that solves our intersection equation. This means that:

1. In case 1., we should report that there is no intersection

2. In case 2., we should report there is no intersection if $t < 0$ (which happens to be equivalent to $B > 0$ since $A > 0$); otherwise, we should return the solution $t = -B/(2A)$.

3. In case 3., we should return the least positive number of the two t's. If both are negative, we should report no intersection.

4 Ray-Triangle Intersection

A 3D triangle is defined by its vertices, three points in 3D. Let's call them a_1, a_2 and a_3. To decide whether a ray intersects a triangle and compute the intersection point, we proceed in two steps.

1. Compute the intersection of the triangle's plane and the ray; if there is no intersection, report 'no intersection'

2. Decide whether the intersection point is inside or outside the triangle.

4.1 Step 1.

The normal vector to the triangle's plane can be computed by

$$\vec{n} = \vec{a_1 a_2} \times \vec{a_1 a_3}.$$

Then, the plane can be described by the following property: a point p belongs to it if and only if

$$\vec{a_1 p} \cdot \vec{n} = 0.$$

Substituting $p := o + t\vec{d}$ yields:

$$0 = (o + t\vec{d} - a_1) \cdot \vec{n} = \vec{a_1 o} \cdot \vec{n} + t\vec{d} \cdot \vec{n},$$

and so

$$t = -\frac{\vec{a_1 o} \cdot \vec{n}}{\vec{d} \cdot \vec{n}}.$$

Similarly to the sphere case, if $t < 0$, we can safely report 'no intersection'. Otherwise, we still need to test whether the intersection point (which can be computed from the ray equation, as $o + t\vec{d}$), is inside the triangle or not.

4.2 Step 2., using the Cross product

This is one of numerous ways to do this which I happen to like best.

Look at the directions of the cross products $\vec{pa}_1 \times \vec{pa}_2$, $\vec{pa}_2 \times \vec{pa}_3$ and $\vec{pa}_3 \times \vec{pa}_1$. Notice that they need to be parallel to the plane's normal vector. If all of them point in the same direction, p is inside the triangle. Otherwise, it is not. To check whether two vectors point in the same direction or not, compare the signs of one of their coordinates. To avoid numerical problems, select a coordinate which has large (or better, largest) magnitude.

5 Local illumination models

Here are vectors we are going to use to compute the amount of light scattered at p toward the viewpoint.

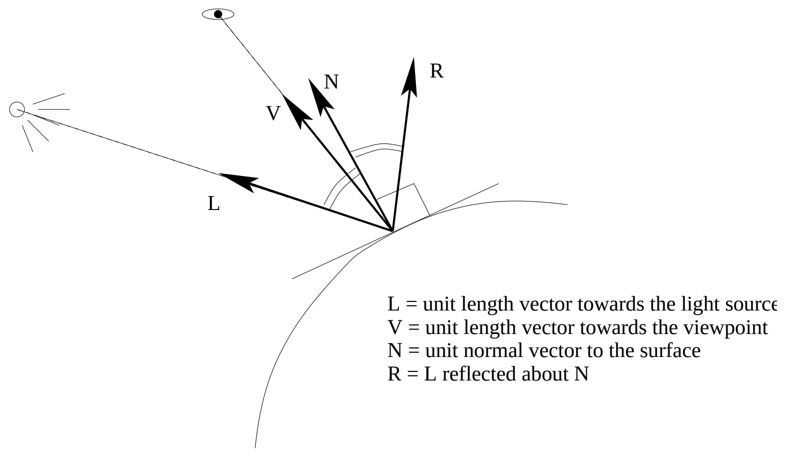
R
N
V
L
L = unit length vector towards the light sourc
V = unit length vector towards the viewpoint
N = unit normal vector to the surface
R = L reflected about N

Figure 1: Vectors needed for lighting calculations

The reflected vector can be computed from N and L using the formula

$$\vec{R} = -\vec{L} + 2 * (\vec{N} \cdot \vec{L})\vec{N}.$$

The illumination model we are going to use has three terms: diffuse, specular and ambient.

The diffuse term is modelled after the Lambert cosine law, which describes light scattering for lambertian (or diffuse) surfaces. A diffuse surface has the property that it appears of the same brightness regardless of the viewing angle. The brightness of such surface is proportional to the energy of light that hits it per unit area per unit time. For a beam of light of cross-section area A hitting the surface, the energy of the beam is distributed over area $A/\cos\angle(L,N) = A/(N \cdot L)$ where N is the surface's normal and L is the vector pointing to the direction the light is comming from. Thus, the brightness will be proportional to $N \cdot L$. This leads to the diffuse term $I_i k_d (N \cdot L)$, where I_i is the intensity of the incident light, k_d is the diffuse reflection coefficient (amount of light scattered in a diffuse manner). k_d is a property of the primitive's material.

Most surfaces in real world are not perfectly diffuse. Quite often, they are in between diffuse surface and a mirror, reflecting most light in the direction of R. The specular term attempts to model such behavior. The formula is $I k_s \max(0, R \cdot V)^n$, where k_s is the specular reflection coefficient and n is the specular exponent (both are material properties). Notice that $R \cdot V$ is largest if R and V overlap, i.e. if the surface is looked at from the reflected vector direction. As the angle between the two vector increases, the specular term decreases. The specular exponent controls how fast it decreases: the larger n

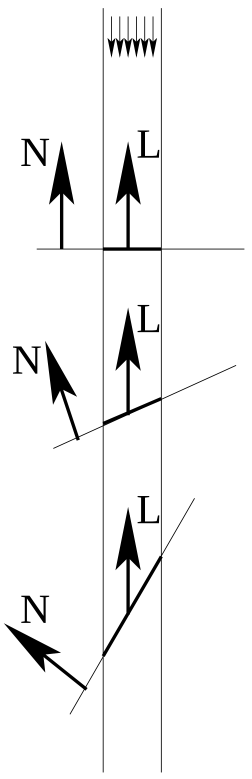

Figure 2: area of intersection of a beam with a surface is inversely proportional to $N \cdot L$.

is, the faster the decay. By increasing n we can cause the surface reflect light more and more like a perfect mirror.

Finally, the ambient term attempts to model multiply scattered light. Here, we model the ambient light by $I_a k_a$, where I_a is a global constant (ambient light intensity) and k_a is the ambient reflection coefficient, basically describing how much ambient light a particular material scatters back into the environment. Obviously, this is just a hack having nothing to do with the actual physics. Nevertheless, we need something like this, otherwise shadows would be completely dark.

To sum up, the illumination formula is:

$$k_a I_a + I_i(k_d(N \cdot L) + k_s \max(0, R \cdot V)^n).$$

If the point p is in shadow, we want to use only the ambient term.

To simulate the effect of objects further away from the light getting darker, we can add the attenuation term. In physics, the energy reaching a unit surface area is inversely proportional to the squared distance r from the light to the surface. This leads to

$$k_a I_a + \frac{1}{r^2} I(k_d(N \cdot L) + k_s \max(0, R \cdot V)^n),$$

where I is the light source intensity. Because of imperfect modeling of physics and relatively low dynamic range of today's displays, it makes more sense to make the attenuation term decay slower. Usually, instead of $\frac{1}{r^2}$ one uses an inverse of a general quadratic polynomial, $\frac{1}{c_2 r^2 + c_1 r + c_0}$.

For color objects, we have separate formulas for the red, green and blue components:

Case 1 If $\vec{L} \cdot \vec{N} \geq 0$:

$$I_r = k_{ar} I_a + \frac{1}{c_2 r^2 + c_1 r + c_0} I(k_{dr}(\vec{L} \cdot \vec{N}) + k_s(\max(0, \vec{R} \cdot \vec{V})^n))$$

$$I_g = k_{ag} I_a + \frac{1}{c_2 r^2 + c_1 r + c_0} I(k_{dg}(\vec{L} \cdot \vec{N}) + k_s(\max(0, \vec{R} \cdot \vec{V})^n))$$

$$I_b = k_{ab} I_a + \frac{1}{c_2 r^2 + c_1 r + c_0} I(k_{db}(\vec{L} \cdot \vec{N}) + k_s(\max(0, \vec{R} \cdot \vec{V})^n)).$$

Case 2 If $\vec{L} \cdot \vec{N} < 0$ (or the point we are looking at is in shadow), we use only the ambient light terms: $I_r = k_{ar} I_a$, $I_g = k_{ag} I_a$, and $I_b = k_{ab} I_a$.

All the k's are properties of the surfaces.

Color light sources can be modelled by specifying three components of I (red, green and blue). Illumination from multiple light sources is dealt with by doing the computations described above for each light source separately and summing the results.