

# **Ray Tracing in Practice**

**CSCI 4239/5239**

**Advanced Computer Graphics  
Spring 2026**

# Simple Ray Tracing Algorithm

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

# Ex 26: Three Ray Traced Spheres

- Simple scene
  - Three highly reflective spheres
  - Two white lights (one close, one far)
  - OpenMP for parallel processing
- 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
  - OpenMP parallel calls to `RayTracePixel()`
  - 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

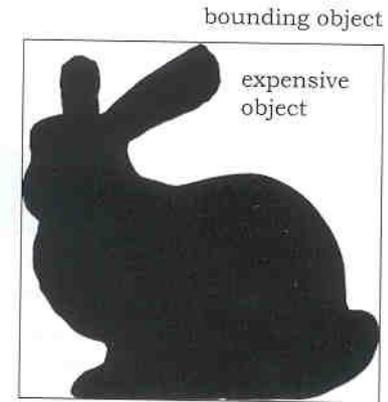
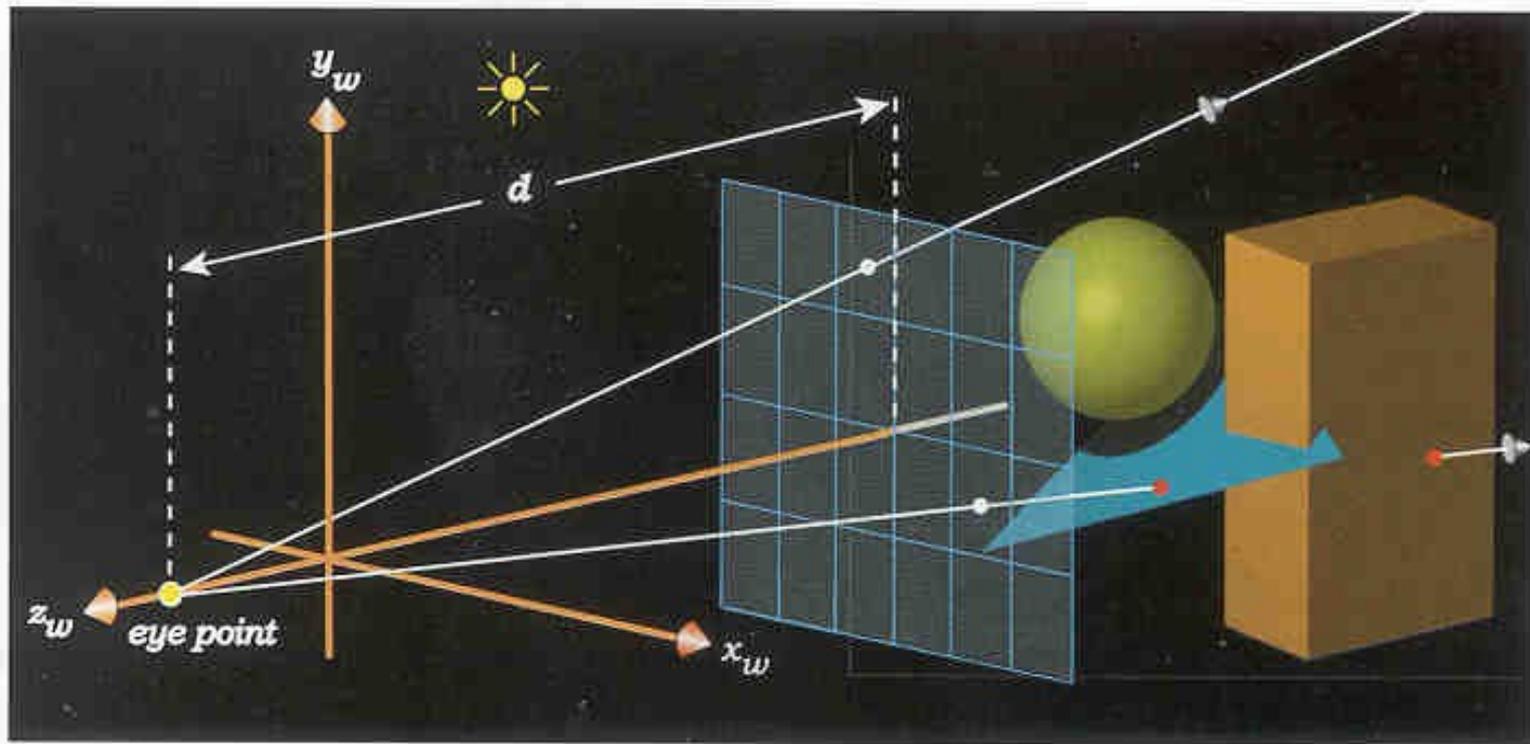


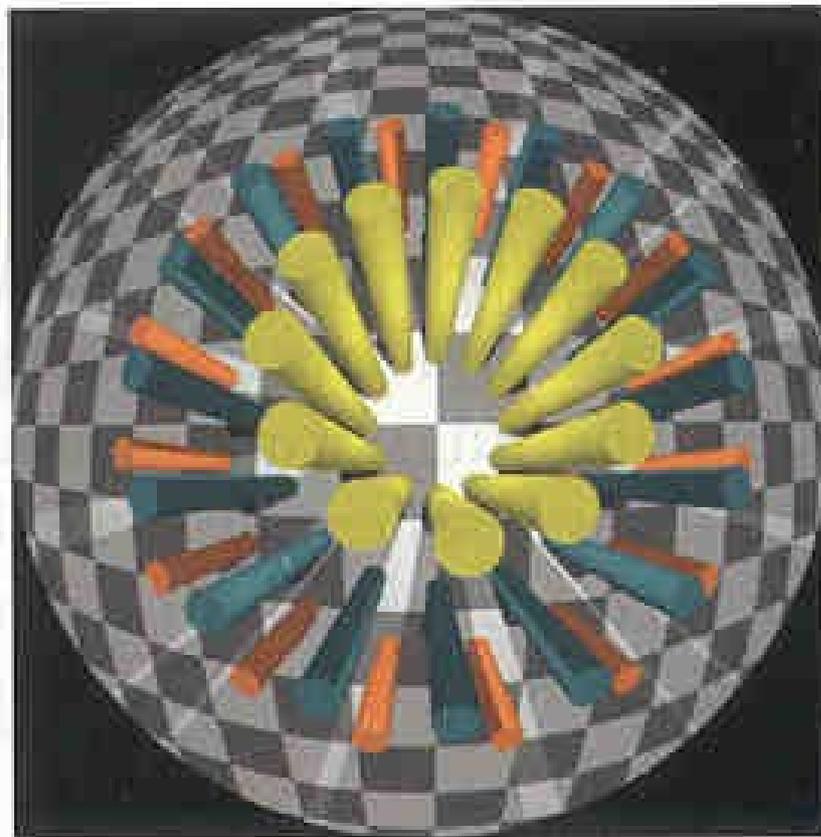
Figure 19.1. The Stanford bunny and a bounding box.

# Perspective Ray Tracing

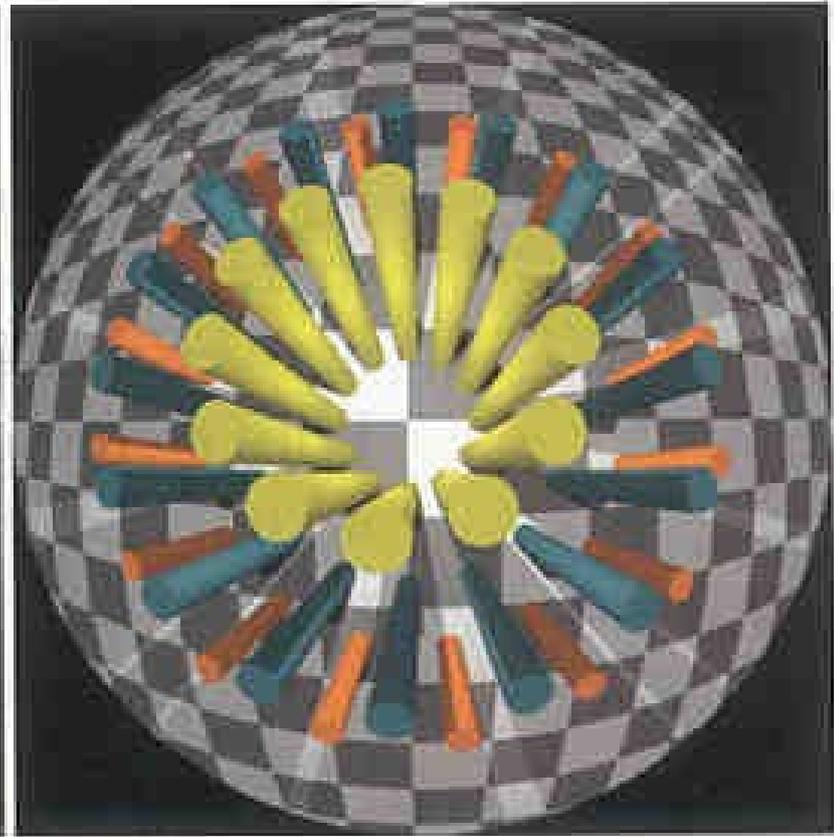


**Figure 8.14.** Set-up for axis-aligned perspective viewing with the eye point and two rays going through pixel centers.

# Stereoscopy



left-eye view



right-eye view

# Installing PBRTv3

- **Build code from github**

- `git clone --recursive https://github.com/mmp/pbrt-v3.git`
  - `cd pbrt-v3`
  - `mkdir build`
  - `cd build`
  - `cmake ..`
  - `make -j8`
  - `sudo make install`

- **Run using** `pbrt foo.pbrt`

- **Examples** `ex27-3.pbrt` **and** `ex27-3.png`

# Installing PBRTv4

- **Build code from github**

- `git clone --recursive https://github.com/mmp/pbrt-v4.git`
  - `git clone git://git.pbrt.org/pbrt-v4-scenes`
  - `cd pbrt-v4`
  - `mkdir build`
  - `cd build`
  - `cmake PBRT_OPTIX7_PATH=xxxx ..`
  - `make -j8`
  - `sudo make install`

- **Run using** `pbrt --gpu foo.pbrt`

- **Examples** `ex27-4.pbrt` **and** `ex27-4.png`

- **See differences in input with**

- `diff ex27-3.pbrt ex27-4.pbrt`