Ray Tracing: Implementation

CSCI 4830/7000
Advanced Computer Graphics
Spring 2010

How does it work?

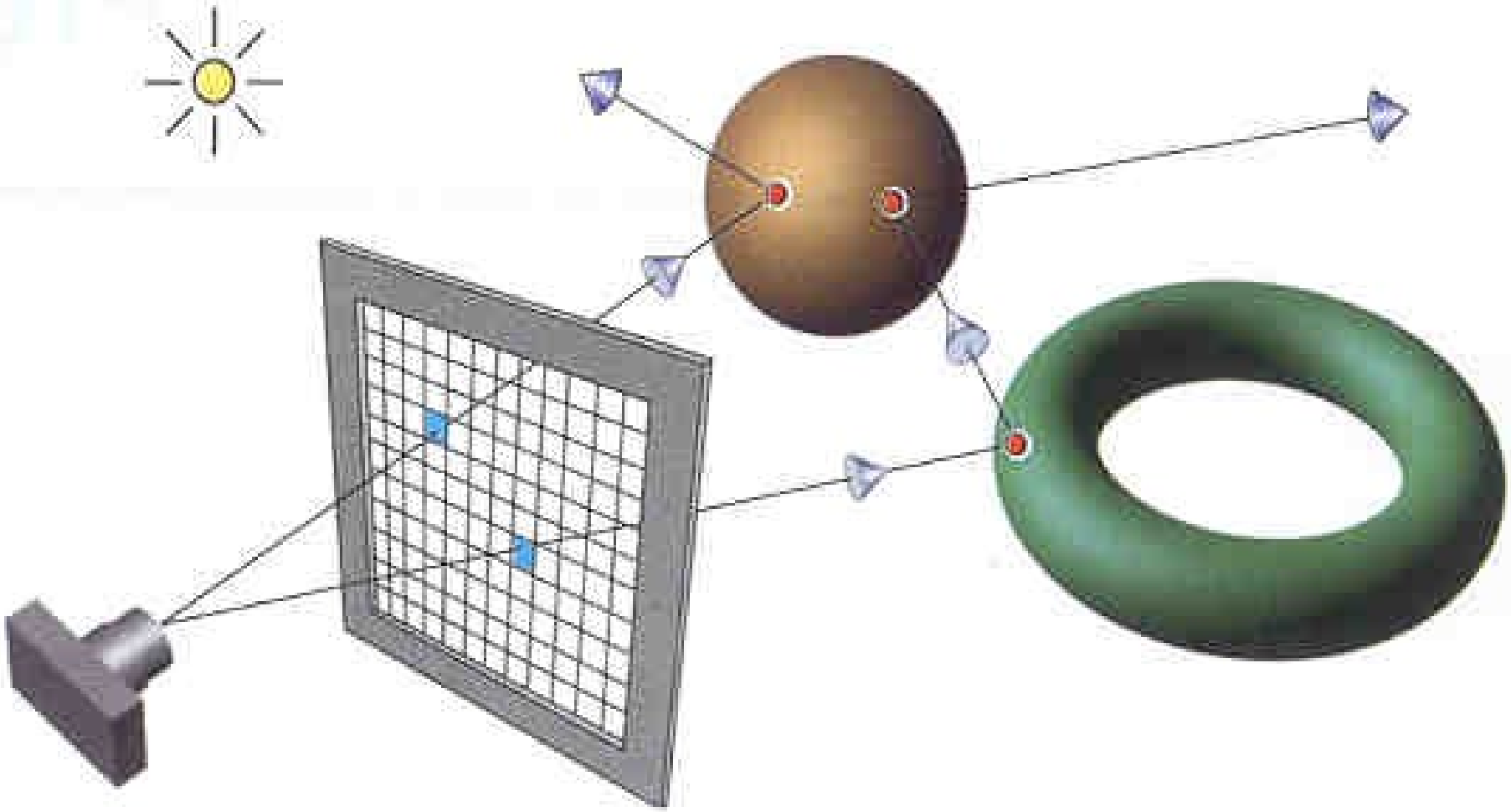
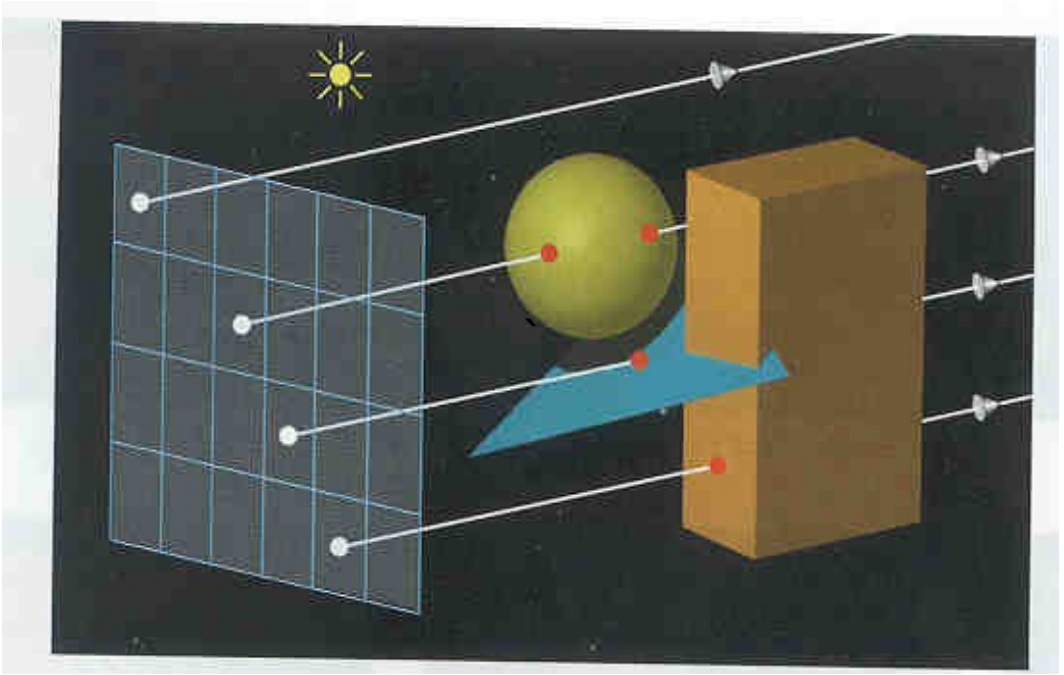


Figure 1. The ray-tracing process.

How ray tracing works

- Define scene and view
  - objects
  - lights
  - eye
- For each pixel
  - Shoot ray from pixel
  - Find nearest hit
  - Use object properties and lights to calculate color, or set to black if no hits



Interaction between Lights and Objects

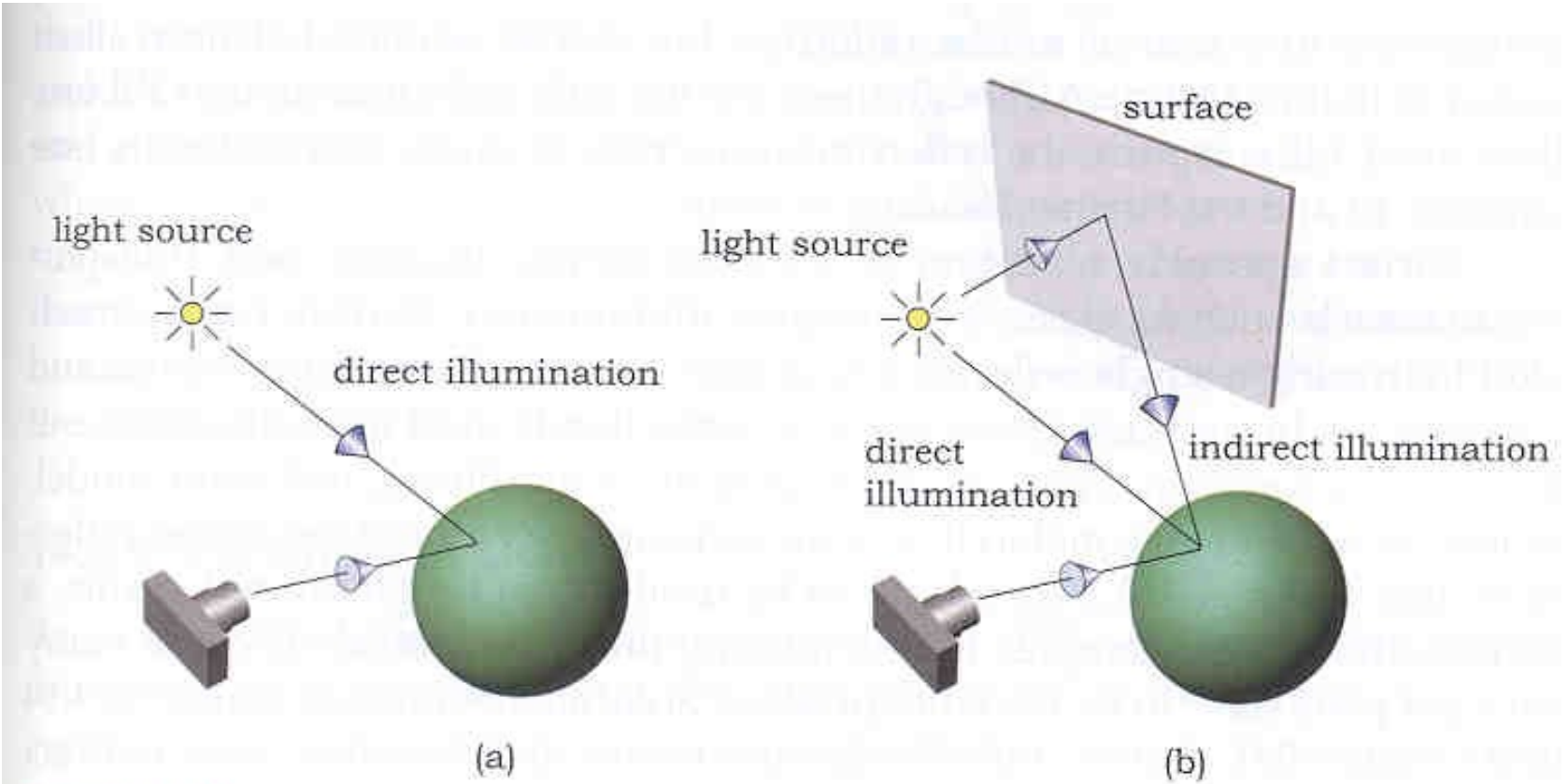


Figure 14.2. (a) Direct illumination hits the surface of an object directly from a light source; (b) indirect illumination hits a surface after being reflected from at least one other surface.

Bouncing Rays from Surfaces

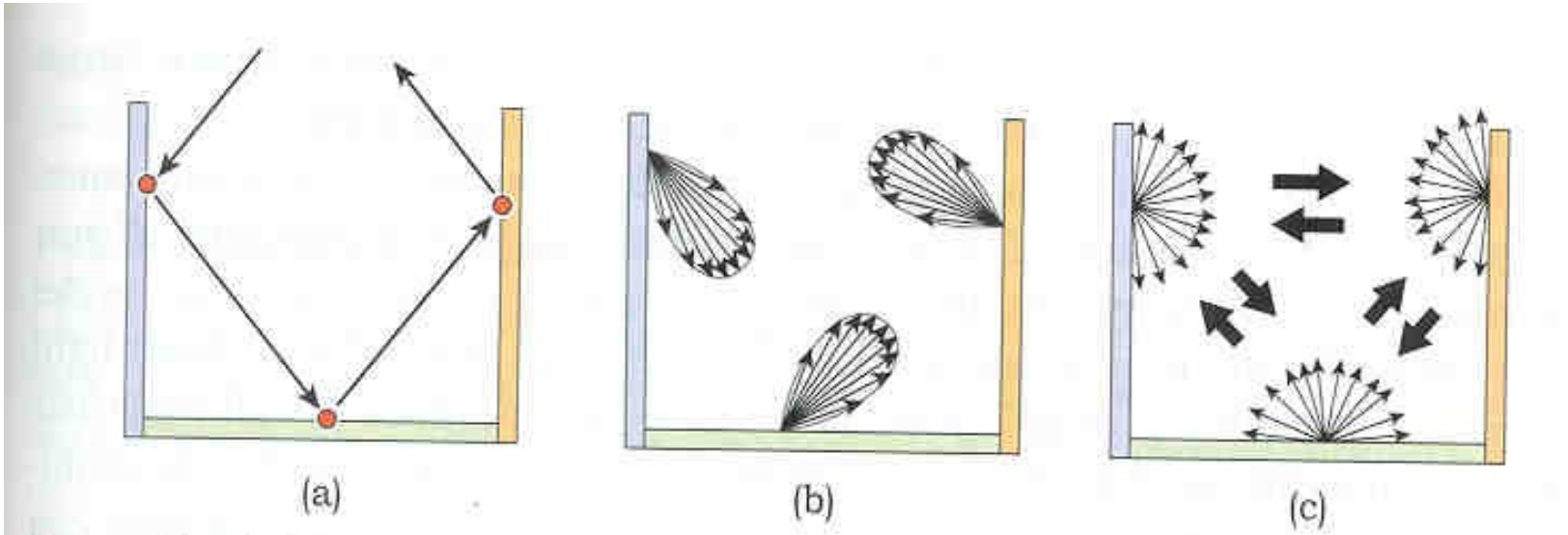


Figure 14.4. (a) Mirror reflection can be modeled by tracing a single reflected ray at each hit point; (b) modeling glossy specular light transport between surfaces requires many rays to be traced per pixel; (c) modeling perfect diffuse light transport between surfaces also requires many rays to be traced per pixel.

Light Reflection

- Diffuse (Lambertian) reflection
  - Intensity Factor N•L

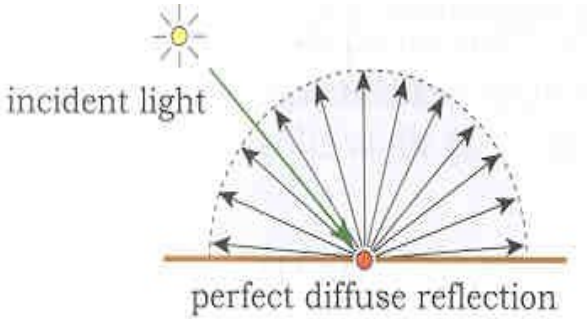


Figure 13.6. Light being scattered from a perfectly diffuse surface.

- Specular reflection
  - R = 2(N•L)N-L
  - Intensity Factor α

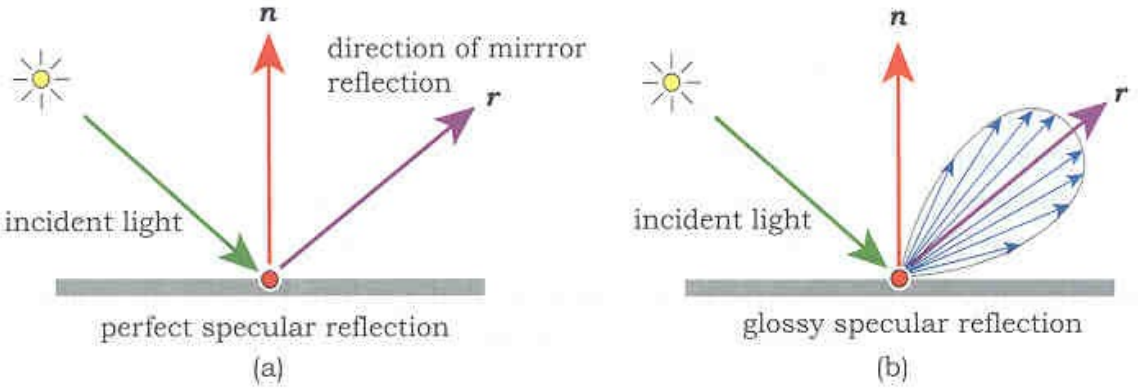


Figure 14.3. (a) Perfect specular reflection; (b) glossy specular reflection.

Specular Reflected Light

- Assume the ray (from the eye) hits objects 1,2,3,... with reflection coefficients $\alpha_1,\alpha_2,\alpha_3,...$
- Specular Reflection Color

  $\alpha_1(C_1 + \alpha_2(C_2+\alpha_3(C_3+...)))$

  $= \alpha_1C_1 + \alpha_1\alpha_2C_2+\alpha_1\alpha_2\alpha_3C_3+...$
- Since light is assumed to be linearly additive, just keep track of α and add light along successive bounces of the ray
- White specular means α can be a scalar

Simple Ray Tracing Algorithm

- Initialize ray (**O**,**d**)
  - color = black
  - coef = 1
- Find closest intersection **P**
  - color += coef*ambient*material
  - *if not in shadow color* += coef*$\mathbf{N}\bullet\mathbf{L}$*diffuse*material
  - coef *= reflectivity
  - redirect ray from **P** to $\mathbf{d} - 2(\mathbf{d}\bullet\mathbf{N})\mathbf{N}$
- Stop when no intersection, or coef<<1, or maximum number of bounces

Ex 49: Three Ray Traced Spheres

- Simple scene
  - Three highly reflective spheres
  - Two white lights (one close, one far)
- Support classes
  - Vec3, Mat3, Color
- Base classes
  - Ray, Material, Light
- Object classes
  - Sphere

Implementation Notes

- Written in ***very bad*** C++
  - *KISS*
  - No object abstraction
- Use STL vector<> class for lists
- Calculate array of pixel values *width* x *height*
  - View by transforming pixel location
  - Copy to screen using *glDrawPixels*
- All calculations in ***global*** coordinates
  - Preprocess scene as needed

Building a real Ray Tracer in C++

- Base classes
  - Ray
  - Object
  - Light
  - Material
- Derived Object Classes
  - Sphere
  - Cube
  - Triangle
  - Triangle Mesh

Object Class

- Type of object
  - Implicit Surface
    - Sphere
    - Torus, cylinder, cube, ...
  - Compound objects
    - Triangular mesh
- Intersection with a ray
  - Point of intersection
  - Normal
  - Textures, etc

Virtual Methods

- Base class
  - hit
  - sample
  - color
- Each object class overrides the base class

Intersecting a Complex Object

- Defining a complex object
  - Triangle mesh on vertexes
  - Gouraud shading
- Expensive to ray trace
  - Test every ray against every triangle in the object
  - Test bounding box of entire object
- Intersections
  - Plane
  - Axis-aligned box
  - Generic triangle

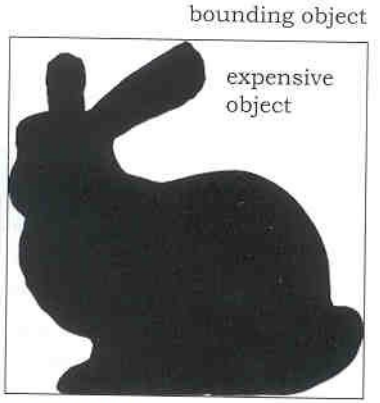


Figure 19.1. The Stanford bunny and a bounding box.

Perspective Ray Tracing

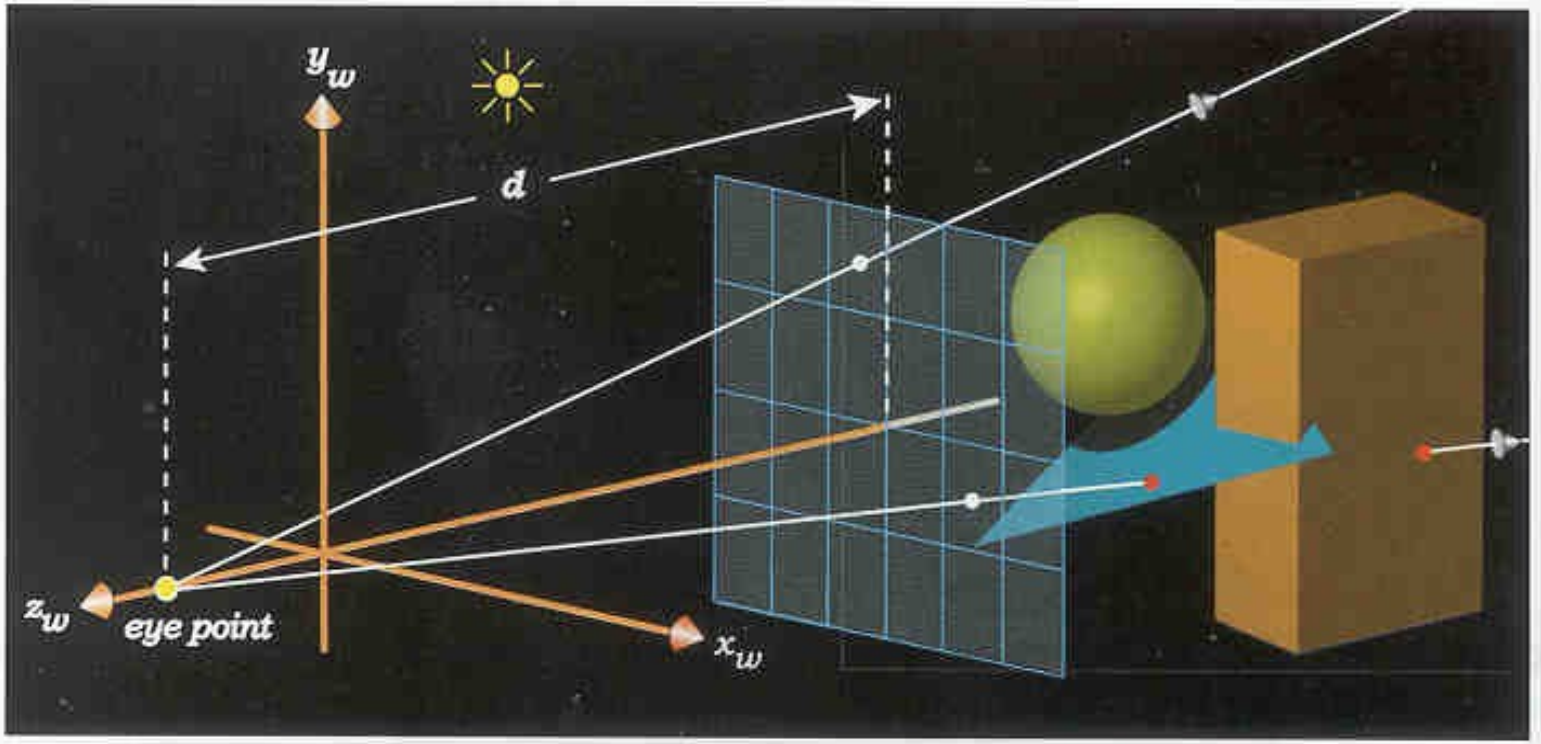
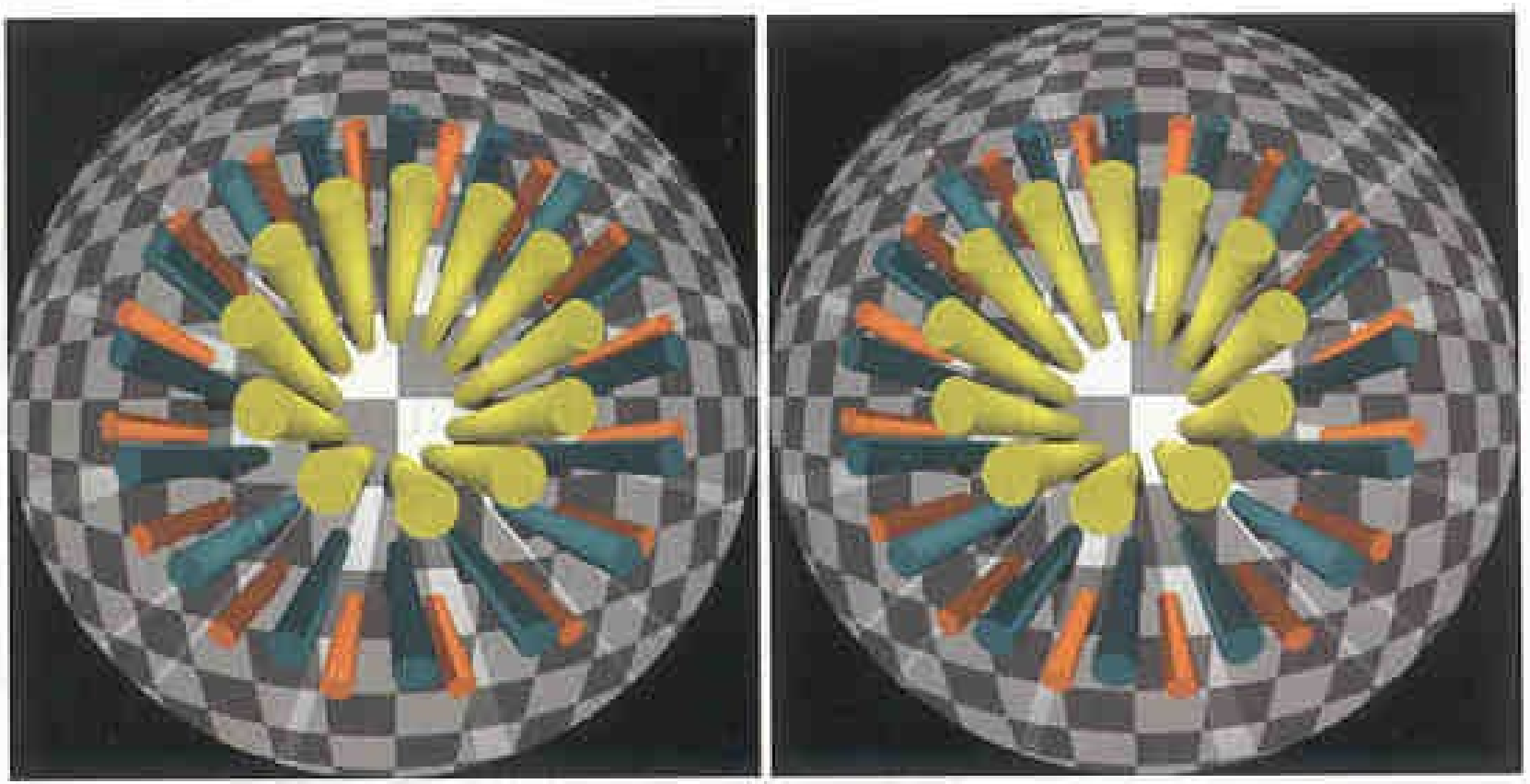


Figure 8.14. Set-up for axis-aligned perspective viewing with the eye point and two rays going through pixel centers.

Stereoscopy



left-eye view

right-eye view