Image Formation

Richard (Hao) Zhang

Introduction to Computer Graphics
CMPT 361 – Lecture 1

Readings

- Chapter 12.2 Ray tracing
- Chapter 8.11.5 Z-buffer

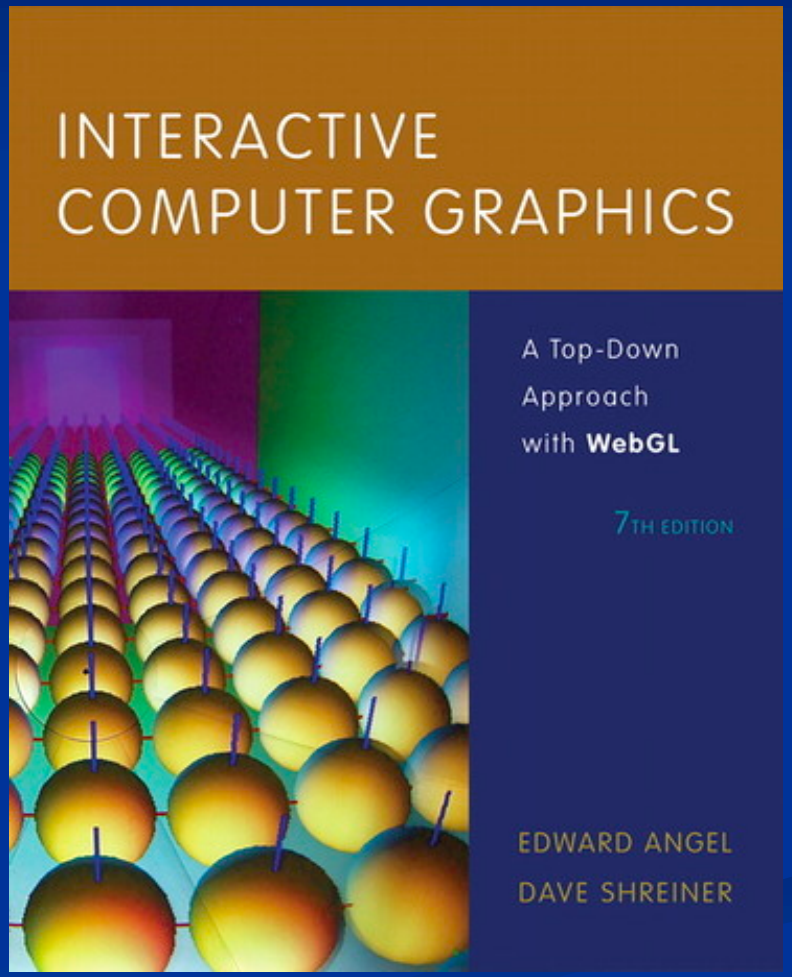
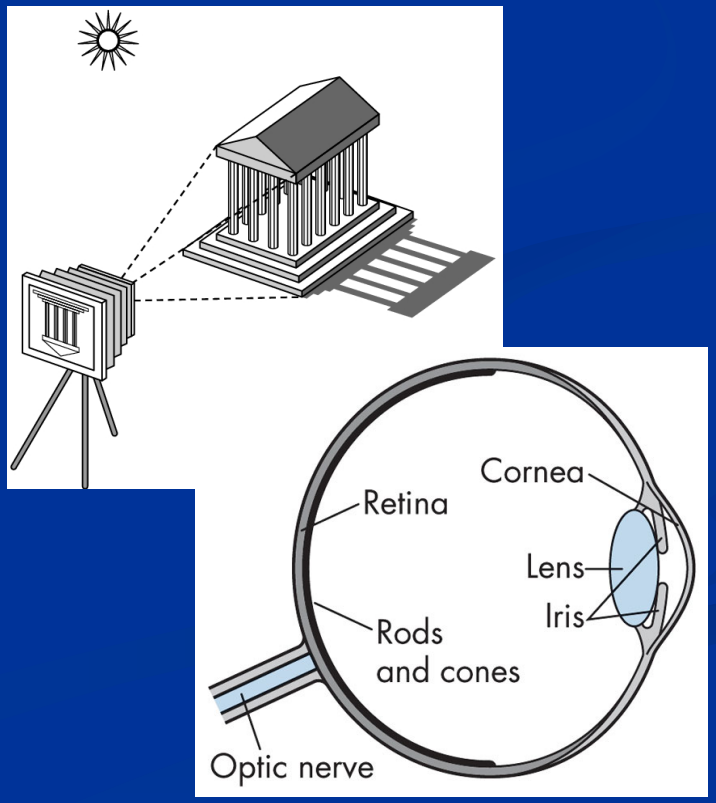
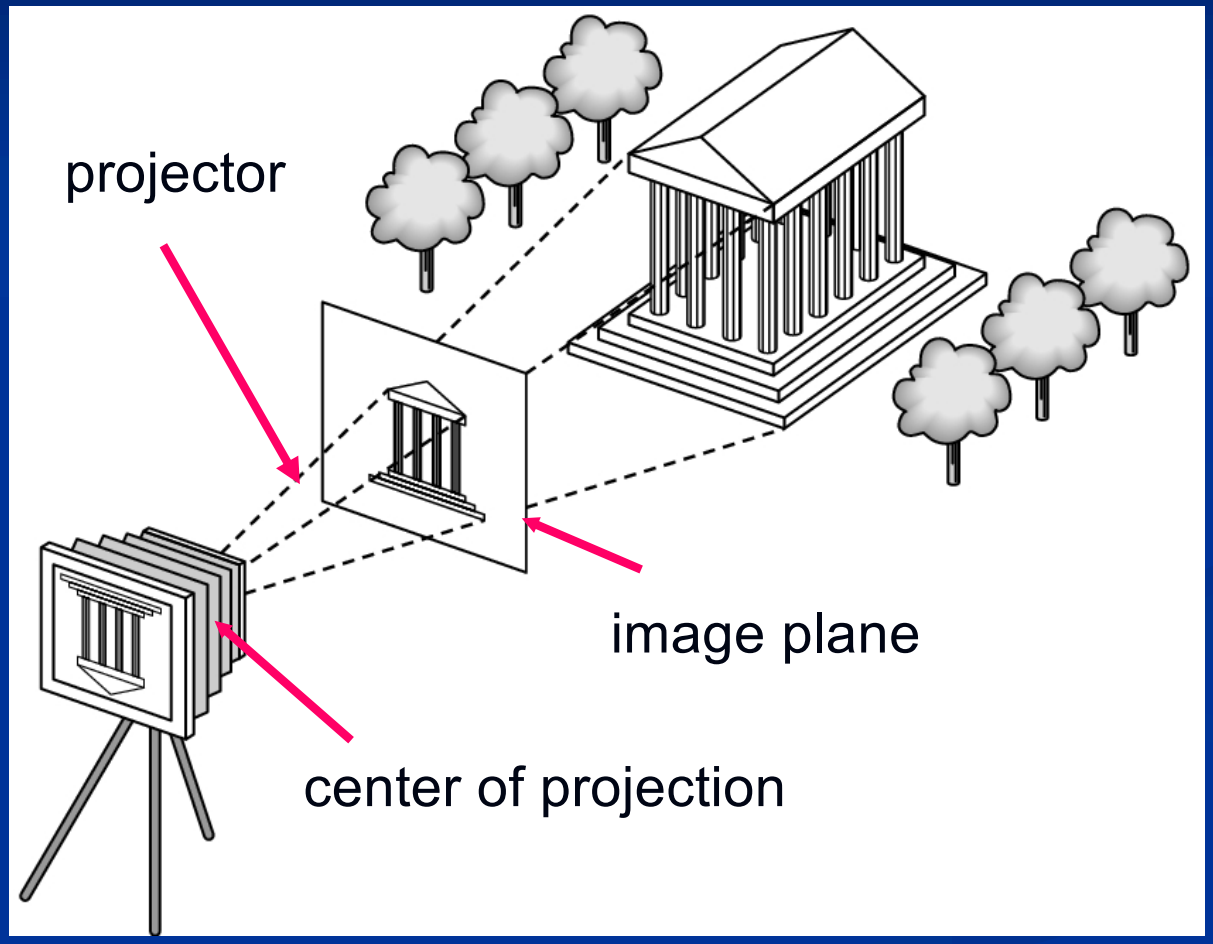


Image formation in graphics

- Form 2D images via a process analogous to how images are formed by physical imaging systems
  - Cameras
  - Microscopes
  - Human visual system



Synthetic camera model



Imaging model adopted by 3D computer graphics

Elements of image formation

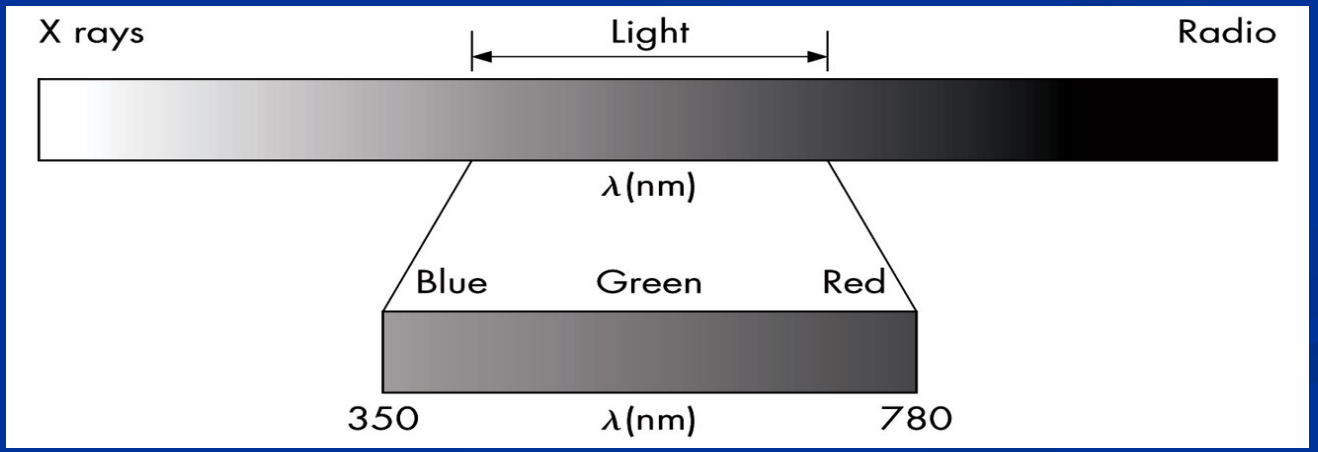
- Object:
  - Exists independent of image formation and viewer – defined in its own **object space**
  - Formed by geometric primitives, e.g., triangles
- Viewer:
  - Forms the image of the objects via a **projection**
  - Image is produced in 2D, e.g., on retina, film, etc.
  - Objects transformed from object to **image space**

Light and color

- No light $\Rightarrow$ nothing is visible
- How do we, or the camera, see?
  - **Light reflected** off objects in the scene, or
  - **Light transmitted** directly into our eyes, e.g., from light source
- What is reflected color?
  - Light and object have color
  - When colored light reaches a colored object, some colors are **reflected** and some **absorbed**

Light

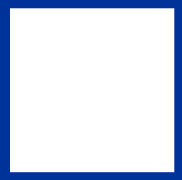
- *Light* is the (visible) part of the electromagnetic spectrum that causes a reaction in our visual systems
- Generally these are wavelengths in the range of about 350-750 nm (nanometers)
- Long wavelengths appear as red and short ones as blue



The electromagnetic spectrum

Simplistic color representation

- RGB-color system (R: red, G: green, B: blue)
- Each displayed color has 3 components: R, G, B
- 8 bits per component in 24-bit true-color display; component value: 0 – 255



(R,G,B) = (255,255,255)

(0,0,0)

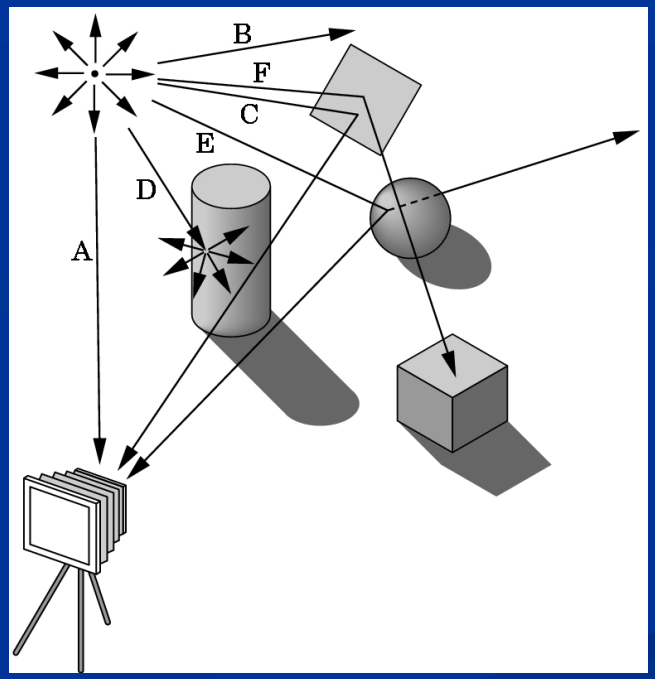
(255,0,0)

(255,255,0)

(0,255,255)

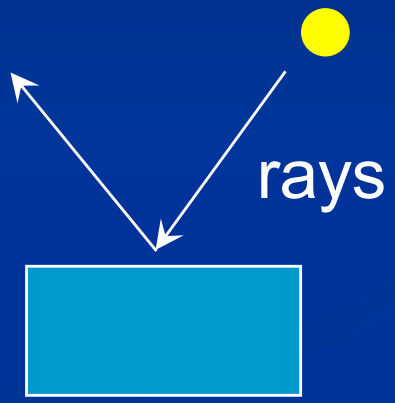
First imaging algorithm

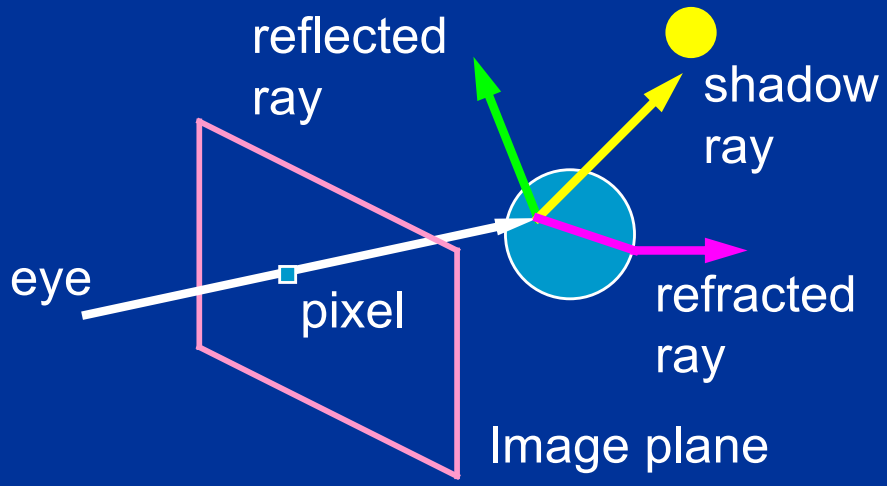
- Directly derived from the synthetic camera model
  - Follow rays of light from light sources
  - Determine which rays enter the lens of the camera through image window
  - Compute color of projection
- Why is this not a good idea?



Ray tracing

- When following light from a light source, many (reflected) rays do not intersect the image window and do not contribute to the scene

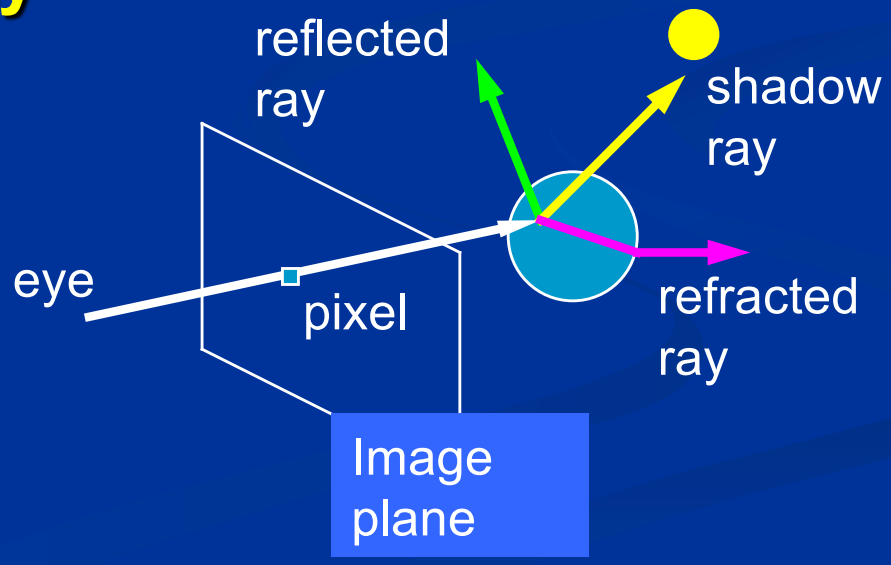




- **Reverse the process**
- Cast one ray per pixel from the eye and shoot it into the scene

Ray tracing: the basics

- A point on an object may be illuminated by
  - Light source directly – through **shadow ray**
  - Light reflected off an object – through **reflected ray**
  - Light transmitted through a transparent object – through **refracted ray**



Ray tracing: the algorithm

for each pixel on screen

determine ray from eye through pixel

if ray shoots into infinity, return a background color

if ray shoots into light source, return light color appropriately

find closest intersection of ray with an object

cast off **shadow ray** (towards light sources)

Most expensive

if shadow ray hits a light source, compute light contribution (1) according to some illumination model

cast **reflected** (2) and **refracted ray** (3), **recursively** to calculate pixel color contributions

return pixel color **as a sum** of (1)-(3) after some **absorption**

The WingsPov Civilisation Museum by *Eric Ouvrard*

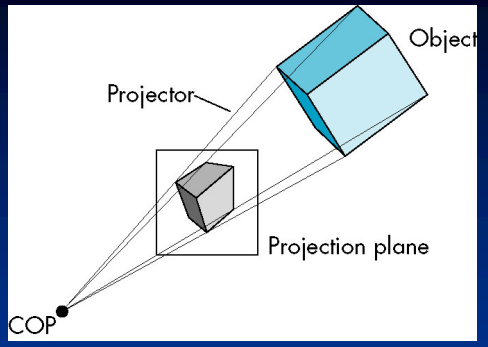
Ray tracing: pros and cons

- Pros:
  - Follows (approximately) the physics of optical flow
  - Visual realism – can model light-object interactions and **inter-surface reflections and refractions**
- Cons
  - **Expensive**: intersection tests and the levels of recursion (rays generated)
  - Inadequate for modeling **non-reflective** (dull-looking) objects – why is that?

z-buffer algorithm

- Works in the image space, like ray tracing
- **Per-polygon** operations vs. per-pixel for ray tracing
- Simple and often accelerated with hardware — **fast**
- Works regardless of the order in which the polygons are processed — no need to sort them back to front
- A **visibility algorithm** and not designed to compute colors

The algorithm




```
for each polygon in the scene
    project its vertices onto viewing (image) plane
    for each pixel inside the polygon formed on viewing plane
        determine point on polygon corresponding to this pixel
        get depth value for this pixel (distance from point to plane)
        if depth value < stored depth value for the pixel
            compute pixel color based on some illumination model
            update pixel color in frame buffer
            update depth value in depth buffer (z-buffer)
        end if
end if
```

Question: What does the depth buffer store after running the z-buffer algorithm for the scene? Does polygon order change depth? Image?