HW4 Ray-Tracing

sikun@ucsb.edu

Requirements

- **Implement a simple ray tracer:**
  - Parse a scene description file and draw the scene using ray tracing
  - Only one ray needs to be traversed per pixel
  - One shadow ray is traced per light source
  - One reflective ray is traced for each ray-object intersection
  - No refractive ray
- **Deadline: 11:59pm, December 8th (Friday)**
- **Source code & Makefile**
  - ./raytracer scene_description_file output_image

Scene Description File Format

- Consists of three sections:
  - Camera
  - Object
    - sphere
    - plane
  - Light
- Only one camera but may have multiple objects and lights.

Sample scene file

```
camera 400

sphere
        dimension 1
        center 0 1 -5
        reflectivity .7
        color .3 .3 .3
plane
        dimension 4 4
        center 0 -1 -5
        normal 0 1 0
        headup 0 0 1
        texture wood_tex.ppm


light
        location -1 2 -2
        color 1 .7 .7

light
        location 1 0 -6
        color .3 .3 1
```

Camera

- perspective view
- Eye is located at the global origin.
- The camera axes line up with the global axes.
- The image plane is located at z = -1 and of a size 1x1 centered on the z-axis.
- The only parameter the camera has is its spatial resolution. Hence, a line like

$$\text{camera } 1000$$

- Aspect ratio is always 1:1

Object

- sphere & plane
- Common attributes
    - Center location
    - Dimension
    - Color
    - Reflectivity
    - Texture

Object - sphere

sphere

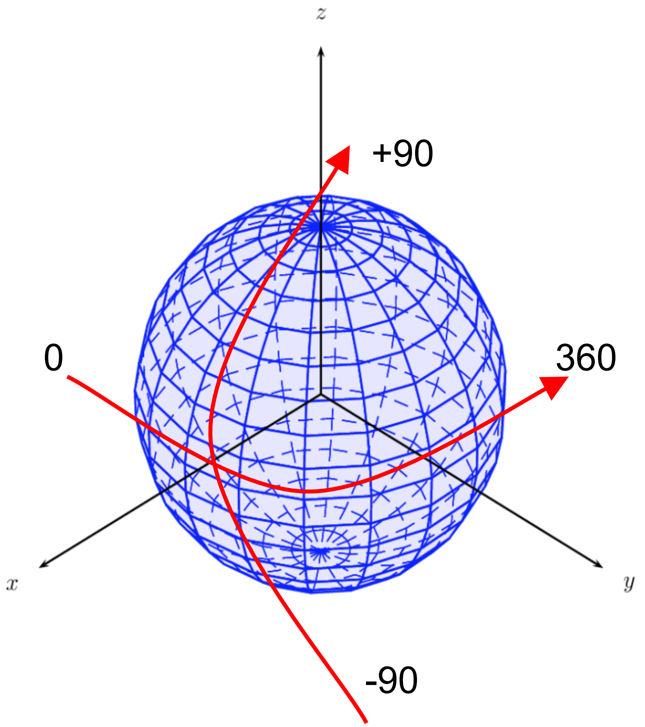
dimension 5

center 10 10 -10

color 1.0 0 0

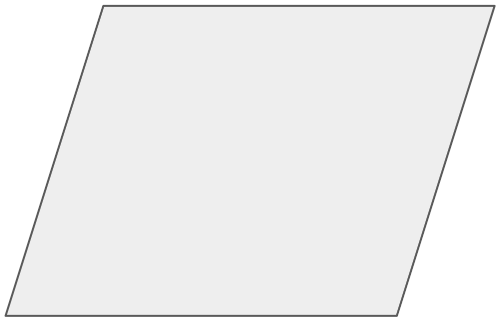
reflectivity 0.5

texture wood.ppm



Object - plane

```
plane
	dimension 5 10
	center 10 -5 -10
	color 1.0 0 1.0
	normal 0 0 1
	headup 1 1 0
	reflectivity 0.7
	texture wood.ppm
```



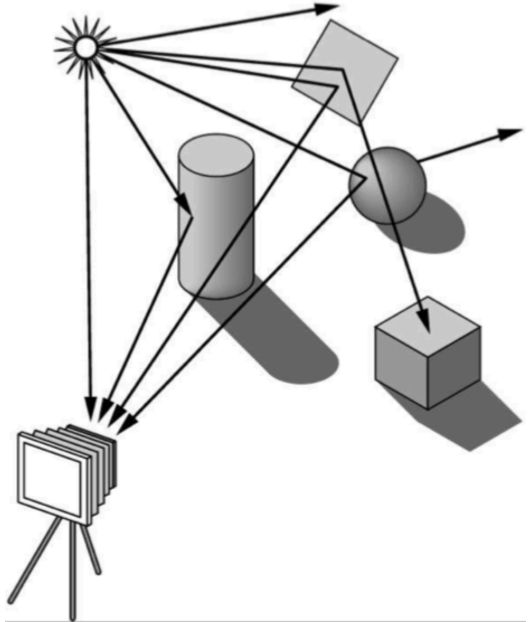
Light

- Point light sources
- Each light has two sets of parameters:
  - location(x, y, z)
  - color(r, g, b)   0<= r,g,b <=1

```
light
    location 20 -5 -10
    color 1.0 1.0 1.0
```

Forward Ray Tracing

- Rays as paths of photons in world space
- Forward ray tracing: follow photon from light sources to viewer
- Problem: many rays will not contribute to image!



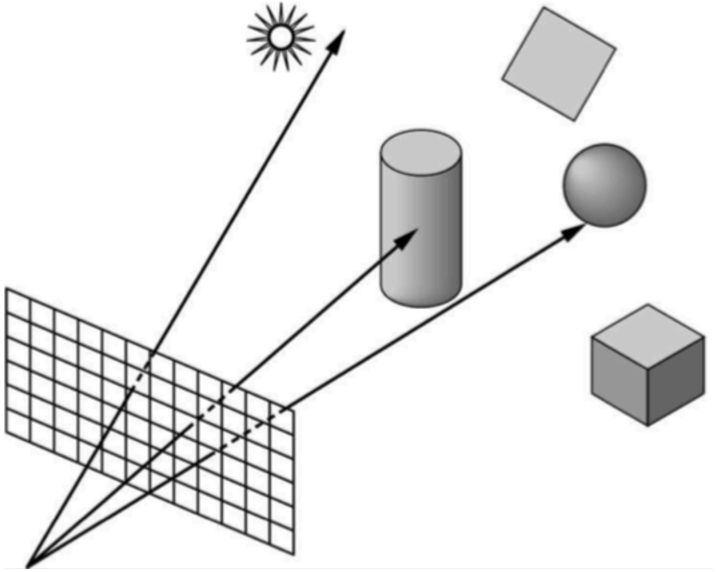
Backward Ray Tracing

- Ray-casting: one ray from center of projection through each pixel in image plane

- Illumination

1. Phong (local as before)
2. Shadow rays
3. Reflection
4. Refraction

- 3 and 4 are recursive



Construct a Ray

- $\mathbf{p}(t) = \mathbf{e} + t\,(\mathbf{s}-\mathbf{e}) = t\mathbf{s}$
- **e**: eye (camera) position (known)

  s: pixel position (known after knowing the resolution)
- Pixel position: usually pick the center of a pixel (half)

Ray-Sphere Intersection

- Problem: Intersect a line with a sphere
  - ✓ A sphere with center $\mathbf{c} = (x_c, y_c, z_c)$ and radius R can be represented as:

    $$(x-x_c)^2 + (y-y_c)^2 + (z-z_c)^2 - R^2 = 0$$

  - ✓ For a point **p** on the sphere, we can write the above in vector form:

    $(\mathbf{p}-\mathbf{c}).(\mathbf{p}-\mathbf{c}) - R^2 = 0$ (note '**.**' is a dot product)

  - ✓ We can plug the point on the ray $\mathbf{p}(t) = \mathbf{e} + t\,\mathbf{d}$

    $(\mathbf{e}+t\mathbf{d}-\mathbf{c}).(\mathbf{e}+t\mathbf{d}-\mathbf{c}) - R^2 = 0$ and yield

    $$(\mathbf{d}.\mathbf{d})\, t^2 + 2\mathbf{d}.(\mathbf{e}-\mathbf{c})t + (\mathbf{e}-\mathbf{c}).(\mathbf{e}-\mathbf{c}) - R^2 = 0$$

Ray-Sphere Intersection

- When solving a quadratic equation

  $at^2 + bt + c = 0$

  We have
- Discriminant $d = \sqrt{b^2 - 4ac}$
- and Solution $t_\pm = \frac{-b \pm d}{2a}$

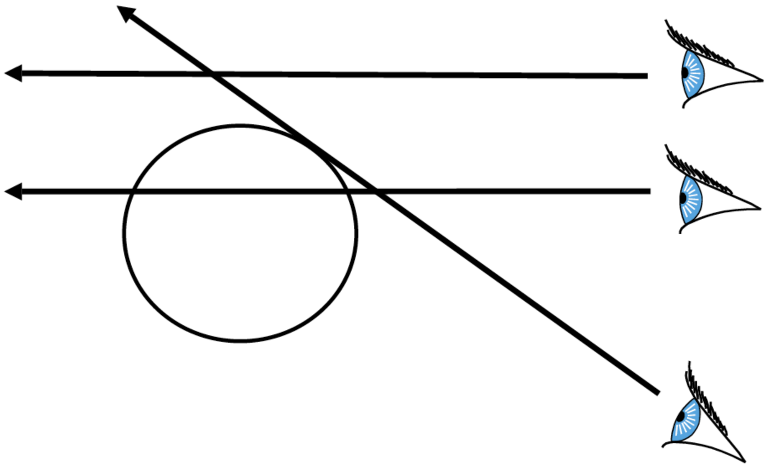
Ray-Sphere Intersection

$$d = \sqrt{b^2 - 4ac}$$

$b^2 - 4ac < 0 \Rightarrow$ **No intersection**

$b^2 - 4ac > 0 \Rightarrow$ **Two solutions (enter and exit)**

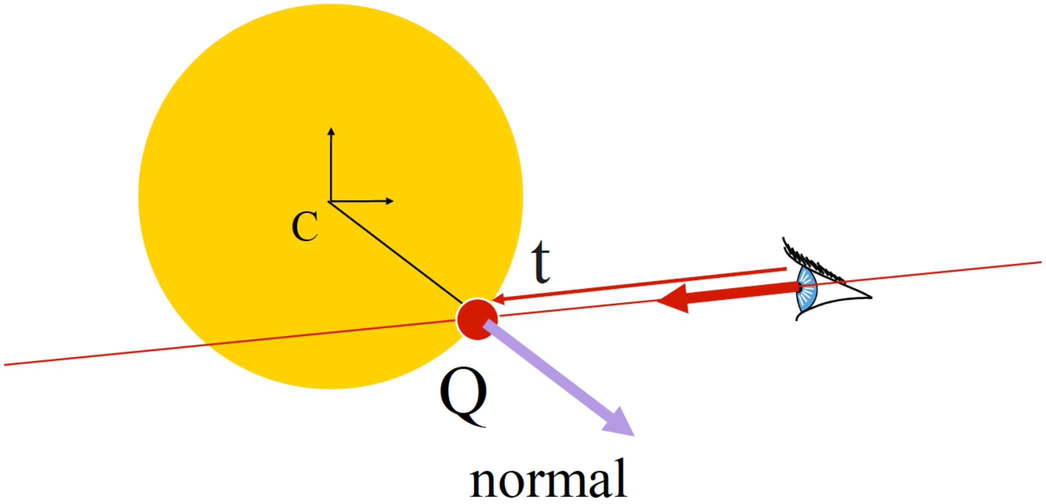
$b^2 - 4ac = 0 \Rightarrow$ **One solution (ray grazes sphere)**



Calculating Normal

- Needed for computing lighting

  Q = P(t) – C … and remember Q/||Q||



Ray-plane intersection

- Given plane normal (a,b,c) and one point on the plane (center):

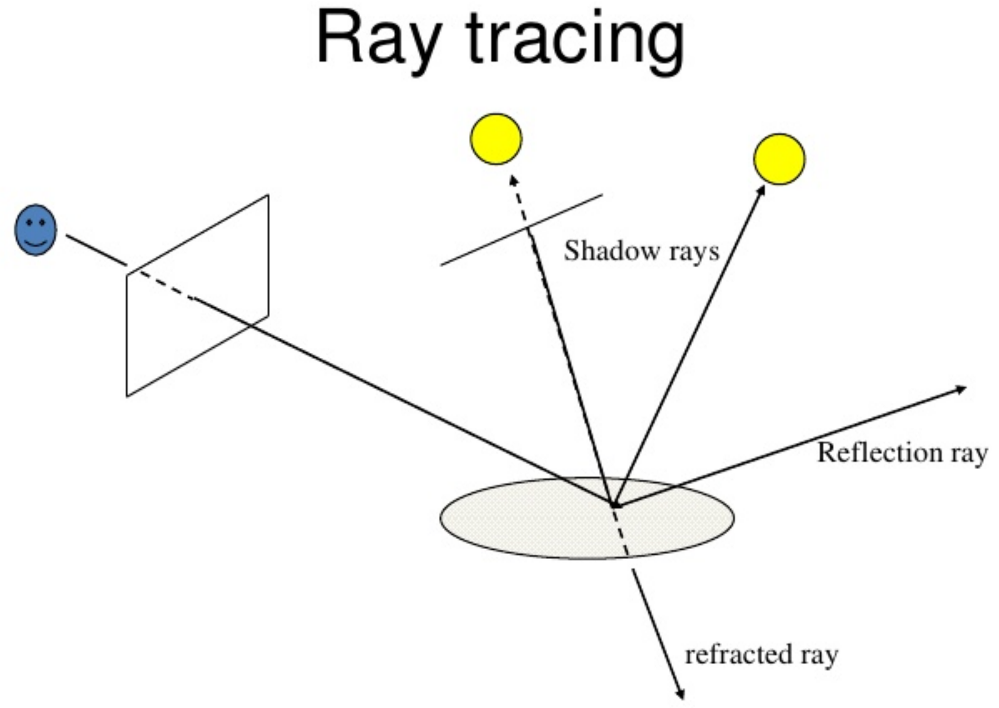
  plug in the point coordinate to get the plane equation

  $ax+by+cz+d=0$

- Calculate the intersection point: plug in the ray equation

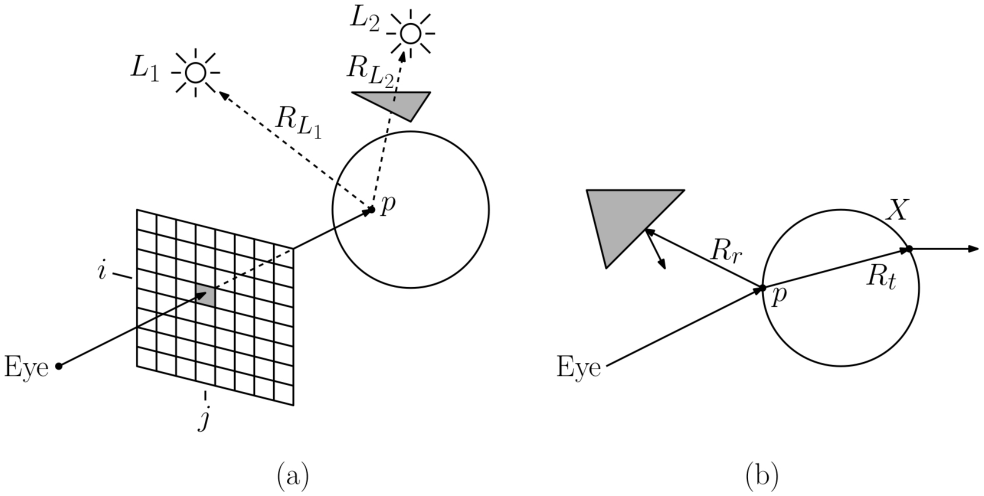
  $a(e_x+td_x) + b(e_y+td_y) + c(e_z+td_z) + d = 0 \Rightarrow$ get t $\Rightarrow$ get point $\mathbf{e}+t\mathbf{d}$

Casting shadows: hit-point to each light source



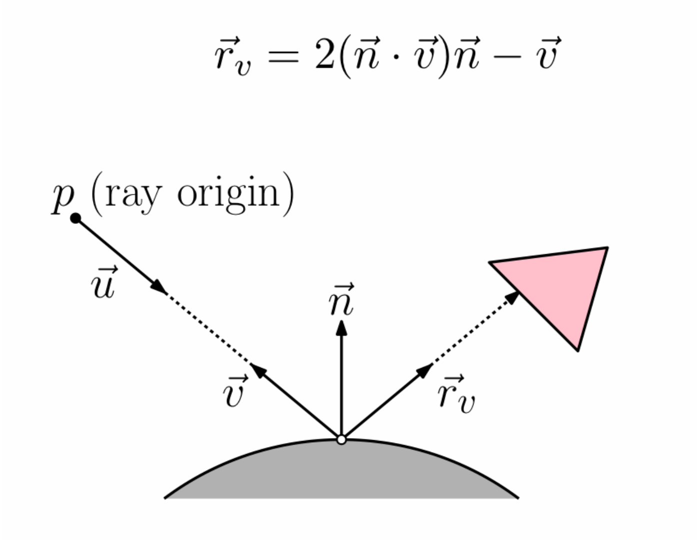
Reflections

- Recursive (stop when hitting a non-reflective object, return its color)

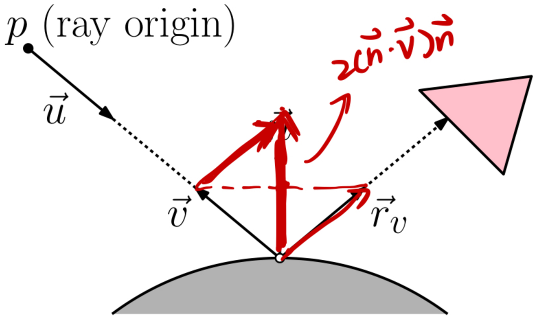


- What if the lights go back and forth between two mirrors?

Reflective direction



v is normalized –u
n is normalized normal



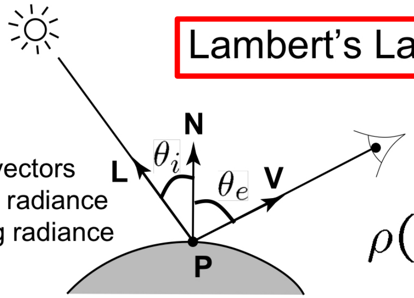
Illumination model

- Phong

 - k_a: const. e.g. k_a = 0.1 ➔ contribute $\boxed{k_a * I_a}$ (I_a is outgoing radiance: color_obj)

 - If not in shadow ➔ light source I_i (incoming radiance) has a contribution:

Determined by the cosine law.
N, L are unit vectors
k_d varies for different materials
(you can simply choose that to be color_obj)



L, **N**, **V** unit vectors
I_e = outgoing radiance
I_i = incoming radiance

$$\boxed{\text{Lambert's Law: } I_e = k_d \mathbf{N} \cdot \mathbf{L} I_i}$$

BRDF for **Lambertian surface**

$$\rho(\theta_i, \phi_i, \theta_e, \phi_e) = k_d cos\theta_i$$

Contribution of reflection

- += Final color * reflectivity
- You can try different things like …

    (Final color)^n * reflectivity (larger n, smaller range of reflection)

- Illumination model and parameters are not fixed, you can play with them and choose what you like the best

Program Skeleton

```
for (each scan line) {
    for (each pixel in scan line) {
        compute ray direction from eye to pixel
        for (each object in scene) {
            if (intersection and closest so far) {
                record object and intersection point
            }
            accumulate pixel colors
            -   shadow ray color
            -   reflected ray color (recursion)
        }
    }
}
```

Demo

Q & A