

1 Rays

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

2 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 $t \geq 0$ 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.

3 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.

3.1 Step 1.

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

$$\vec{n} = a_1\vec{a}_2 \times a_1\vec{a}_3.$$

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

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

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

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

and so

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

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.

3.2 Step 2.

There are a few ways this can be done.

3.2.1 Projecting to 2D

One can reduce the problem to 2D. Imagine you project the triangle onto some plane (parallel projection). If the intersection point is inside, the projected point will still be inside the projected triangle. If it is outside, the projected point will also be outside the projected triangle. We talked about two ways of deciding whether a point p is inside or outside a 2D triangle:

1. Shoot a ray from the point and count its intersections with the edges. If the number of intersections is odd, the point is inside. If it's even, the point is outside.
2. For each vertex, check whether p is on the same side of the line passing through the other two vertices. If you get three 'yes' answers, the point is inside the triangle. Otherwise, it is outside.

3.2.2 Using the Cross product

Look at the directions of the cross products $p\vec{a}_1 \times p\vec{a}_2$, $p\vec{a}_2 \times p\vec{a}_3$ and $p\vec{a}_3 \times p\vec{a}_1$. 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.

4 Local reflection models

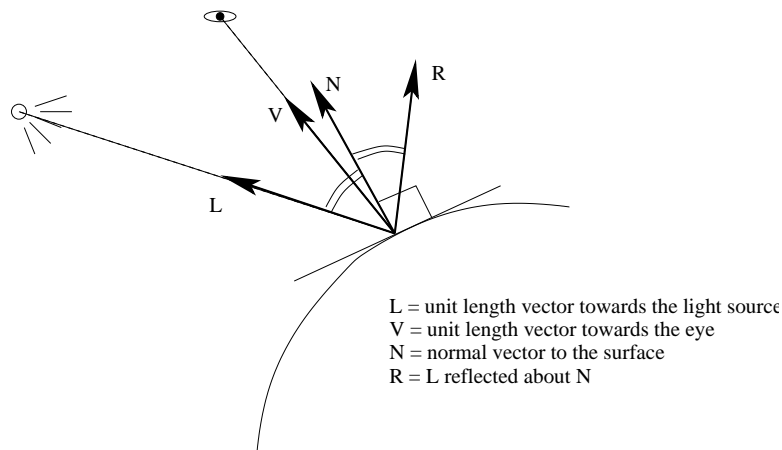


Figure 1: Vectors needed for lighting calculations

... Take a look at Section 6.2.1, which I followed in class. Be sure to know

the formula

$$R = -L + 2 * (N \cdot L)N$$

and understand where it comes from. For colorful objects, we had formulas for the red, green and blue components:

$$I_r = k_{ar}I_a + I_i(k_{dr}(L \cdot N) + k_s(R \cdot V)^n)$$

$$I_g = k_{ag}I_a + I_i(k_{dg}(L \cdot N) + k_s(R \cdot V)^n)$$

$$I_b = k_{ab}I_a + I_i(k_{db}(L \cdot N) + k_s(R \cdot V)^n).$$

All the k 's are properties of the surfaces.

Also, remember that if $R \cdot V$ is negative, we skip the specular term: light is reflected specularly only towards directions no more than 90 degrees off the reflected vector. Similarly, if the second dot product, $L \cdot N$, is negative, then the surface is facing away from the light source. Therefore, only the ambient term should be used.

5 Putting it all together: a simple ray tracer

A simple ray tracer computes a color for a pixel, say with coordinates (i, j) on an $m \times n$ -pixel screen as follows.

First, it computes the closest intersection point of a ray (call it an eye ray) originating from the eye and passing through the center of the pixel (i, j) with the objects in the scene. The screen is described by a point l , its lower left corner, and two vectors \vec{h} and \vec{v} , running along its horizontal and vertical edges (respectively). This intersection point will tell us what is visible 'through' that pixel. The ray is described by:

Origin: e , the eye.

Direction: $(l + \frac{(i+.5)}{m} * \vec{h} + \frac{(j+.5)}{n} * \vec{v}) - e$.

The formula for the ray direction can be obtained like this: to reach the lower left corner of the pixel from l (the lower left corner of the screen) we need to move i times by the vector running along the horizontal edge of a pixel and j times along the vector running along a vertical edge of a pixel. These vectors are \vec{h}/m and \vec{v}/n . Then, we need to move from the lower left corner to the center of the pixel, i.e. by $.5 * \vec{h}/m + .5 * \vec{v}/n$. Hence the formula for the center of the pixel (i, j) :

$$(l + \frac{(i + .5)}{m} * \vec{h} + \frac{j + .5}{n} * \vec{v}).$$

Subtracting e gives the coordinated of the vector running from e to the center of the pixel, i.e. the direction vector for the eye ray.

To compute the intersection of that ray with the objects in the scene, one can simply go over all the objects and compute their intersections with the

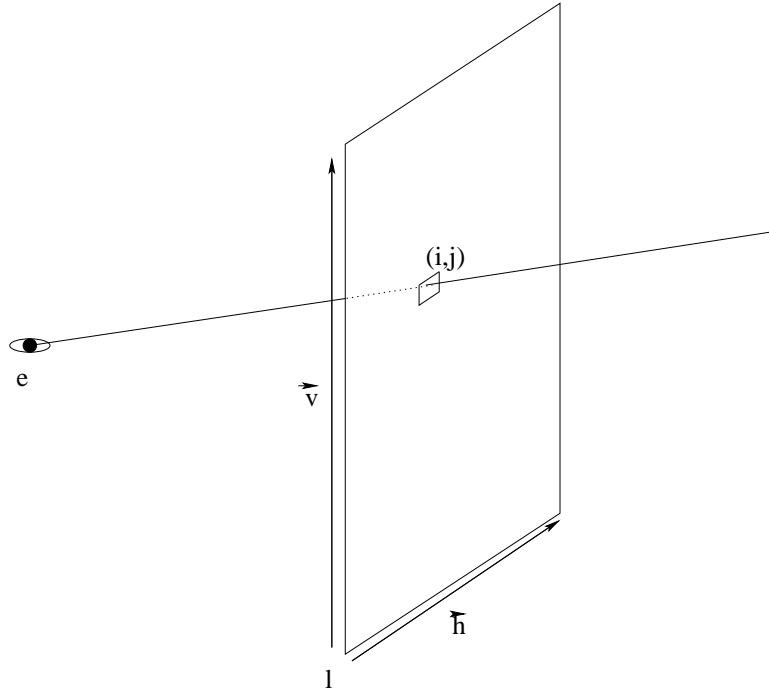


Figure 2: Screen in the virtual world and ray through the center of a pixel (i, j) .

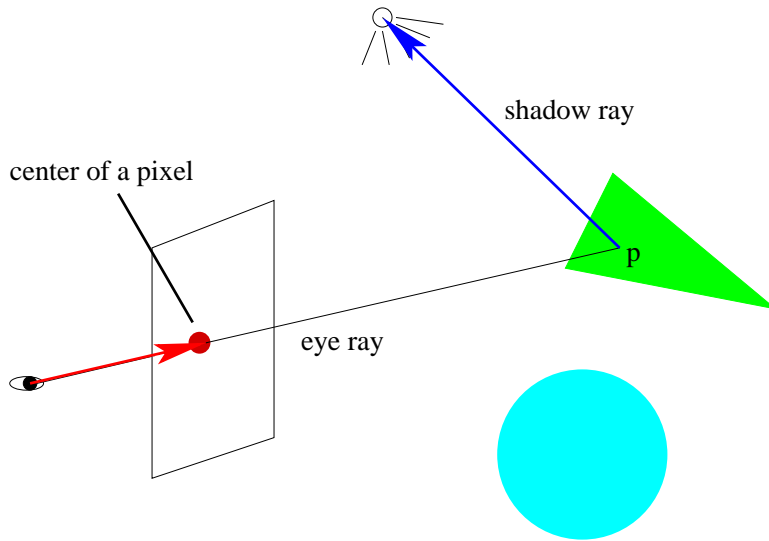


Figure 3: Eye ray and shadow ray

ray, keeping track of the closest one (i.e. smallest t parameter for which an intersection happens).

Once the closest intersection point (call it p) is computed, we need to decide whether it is in shadow or not. First, notice that if p is facing away from the light then it is certainly in shadow. To test whether this is so, look at the dot product $N \cdot \vec{pb}$, i.e. the dot product of the normal vector at p and the vector starting at p and pointing towards the light source. If it is negative, then p is facing away from the light and therefore it cannot be lit. Otherwise, we need to test whether other primitives block the light from hitting p . To do this, we consider the ray (shadow ray) starting at p and ending at the light source b (the direction vector is therefore $b-p$). We need to decide whether that ray intersects a primitive *different from the one containing p* before it reaches the light source (then p is in shadow) or not (then, p is lit). We can reuse the closest intersection routine (however, it has to know it needs to skip the primitive containing p): if t is the parameter where the closest intersection happens and $t < 1$, it means that the intersection is between p and b and therefore p is in shadow. If either there is no intersection or $t > 1$ then the point p is lit.

Now, if p is lit then we need to use the full formula for the reflected light intensity as in Section 4. If p is in shadow, we use only the ambient term and skip the diffuse and specular terms.