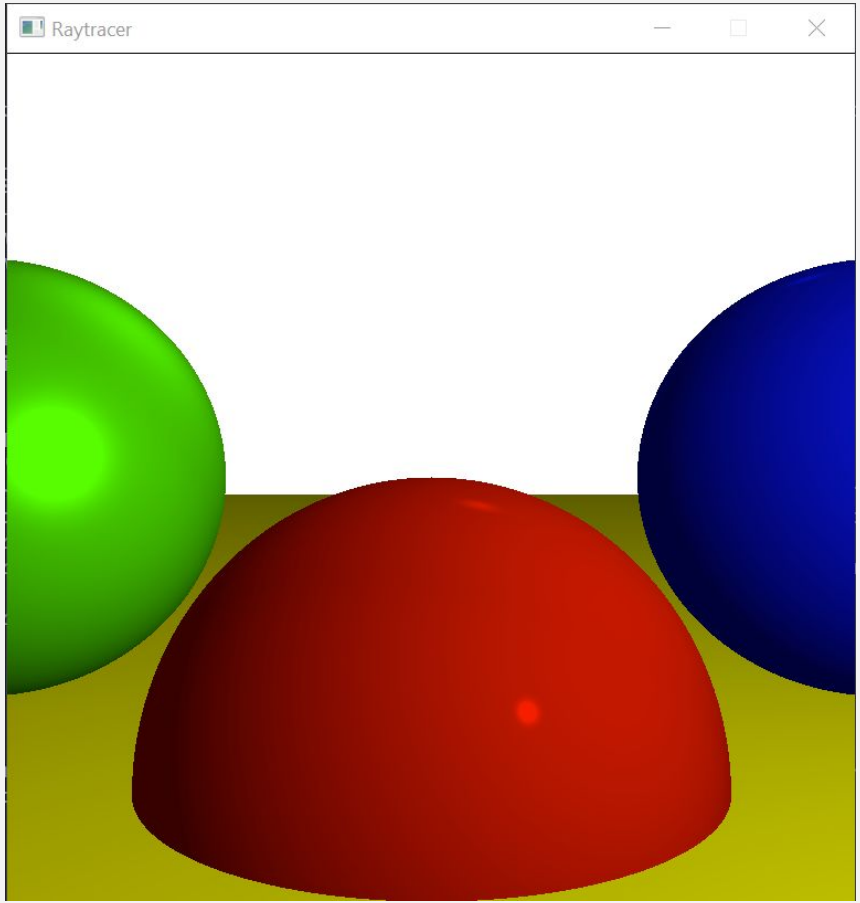
2
Lights

Based on *Computer Graphics from Scratch* by Gabriel Gambetta

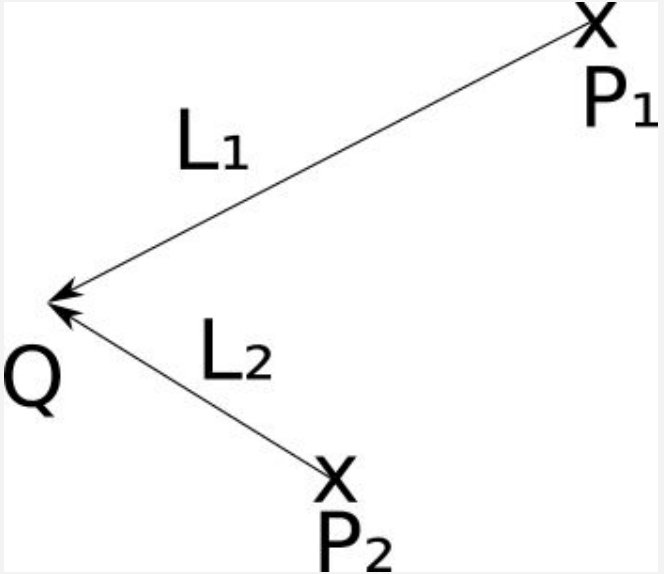


Modelling Lights

In order to simplify our modelling of lights, we make two assumptions:
- Lights are white, so their intensity is a single value, *i*.
- We ignore atmospheric dimming of far away lights.

We define three types of lights.

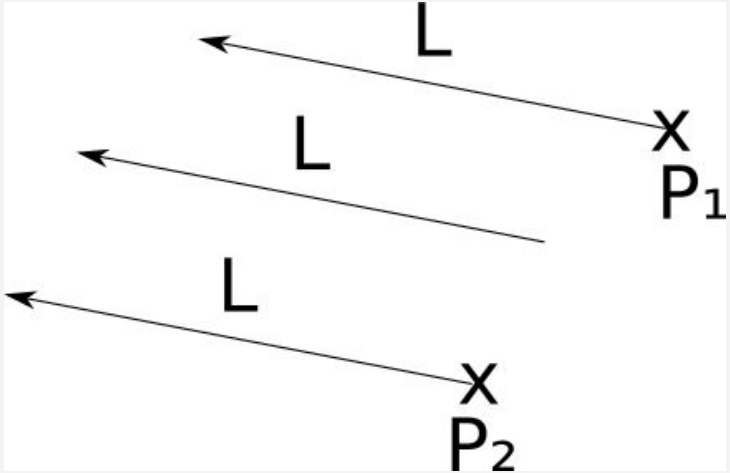
Point Lights
They emit light from a fixed position, *Q*, in all directions, with an intensity of *i*. The Light vector, *L*, is different for every point *P* in the scene and is calculated with *(Q - P)*.



Modelling Lights 2

Directional Lights

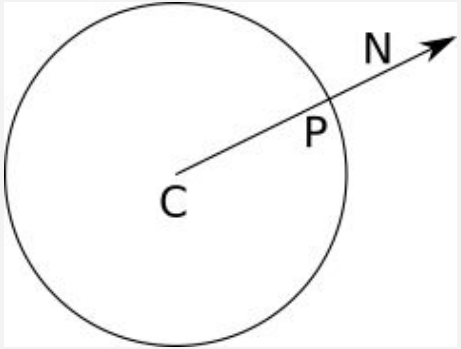
They are defined by a fixed direction vector, *L*, and an intensity *i*.
L is the same for every point in the scene.



Ambient Light

They contribute light to every point in the scene. They can be characterized with a single value, their intensity *i*.

Normals



In order to model reflections, we need a way to characterize surfaces. We use normals for that. The normal *N* of a surface is a vector of length 1 that is perpendicular to the surface.

In a sphere, the normal for any point *P* on the sphere is the direction from the center *C* to that point, *P* - *C*.
The normal has to be length 1, so we normalize it.

$$\vec{N} = \frac{P - C}{|P - C|}$$

Diffuse vs Specular Reflections

When a ray of light hits a matte object, the ray is scattered back into the scene equally in every direction, we call this process *diffuse* reflection.

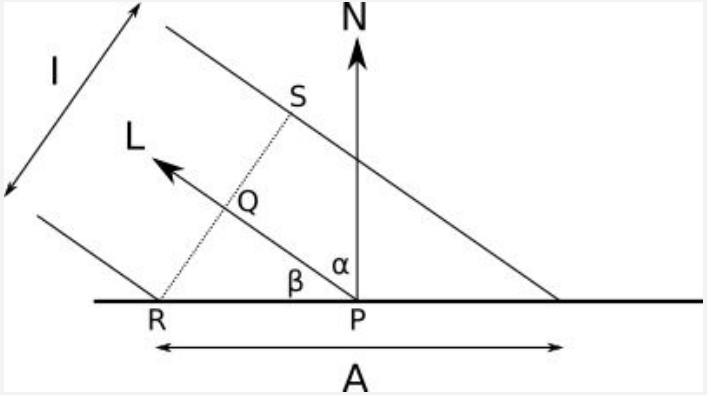
Unlike matte objects, shiny objects look slightly different depending on where you're looking from. That reflection is its *specular* reflection.

Diffuse Reflection

We call the vector perpendicular to the surface at a point *P*, the normal vector *N*.

A ray of light with direction *L* and intensity *I* hits a surface with normal *N*.

There's a geometrical process at:
https://gabrielgambetta.com/computer-graphics-from-scratch/03-light.html#modeling-diffuse-reflection



The gist, as I understand it, is: We simplify the physical act of a ray of light with an intensity *I* bouncing on a surface by modelling it as a ray with width *I* that spreads itself over an area *A*. The coefficient of *I* and *A* is the effective intensity of the light at that point.

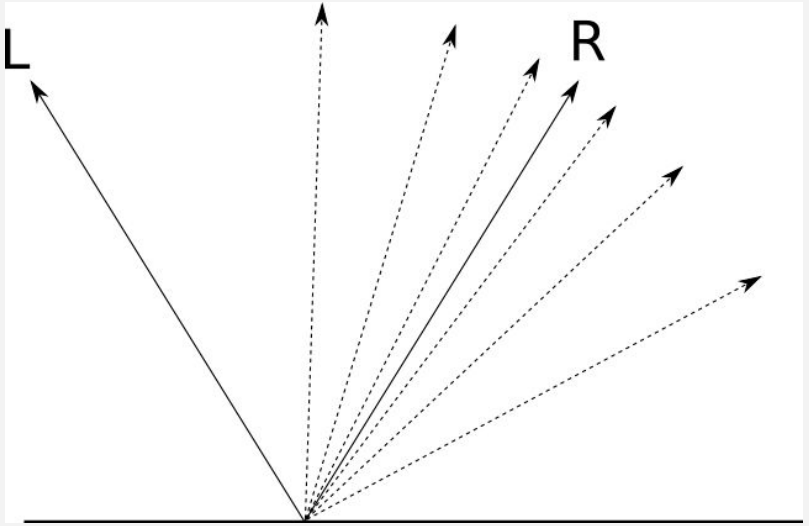
The conclusion is the following equation for the intensity of diffuse lighting at a certain point where a ray of light hits. i/a = nl etc. remember to max(0, i/a)

$$\frac{I}{A} = \frac{\langle \vec{N}, \vec{L} \rangle}{|\vec{N}||\vec{L}|}$$

Specular Reflection

For surfaces that aren't perfectly polished, the closer a direction *V* is to *R* the more rays of light are reflected in that direction. When the angle between *V* and *R* is 0, all light is reflected. When it is 90, no light is reflected.

Although it isn't based on physics, we can model this behaviour using the *cos* function.



In order to model the "shininess" of the object, we can compute the power of cos(a) to some exponent *s*

There's a better explanation at:
https://www.gabrielgambetta.com/computer-graphics-from-scratch/03-light.html#modeling-specular-reflection

The Illumination Equation

For every point and directional light, calculate its diffuse and specular component, sum them and multiply the resulting value by the intensity of the light. Sum all of these values and add the ambient intensity too.

$$I_P = I_A + \sum_{i=1}^{n} I_i \cdot \left[\frac{\langle \vec{N}, \overrightarrow{L_i} \rangle}{|\vec{N}||\overrightarrow{L_i}|} + \left(\frac{\langle \overrightarrow{R_i}, \vec{V} \rangle}{|\overrightarrow{R_i}||\vec{V}|} \right)^s \right]$$

Source Code

A C implementation of the algorithm is available at:
https://github.com/tato/Raytracer/blob/dd70db0763836bb7455e5c1e6f4ec3fad23809ad/ray/ray.c

241 lines (197 sloc) | 5.94 KB

Raw Blame

```c
#include <ray.h>
#include <stretchy_buffer.h>

#include <stdio.h>
#include <stdbool.h>
#include <math.h>

#define INF 1000000.f

typedef struct {
    float x, y, z;
} V3;

typedef struct {
    V3 center;
    float radius;
    RGBA color;
    float specular;
} Sphere;

typedef enum {
    AMBIENT, POINT, DIRECTIONAL
} LightType;
```