Tutorial 6

TA: Mengyun Liu, Hongtu Xu

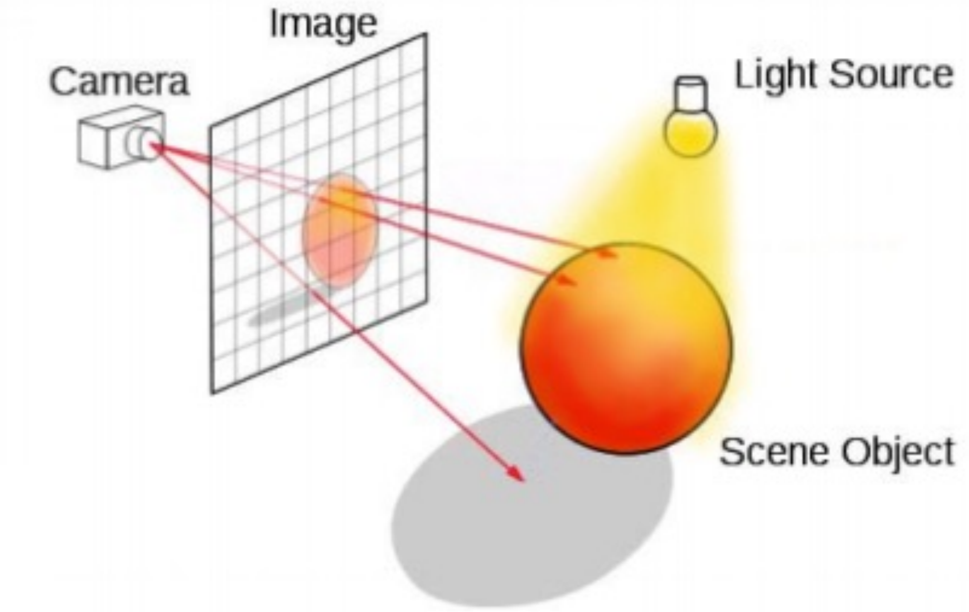
Agenda

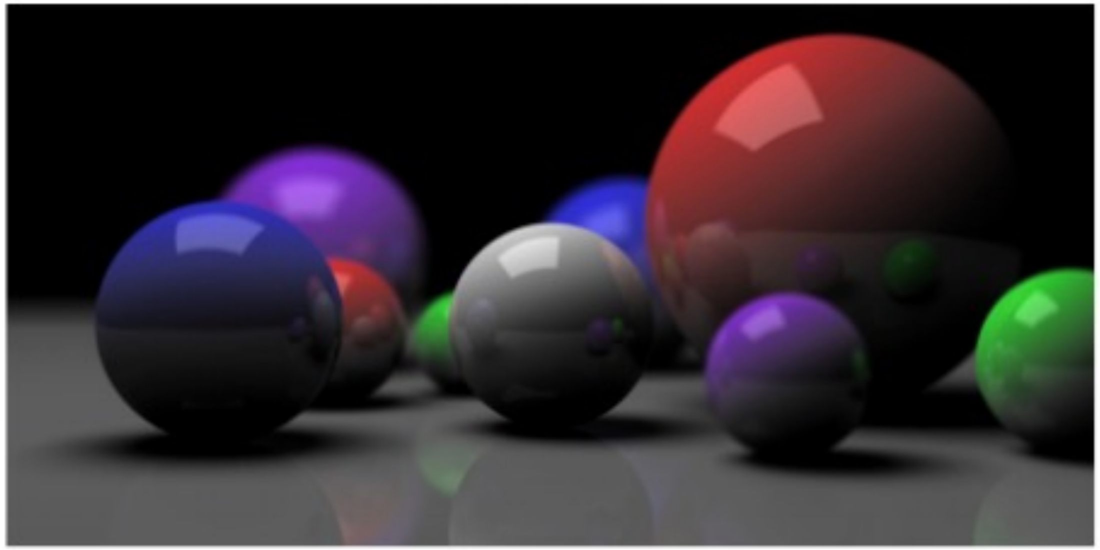
- Ray Tracing
- Direct Lighting
- Radiance
- Assignment 3

Ray Tracing

Ray Tracing

- Simulation of the realistic imaging process.
- Recall the imaging process for pin-hole cameras.



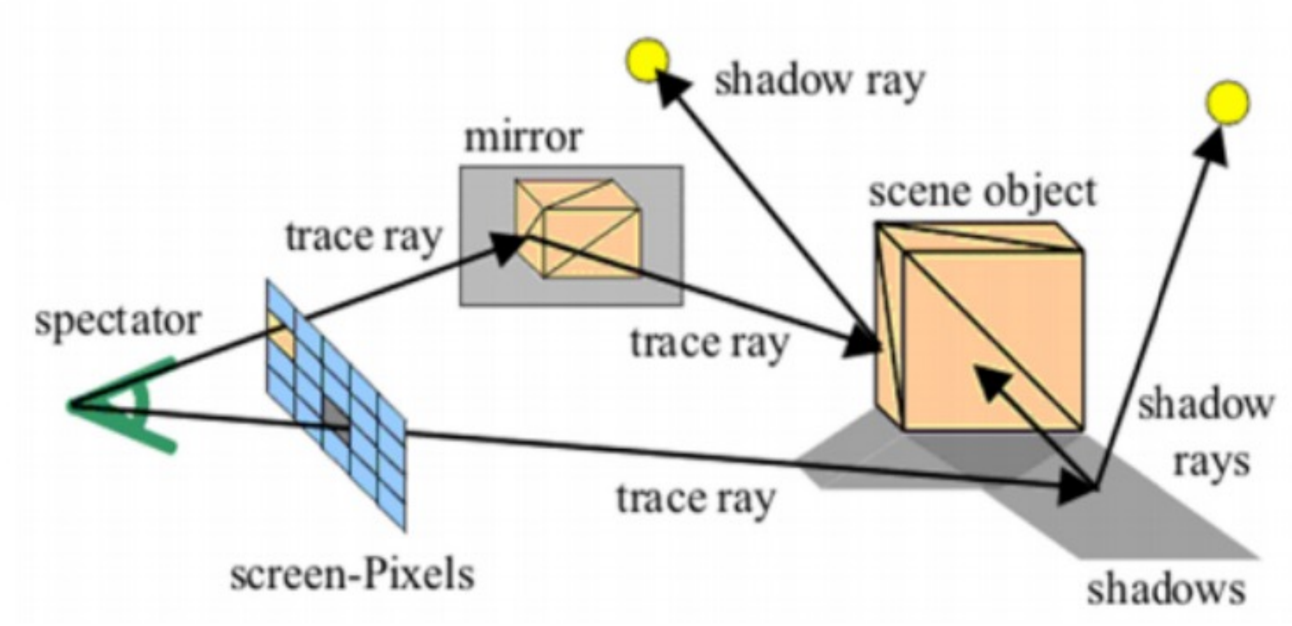


Ray Tracing

- Differing from projection-based rendering, e.g., OpenGL, we cast rays into the scene to evaluate the radiance brought back by ray tracing.
- Ray tracing routine
  - In general, starts from the camera's pin-hole and directs to a point on the imaging plane.
  - When the ray intersects with some objects, perform reflections on the interaction point.
  - Repeat until the ray hits any light sources.
  - Traceback and compute the radiance brought back.

Ray Tracing

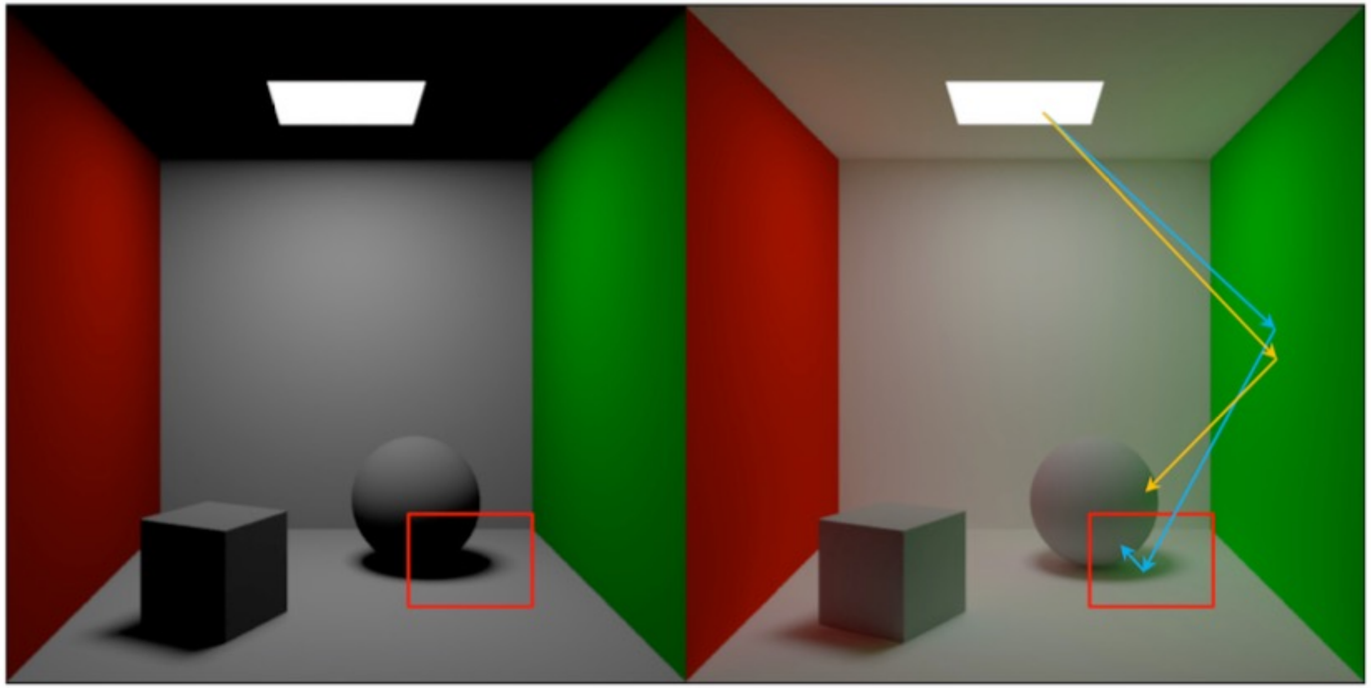
- For simplification, we consider merely two kinds of reflection
  - Specular reflection (e.g., mirror, icy surface)
  - Diffusion reflection (e.g., rough wall)



Direct Lighting

Direct Lighting

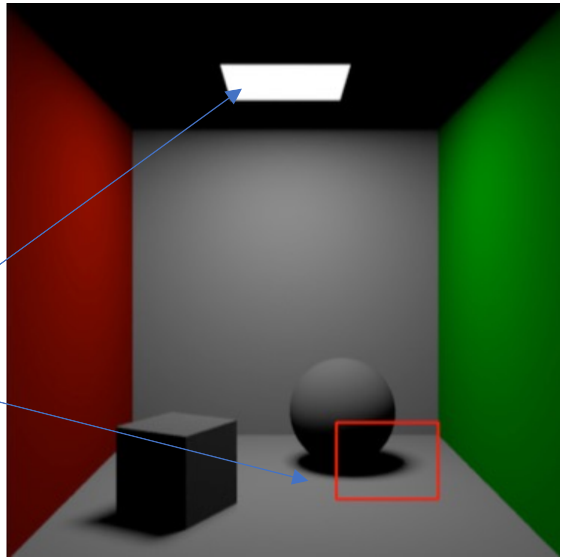
- This routine is somewhat problematic and inefficient since the ray could be reflected thousands of times until meeting any light source.
- A practical method is to decompose the lighting into **direct** and **indirect**.



Direct Lighting

- Direct lighting: once the shot rays hit any object, we directly reflect rays to the light sources.
- Shadow: But those rays may hit other objects when they are reflected to the light sources (sheltering/occlusion)
  - Which will result in the shadow
- If the light sources are area lights, some shadow has chances to be lighted
  - Which will result in the soft shadow.

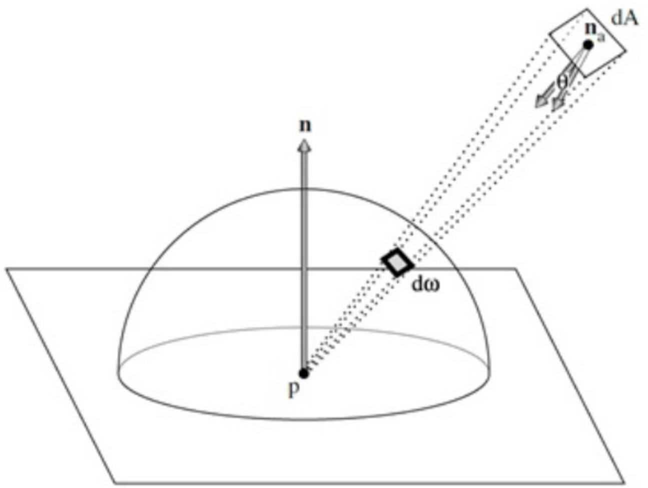
Area light, soft shadow, but direct illumination



Direct Lighting

- Rendering equation:
  - $L_o(p, \omega_o) = \int_S f(p, \omega_i, \omega_o) L_i cos\theta_i d\omega_i$
  - For discrete case, integral can be converted to summation
  - For Phong shading model, BRDF is modeled as diffusion + specular
- Discrete + Phong Shading (diffusion)
  - $L_o(p, \omega_o) = \sum_{L_i} diff \cdot L_i cos\theta_i$

$$L_o(\mathrm{p}, \omega_o) = \int_A f(\mathrm{p}, \omega_o, \omega_i)\, L_i(\mathrm{p}, \omega_i)\, |\cos\theta_i|\, \frac{\mathrm{d}A\ \cos\theta}{r^2}$$



Radiance

Radiance

- Notice that what we consider now is radiance for each pixel on the imaging film. Not RGB color yet.
- Color represents waves with different wavelengths, similar to radiance. In computer area, lights are decomposed to 3 components: Red, Green, Blue.
- For simplification in Phong lighting model, we can decompose radiance to RGB components and compute the radiance for the three channels, respectively.
- Finally, convert radiance to color via tone mapping (Gamma correction).

Assignment 3

Assignment 3: Ray Tracing with Direct Lighting

- **[must]** You are required to implement a pin-hole camera, which is able to shoot rays. And there should be at least one ray per pixel.
- **[must]** You are required to implement algorithms for the ray-triangle intersection (without the acceleration structure).
- **[must]** You are required to implement anti-aliasing for ray-tracing by using super sampling with the rotated grid.
- **[optional]** You may implement texturing.
- **[optional]** You may implement a mipmap for texturing in ray tracing.
- **[optional]** You may implement normal mapping.
- **[optional]** You may implement environment lighting with importance sampling.

Code Structure

include	
camera.h	Data structure and methods for manipulating cameras.
core.h	Header files and declarations
film.h	Data structure storing each pixel’s radiance.
geometry.h	Represents 3D geometries in the scene
integrator.h	Used to solve the rendering equation
interaction.h	Basic data structures for ray-object intersection
light.h	Represents light sources in the scene
material.h	Material information for geometries like Phong model
ray.h	Basic data structures for ray-object intersection
scene.h	Contains geometries and lights, with a camera attached
texture.h	Texture information for geometries

Working Flow

Create a scene with a camera

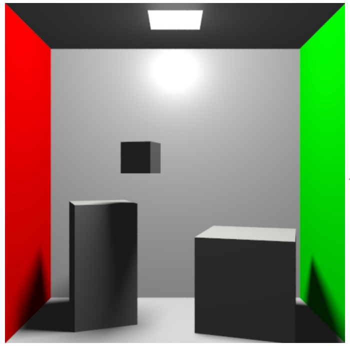
Add geometries and set a light

Send the scene to an integrator

Integrator traces rays in the scene and computes radiance

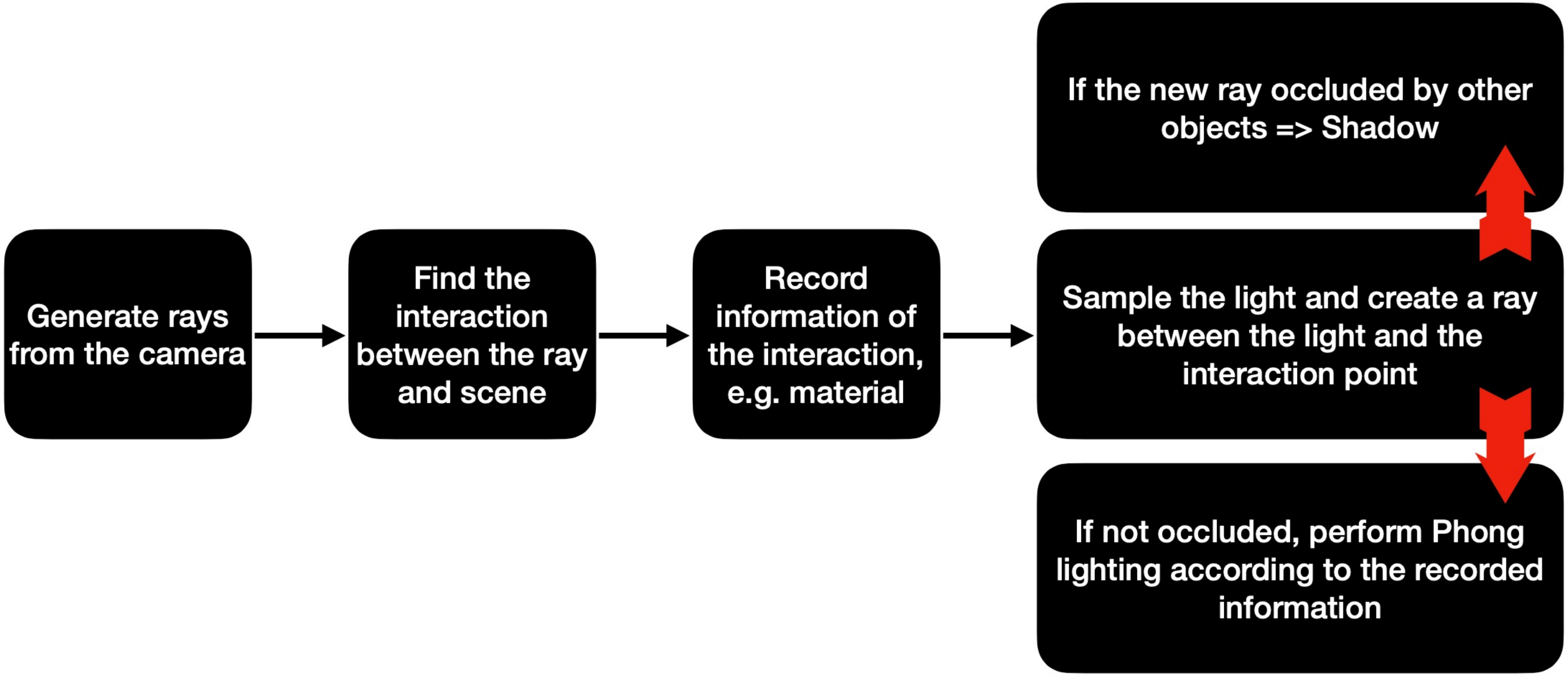
Convert radiance to color

Save the RGB image



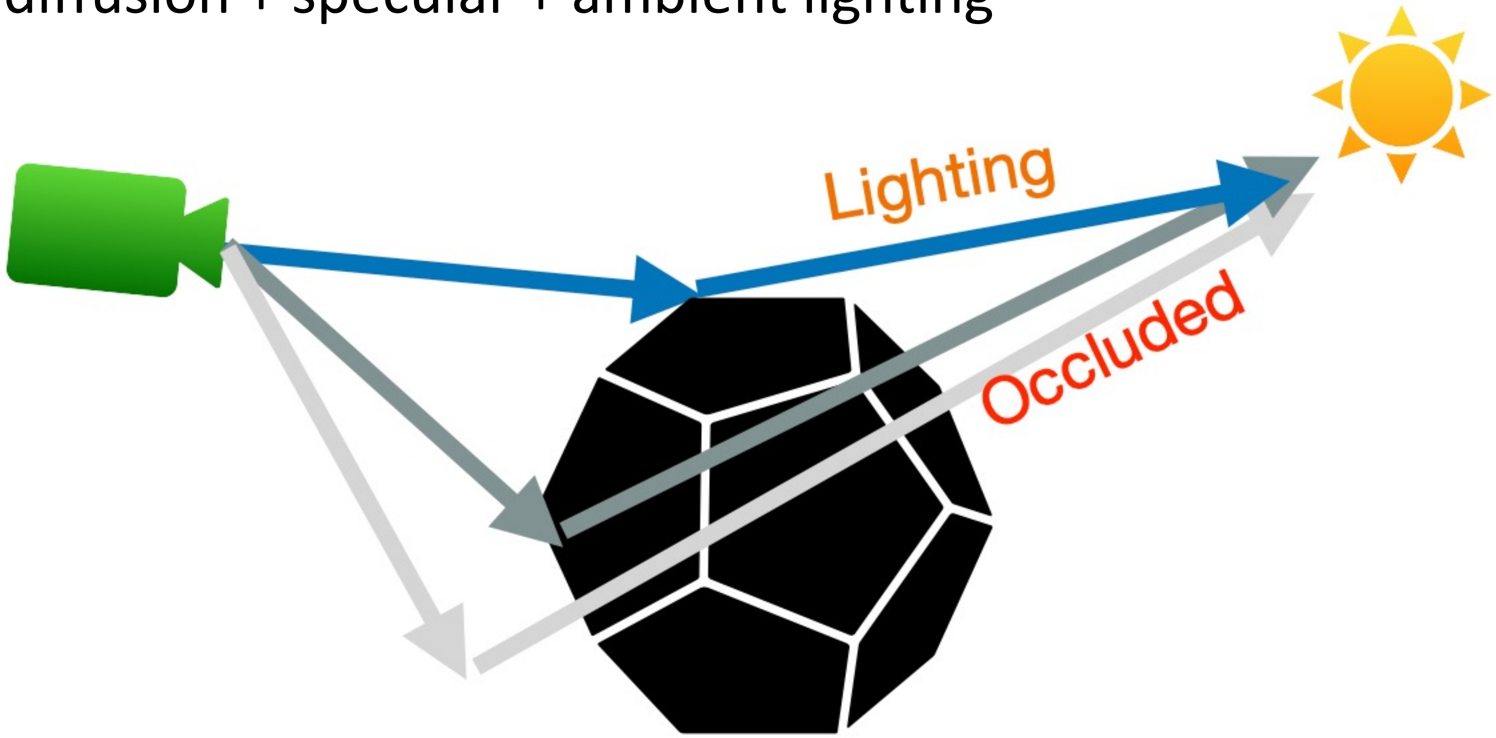
Done !

Working Flow in the Integrator



Algorithm for Direct Phong Lighting

- Create a new ray from P to a light
  - If intersects(scene, camera ray) at P source
  - If intersects(scene, new ray)
    - Return ambient lighting
  - Return diffusion + specular + ambient lighting



Ray-Triangle intersection

- Define
  - The point in the triangle plane as p1 = $(1 - b_1 - b_2)\mathrm{p}_0 + b_1\mathrm{p}_1 + b_2\mathrm{p}_2$
  - The point in the ray as p2 = O + t**d**
- We have the relationship p1 = p2

$$\begin{pmatrix} -\mathbf{d} & \mathbf{e}_1 & \mathbf{e}_2 \end{pmatrix} \begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \mathbf{s}$$

- Solve the linear equations using **Cramer's Rule**

$$\begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \frac{1}{|-\mathbf{d} \quad \mathbf{e}_1 \quad \mathbf{e}_2|} \begin{bmatrix} |\quad \mathbf{s} & \mathbf{e}_1 & \mathbf{e}_2| \\ |-\mathbf{d} & \mathbf{s} & \mathbf{e}_2| \\ |-\mathbf{d} & \mathbf{e}_1 & \mathbf{s}| \end{bmatrix}$$

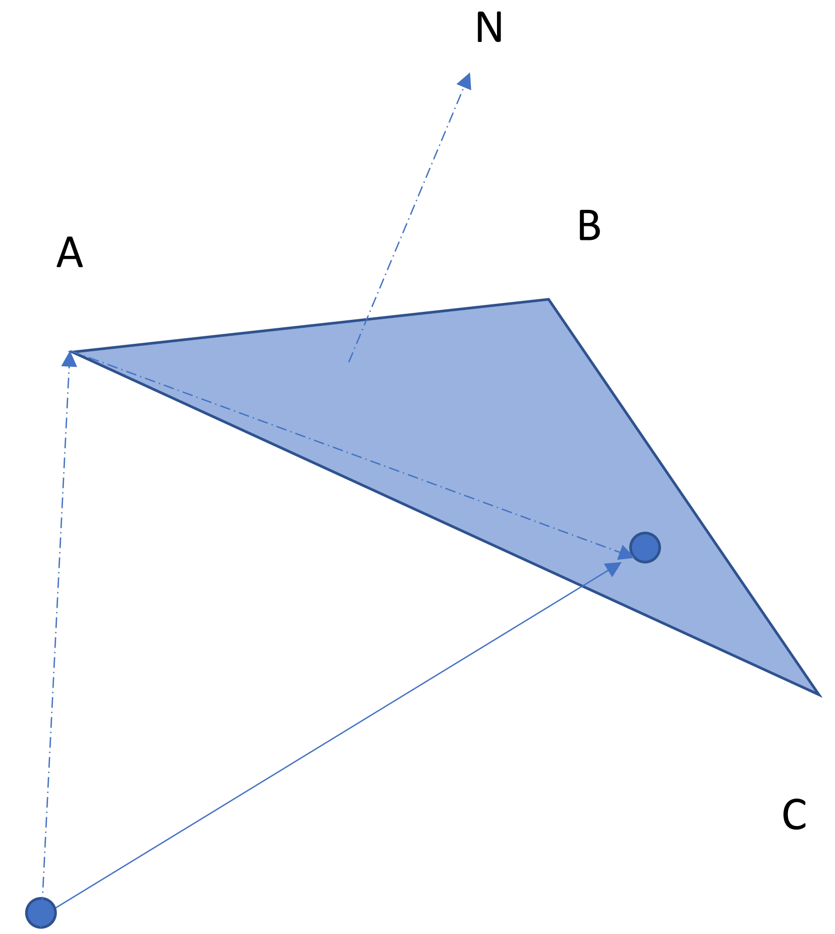
Ray-Triangle intersection

- **Is this solver efficient?**
  - No!
- **More observation**
  - Determinant identification in 3D

$$|\,\mathbf{a} \quad \mathbf{b} \quad \mathbf{c}\,| = -(\mathbf{a}\times\mathbf{c})\cdot\mathbf{b} = -(\mathbf{c}\times\mathbf{b})\cdot\mathbf{a}$$

$$\Downarrow$$

$$\begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \frac{1}{(\mathbf{d}\times\mathbf{e}_2)\cdot\mathbf{e}_1}\begin{bmatrix} (\mathbf{s}\times\mathbf{e}_1)\cdot\mathbf{e}_2 \\ (\mathbf{d}\times\mathbf{e}_2)\cdot\mathbf{s} \\ (\mathbf{s}\times\mathbf{e}_1)\cdot\mathbf{d} \end{bmatrix} \Rightarrow \begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \frac{1}{\mathbf{s}_1\cdot\mathbf{e}_1}\begin{bmatrix} \mathbf{s}_2\cdot\mathbf{e}_2 \\ \mathbf{s}_1\cdot\mathbf{s} \\ \mathbf{s}_2\cdot\mathbf{d} \end{bmatrix}$$



Thanks