CSC 4356
Interactive Computer Graphics
Lecture 21: Ray Tracing (Part 1)

Jinwei Ye

http://www.csc.lsu.edu/~jye/CSC4356/

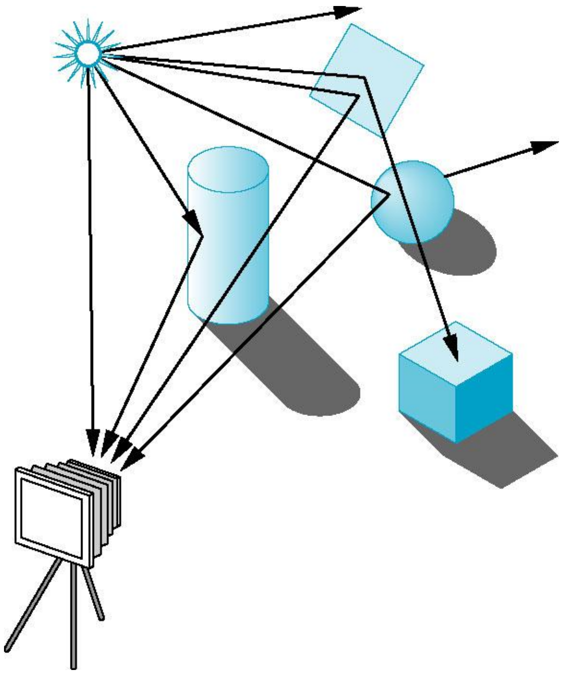
Tue & Thu: 10:30 - 11:50am
218 Tureaud Hall

Illumination Models

- Interaction between light sources and objects in scene that results in perception of intensity and color at eye
- Local vs. global models
  - **Local illumination**: Perception of a particular primitive only depends on light sources directly affecting that one primitive
    - Geometry
    - Material properties
  - **Global illumination**: Also take into account indirect effects on light of other objects in the scene
    - Shadows cast
    - Light reflected/refracted

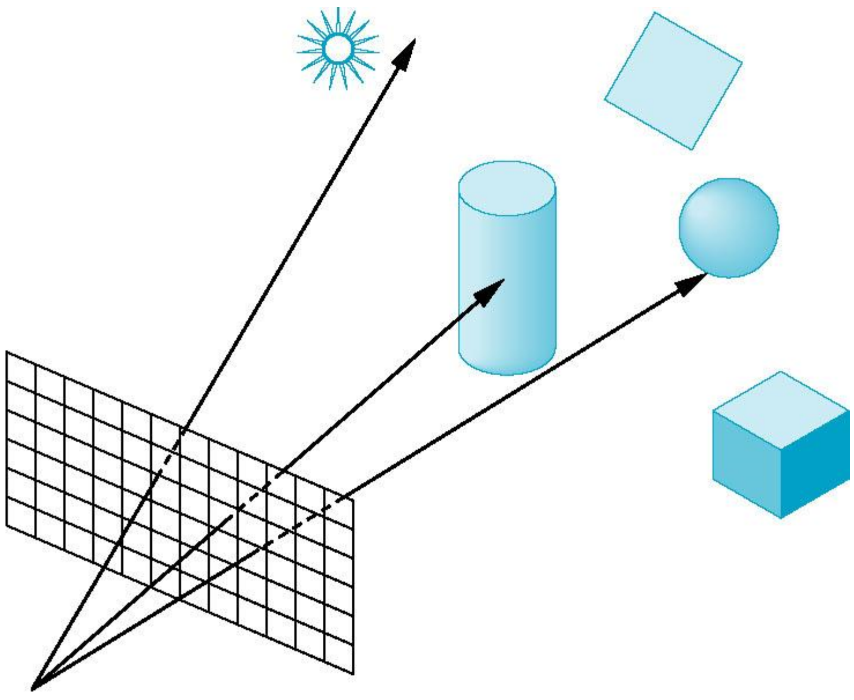
“Forward” Ray Tracing

- Proper global illumination means simulation of physics of light
  - Rays are emitted from light source, bounce off objects in the scene, and some eventually hit our eye, forming an image
- Problem: Not many rays make it to the image
  - Waste of computation for those that don’t



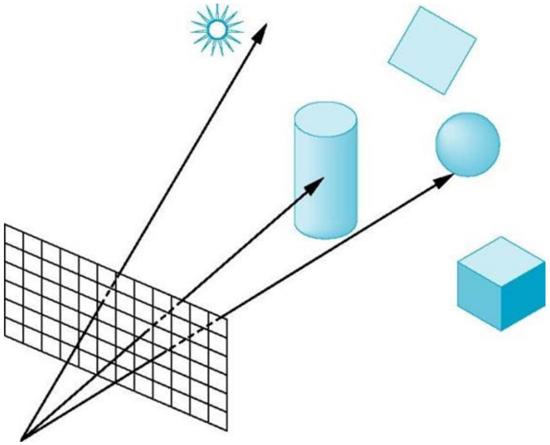
“Backward” Ray Tracing

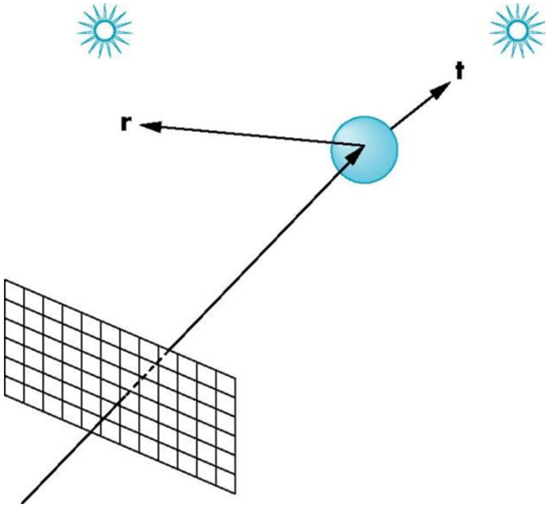
- Idea: Only consider those rays that create the image
  - Trace rays from pixels



Backward Ray "Following": Types

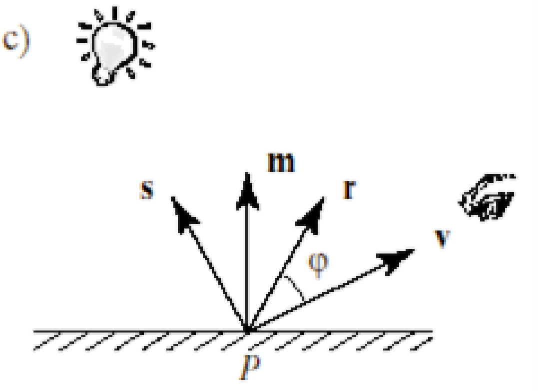
- **Ray casting**: Compute illumination at first intersected surface point only
  - – Takes care of hidden surface elimination
- **Ray tracing**: Recursively spawn rays at hit points to simulate reflection, refraction, etc.





Lighting a point

- Let c = (r, g, b) be **perceived** material color, **s**(l) be color of light l
- Sum over all lights l for each color channel (clamp overflow to [0, 1]):



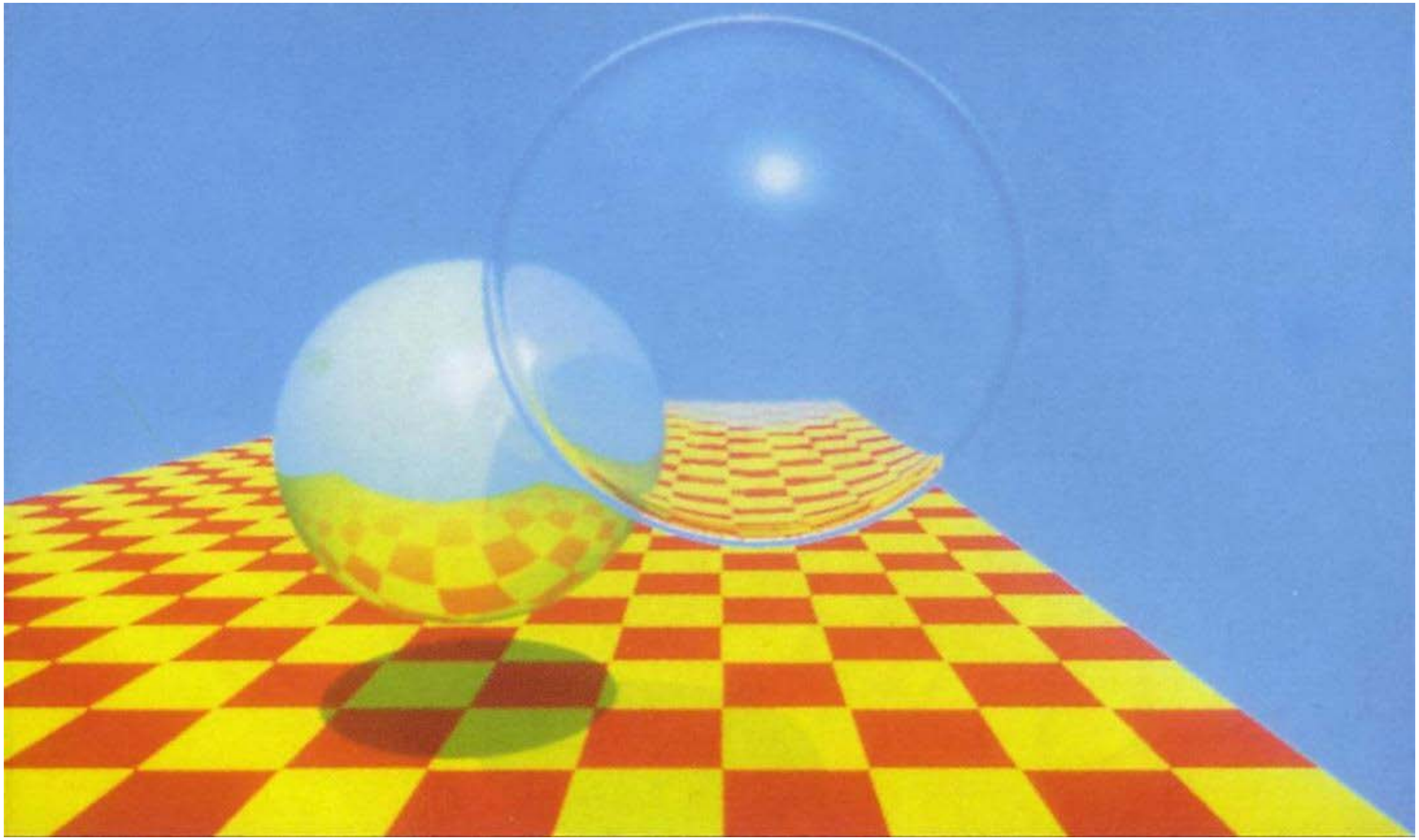
$$c_{total} = \sum_{l} c_{amb}(l) + c_{diff}(l) + c_{spec}(l)$$

$$c_{amb}(l) = m_{amb} \otimes s_{amb}(l)$$

$$c_{diff}(l) = \max(0, \mathbf{n} \cdot \mathbf{l}(l)) m_{diff} \otimes s_{diff}(l)$$

$$c_{spec}(l) = \max(0, \mathbf{v} \cdot \mathbf{r}(l))^{shine} m_{spec} \otimes s_{spec}(l)$$

One of the earliest ray-traced scenes



Ray Tracing: Example

Ray Tracing: More recent example

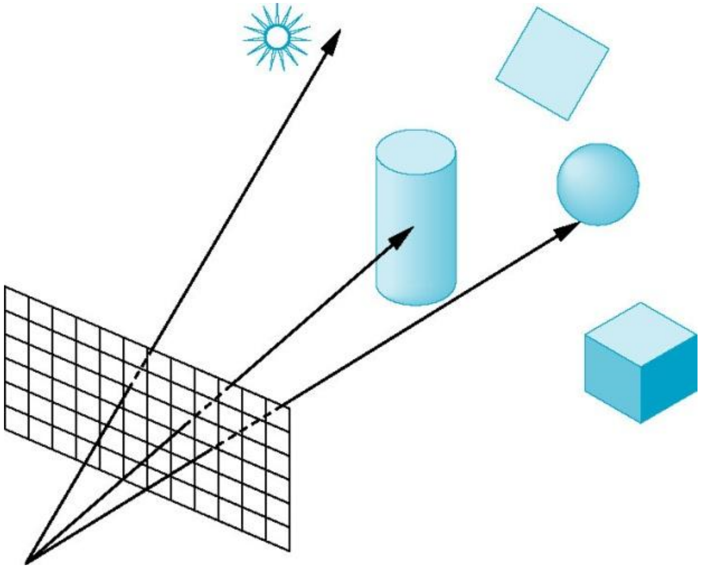
Ray Tracing: Example from “Cars”

Ray Tracing: Another car

From a CAD model using Nvidia’s mental ray
(http://www.nvidia-arc.com/products/nvidia-mental-ray)

Ray Casting

- Simulation of irradiance (incoming light ray) at each pixel
- Send a ray from the focal point through each pixel and out into the scene and see if it **intersects** an object
  - Use background color if nothing is hit

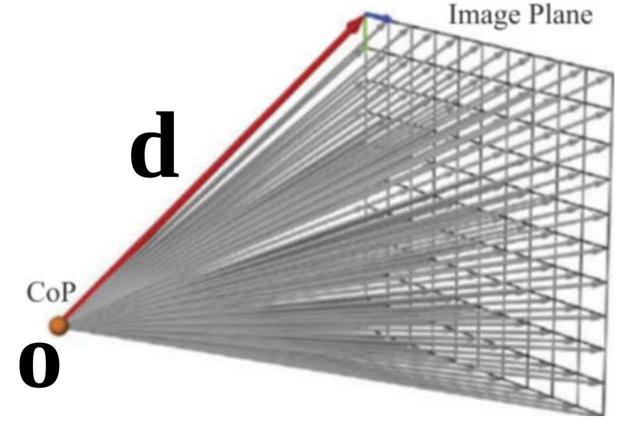


- Local shading model is applied to **first** point hit

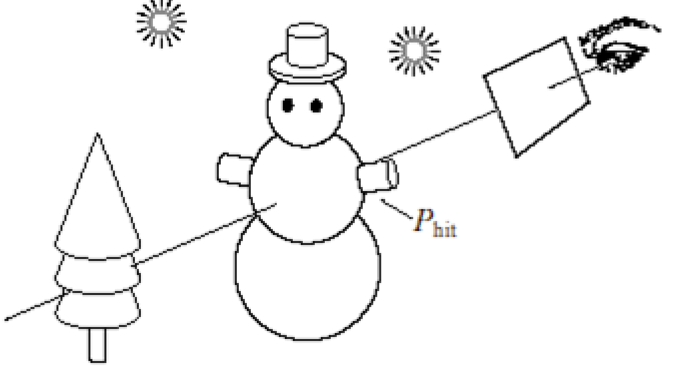
Ray Casting: Details

- Must compute 3D ray into scene for each 2D image pixel

$$\mathbf{p} = \mathbf{o} + t\mathbf{d}$$



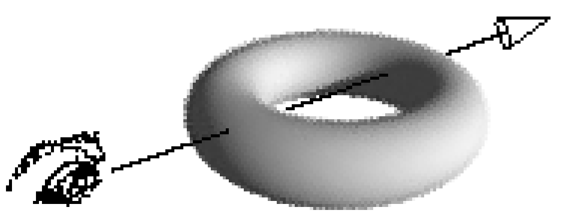
- Compute 3-D **position** of ray's intersection with nearest object and normal at that point
- Apply lighting model such as Phong to get color at that point and fill in pixel with it



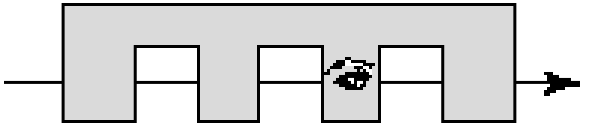
Does Ray Intersect any Scene Primitives?

- Test each primitive in scene for intersection individually
- Different methods for different kinds of primitives
  - Polygon
  - Sphere
  - Cylinder, torus
  - Etc.
- Make sure intersection point is **in front of eye** and **nearest one**

a)



b)



Ray-Sphere Intersection I

- Combine implicit definition of sphere

$$|\mathbf{p}-\mathbf{p}_c|^2 - r^2 = 0$$

with ray equation

$$\mathbf{p} = \mathbf{o} + t\mathbf{d}$$

Thus we have

$$|\mathbf{o}+t\mathbf{d}-\mathbf{p}_c|^2 - r^2 = 0$$

Ray-Sphere Intersection II

- Substitute $\Delta \mathbf{p} = \mathbf{p}_c - \mathbf{o}$ and use

$$|\mathbf{a}+\mathbf{b}|^2 = |\mathbf{a}|^2 + 2\mathbf{a}\cdot\mathbf{b} + |\mathbf{b}|^2$$

- To solve for t, resulting in a quadratic equation with roots given by:

$$t = d \cdot \Delta p \pm \sqrt{(d \cdot \Delta p)^2 - (|\Delta p|^2 - r^2)}$$

  – d is a unit vector |d| = 1

- Notes
  – Real solutions mean there actually are 1 or 2 intersections -- what does this correspond to?
  – Negative solutions are behind eye

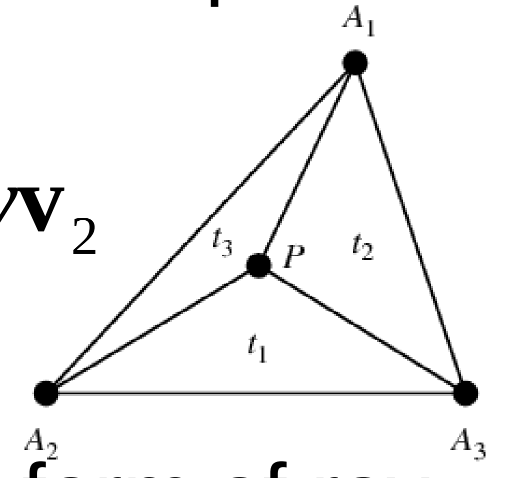
Ray-Polygon Intersection

- Express point **p** on a ray as some distance t along direction d from origin **o**: **p** = **o** + t**d**
- Use plane equation **n** · **x** + m= 0, substitute **o** + t**d** for **x**, and solve for t
- Only positive t's mean the intersection is in front of the eye
- Then plug t back into **p** = **o** + t**d** to get **p**
- Is the 2-D location of p on the plane inside the 2-D polygon?
  - For convex polys, Cohen-Sutherland-style outcode test will work

Ray-Triangle Intersection

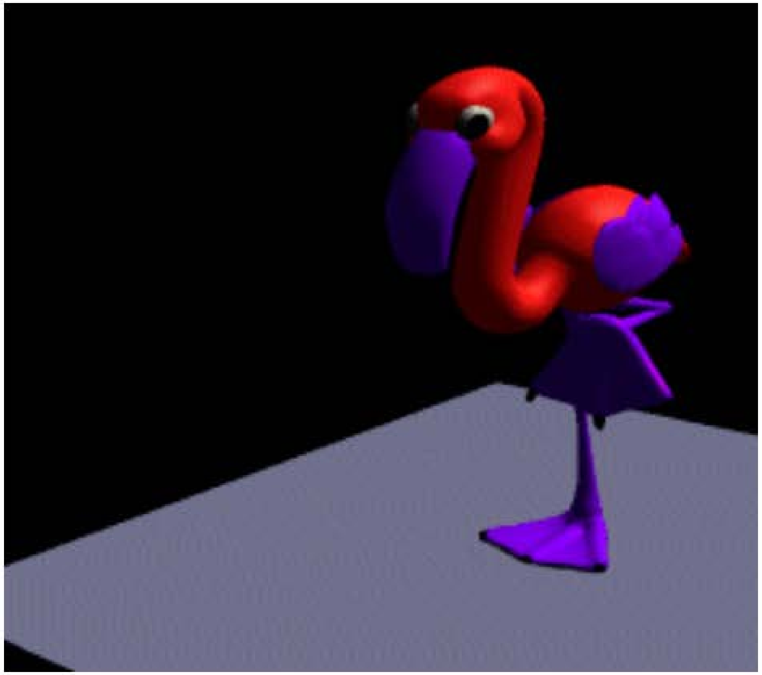
- Direct barycentric coordinates expression

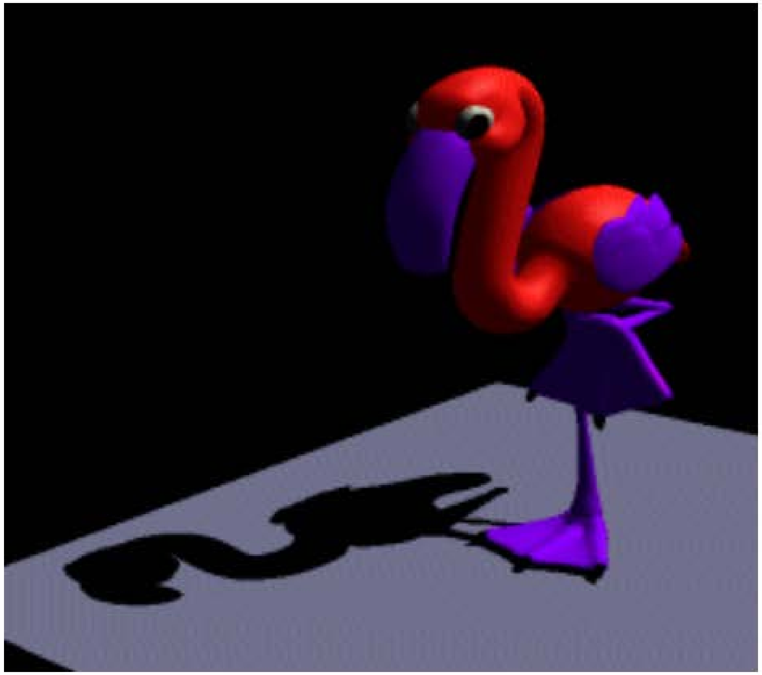
$$\mathbf{t}(u,v) = (1-u-v)\mathbf{v}_0 + u\mathbf{v}_1 + v\mathbf{v}_2$$



- Set this equal to parametric form of ray **o** + t**d** and solve for intersection point (t, u, v)
- Only inside triangle if u, v, and 1 – u – v are between 0 and 1

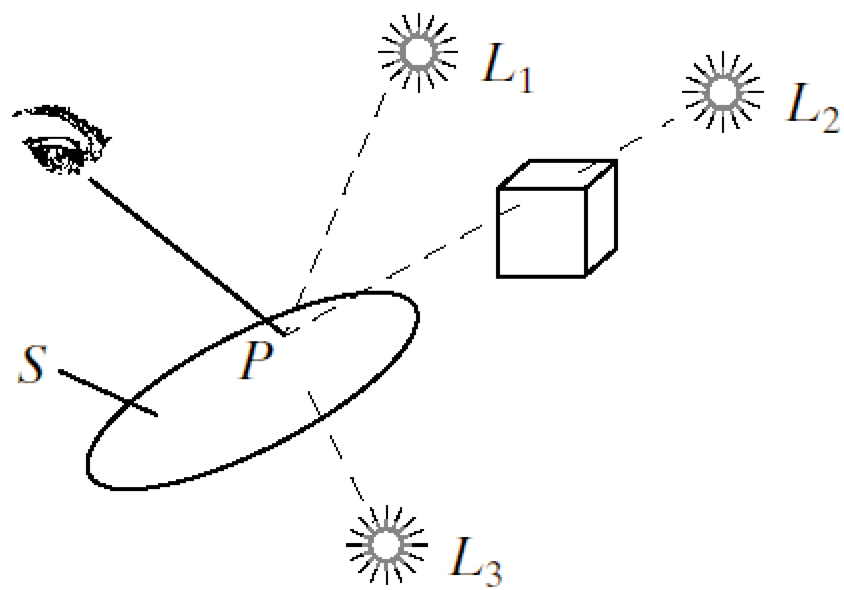
How to render shadow?





Shadow Rays

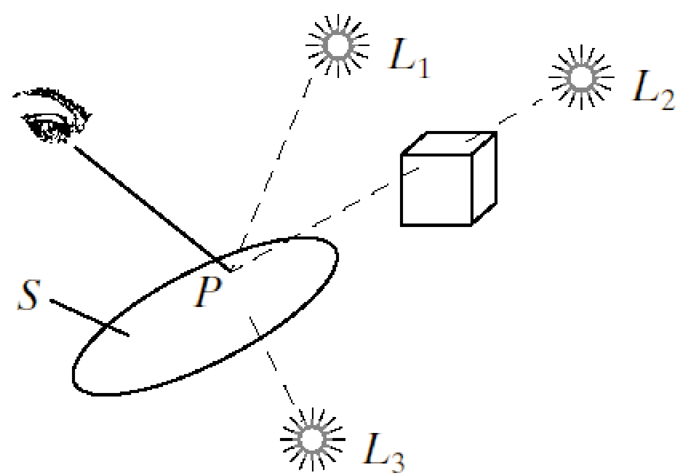
- For point being locally shaded, spawn new ray in each light direction and check for intersection to make sure light is "visible"



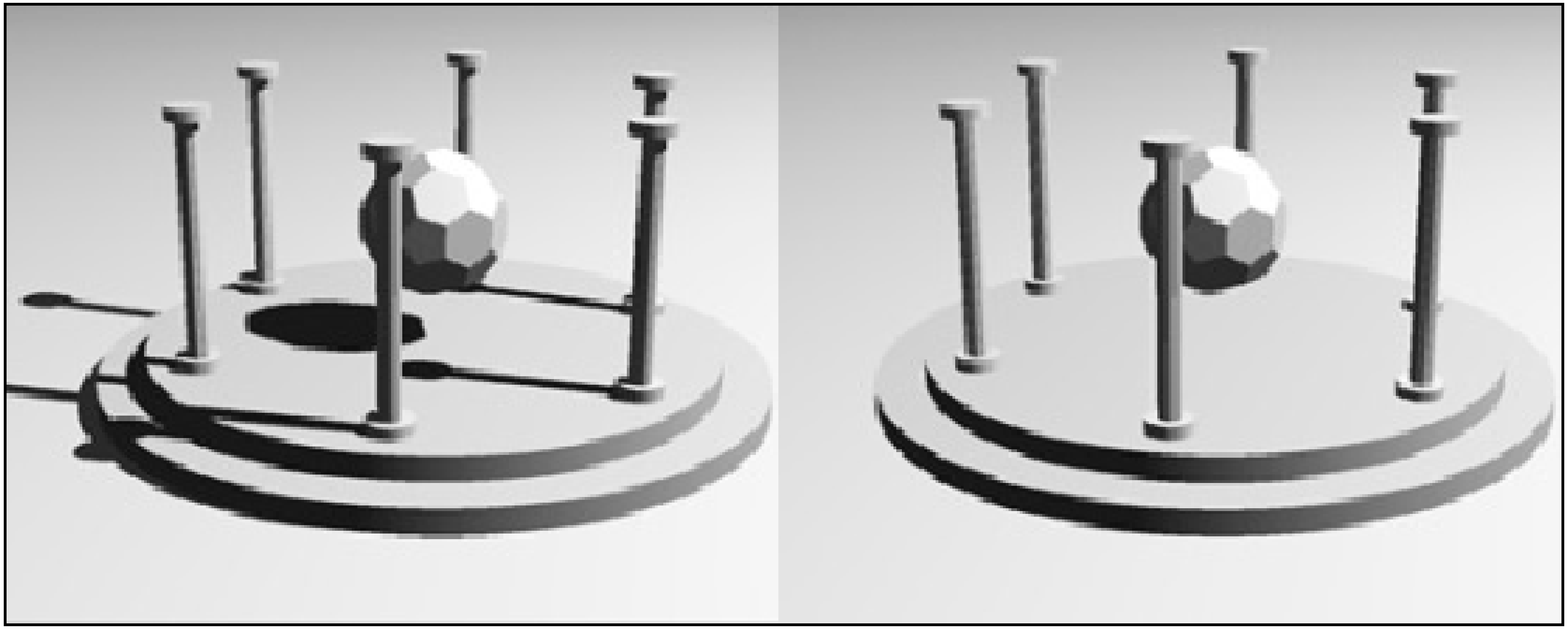
Shadow Rays

- For point **p** being locally shaded, only add diffuse & specular components for light l if light is not blocked
- Test for occlusion of l for **p**:
  - Spawn **shadow ray** for l with origin **p**, direction **l**(l)
  - Check whether shadow ray intersects any scene object
  - Intersection only "counts" if:

$$0 < t < |\mathbf{p}_l - \mathbf{p}|$$



Ray-Cast Scene with and without Shadows



Next Time…

- More about ray tracing
- Programming assignment 3 is due today!
- Office hour change (this week only)
  – Friday (tomorrow) morning 10:00-12:00