

Computer Graphics

Prof. Feng Liu

Fall 2018

<http://www.cs.pdx.edu/~fliu/courses/cs447/>

11/27/2018

Last time

Splines

Today

Raytracing

Final Exam

■ December 4 (Tuesday) 17:30-19:00

■ To-know list available

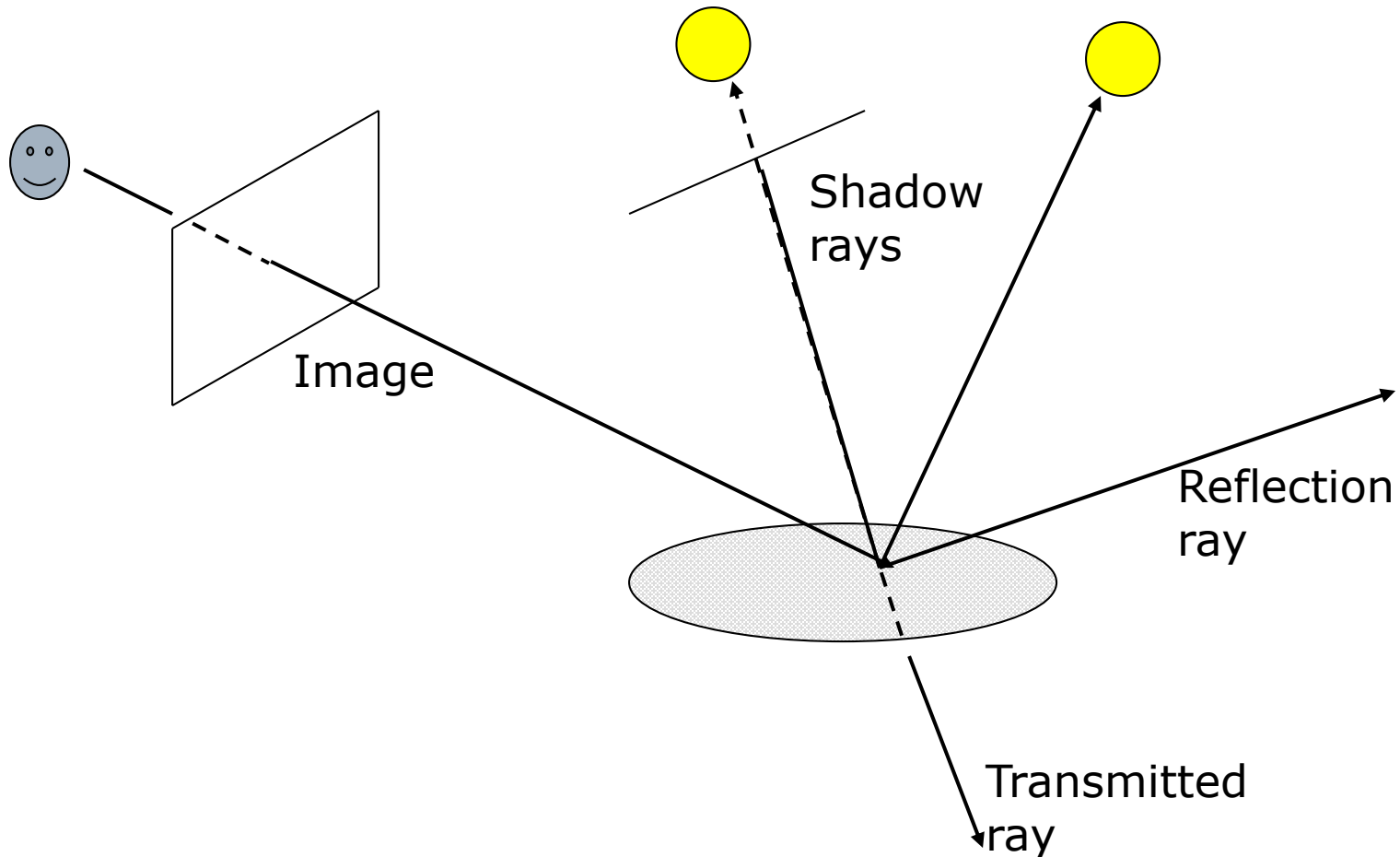
OpenGL Limitations

- Stream processing of geometry
 - No random access to geometric information
 - Can't do any computation that requires all the geometry at once
- Rasterization is limited
 - We saw many ways to represent objects - not all can be rasterized
 - Cannot provide rasterizers for every form of geometry
- Everything get rasterized and drawn
 - Figuring out what you can see before rasterizing is possible but hard
- Computation loops over geometry, then pixels
 - for all objects { for all pixels in object ... }

Raytracing

- Cast rays out from the eye, through each pixel, and determine what the rays hit
 - Builds the image pixel by pixel, one at a time
- Cast additional rays from the hit point to determine the pixel color
- Rays test visibility - what do I see from this point in this direction?
 - Ray casting is widely used in graphics to test visibility

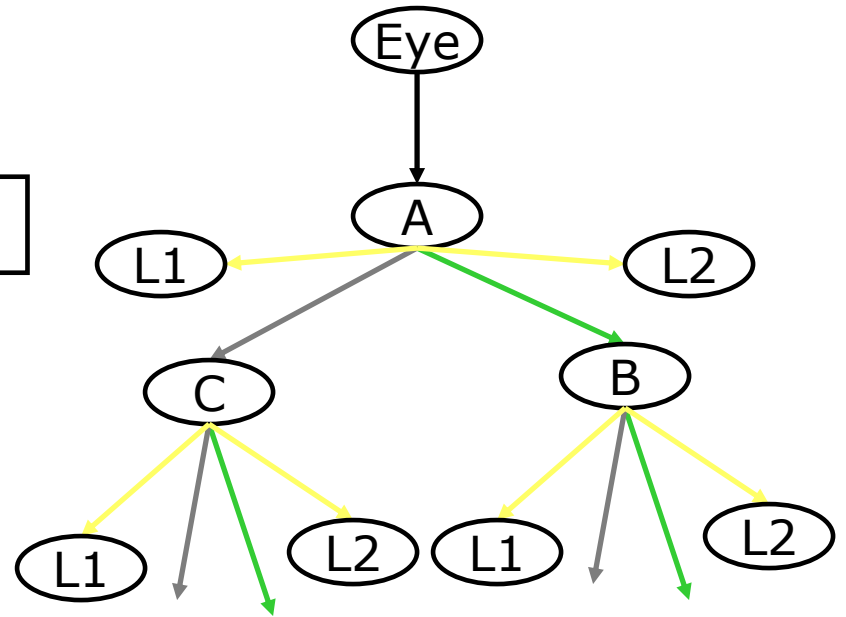
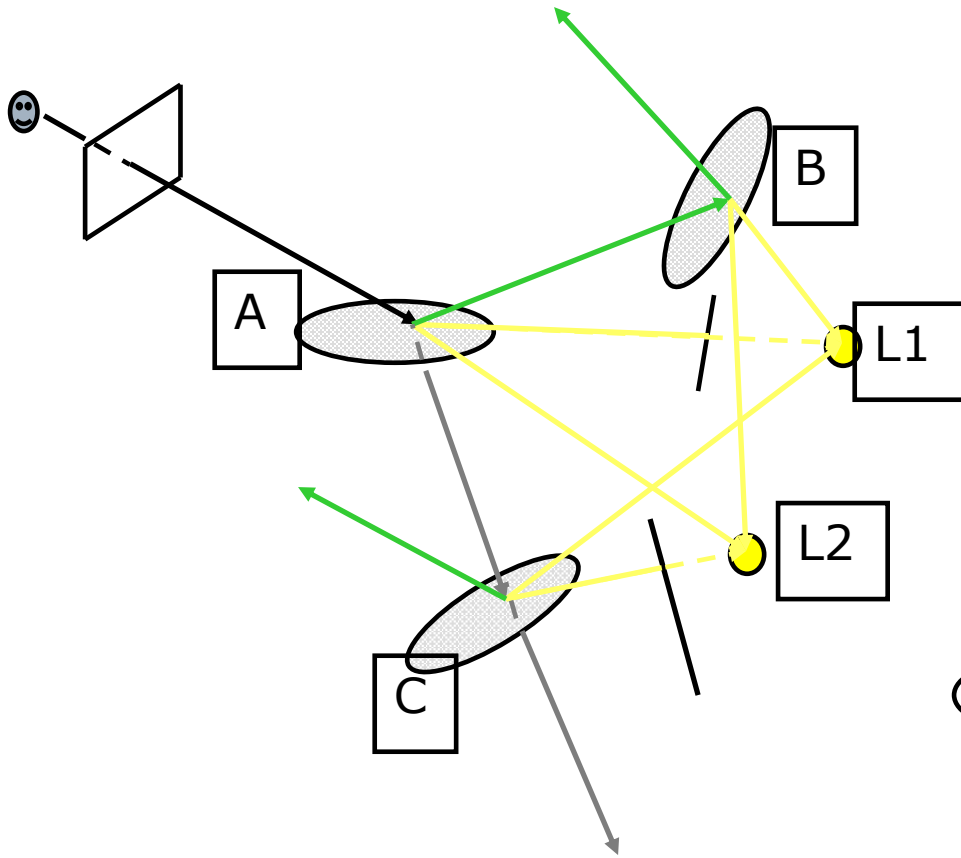
Raytracing



Recursive Ray Tracing

- When a reflected or refracted ray hits a surface, repeat the whole process from that point
 - Send out more shadow rays
 - Send out new reflected ray (if required)
 - Send out a new refracted ray (if required)
 - Generally, reduce the weight of each additional ray when computing the contributions to surface color
 - Stop when the contribution from a ray is too small to notice
- The result is a *ray tree*

Ray Tree

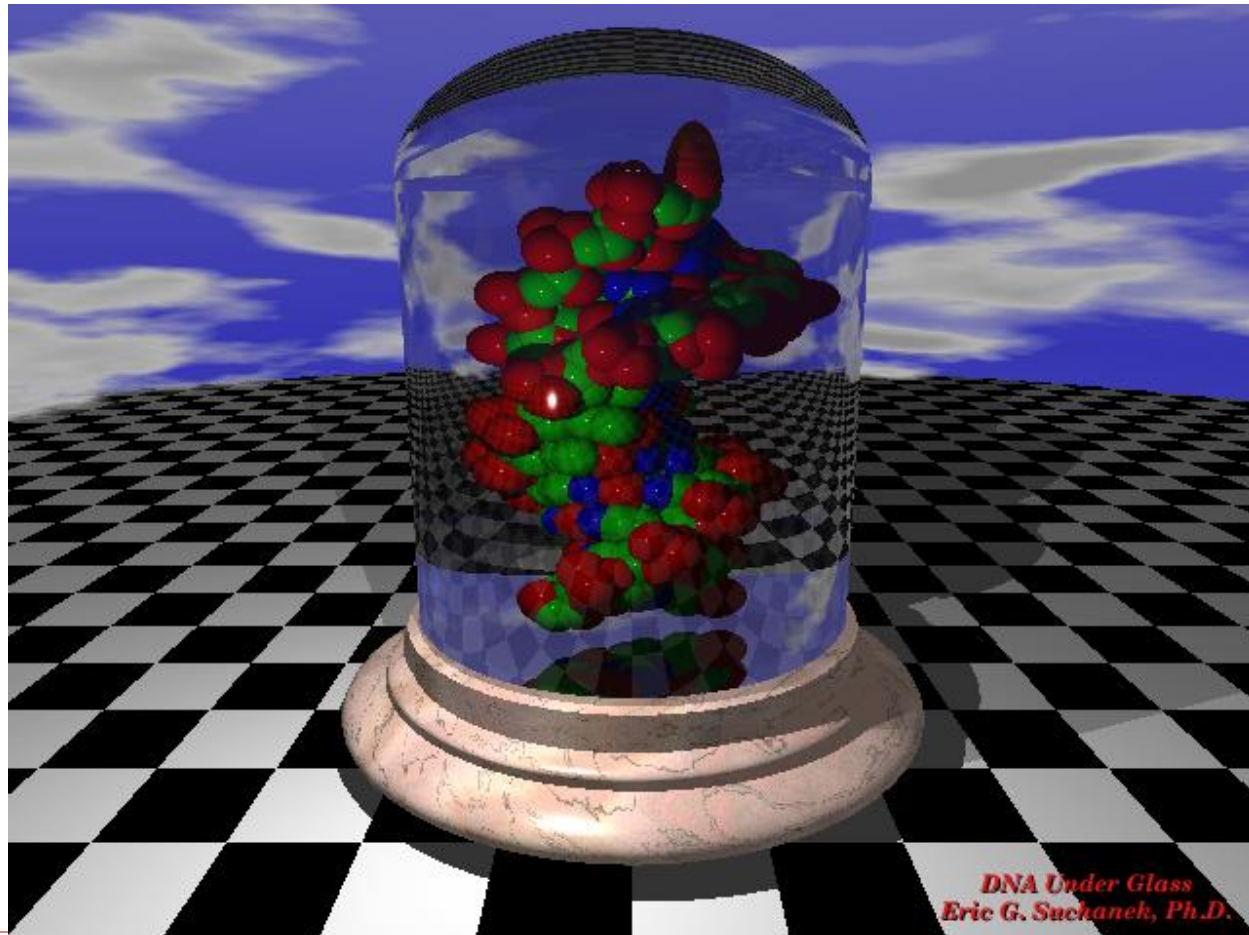




PCKTWTCH by
Kevin Odhner,
POV-Ray



Kettle,
Mike Miller,
POV-Ray



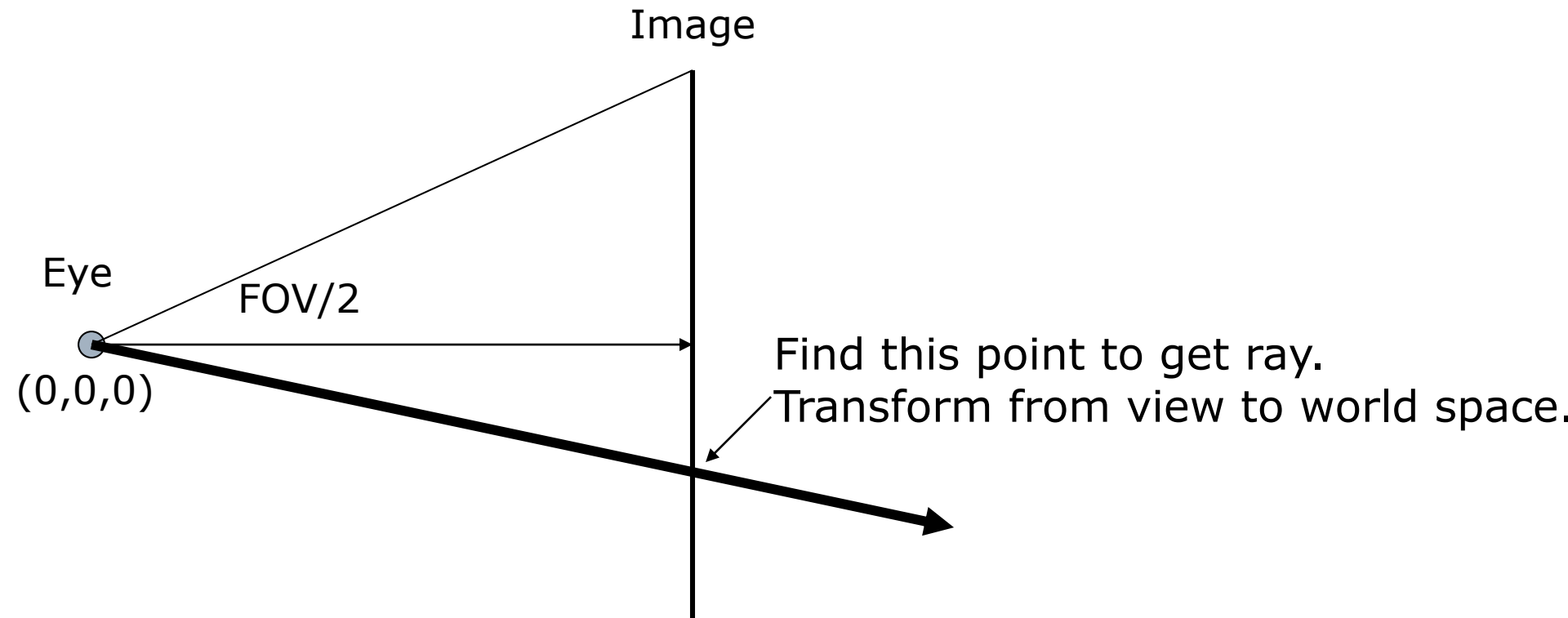
Raytracing Implementation

- Raytracing breaks down into two tasks:
 - Constructing the rays to cast
 - Intersecting rays with geometry
- The former problem is simple vector arithmetic
- The intersection problem arises in many areas of computer graphics
 - Collision detection
 - Other rendering algorithms
- Intersection is essentially root finding (as we will see)
 - Any root finding technique can be applied

Constructing Rays

- Define rays by an initial point and a direction
- Eye rays: Rays from the eye through a pixel
- Shadow rays: Rays from a point on a surface to a light
 - If the ray hits something before it gets to the light, then the point is in shadow
- Reflection rays: Rays from a point on a surface in the reflection direction
 - Only for reflective surfaces
- Transmitted rays: Rays from a point on a transparent surface through the surface
 - Use Snell's law to get refraction direction

Eye Rays



Ray-Object Intersections

- Aim: Find the parameter value, t_i , at which the ray first meets object i
- Transform the ray into the object's local coordinate system
 - Makes ray-object intersections generic: ray-sphere, ray-plane, ...
- Write the surface of the object implicitly: $f(\mathbf{x})=0$
 - Unit sphere at the origin is $\mathbf{x}\cdot\mathbf{x}-1=0$
 - Plane with normal \mathbf{n} passing through origin is: $\mathbf{n}\cdot\mathbf{x}=0$
- Put the ray equation in for \mathbf{x}
 - Result is an equation of the form $f(t)=0$ where we want t
 - Now it's just root finding

Ray-Sphere Intersection

$$\text{Ray : } \mathbf{x}(t) = \mathbf{x}_0 + t\mathbf{d}$$

$$\text{Sphere : } \mathbf{x} \bullet \mathbf{x} - 1 = 0$$

$$\text{Substitute: } (\mathbf{x}_0 + t\mathbf{d}) \bullet (\mathbf{x}_0 + t\mathbf{d}) - 1 = 0$$

$$: (\mathbf{d} \bullet \mathbf{d})t^2 + 2(\mathbf{x}_0 \bullet \mathbf{d})t + (\mathbf{x}_0 \bullet \mathbf{x}_0 - 1) = 0$$

- Quadratic in t
 - 2 solutions: Ray passes through sphere - take minimum value that is > 0
 - 1 solution: Ray is tangent - use it if > 0
 - 0 solutions: Ray does not hit sphere

Ray-Plane Intersections

$$\text{Ray : } \mathbf{x}(t) = \mathbf{x}_0 + t\mathbf{d}$$

$$\text{Plane : } \mathbf{n} \bullet \mathbf{x} = 0$$

$$\text{Substitute: } \mathbf{n} \bullet (\mathbf{x}_0 + t\mathbf{d}) = 0$$

$$: (\mathbf{n} \bullet \mathbf{d})t + \mathbf{n} \bullet \mathbf{x}_0 = 0$$

$$: t = \frac{-\mathbf{n} \bullet \mathbf{x}_0}{\mathbf{n} \bullet \mathbf{d}}$$

- To do polygons, intersect with plane then do point-in-polygon test...
- Faster tests for triangles, but this is the start point

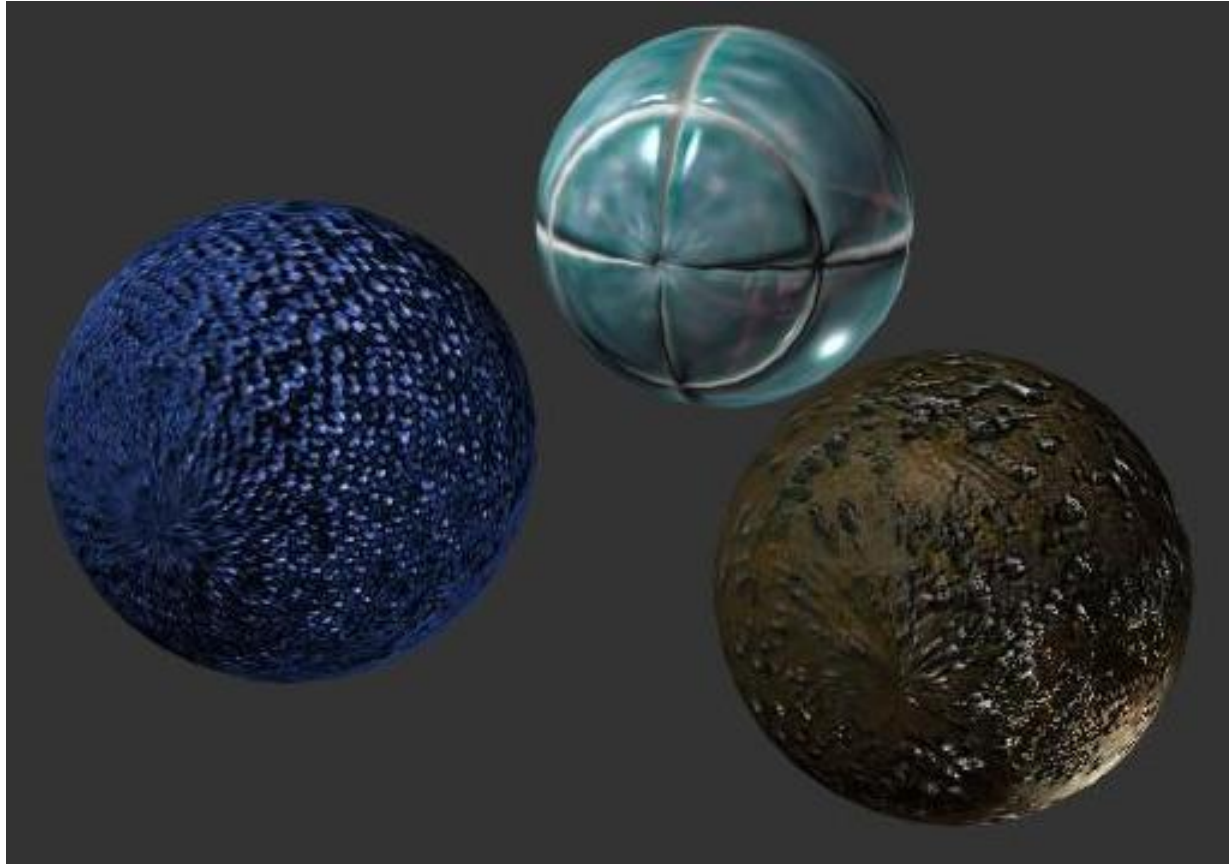
Details

- Must find *first* intersection of ray from the eye
 - Find all candidate intersections, sort them and take soonest
 - Techniques for avoiding testing all objects
 - Bounding boxes that are cheap to test
 - Octrees for organizing objects in space
 - Take care to eliminate intersections behind the eye
 - Same rules apply for reflection and transmission rays
- Shadow ray just has to find *any* intersection shadowing the light source
 - Speedup: Keep a cache of shadowing objects - test those first

Mapping Techniques

- All raytracing calculations are done for every pixel
- Raytracing provides a wealth of information about the visible surface point:
 - Position, normal, texture coordinates, illuminants, color...
- Raytracing also has great flexibility
 - Every point is computed independently, so effects can easily be applied on a per-pixel basis
 - Reflection and transmission and shadow rays can be manipulated for various effects
 - Even the intersection point can be modified

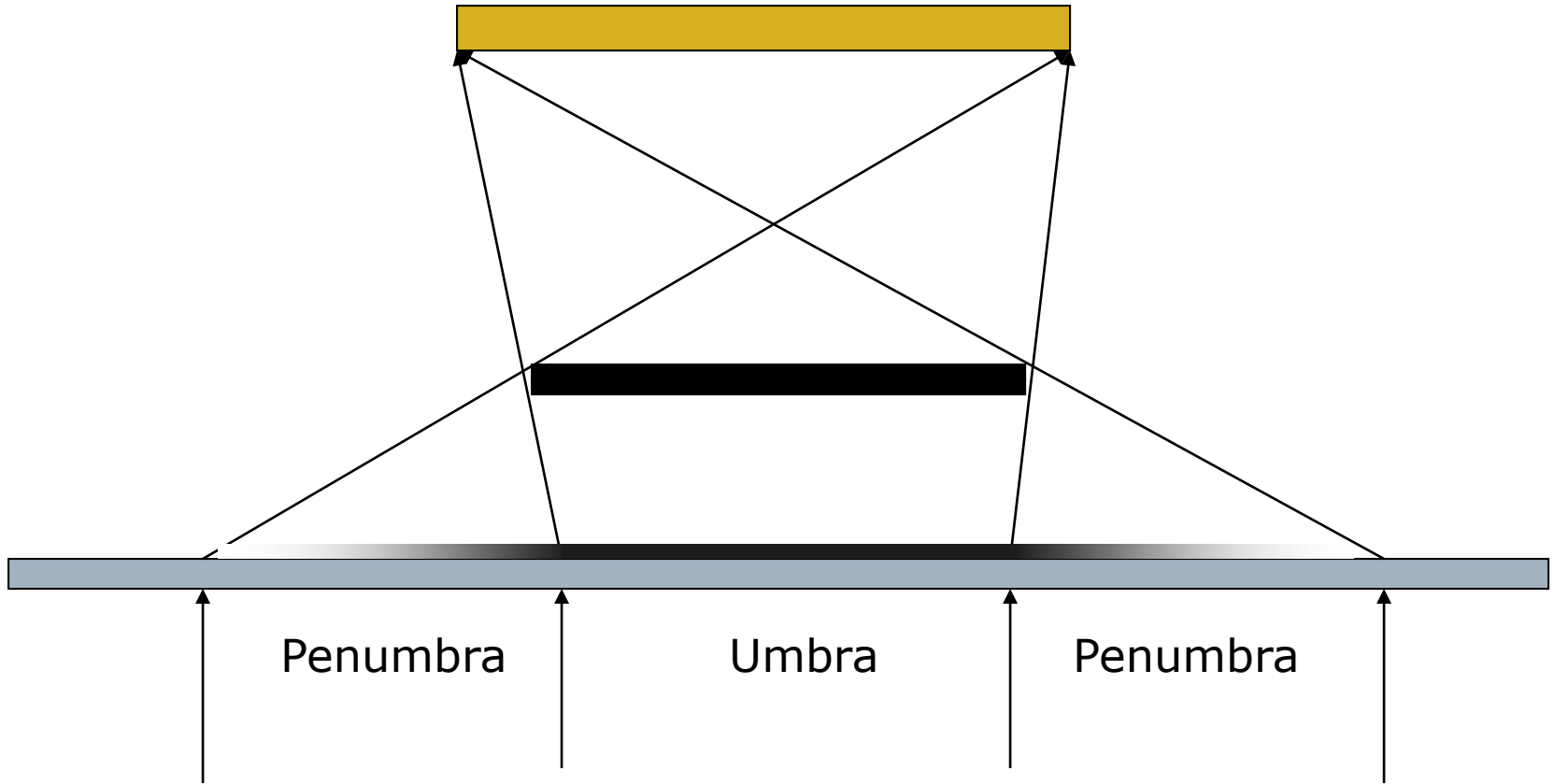
Bump Mapping Examples



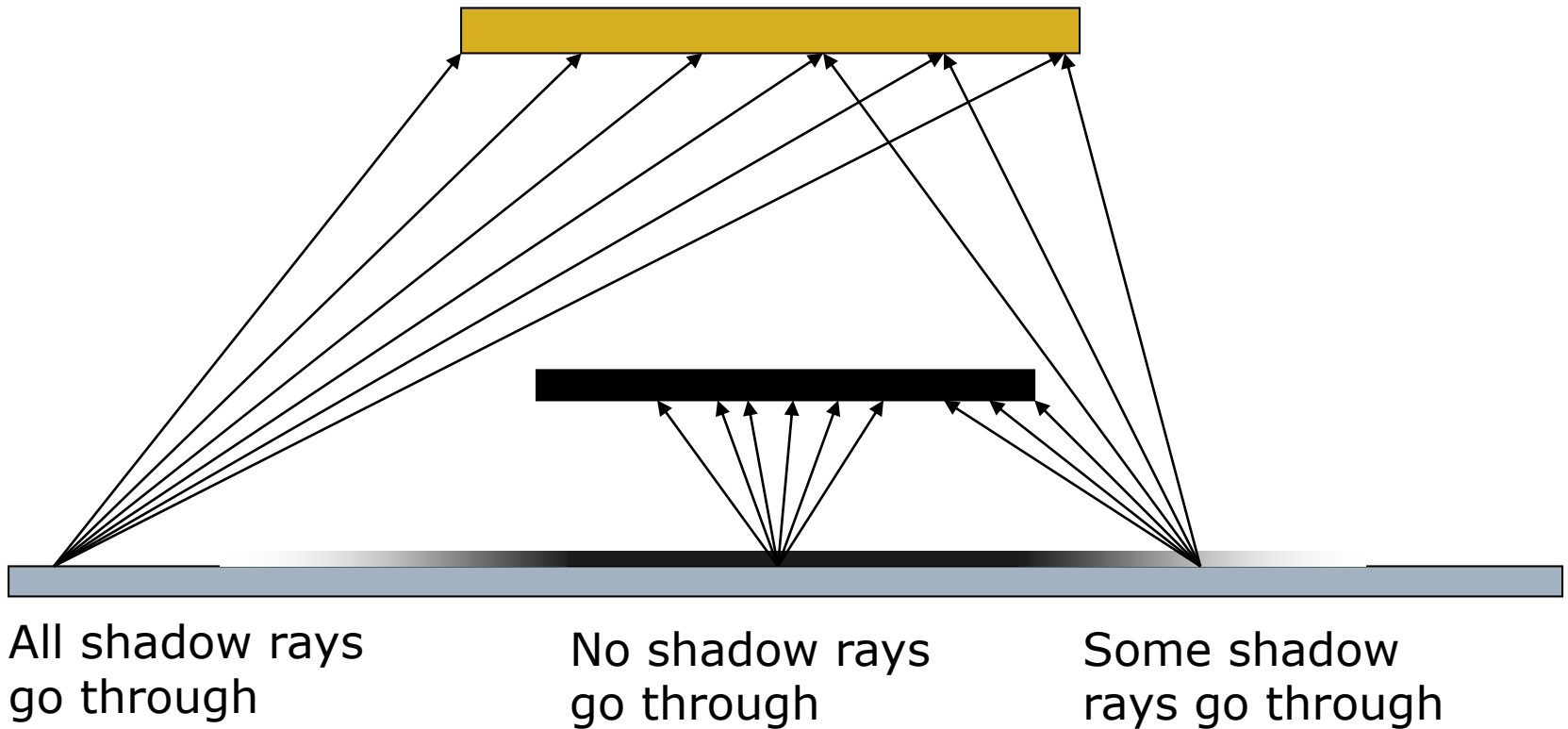
Soft Shadows

- Light sources that extend over an area (*area light sources*) should cast soft-edged shadows
 - Some points see all the light - fully illuminated
 - Some points see none of the light source - the *umbra*
 - Some points see part of the light source - the *penumbra*
- To ray-trace area light sources, cast multiple shadow rays
 - Each one to a different point on the light source
 - Weigh illumination by the number that get through

Soft Shadows



Soft Shadows



Ray-Tracing and Sampling

- Basic ray-tracing casts one ray through each pixel, sends one ray for each reflection, one ray for each point light, etc
- This represents a single sample for each point, and for an animation, a single sample for each frame
- Many important effects require more samples:
 - Motion blur: A photograph of a moving object smears the object across the film (longer exposure, more motion blur)
 - Depth of Field: Objects not located at the focal distance appear blurred when viewed through a real lens system
 - Rough reflections: Reflections in a rough surface appear blurred

Distribution Raytracing

- Distribution raytracing casts more than one ray for each sample
 - Originally called *distributed raytracing*, but the name's confusing
- How would you sample to get motion blur?
- How would you sample to get rough reflections?
- How would you sample to get depth of field?

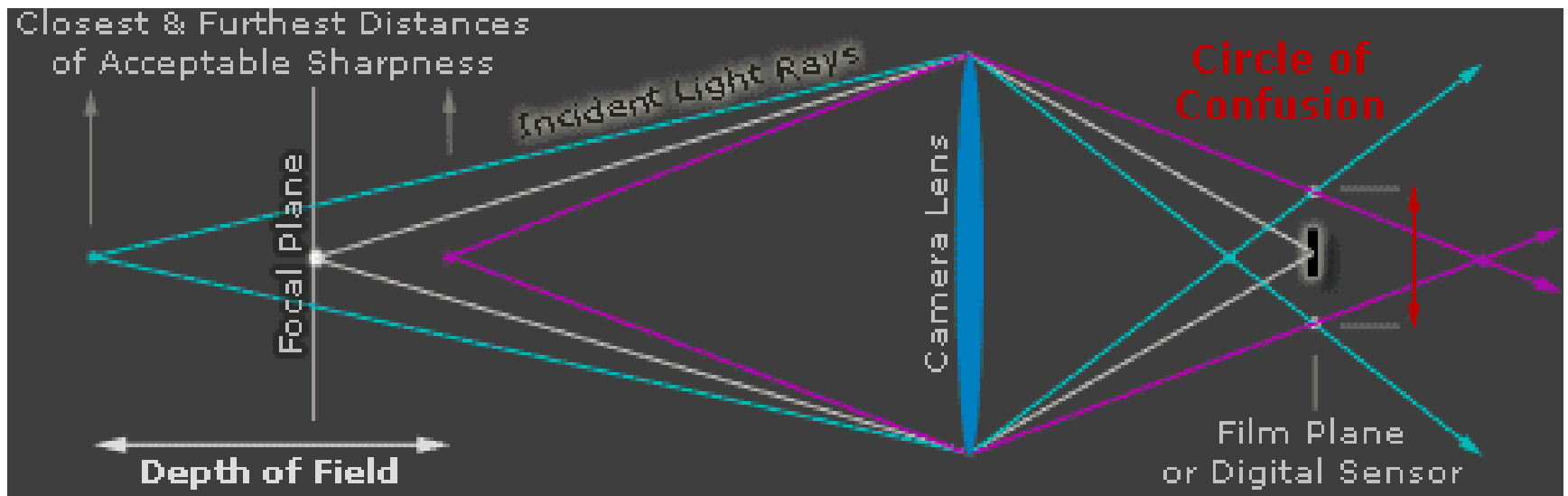
Distribution Raytracing

- Multiple rays for each pixel, distributed in time, gives you *motion blur*
 - Object positions have to vary continuously over time
- Casting multiple reflection rays at a reflective surface and averaging the results gives you rough, blurry reflections
- Simulating multiple paths through the camera lens system gives you *depth of field*

Motion Blur



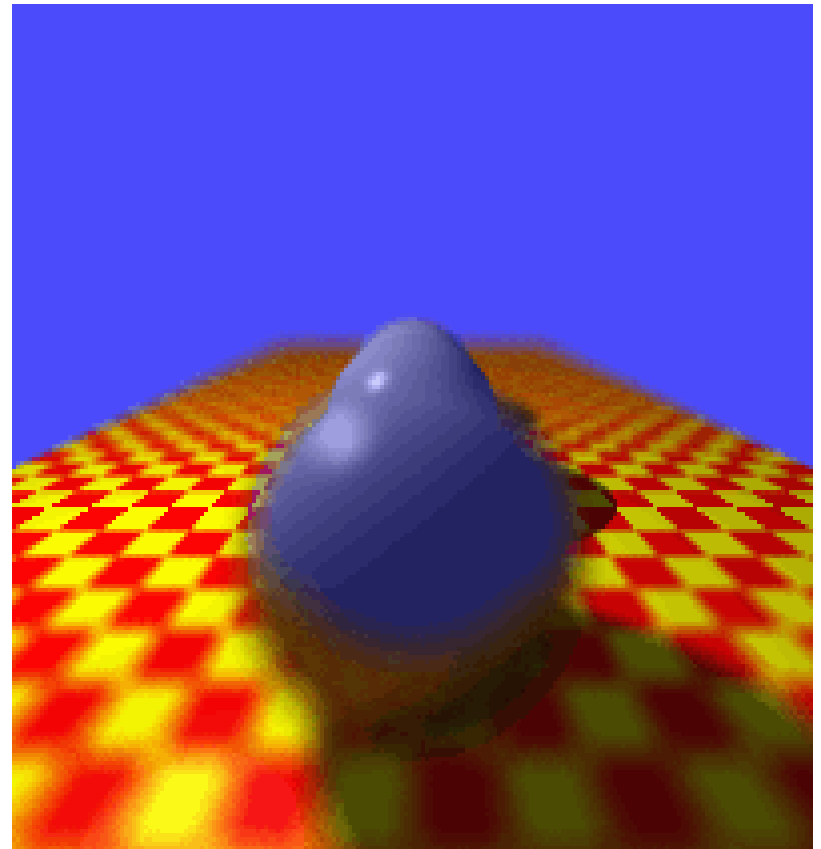
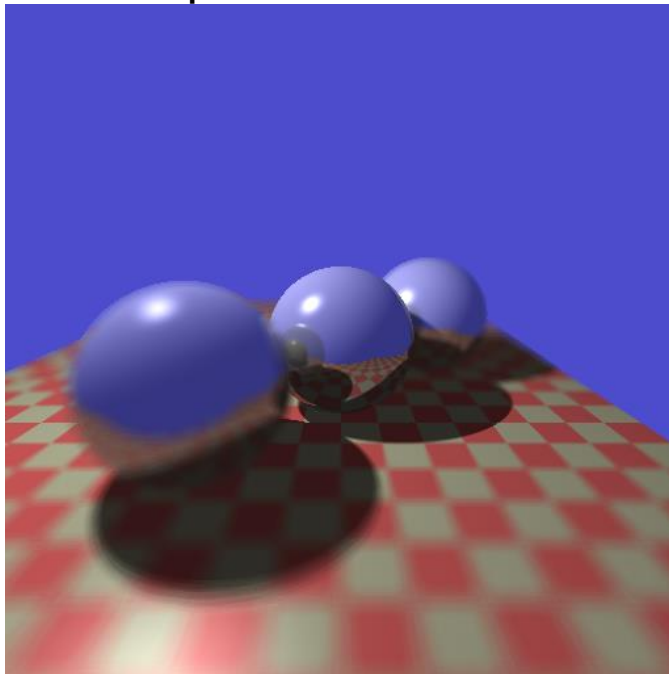
Depth of Field



<http://www.cambridgeincolour.com/tutorials/depth-of-field.htm>

Distribution Raytracing

Depth of Field



From Alan Watt, "3D Computer Graphics"

Next Time

Animation