Recursive Ray Tracing

We can "see" objects around us because light rays hit the object and travel to our eyes. Each object has a characteristic texture, color, shininess which is a result of the way in which the light rays interact with the object. For example, a red object appears red because red light is reflected and blue and green light is absorbed. Mirrors reflect light essentially unchanged while objects with dull surfaces tend to disperse an incoming light ray in lots of directions.

Ray tracing is a method of generating realistic images in computer graphics by simulating the illumination and reflection properties of materials with light. By simulating the flow of light rays. In the process, one in some sense gets for free hidden surface removal and shadows.

The number of lights we see are essentially infinite so we can not trace all rays. We are in fact only interested in those rays that reach our eyes. As a result we trace backwards, from our eye to a light source. If no such ray exists then we don't see anything.

Furthermore, when looking at a computer screen we are only concerned with what color to set each pixel on the screen. As a result, we really only care about the light rays that pass through a pixel on the screen and hit our eyes. Thus if we are drawing a screen in a 100x100 window we really only need to trace 10,000 rays. Of course, this number is increased because each ray follows an erratic path as it is reflected off various objects.

Components of Ray Tracing

- Ray Tracing: from eye to back to light source given reflections and refraction with multiple objects.
- Illumination Models: modelling the interaction of light with objects. complex - must make many simplifying assumptions.
- Object representation: in some sense this is no different than with polygon rendering. However, here we must also be able to easily compute intersections of the light rays with objects.
- Efficiency issues: Ray tracing is computationally intense especially when there are many objects in a scene. There are many techniques for making the process more efficient (e.g. determining the closest object by using some sort of hierarchical structure to keep track of location of objects.

Look at examples of ray tracing in books.

The simplest Algorithm

See sphere example with only ambient light. What it does:

- uses ambient light: "background light" - extremely simplified way of modelling very diffuse light. Not attributable to a light source.
- visible surface determination
- sets color of an object with an intensity dependent on the amount of ambient light.

What it doesn't do:

- No reflection or refraction of rays. (or rather only reflection of ambient light)
- no light sources, assumes "ambient" lighting =¿ objects appear flat.

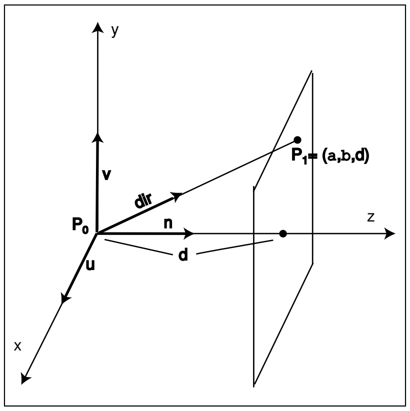
To implement, we must

- place objects in a scene (must determine representation of objects as well). We will start with spheres, planes, boxes.
- Set up location of screen and viewer.
- For each ray, compute equation of ray and determine if it intersects any objects. If yes, determine which object is closest.
- Determine color of reflected ray at the point it intersects the closest object based on ambient light model. Use this to set pixel color.

See pseudocode slide (prog 13.3 Foley)
See also implementation details in lab handout.

Computing the Ray

We define a ray by a starting point P_0 (at the camera) and a direction defined by the line connecting the camera position and the pixel on the screen, P_1.



$$\begin{aligned} dir &= \text{unit vector in direction from } P_0 \text{ to } P_1 \\ &= \frac{P_1 - P_0}{||P_1 - P_0||}. \end{aligned}$$

The ray can then be expressed *parametrically* as

$$P = P_0 + t\ dir$$

where $t = 0$ corresponds to P_0. Of course, the goal is to find the closest object that intersects this ray.

Computing Intersections for Spheres

A sphere can be represented by it center $P_c = (a, b, c)$ and radius r. Given these, the equation for the sphere can be written as

$$(x - a)^2 + (y - b)^2 + (z - c)^2 = r^2.$$

In vector notation, this becomes

$$(P - P_c) \cdot (P - P_c) = r^2.$$

To find the intersection of the sphere and the ray we just insert the equation of ray in for P to give

$$(P_0 + t\ dir - P_c) \cdot (P_0 + t\ dir - P_c) = r^2$$

and solve for t. Multiplying out we obtain

$$(P_0 - P_c) \cdot (P_0 - P_c) + 2\ dir \cdot (P_0 - P_c)t + \ dir \cdot dir\ t^2 = r^2$$

which is a quadratic equation in t. Defining the scalars B and C as

$$\begin{aligned} B &= 2\ dir \cdot (P_0 - P_c) \\ C &= (P_0 - P_c) \cdot (P_0 - P_c) - r^2 \end{aligned}$$

and noting that $dir \cdot\ dir = 1$, we obtain the equation $t^2 + Bt + C = 0$. The solution is simply

$$t = \frac{-B \pm \sqrt{B^2 - 4c}}{2}.$$

There are 0, 1, or 2 real solutions to this depending on the value of the discriminant $D = \sqrt{B^2 - 4C}$. In particular,

$D < 0$	no intersections.
$D = 0$	1 intersection, ray grazes sphere.
$D > 0$	2 intersections, take smaller t value.

And if $t < 0$ then the object is behind viewer.

Example: Let the sphere be defined by: $P_c = \begin{pmatrix} 3 \\ 3 \\ 1 \end{pmatrix}$, $r = 2$ and the ray by $P_0 = \begin{pmatrix} 0 \\ -2 \\ 0 \end{pmatrix}$, $P_1 = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$. (Use a left handed coordinate system.)

Then we find that

$$\begin{aligned} P_0 - P_c &= \begin{pmatrix} -3 \\ -5 \\ -1 \end{pmatrix} \\ dir &= \frac{1}{\sqrt{6}} \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}, \ B = \frac{-28}{\sqrt{6}} = -11.43, \ C = 31 \end{aligned}$$

to give

$$\begin{aligned} 0 &= t^2 - \frac{28}{\sqrt{6}} t + 31 \\ D &= B^2 - 4C = 6.67 \\ t &= \frac{-B \pm \sqrt{D}}{2} \\ &= 7.01(+) \text{ and } 4.42(-) \end{aligned}$$

So the intersection point is

$$P = P_0 + t \ dir = \begin{pmatrix} 1.80 \\ 1.61 \\ 1.80 \end{pmatrix}$$

Computing Intersections with Planes

A plane is defined by its normal N and a point on the plane Q_0. Recall that the equation of the plane is then given by

$$(P - Q_0) \cdot N = 0$$

To find the intersection of the plane and the ray we insert the expression for the ray into the above equation to obtain

$$(P_0 + t \ dir - Q_0) \cdot N = 0$$

and solving for t gives

$$t = \frac{(Q_0 - P_0) \cdot N}{dir \cdot N}$$

Note, a zero denominator means that the line and plane are parallel. In such a case there are either no intersections if the ray does not lie in the plane, or there are an infinite number of intersections.

Boxes aligned with the axes are also easy because they are just planes with the range constrained.

Setting the Pixel Value

In this simple model we start with *ambient* light. Ambient light is the "background" light that results from many many rays from all possible light sources bouncing off of objects multiple times. It is ambient light that allows us to see all surfaces in a room even if there is no direct light on the surfaces. Since it would be too complex to model every single ray, we instead model the average behavior. The modelling is very simplistic in that we assume that ambient light is constant over the entire scene. Although ambient light is a result of light sources in the room we model it as independent of any light source and only dependent on the scene.

Because ambient light is constant, it illuminates all surfaces with an equal brightness making objects monochrome.

Contribution of ambient light to the pixel color is:

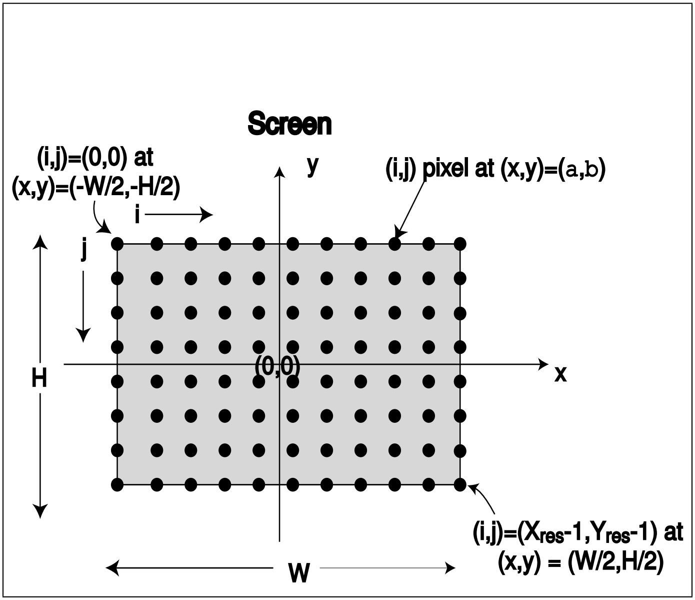
$$C_a = k_a(c_r I_{a,r}, c_g I_{a,g}, c_b I_{a,b})$$

where

- k_a = ambient reflection coefficient, $0 \leq k_a \leq 1$
  k_a is a characteristic of the object, depending on the material make-up of the object.
- $c = (c_r, c_g, c_b)$ = color of the object which is also a characteristic of the object, depending on the material make-up of the object. The range for each component is $0 \leq c_{r,g,b} \leq 255$. Sometimes k_a and c are combined. However, it is useful to keep them separate so that one can adjust the ambient coefficient while maintaining the same balance of color.
- $I_a = (I_{a,r}, I_{a,g}, I_{a,b})$ = the intensity of the ambient light. I_a is the same for all objects in the scene.
- Note, the final pixel color should never exceed 255.

Screen Coordinate System

We are now able to determine the color of the pixel at position P_1. However, P_1 is a point specified in the world coordinate system and not the screen (i.e. pixel) coordinate system. We need to determine be able to determine the world coordinates of a pixel at screen position (i, j).



We assume that the pixels are spread evenly across the screen and that we are given the following information (see figure above as well):

$$\begin{aligned}
P_0 &= \text{location of camera}\\
\text{VPN} &= \text{normal to view plane}\\
\text{VUP} &= \text{up direction}\\
d &= \text{distance of camera from view screen}\\
H &= \text{height of screen}\\
W &= \text{width of screen}\\
X_{res} &= \text{number of pixels per column}\\
Y_{res} &= \text{number of pixels per row}
\end{aligned}$$

We can compute unit vectors along the view coordinate axes, where here we are using a left-handed coordinate system.

$$\begin{aligned}
\hat{n} &= \frac{\text{VPN}}{||\text{VPN}||}\\
\hat{u} &= \frac{\text{VUP} \times \text{VPN}}{||\text{VUP} \times \text{VPN}||}\\
\hat{v} &= \hat{n} \times \hat{u}
\end{aligned}$$

Thus, in world coordinates, the difference $P_1 - P_0$ can be written as

$$P_1 - P_0 = \alpha\hat{u} + \beta\hat{v} + d\hat{n}$$

for some α and β. To determine α and β we assume that pixels are spread evenly over the screen and then convert from pixel (i, j) to a location on the screen as follows

$$(\alpha, \beta) = \left(-\frac{W}{2} + \frac{W \cdot i}{X_{res} - 1}, \frac{H}{2} - \frac{H \cdot j}{Y_{res} - 1}\right)$$

Example: Suppose $P_0 = \begin{pmatrix} -2 \\ -2 \\ 0 \end{pmatrix}$, VPN $= \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$, VUP $= \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$, $d = 1$, $W = 2$, $H = 2$, $X_{res} = 201$, and $Y_{res} = 201$. Then

$$\begin{aligned}
\hat{n} &= \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \\
\hat{u} &= \frac{1}{\sqrt{2}}\begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix} \\
\hat{v} &= \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \\
\alpha &= -1 + \frac{i}{100} \\
\beta &= 1 - \frac{j}{100} \\
P_1 - P_0 &= (-1 + \frac{i}{100})\frac{1}{\sqrt{2}}\begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix} + (1 - \frac{j}{100})\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} + d\frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \\
&= \begin{pmatrix} 2 - \frac{i}{\sqrt{2}100} \\ \frac{1}{\sqrt{2}}(\frac{i}{100}) \\ 1 - \frac{j}{100} \end{pmatrix}
\end{aligned}$$

For pixel $(i, j) = (10, 20)$,

$$P_1 - P_0 = \begin{pmatrix} \frac{1}{\sqrt{2}(2-\frac{1}{10})} \\ \frac{1}{10\sqrt{2}} \\ \frac{4}{5} \end{pmatrix}$$

More Sophisticated Light Models

In general we can consider the color at an intersection point to be the sum of 5 different color components: ambient, diffuse, specular, reflected, and refracted (i.e. transmitted). Diffuse and specular are a result of light coming from a light source and being reflected from an object back to the observer's eyes. Materials that are rough tend to scatter the light in all directions. Such materials tend to have a rather dull finish and the illumination is referred to as diffuse. While materials that are smooth and highly reflective tend to reflect light very non-uniformly. Such objects tend to have what are called specular highlights. While the separation between these two is rather artificial, it is still useful to model the diffuse and specular components as being separate.

1. ambient: Already discussed.

2. diffuse: due to light source.
3. specular
4. reflected from other objects
5. refracted (i.e. transmitted)

The latter 4 all require specifying the light sources. For diffuse and specular we look at light that hits the intersection point directly from the light. We want to compute how much light from the light source that hits the intersection actually makes it back to out eye.

The ray that goes from the light to the object is called the **shadow ray**. Note that if the shadow ray hits an opaque object before reaching the intersection point then that light does not contribute to the light source at the intersection point. need picture.

0.0.1 Diffuse Light

Diffuse light models surfaces that tend to scatter light in all directions. This occurs because the surface is not completely smooth so that at the microscopic level there are bumps that have normals that are not aligned with the normal of the macroscopic object.

Objects that have a dull, matte finish have a large diffuse component. These surfaces appear equally bright from all directions that they are viewed. In fact the brightness depends only on the angle that the shadow ray makes with the surface normal and not the angle of the viewer (need picture). The reason for this is as follows.

Imagine a cylinder of light coming from the light source and falling on the object (here we are ignoring light attenuation). The cross sectional area of this cylinder is A. The energy per unit area hitting the surface when the light is right overhead is E/A where E is the energy emitted from the light source. If the light is off to the side then the area hit by the light is larger and is given by $A_\theta = A/\cos\theta$ so the energy per unit area is $E\cos\theta/A$, i.e. it is smaller by an amount θ. Thus the brightness is a function of $L \cdot N = \cos\theta$ where L is the unit vector pointing from the intersection point to the light source and N is the unit normal to the surface.

Why does the brightness not depend on the viewing angle? Once the light is fixed, we have a fixed energy per unit area hitting the surface. As the viewing angle increases the area viewed increases. However, the amount of this energy that gets reflected decreases so as to offset the increase in viewing area. Thus the view sees the same brightness from any angle.

For a monochrome light source, the diffuse component is given by

$$C_d = k_d \sum_j I_j (L_j \cdot N)$$

where

- k_d = coefficient of diffuse reflection of the object being intersected
- N = unit normal at intersection point
- L_j = unit vector pointing from the intersection point to the j^{th} light source.
- I_j = the intensity of the j^{th} light source.
- Note: if $L_j \cdot N < 0$ and the intersected object is opaque then the diffuse component for that light is zero because the light is behind the object.

If we add color, then the intensity becomes $I_j = (I_{j,r}, I_{j,g}, I_{j,b})$ and we introduce the diffuse color of the object $c = (c_r, c_g, c_b)$ which may or may not be different from the ambient color. The diffuse component then becomes

$$C_d = k_d \sum_j (c_r I_{j,r}, c_g I_{j,g}, c_b I_{j,b})(L_j \cdot N)$$