Light-Based Rendering

Lecture 28

What is rendering?

Generating an **image** from a 3D scene model

Ingredients …

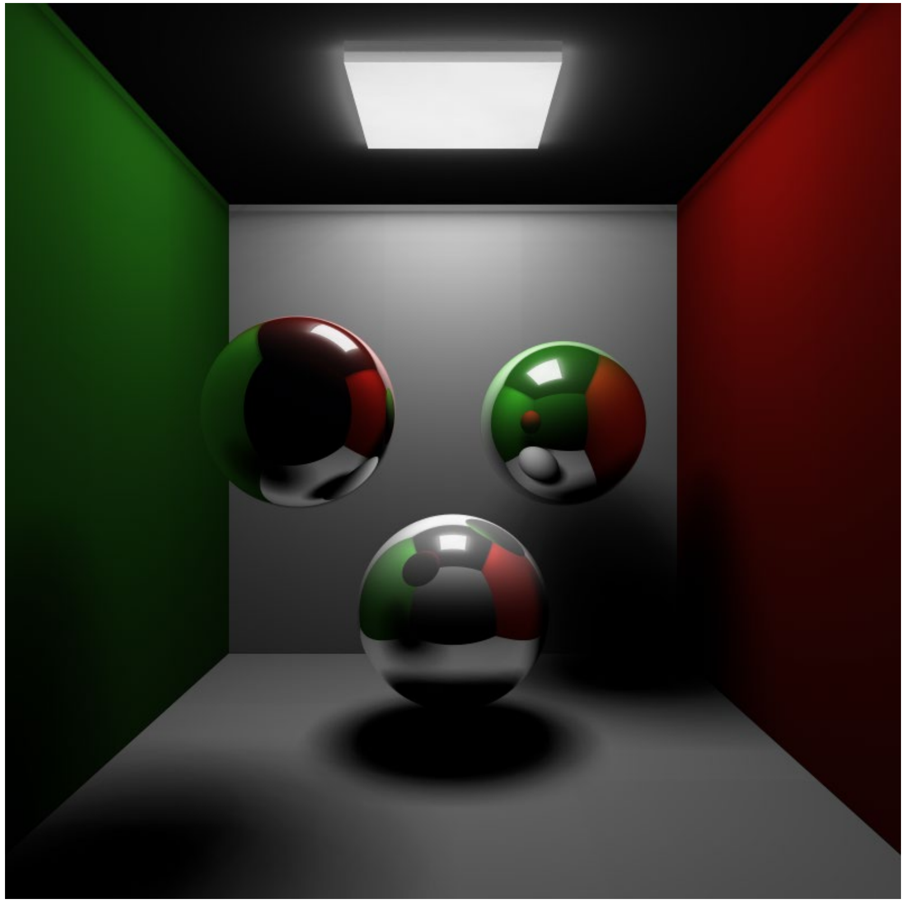
- Representation of 3D geometry
- Specification for camera & lights
- Textures, material specifications, etc

Typically refers to *high-quality* image creation

Phenomena in scenes

Complex reflections

Soft shadows



Things we like to see

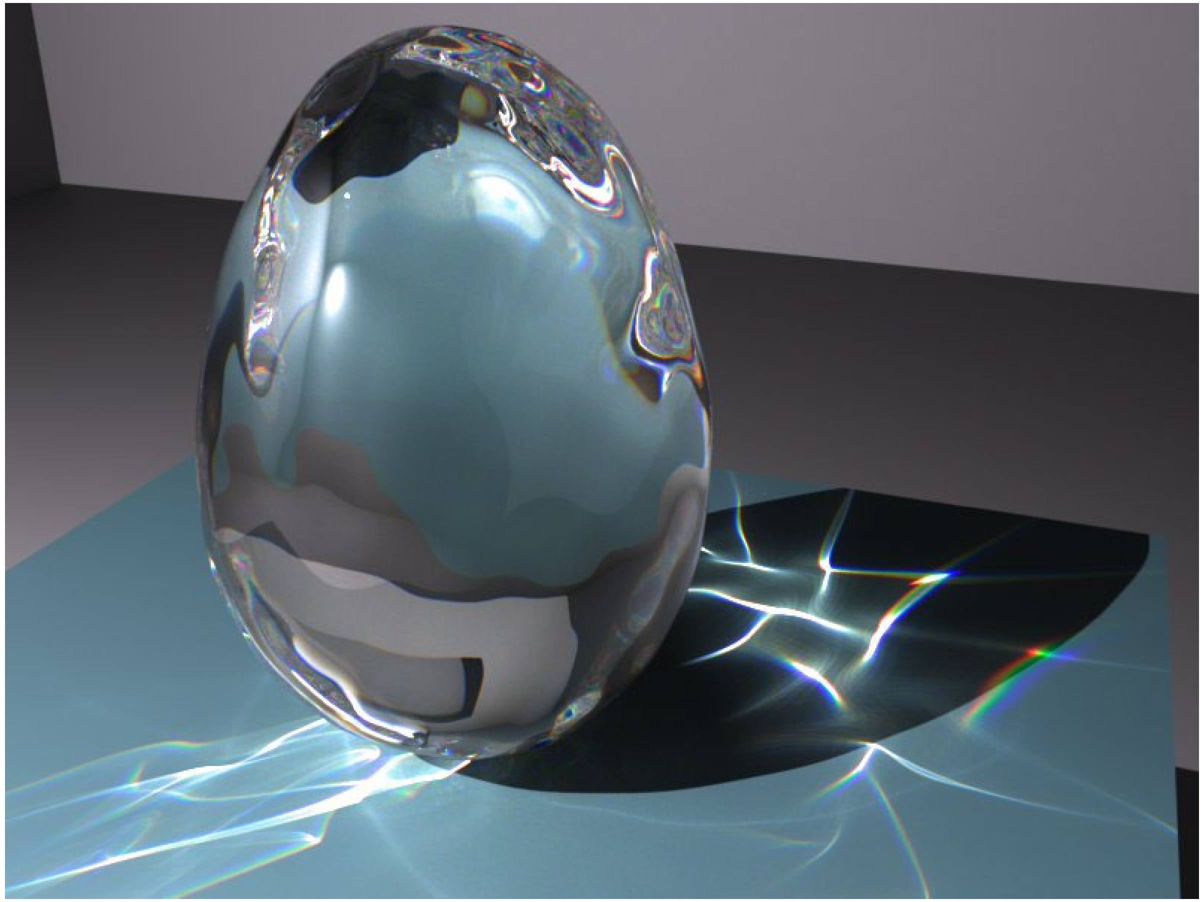
Transparency

Refraction

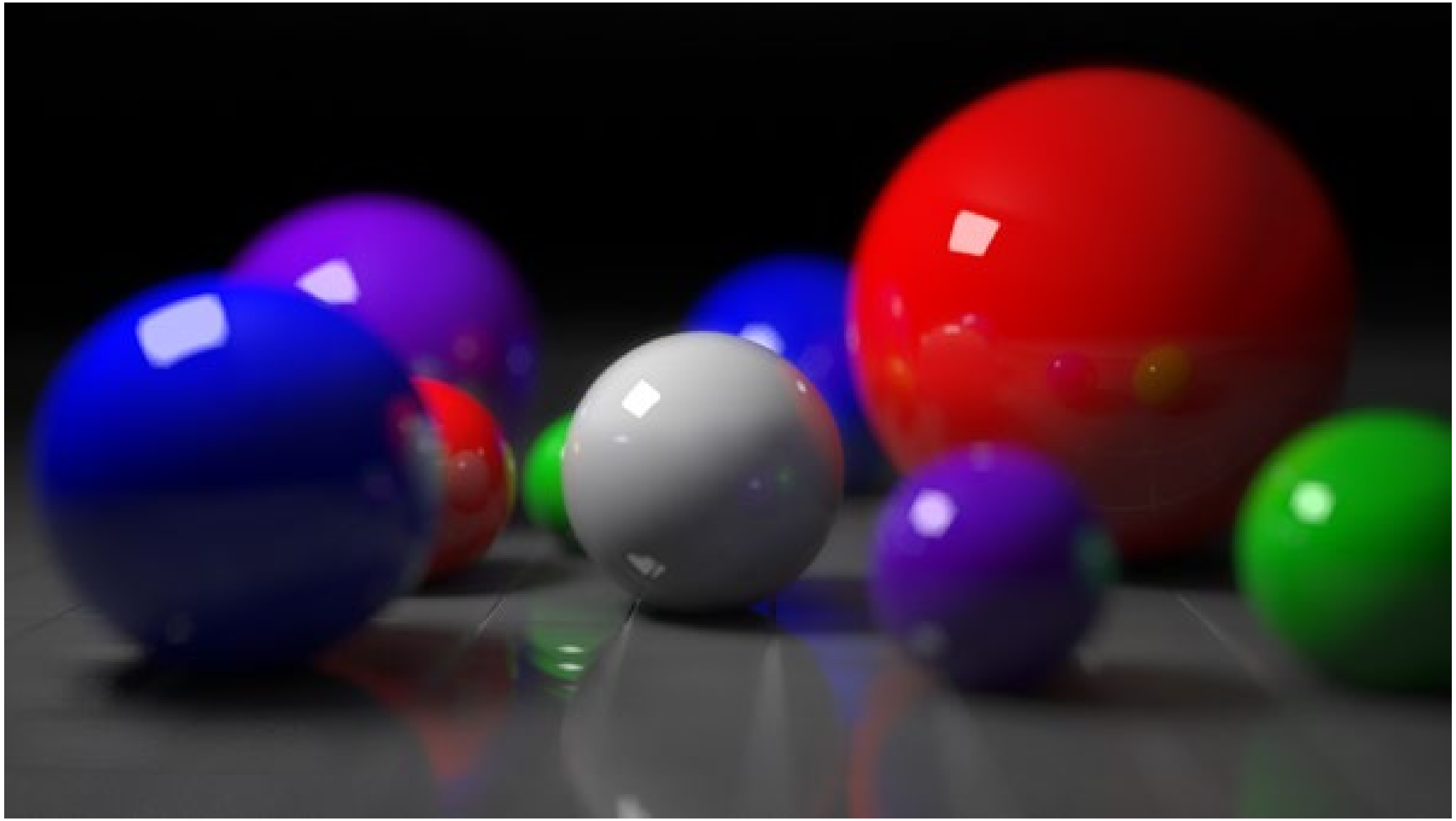
Effects we expect to see

Caustics

Dispersion



Effects of real images



Depth of field

Defocus

Effects of real lighting

Color
bleeding

Effects of real materials

Subsurface scattering

Types of rendering

Model-centric (e.g. OpenGL/WebGL)

- Centered on primitives (triangles)
  - Each object's appearance is (largely) independent of other objects

Types of rendering

Model-centric (e.g. OpenGL/WebGL)

- Advantages:
  - Fast!! (can leverage *both* data-level and pipeline parallelism)
  - Great standardization
  - Can “fake” several effects (with hacks)

Types of rendering

Model-centric (e.g. OpenGL/WebGL)

- Shortcomings:
  - Can't natively do many advanced effects
  - Several effects even difficult to fake
  - Realism is an art

Types of rendering

Light-centric

Simulate the behavior of *light* in a scene

- where light is emitted (and where to) …
- how it disperses when it hits objects …
- how it's attenuated by various media …
- how much of it reaches the camera …

Types of rendering

Light-centric

Advantages:

- Solid physics foundation (light transport)
- Potential for photorealism
- Broad gamut of demonstrated effects
- (… and *some* parallelism)

Types of rendering

Light-centric

- Shortcomings:
  - Not that fast …
    - Ray-parallelism (at best–no pipelining)
    - Requires access to the scene for every ray
  - Photorealism limited by knowledge of details
  - Less control for “stylized” appearance

Types of rendering

Light-centric

General idea:

There *is* a way to render anything

(in theory …)

… but it would be prohibitively expensive

… and we don't fully understand materials

Remedy: Make educated compromises, to obtain acceptable results in realistic render times

Kinds of Light-Based Rendering

Ray / Path based

Simulate how light works

Consider what happens to specific photons

Gather up a lot of them

Global Models

Figure out how much light ends up in places

Consider interactions

In the actual world …

Light is emitted from sources

… bounces off objects (and disperses)

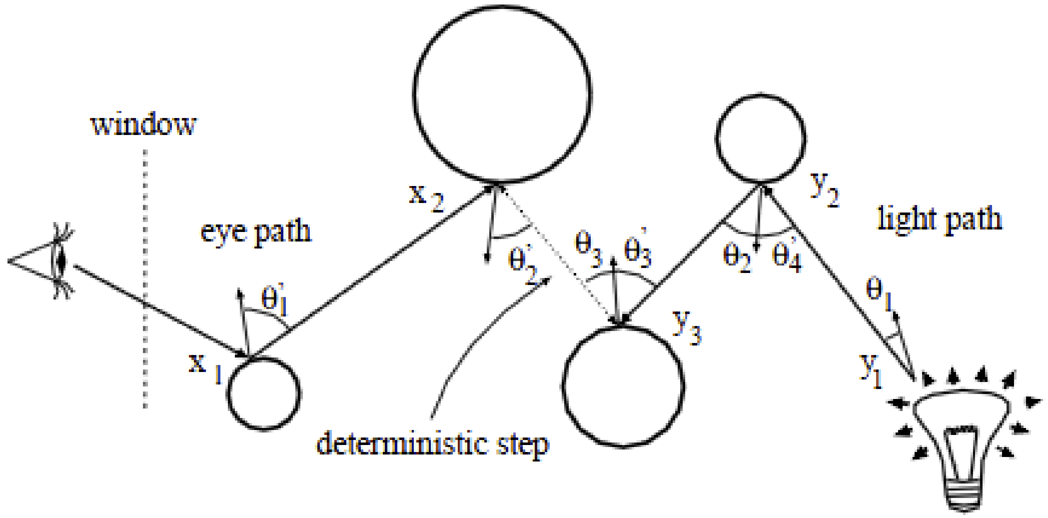
… reaches the eye or camera sensor

“Rays”?

Light *is* quantized (photons)

Photon bundles – rays – more practical

A ray's journey



What happens with each "bounce" ?

Where does a ray go (or come from)?

Where does it bounce?

Distribution of probability

Where did it come from?

Distribution of probability

Bi-Directional Reflectance Distribution (BRDF)

A function…

what direction “in”

what direction “out”

what color

Says “how much reflection in that direction”

Or

How likely would this light path be?

Ray Tracer (v0 – Nature's algorithm)

Make a photon (at a light source)
See where it goes

Each step is random

Some make it to the "camera"

Forward or backward?

Forward ray tracing is physically intuitive …

… but many rays wasted

… need to spawn many rays to ensure

sometimes used (light gathering)

Backward ray tracing

Track rays **from** camera and out into the world (hopefully reaching a light source!)

Ray Tracer (v1 – stochastic)

Start at the eye

Pick a direction (through a pixel)

Figure out where this ray might have come from

Still need lots of rays...

Send out lots of rays per pixel

Each one might get somewhere different
Not all of them get to light sources (or bring light)

Lots of time on “unimportant” rays

Ray Tracer (v2 – simple recursive)

At each bounce…

Where could the ray have come from?

Check the most likely directions

- Mirror reflection (specular)
- Diffuse reflection (towards light)
- Refraction (through surface)
- Others

Types of Rays (really a continuum)

Specular

Directions of high reflectivity

Whatever is there, you'll see it because its in the right direction

Diffuse

Directions direct to a strong source

So many photons you're likely to get some of them

Diffuse (shadow) Rays

In the direction of the light

Check: does it actually get to the light

Color is easy (from the light)

Specular (recursive rays)

In the direction(s) of most reflection/refraction

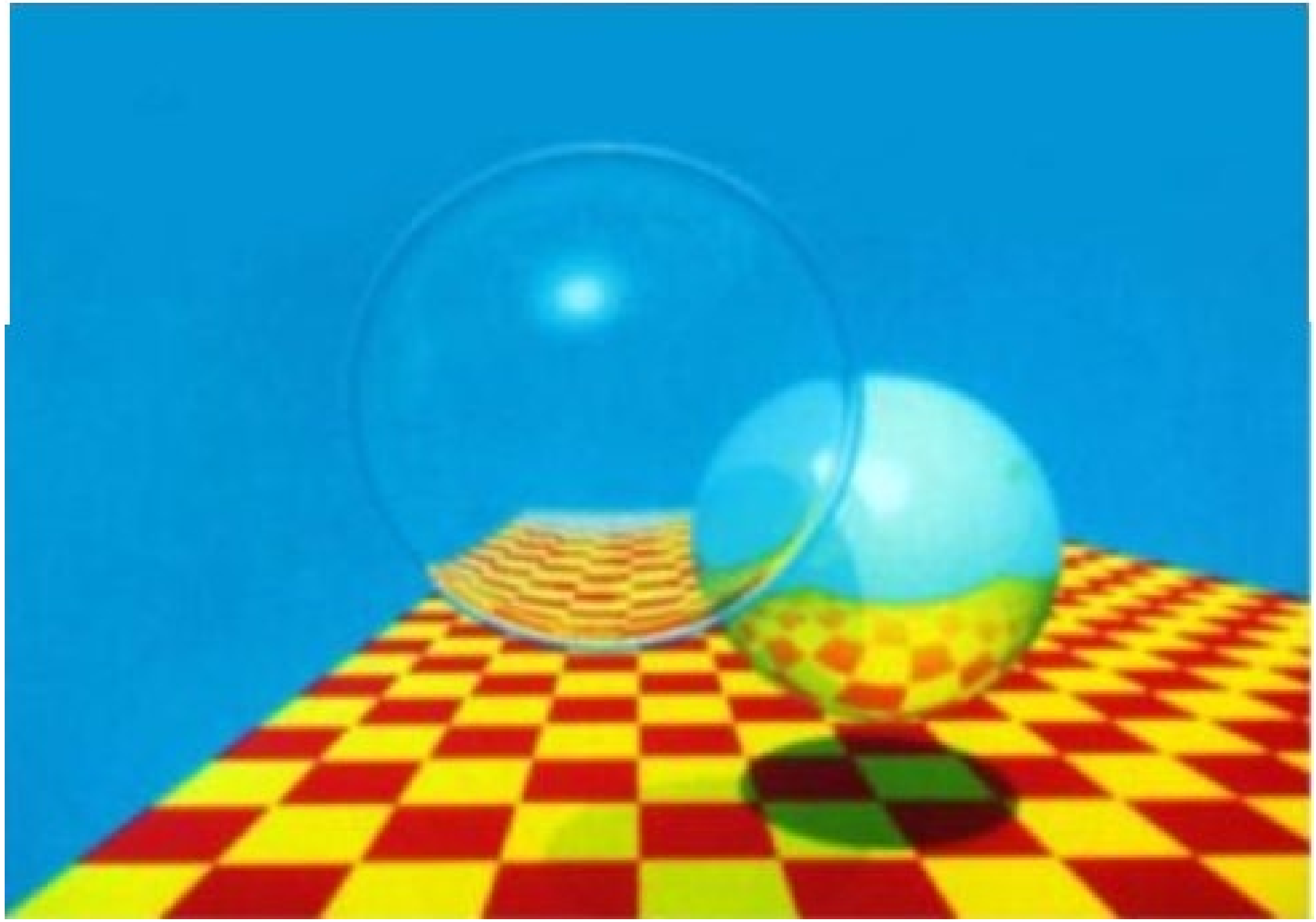
Know the direction to look

What color would those photons be?

Recurse! (same as original problem) (typically bounded)

Recursive ray tracing

Shadows, Reflections, Refractions are Easy!



The basic operation: Ray Casting

Where does a ray hit?

Ray Object Intersection Tests

Need to check all objects

In practice: use fancy data structures to speed things up

Is ray tracing slow? (compared to rasterization)

Rasterization: need to draw each triangle (even if few pixels)

Ray Tracing: need rays per pixel

(intersection might be fast –with good data structures)

But…

Rays in parallel, but need access to scene

Image quality depends on number of rays

The real problem: sampling

Really – more than one photon (all directions possible)

One ray per pixel (aliasing)
One ray per diffuse source (no spill)
One ray per reflection (perfect mirror)
One ray per light (hard shadows / no area lights
(and many more)

Ray Tracer (v3 – Distribution Ray Tracing)

Try lots of rays (a distribution of them)

Sample the different possibilities

Weight the different samples depending on how likely

Distribution vs. Recursion

Example: Anti-Aliasing

One ray per pixel?

Example: Area Light Source

One ray per light source?

Example: General Diffuse (Spill)

Only towards the light source?

(in practice, trying all directions is impractical)

Example: Imperfect Mirror

One direction of reflection?

Example Depth of field

One focal point?

Example: Motion Blur

One Point in Time?

Summary: Light-Based Rendering
Is this better?

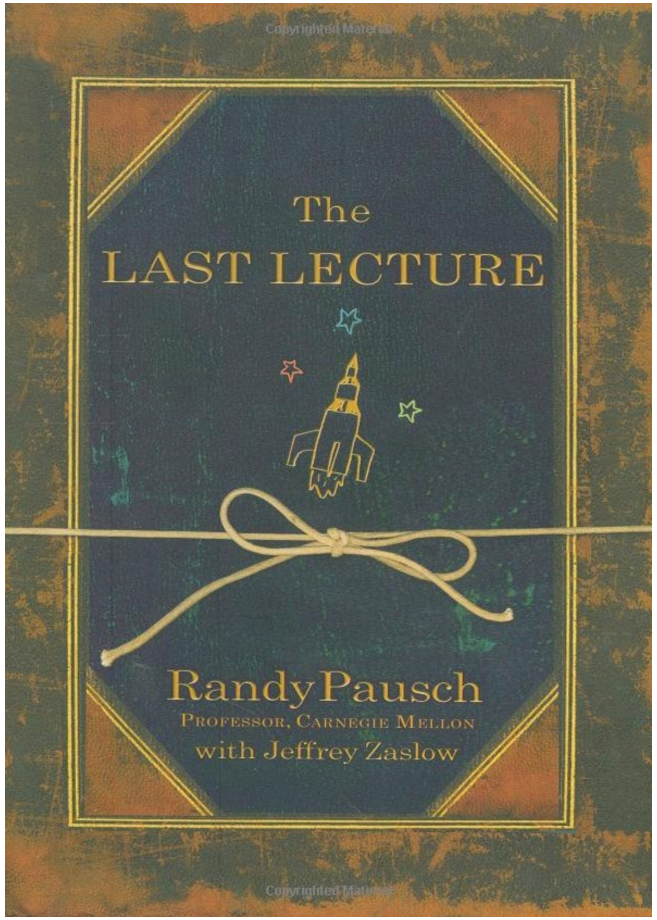
Hard to make fast (speed/quality tradeoff)

Many effects can be achieved (with lots of rays)

Well-principled (use real physics models)

The End…

Lecture 28



"Experience is what you get when you didn't get what you wanted.

And experience is often the most valuable thing you have to offer."

What did we do this semester?

Web programming, javascript, web basics, …

Drawing with APIs, primitives

Coordinate Systems, Hierarchies

Curves and Shapes

3D Drawing (viewing, primitives, meshes)

3D Appearance (lighting, texturing)

How drawing works (pipeline, shaders, efficiency issues)

Brief views of other topics (rendering, surfaces)

What didn’t we do

Image processing / photography
Mathematics of viewing and 3D transformations
Better curve and surface representations
Lighting math

Animation (motion creation)
Modeling (where do shapes come from?)
Rendering
Applications (what pictures should we make?)

Now what?

You should have the basics…

You can make things (so try!)

You can go on to learn about advanced topics