CSE328 Fundamentals of Computer Graphics: Concepts, Theory, Techniques, and Applications

Hong Qin
Department of Computer Science
Stony Brook University (SUNY at Stony Brook)
Stony Brook, New York 11794-2424
Tel: (631)632-8450; Fax: (631)632-8334
qin@cs.stonybrook.edu
http://www.cs.stonybrook.edu/~qin

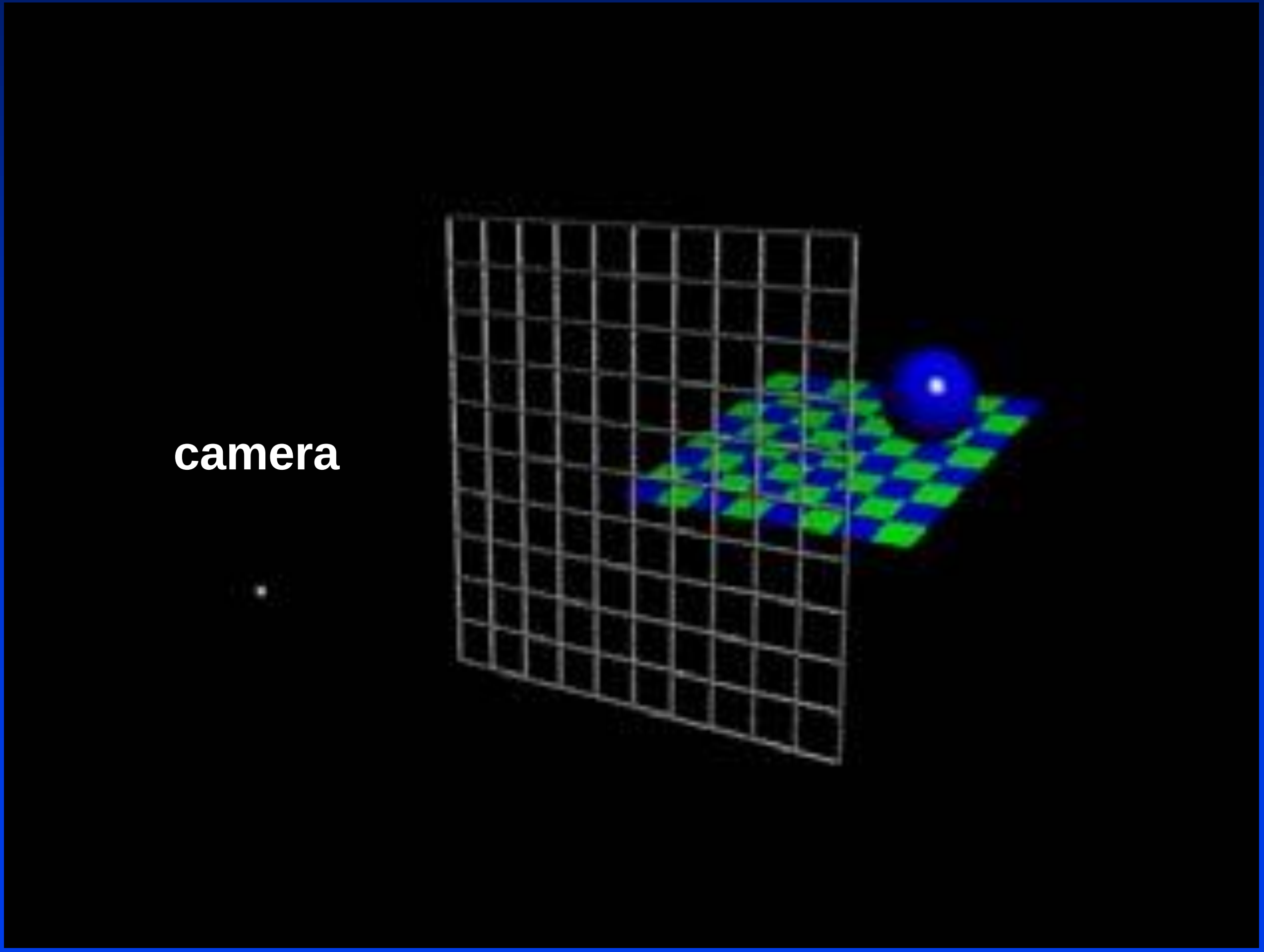
Photo-realistic Examples

Photo-realistic Examples

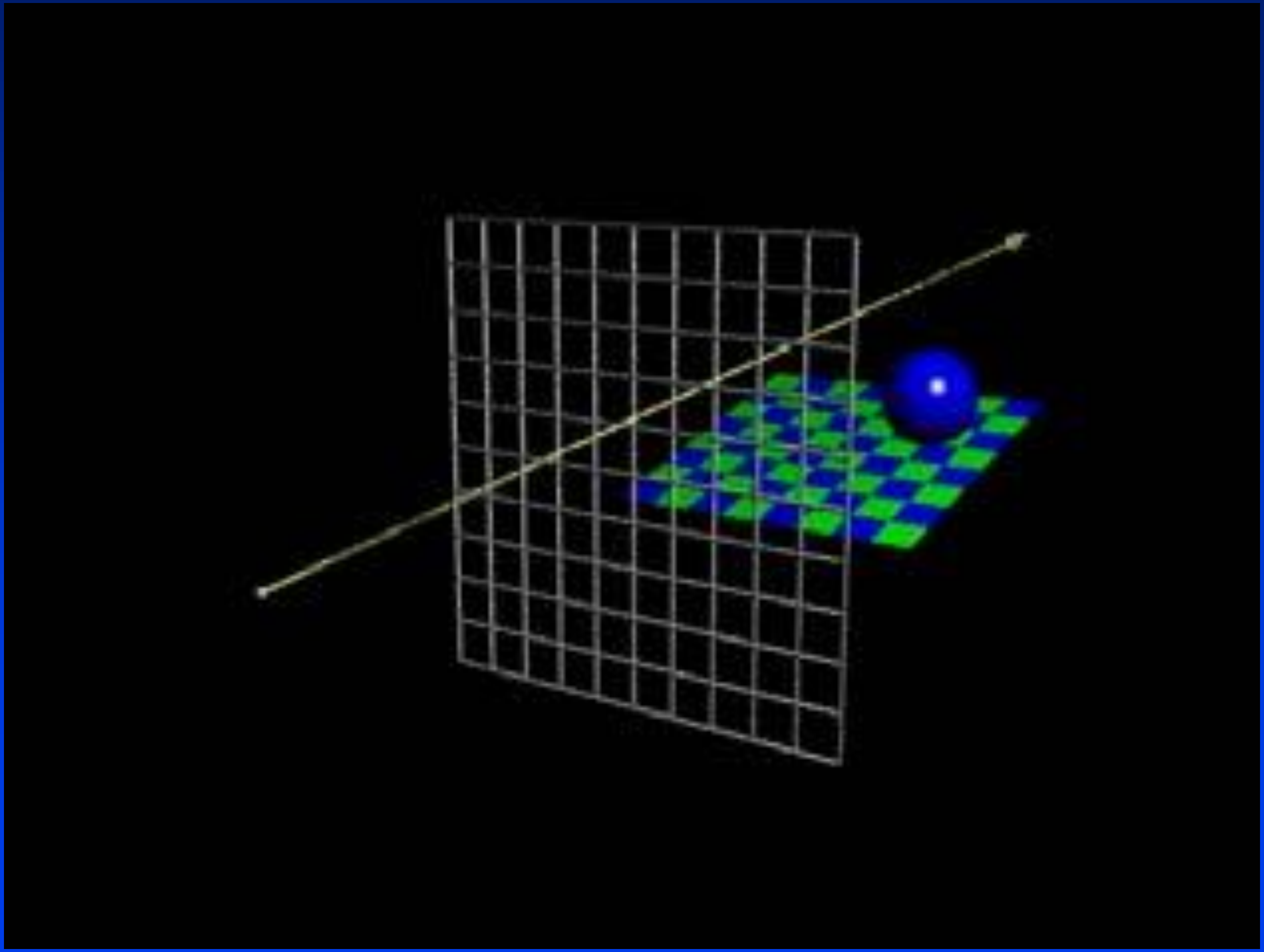
Photo-realistic Examples

Ray Casting: Basic Principles

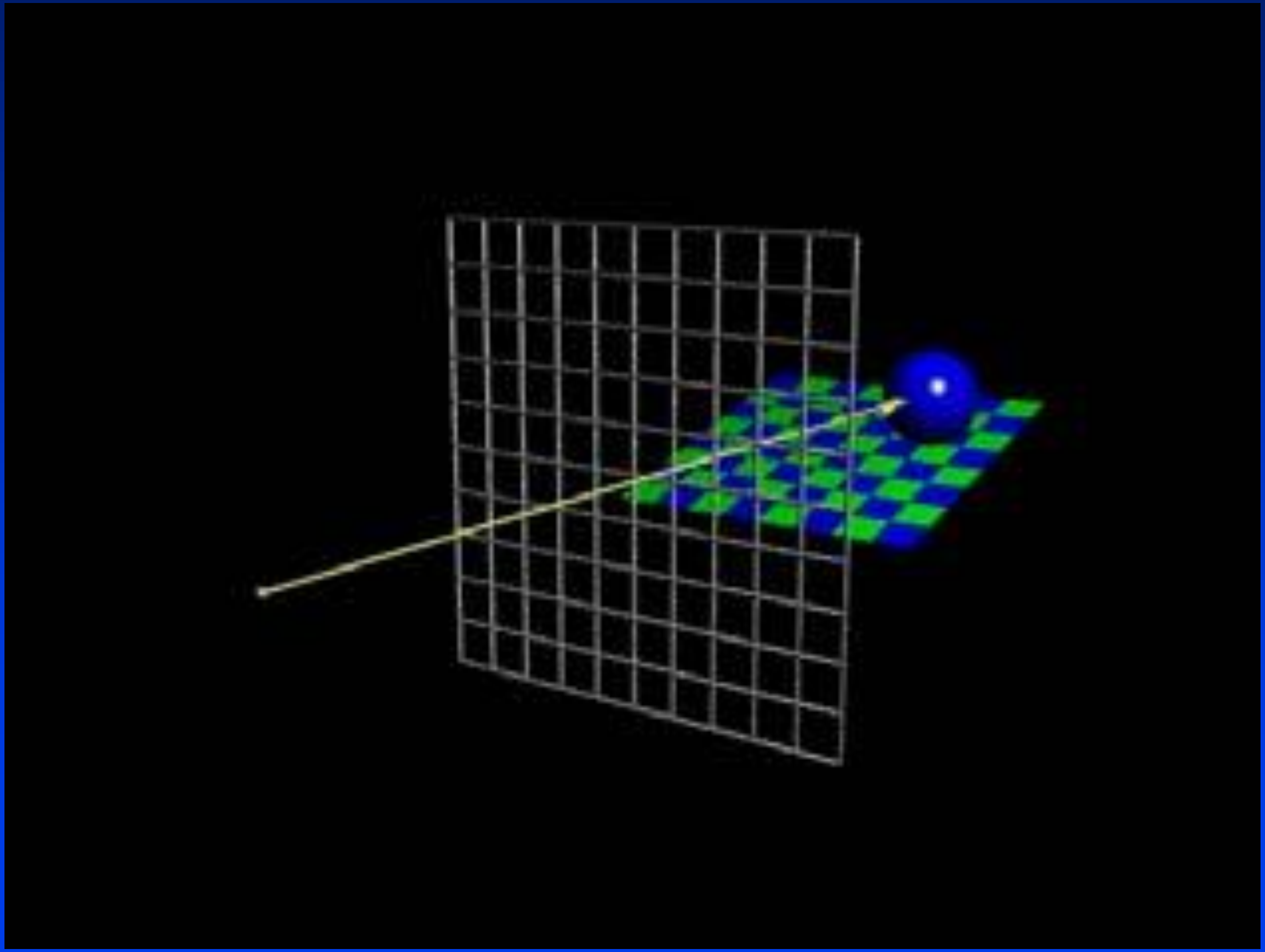
- Camera
- Pixel plane
- Scene



Ray Casting: Basic Principles

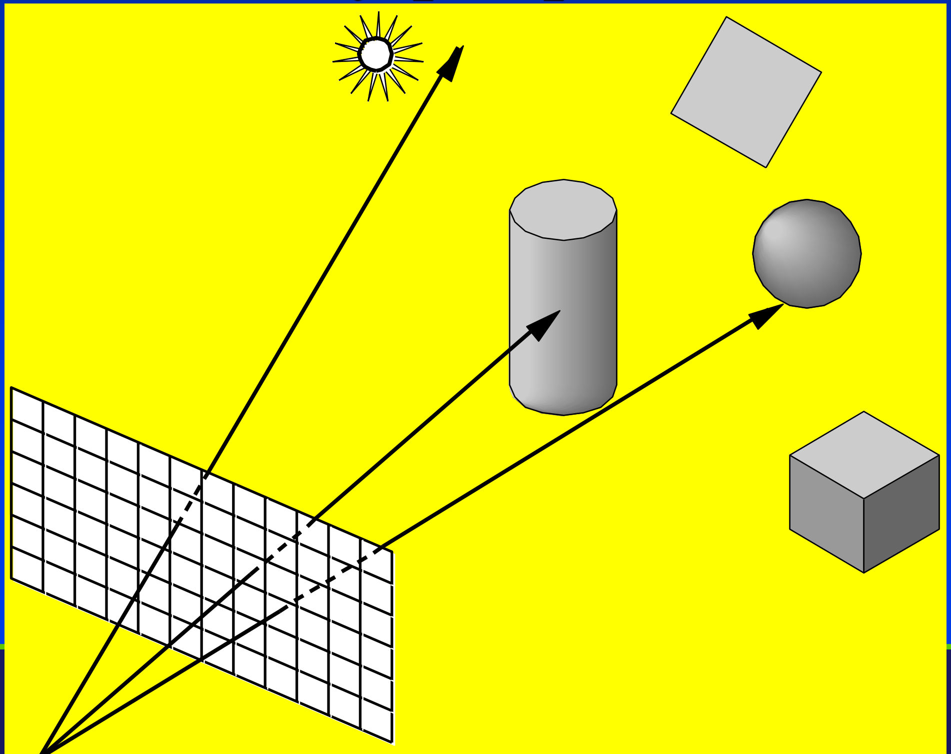


Ray Casting: Basic Principles



Ray Casting: Basic Principle

- Only rays that reach the eye matter
- Reverse direction and cast rays
- Need at least one ray per pixel



Math for Ray Casting

$$P = P_0 + su$$

$$u = \frac{P_{pix} - P_{prp}}{\left| P_{pix} - P_{prp} \right|}$$

Ray-Tracing

Today's Topics

- We will take a look at ray-tracing which can be used to generate extremely photo-realistic images

Ray Tracing

Ray can split and change directions

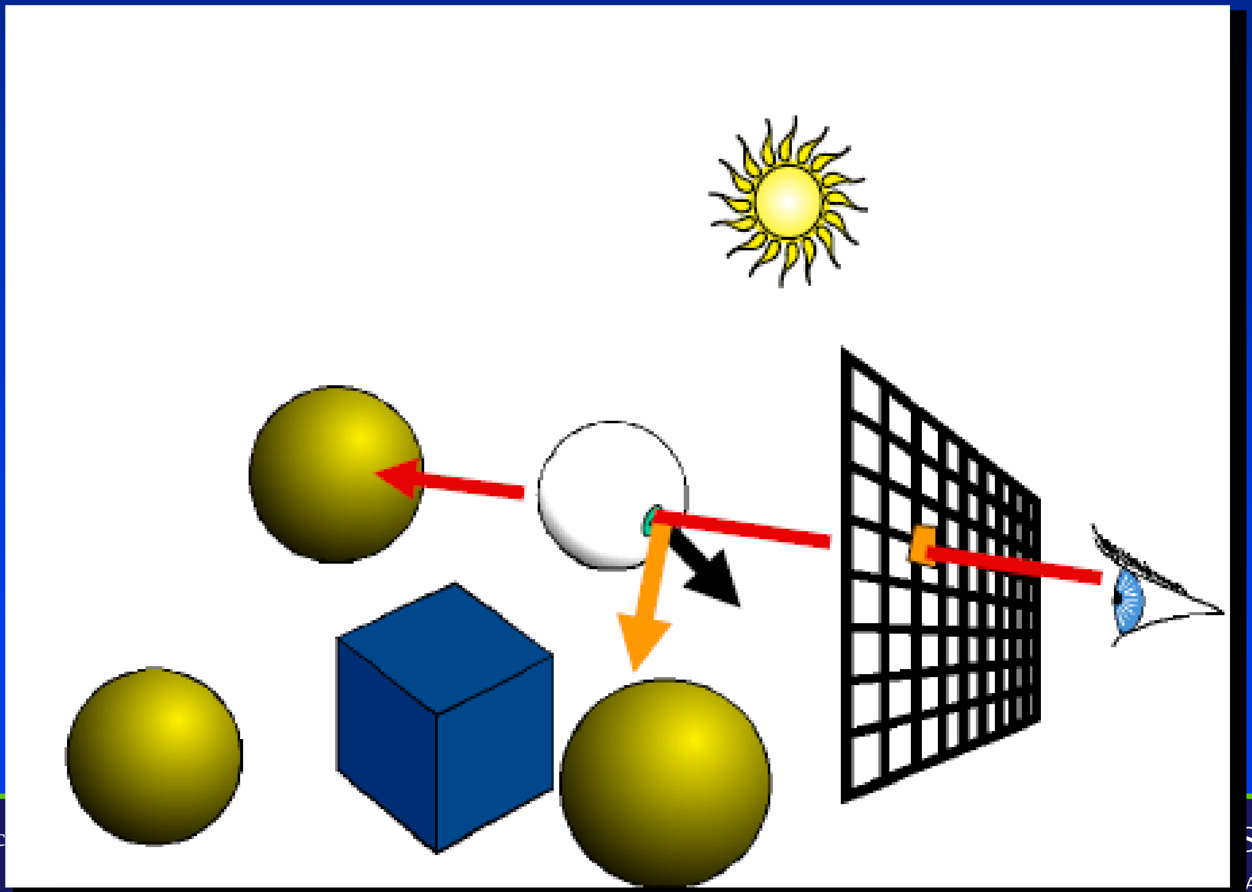
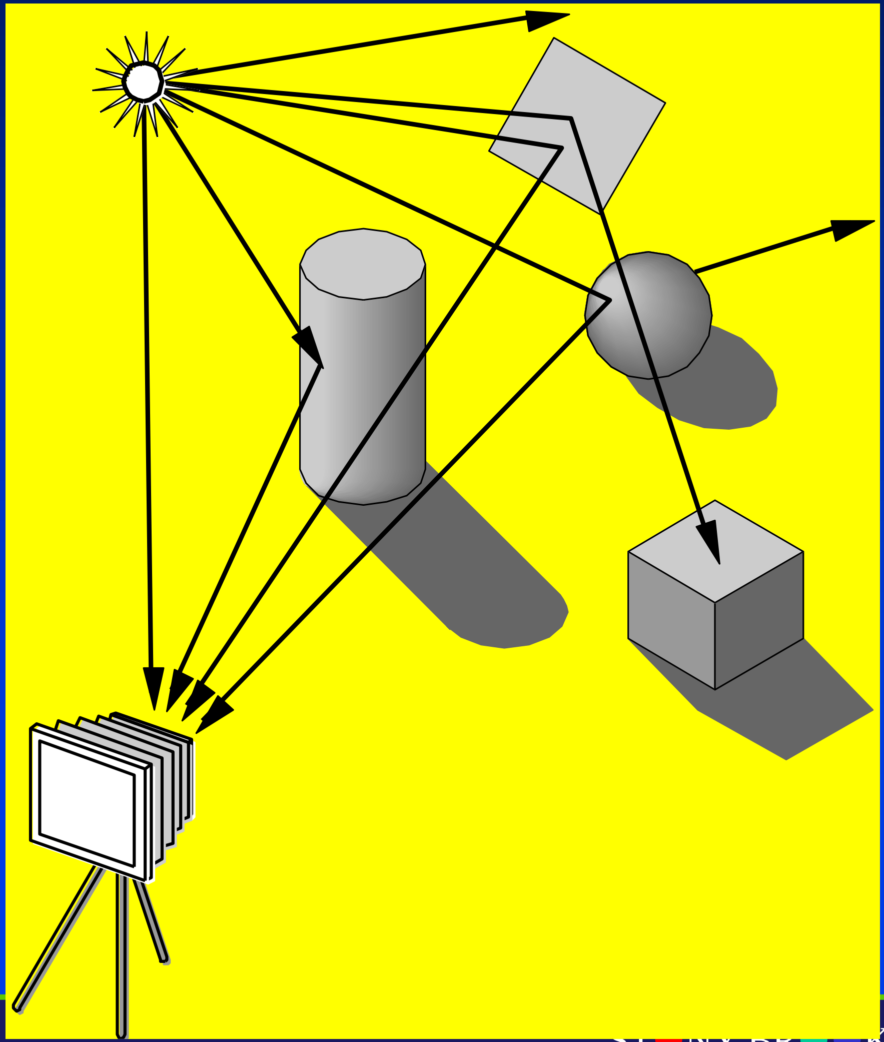


Photo-realistic Examples

Photo-realistic Rendering

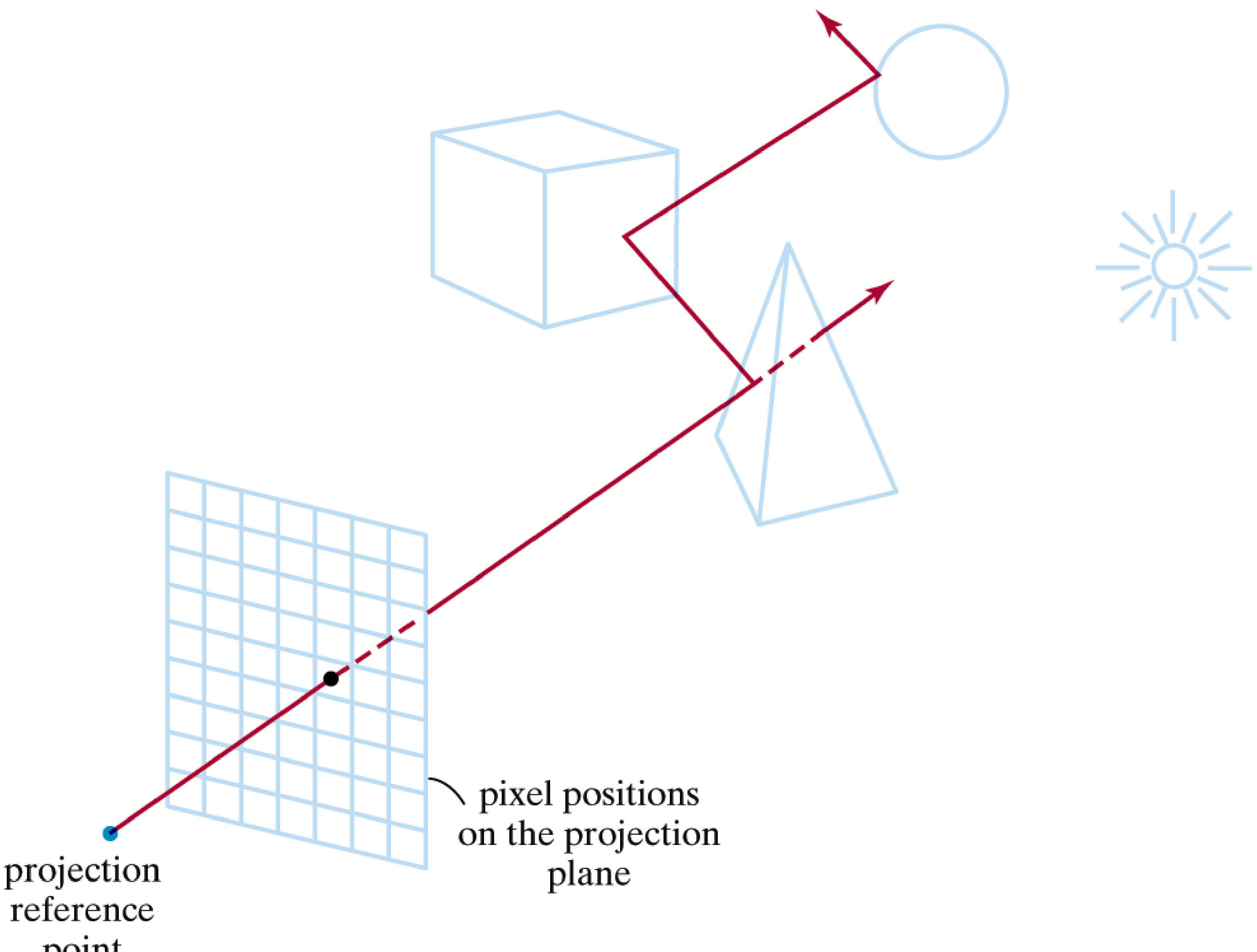
- Simple forward approach: Follow light rays from a point light source
- Can account for reflection and transmission (refraction) during ray transmission from a light source to image plane



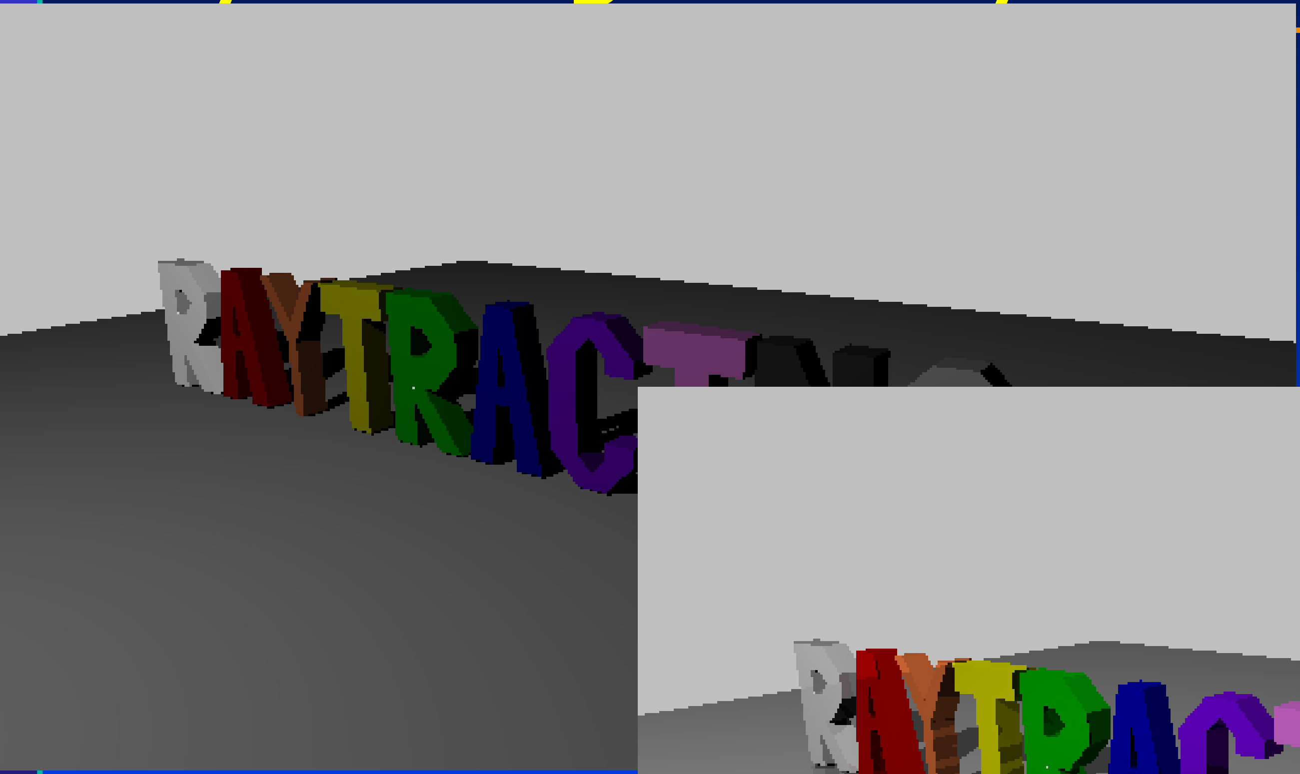
Computation

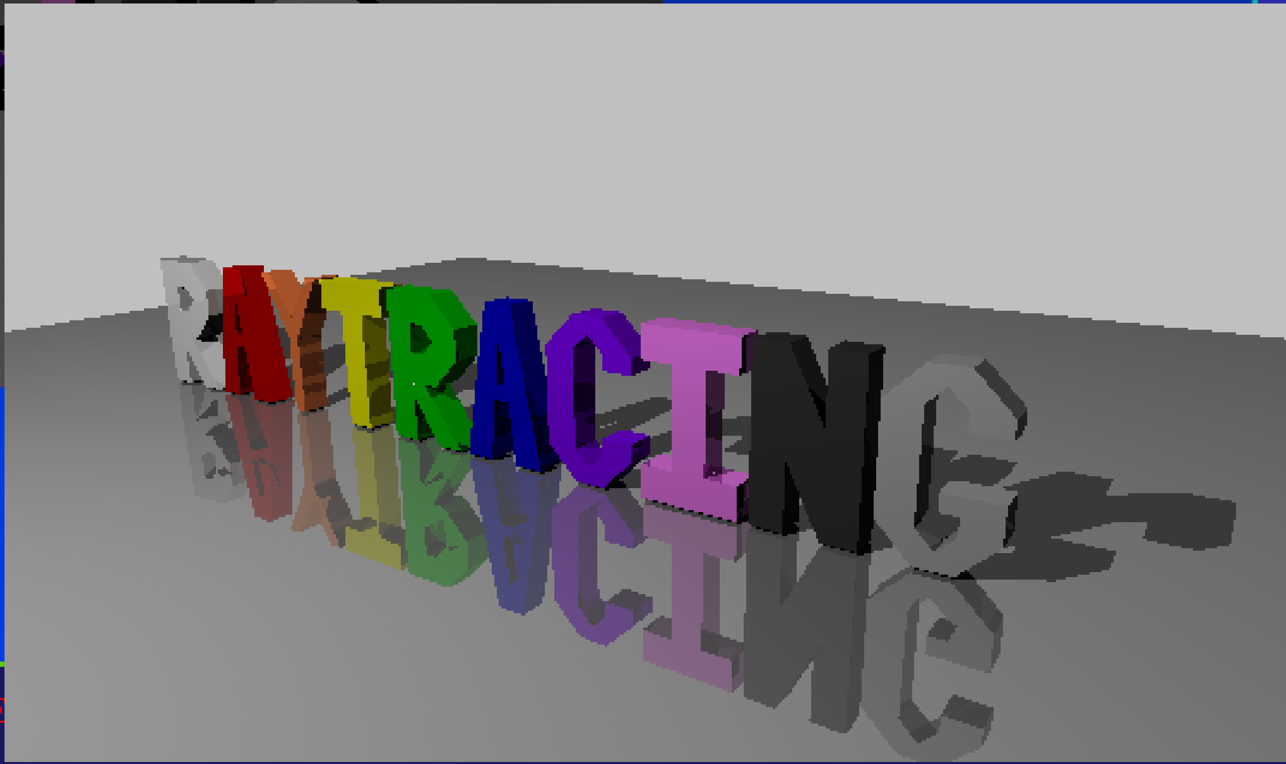
- Should be able to handle all physical interactions between objects and light rays
- Unfortunately, the direct, forward paradigm is not computational tractable at all
- Most rays do not affect what we see on the image plane, because those rays do not penetrate through the image plane at all
- Scattering produces many (infinite) additional rays
- Alternative: ray-casting/ray-tracing

Ray-Tracing: Basic Principles

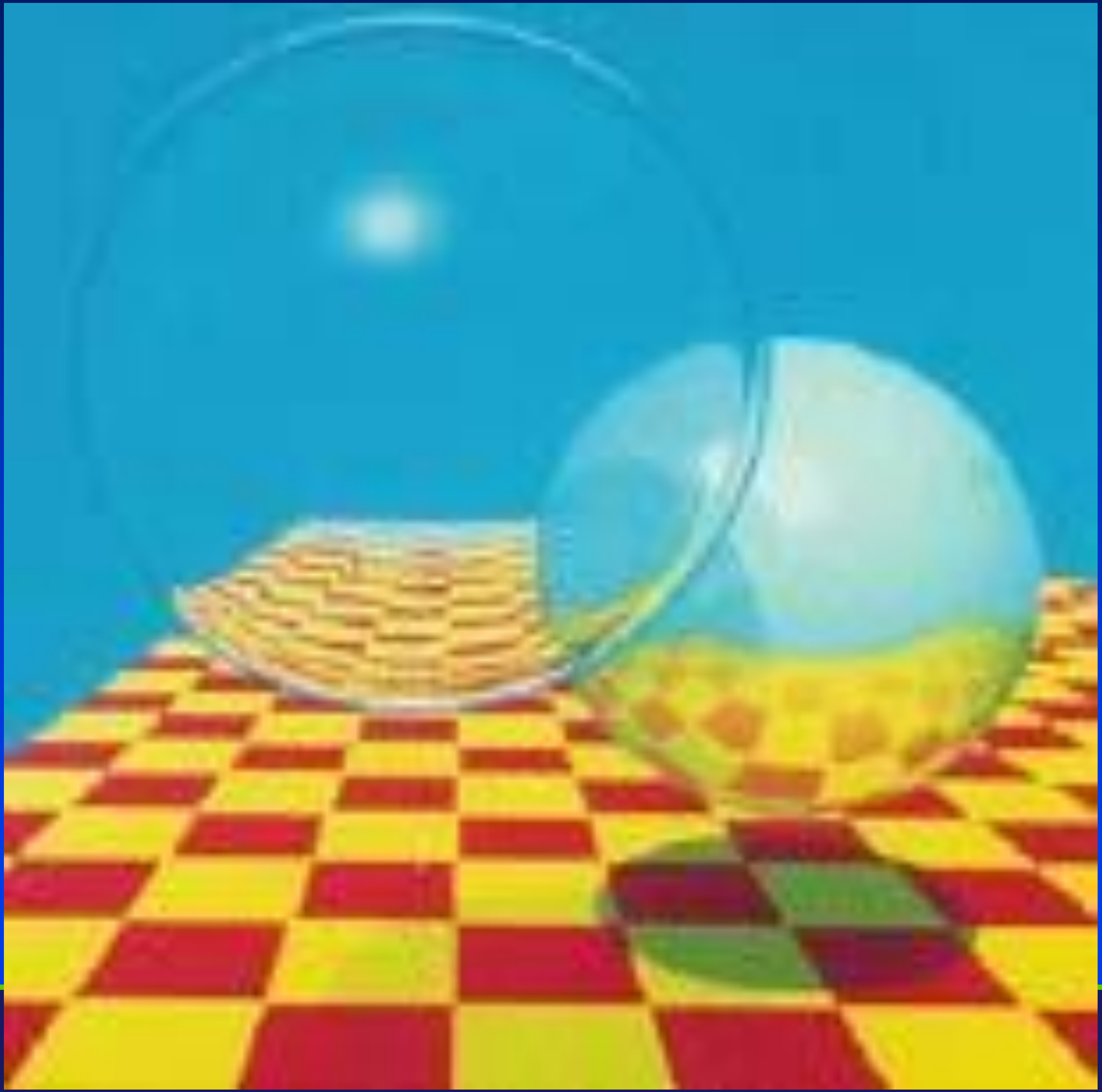


Raycasting vs. Ray Tracing





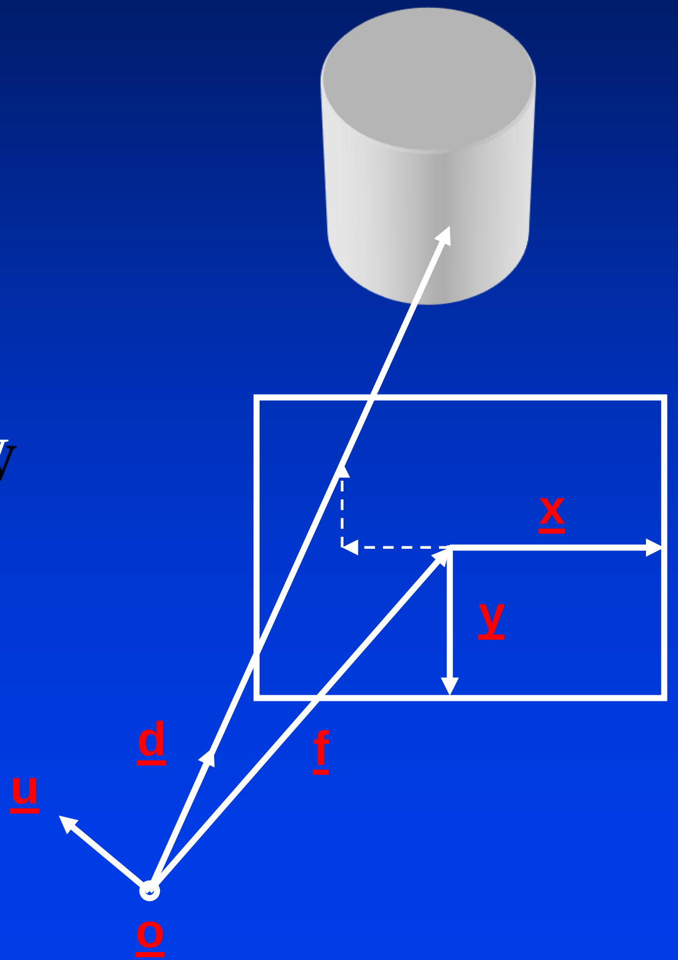
Ray Tracing



Ray Tracing

Ray Generation

- Important parameters
  - o: Origin (point of view)
  - f: Vector to center of view, focal length
  - x, y: Span the viewing window
  - xres, yres: Image resolution



Ray Tracing: Basic Setup

- Assumption: empty space totally transparent
- Surfaces (geometric objects)
  - 3D geometric models of objects
- Optical surface characteristics (appearance)
  - Absorption, reflection, transparency, color, …
- Illumination
  - Position, characteristics of light sources

Fundamental Steps

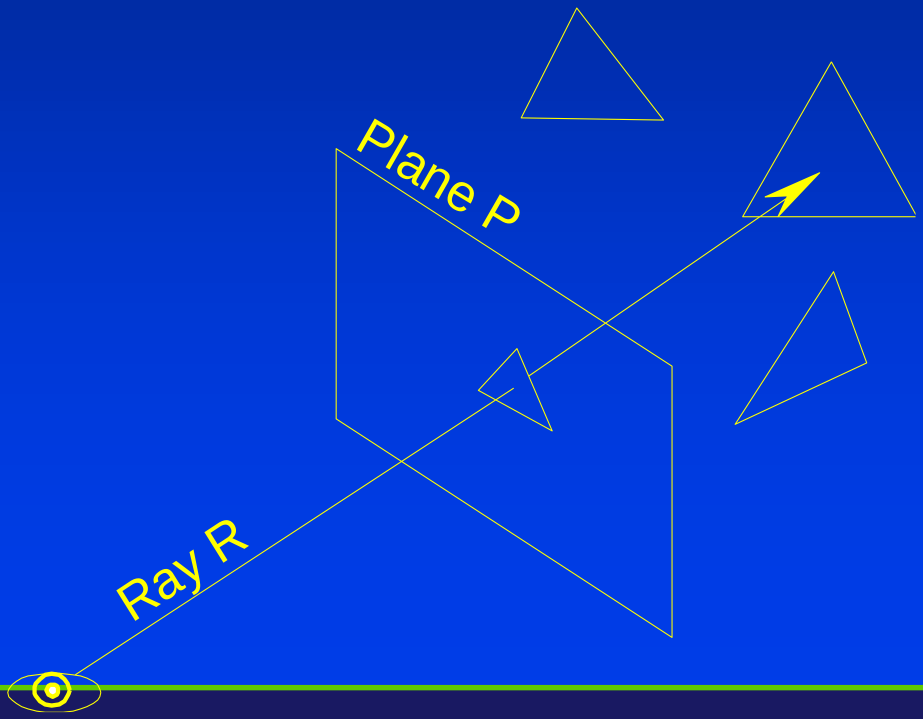
- Generation of primary rays
  - Rays from viewpoint into 3D scene
- Ray tracing & traversal
  - First intersection with scene geometry
- Shading
  - Light (radiance) send along primary ray
  - Compute incoming illumination with recursive rays

Ray Tracing Algorithm

- Input:
  - Description of a 3D virtual scene
    - Described using triangles
  - Eye position and screen position
- Output:
  - 2D projection of the 3D scene onto screen

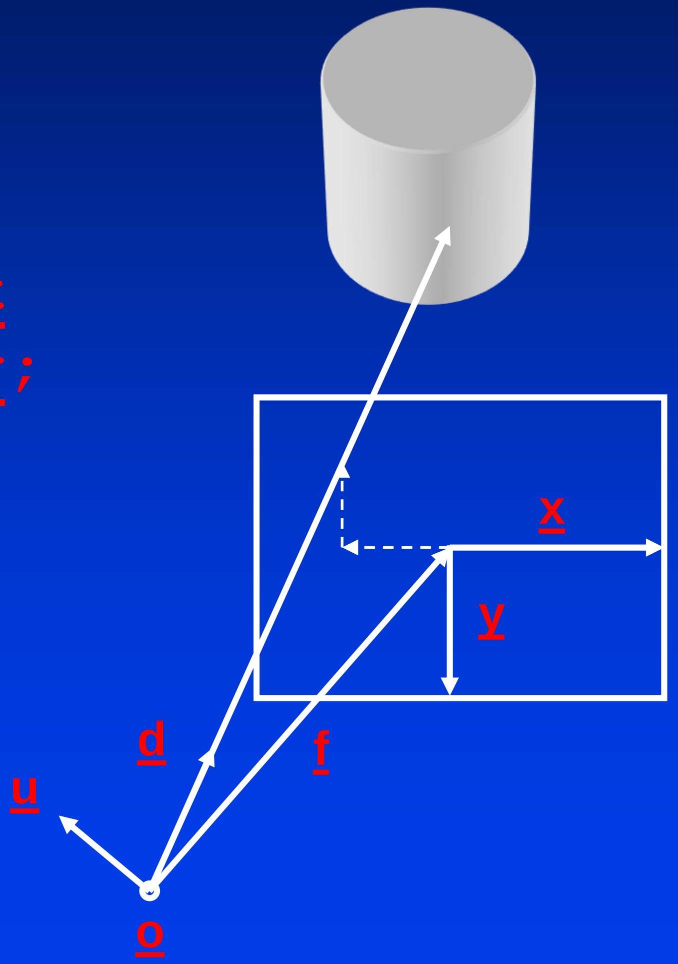
Ray Tracing Algorithm: First Step

- For each pixel in projection plane P
  - Cast ray from eye through current pixel to scene
  - Intersect with each object in scene to find which object is visible



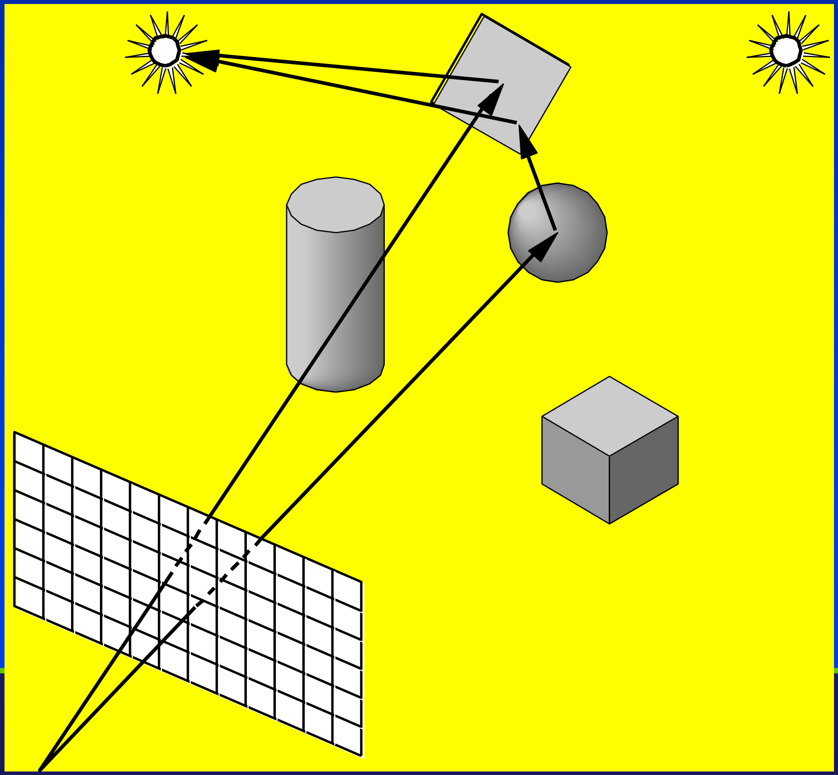
Algorithm

```
for (x= 0; x < xres; x++)
  for (y= 0; y < yres; y++)
  {
    d= f + 2(x/xres - 0.5)·x
         + 2(y/yres - 0.5)·y;
    d= d/|d|; // Normalize
    col= trace(o, d);
    write_pixel(x,y,col);
  }
```

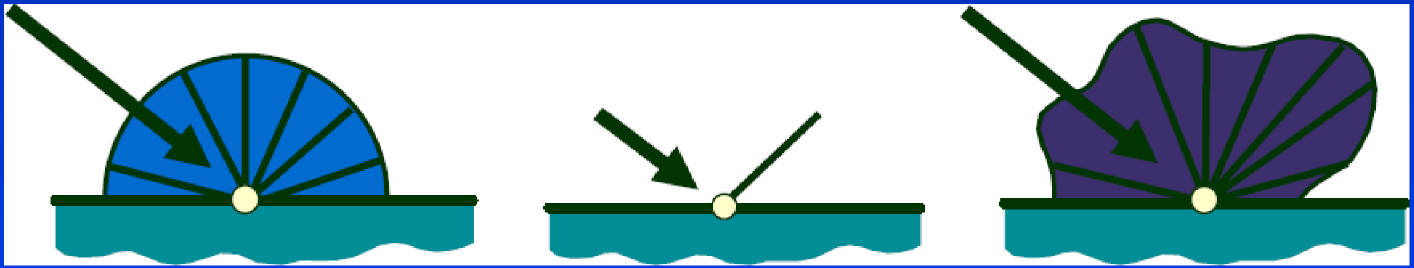


Reflection

- Must follow shadow rays off reflecting or transmitting surfaces
- Process is recursive



Ray Tracing



- Diffuse
- Cos (N.L)

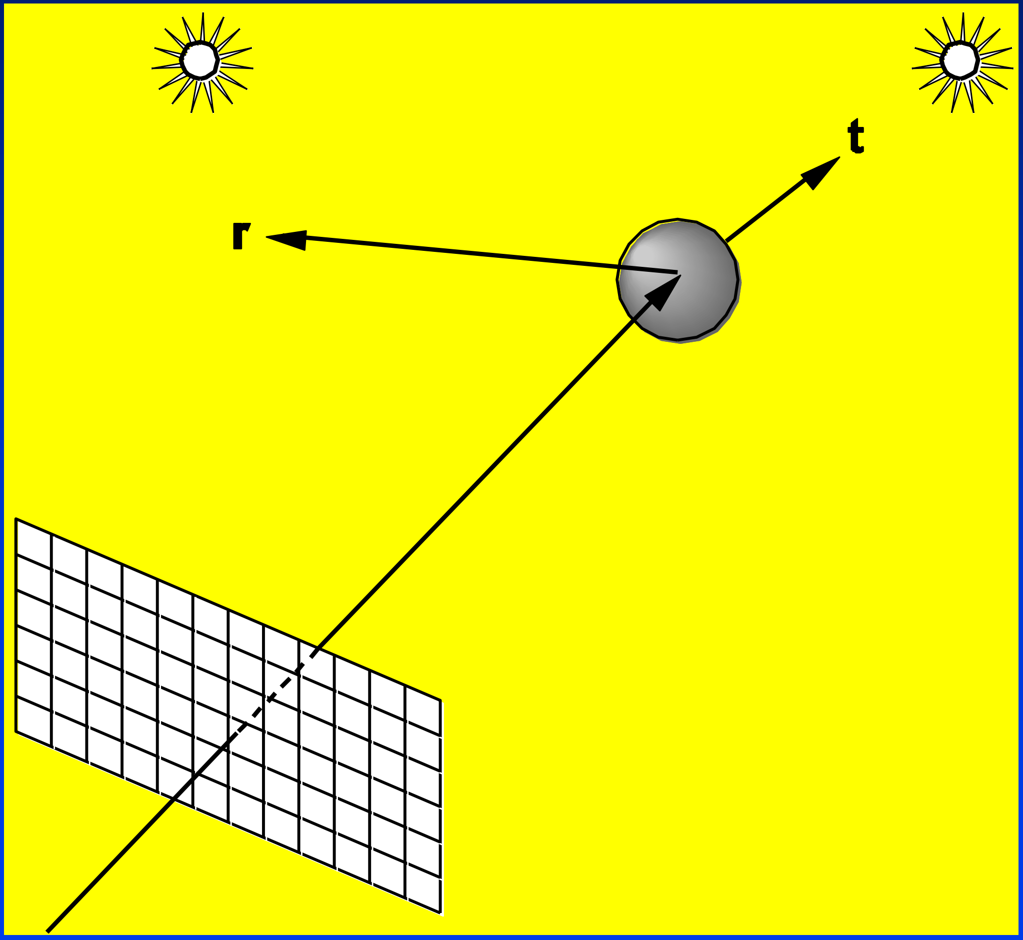
- Specular
- Perfect reflection (N.V) = (N.R)
- Recursive

- Phong shading
- Cos (R.V) of (N.H)
- Exponential n

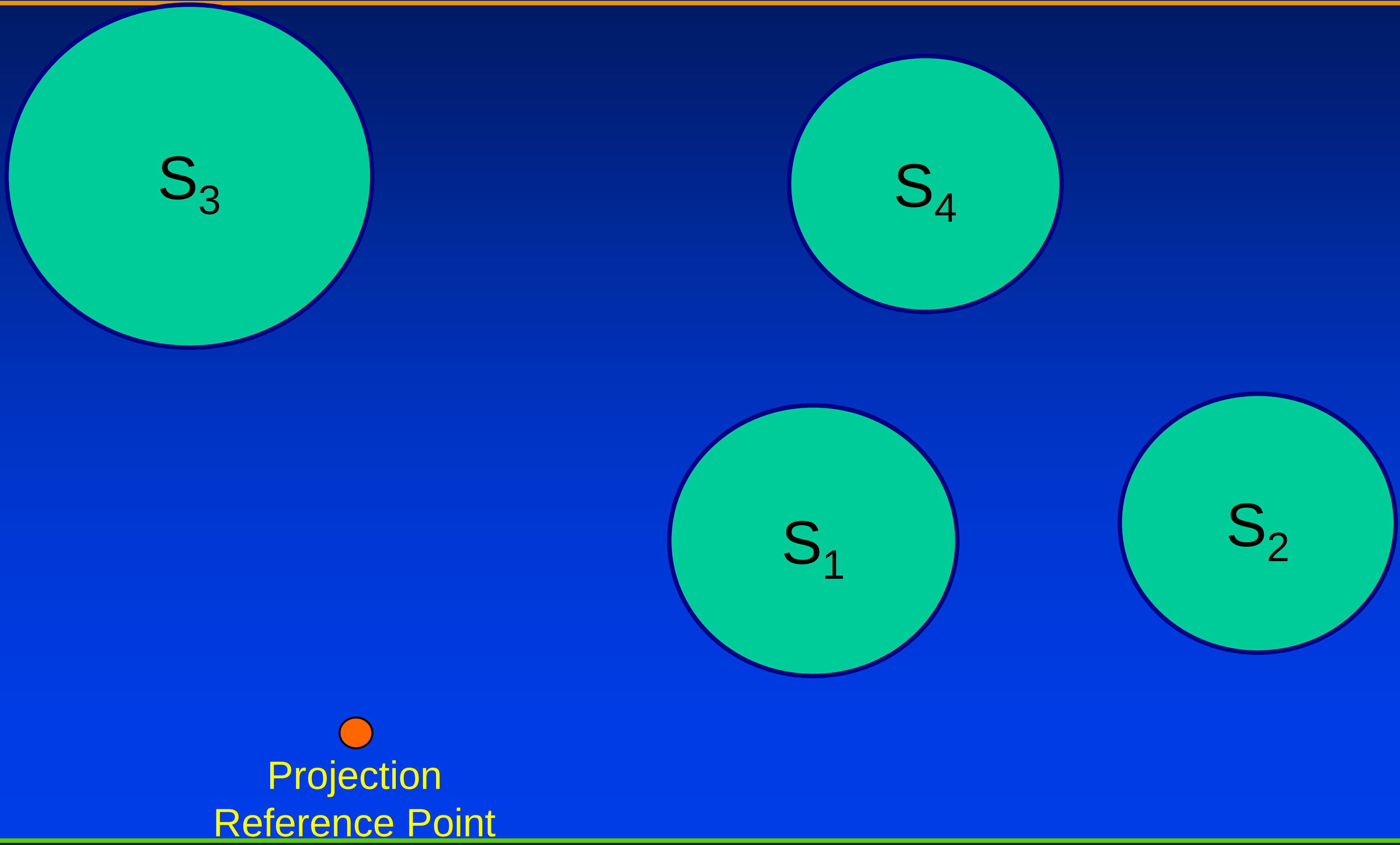
Diffuse Surfaces

- Theoretically the scattering at each point of intersection generates an infinite number of new rays that should be traced (computational intractable, however)
- In practice, we only trace the transmitted and reflected rays but use the Phong model to compute shade at intersection points

Reflection and Transmission

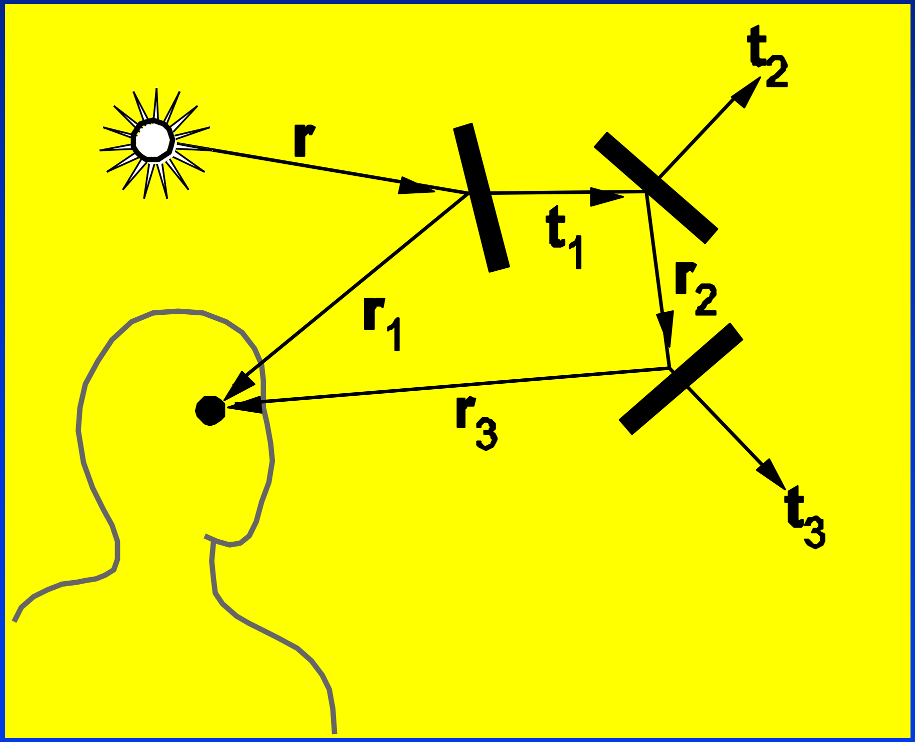


Ray-Tracing Tree Example

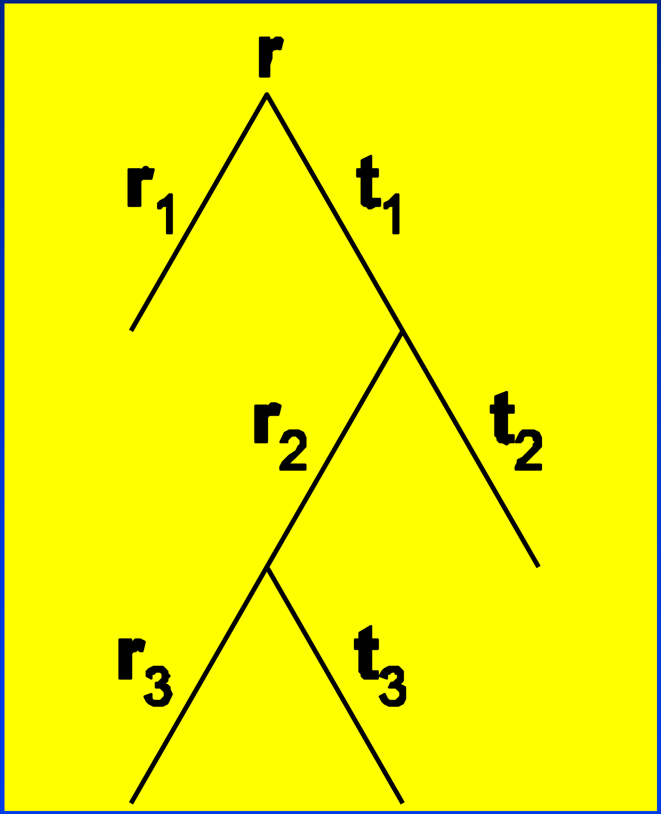


STONY BROOK
STATE UNIVERSITY OF NEW YORK

Ray Trees



Ray Tree



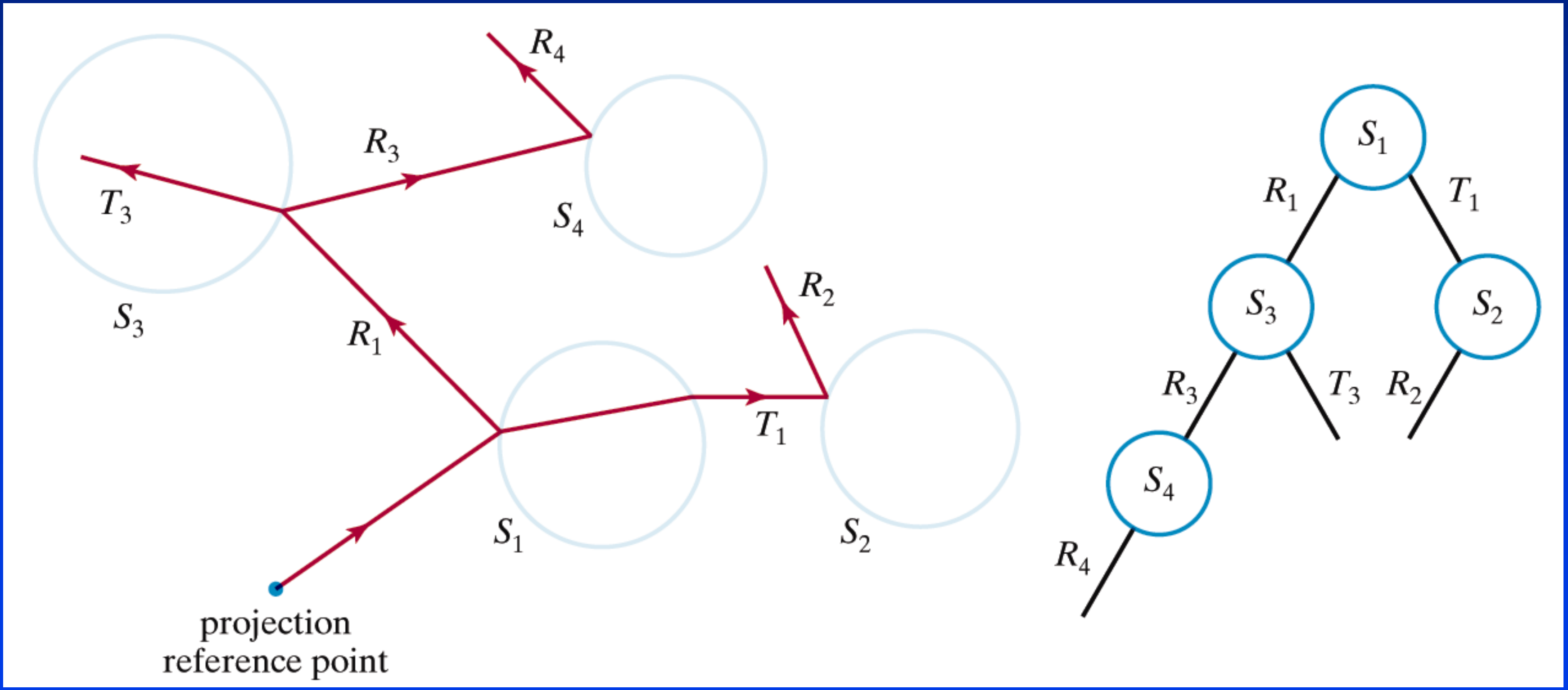
Basic Ray-Tracing

- Ray tracing proceeds as follows:
  - Fire a single ray from each pixel position into the scene along the projection path (a simple ray-casting mechanism)
  - Determine which surfaces the ray intersects and order these by distance from the pixel
  - The nearest surface to the pixel is the visible surface for that pixel
  - Reflect a ray off the visible surface along the specular reflection angle
  - For transparent surfaces also send a ray through the surface in the refraction direction
  - Repeat the process for these secondary rays

Ray-Tracing Tree

- As the rays travel around the scene each intersected surface is added to a binary **ray-tracing tree**
  - – The left branches in the tree are used to represent reflection paths
  - – The right branches in the tree are used to represent transmission paths
- The tree's nodes store the intensity at that surface
- The tree is used to keep track of all contributions to a given pixel

Ray-Tracing Tree Example



STONY BROOK
STATE UNIVERSITY OF NEW YORK

Ray-Tracing Tree

- After the ray-tracing tree has been completed for a pixel the intensity contributions are accumulated
- We start at the terminal nodes (bottom) of the tree
- The surface intensity at each node is attenuated by the distance from the parent surface and added to the intensity of the parent surface
- The sum of the attenuated intensities at the root node is assigned to the pixel

Building a Ray Tracer

- Best expressed recursively
- Can remove recursion later
- Image-based approach and algorithms
  - For each ray ....
- Find intersection with closest surface
  - Need the entire object database available
  - Complexity of calculation limits object types
- Compute lighting at surface
- Trace reflected and transmitted rays

When Do We Stop?

- Some light will be absorbed at each intersection
  - Only keep track of amount left
- Ignore rays that go off to infinity
  - Put large sphere around the scene
- Count steps

Terminating Ray-Tracing

- We terminate a ray-tracing path when any one of the following conditions is satisfied:
  - The ray intersects no surfaces
  - The ray intersects a light source that is not a reflecting surface
  - A maximum allowable number of reflections have taken place

Recursive Ray Tracer

```
color c = trace(point p, vector d,
  int step)
{
   color local, reflected,
  transmitted;
   point q;
   normal n;
   if(step > max)
  return(background_color);
```

Recursive Ray Tracer

```
q = intersect(p, d, status);
if(status==light_source)
  return(light_source_color);
if(status==no_intersection)
  return(background_color);


n = normal(q);
r = reflect(q, n);
t = transmit(q,n);
```

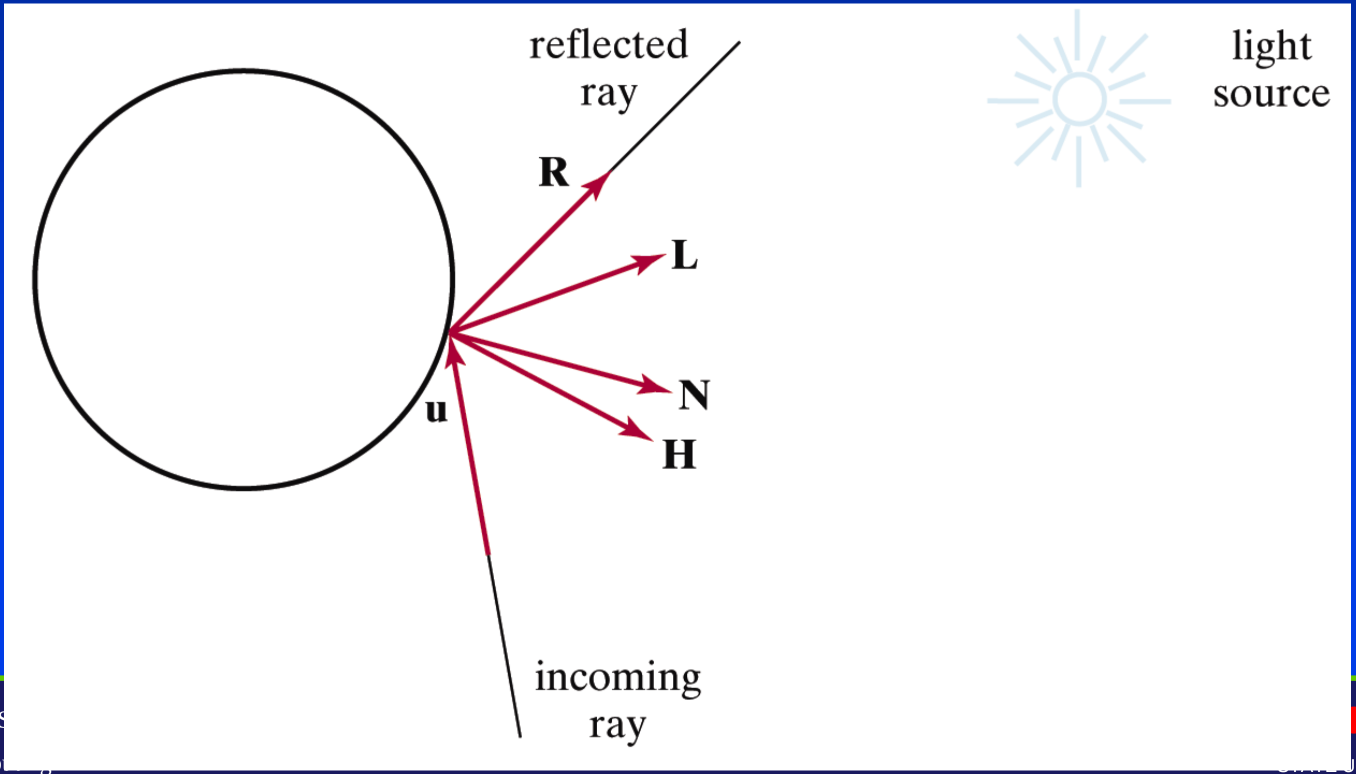
Recursive Ray Tracer

```
local = phong(q, n, r);
reflected = trace(q, r, step+1);
transmitted = trace(q,t, step+1);

return(local+reflected+
transmitted);
```

Ray-Tracing & Illumination Models

- At each surface intersection the illumination model is invoked to determine the surface intensity contribution



Computing Intersections

- Implicit objects
  - Quadrics
- Planes
- Polyhedra
- Parametric surfaces

Planes

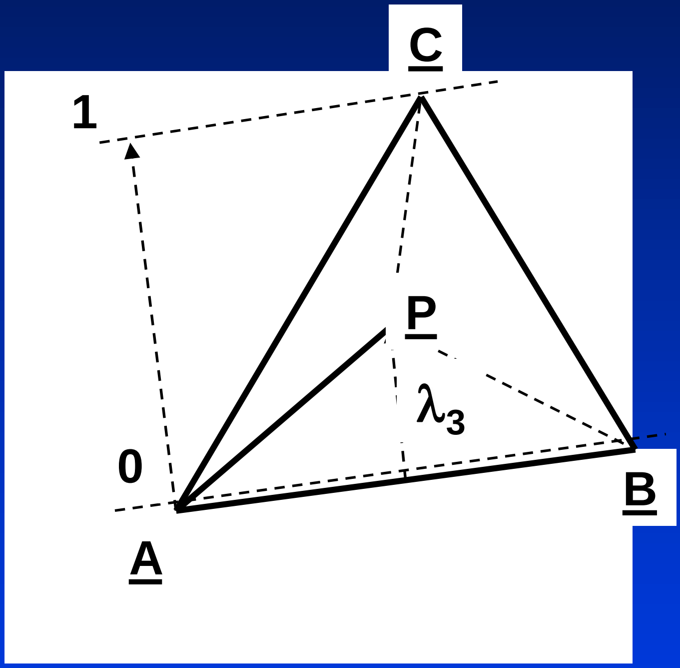
$\mathbf{p} \bullet \mathbf{n} + c = 0$

$\mathbf{p}(t) = \mathbf{p}_0 + t\,\mathbf{d}$

$t = -(\mathbf{p}_0 \bullet \mathbf{n} + c) / \mathbf{d} \bullet \mathbf{n}$

Intersection Ray - Triangle

- Barycentric coordinates
  - Non-degenerate triangle ABC
    $\underline{P} = \lambda_1 \underline{A} + \lambda_2 \underline{B} + \lambda_3 \underline{C}$
  - $\lambda_1 + \lambda_2 + \lambda_3 = 1$
  - $\lambda_3 = \angle(APB) / \angle(ACB)$ etc
    - Relative area



- Hit iff all λ_i greater or equal than zero

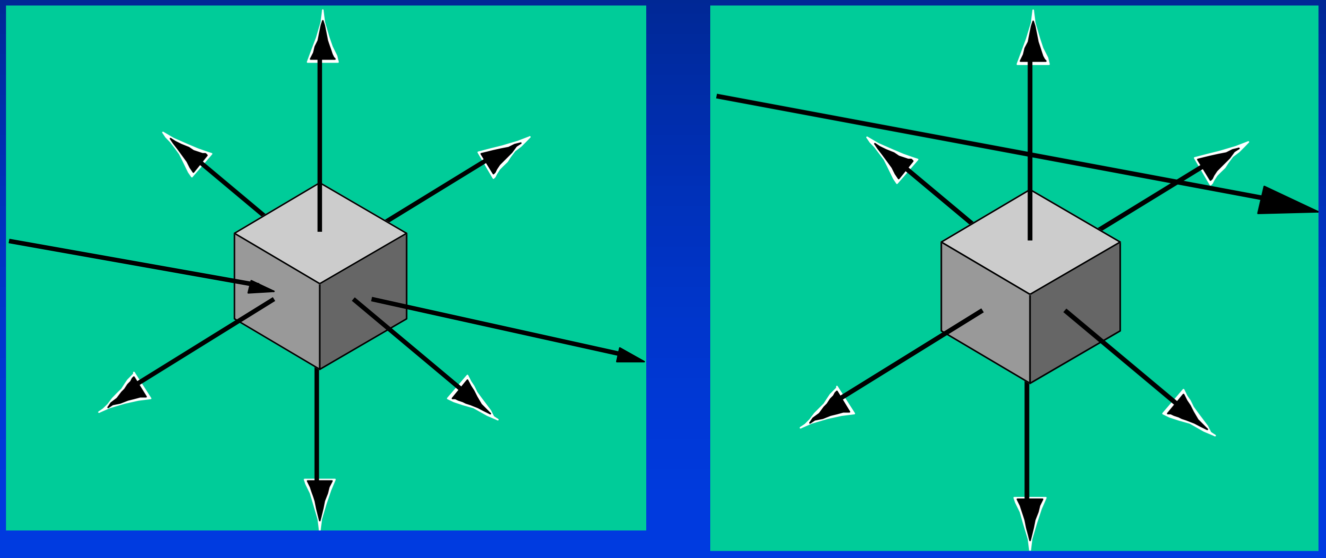
Polyhedra

- Generally we want to intersect with closed objects such as polygons and polyhedra rather than planes
- Hence we have to worry about inside/outside testing
- For convex objects such as polyhedra there are some fast tests

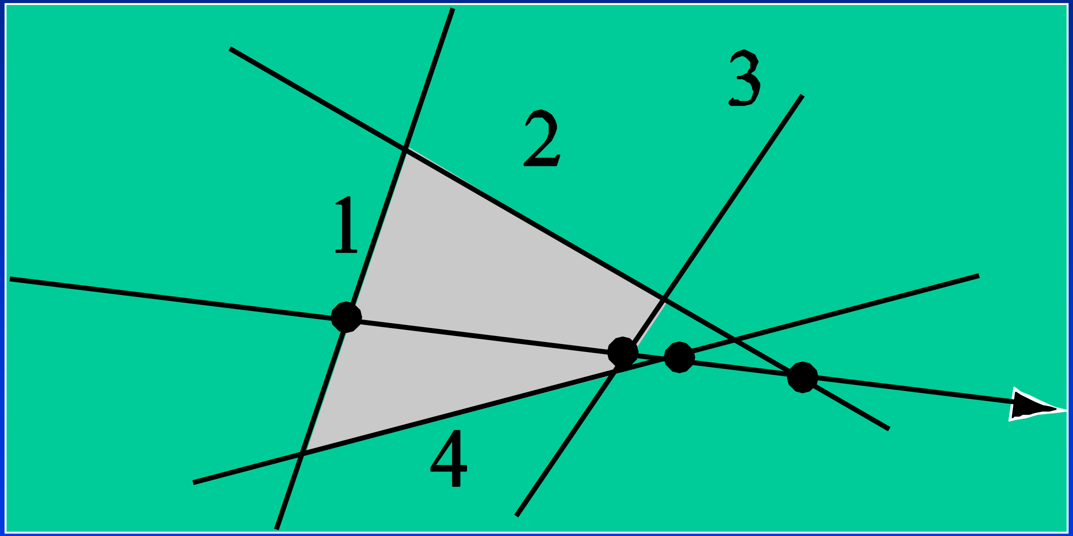
Ray Tracing Polyhedra

- If ray enters an object, it must enter a front facing polygon and leave a back facing polygon
- Polyhedron is formed by intersection of planes
- Ray enters at furthest intersection with front facing planes
- Ray leaves at closest intersection with back facing planes
- If entry is further away than exit, ray must miss the polyhedron

Ray Tracing Polyhedra

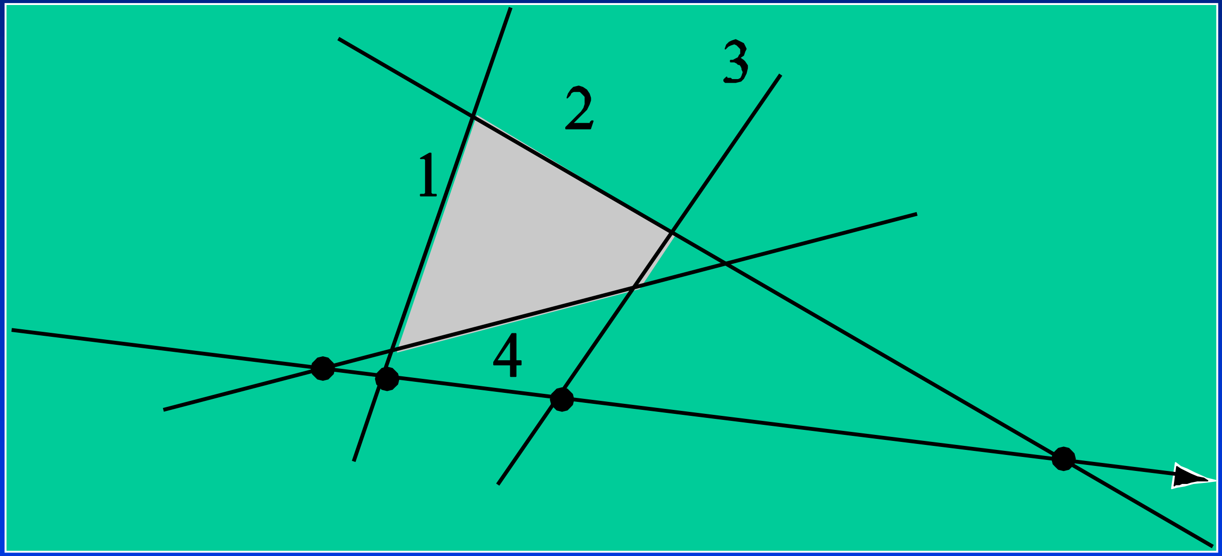


Ray Tracing a Polygon



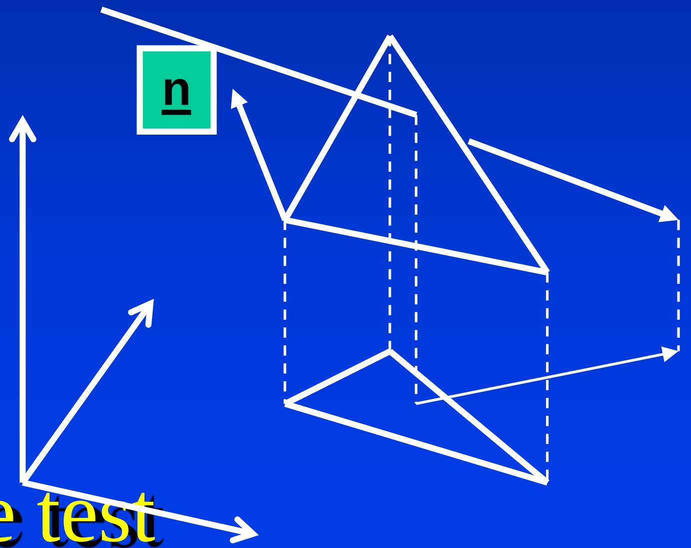
STONY BROOK
STATE UNIVERSITY OF NEW YORK

Ray Tracing a Polygon



Intersection Ray - Triangle

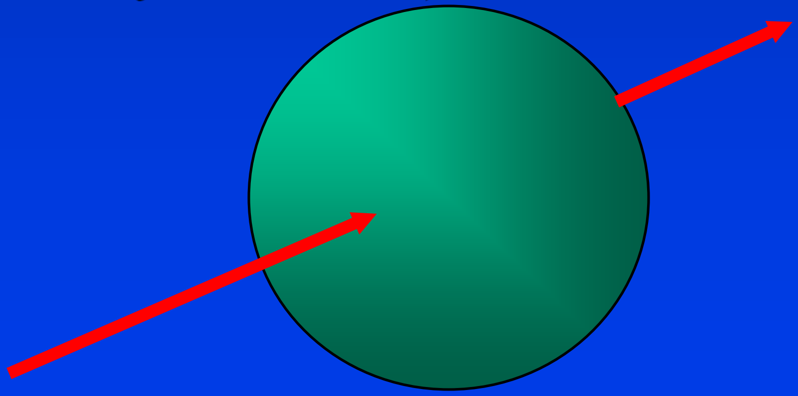
- Compute intersection with triangle plane
- Given the 3D intersection point
  - Project point into xy, xz, yz coordinate plane
  - Use coordinate plane that is most aligned
  - Coordinate plane and 2D vertices can be pre-computed



- Perform barycentric coordinate test

Ray Casting a Sphere

- Ray is parametric
- Sphere is quadric
- Resulting equation is a scalar quadratic equation which gives entry and exit points of ray (or no solution if ray misses)



Sphere

$$(\mathbf{p} - \mathbf{p}_c) \bullet (\mathbf{p} - \mathbf{p}_c) - r^2 = 0$$

$$\mathbf{p}(t) = \mathbf{p}_0 + t\,\mathbf{d}$$

$$\mathbf{p}_0 \bullet \mathbf{p}_0\, t^2 + 2\, \mathbf{p}_0 \bullet (\mathbf{d} - \mathbf{p}_0)\, t + (\mathbf{d} - \mathbf{p}_0) \bullet (\mathbf{d} - \mathbf{p}_0) - r^2 = 0$$

Ray Casting Quadrics

- Ray casting has become the standard way to visualize quadrics which are implicit surfaces in CSG systems
- Constructive Solid Geometry
  - Primitives are solids
  - Build objects with set operations
  - Union, intersection, set difference

Quadrics

General quadric can be written as

$$\mathbf{p}^T\mathbf{A}\mathbf{p} + b^T\mathbf{p} + c = 0$$

Substitute equation of ray

$$\mathbf{p}(t) = \mathbf{p}_0 + t\,\mathbf{d}$$

to get quadratic equation

Implicit Surfaces

Ray from $\mathbf{p}_0$ in direction $\mathbf{d}$

$$\mathbf{p}(t) = \mathbf{p}_0 + t\,\mathbf{d}$$

General implicit surface

$$f(\mathbf{p}) = 0$$

Solve scalar equation

$$f(\mathbf{p}(t)) = 0$$

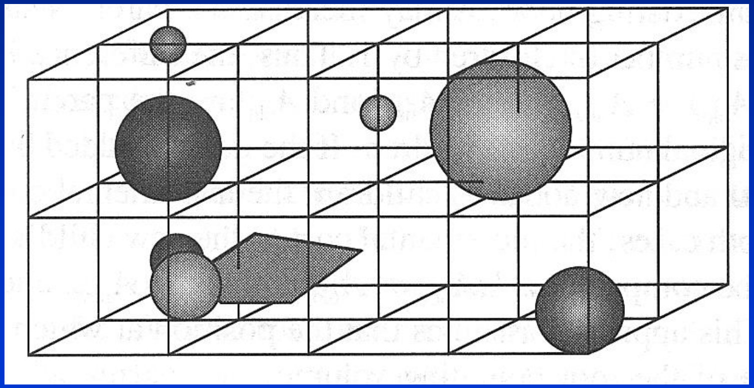
General case requires numerical methods

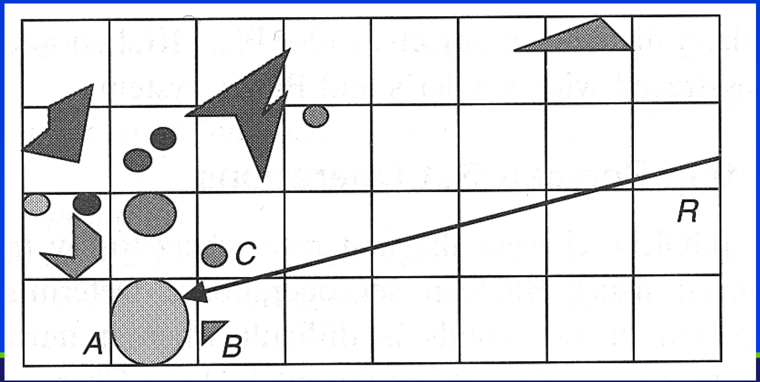
Ray Tracing Acceleration

- Intersect ray with all objects
  - Way too expensive
- Faster intersection algorithms
  - Little effect
- Less intersection computations
  - Space partitioning (often hierarchical)
    - Grid, octree, BSP or kd-tree, bounding volume hierarchy (BVH)
  - 5D partitioning (space and direction)

Spatial Partitioning: Grid Structure

- Building a grid structure
  - Start with bounding box
  - Resolution: often ~ $\sqrt[3]{n}$
  - Overlap or intersection test
- Traversal
  - 3D-DDA
  - Stop if intersection found in current voxel



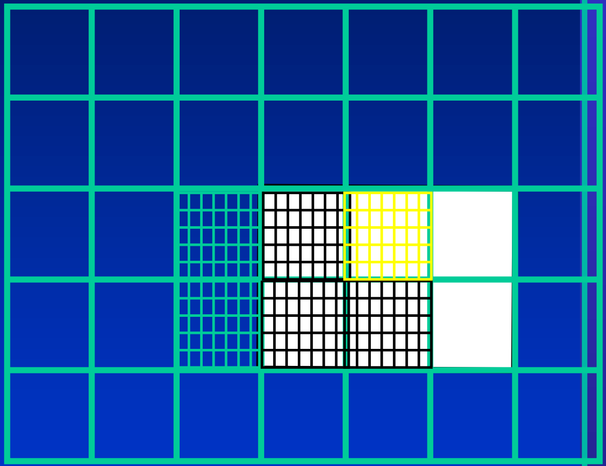


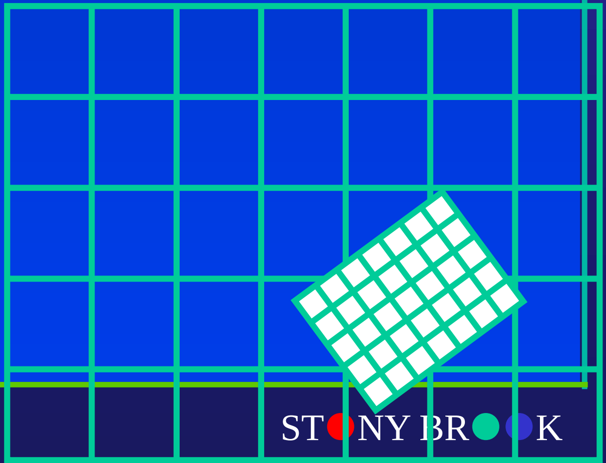
Grid: Issues

- Grid traversal
  - Requires enumeration of voxel along ray → 3D-DDA (Digital Differential Analyzer)
  - Simple and hardware-friendly
- Grid resolution
  - Strongly scene dependent
  - Cannot adapt to local density of objects
    - Problem: “Teapot in a stadium”
  - Possible solution: hierarchical grids

Hierarchical Grids

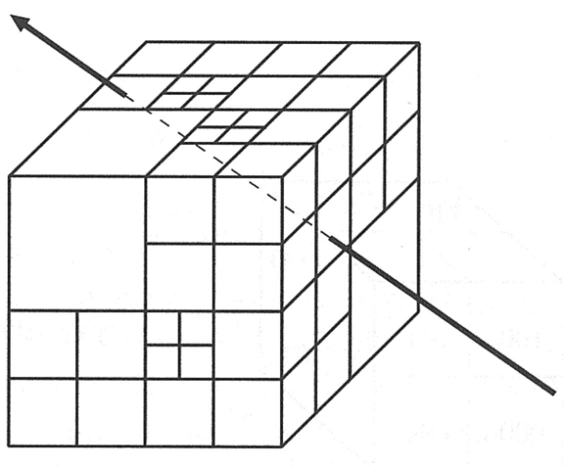
- Simple building algorithm
  – Recursively create grids in high-density voxels
  – Problem: What is the right resolution for each level?
- Advanced algorithm
  – Separate grids for object clusters
  – Problem: What are good clusters?





Octree

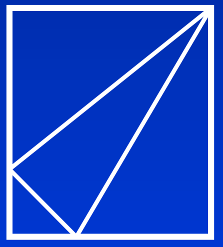
- Hierarchical space partitioning
  - Adaptively subdivide voxels into 8 equal sub-voxels recursively
  - Result in subdivision

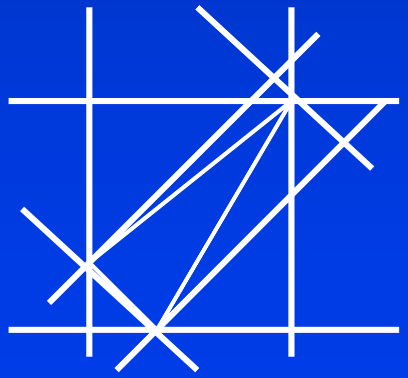


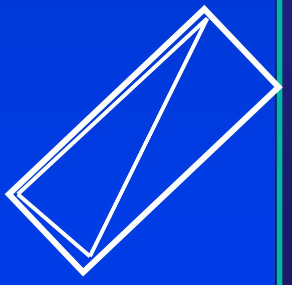
- Problems
  - Rather complex traversal algorithms
  - Slow to refine complex regions

Bounding Volumes

- Idea
  - Only compute intersection if ray hits BV
- Possible bounding volumes
  - Sphere
  - Axis-aligned box
  - Non-axis-aligned box
  - Slabs

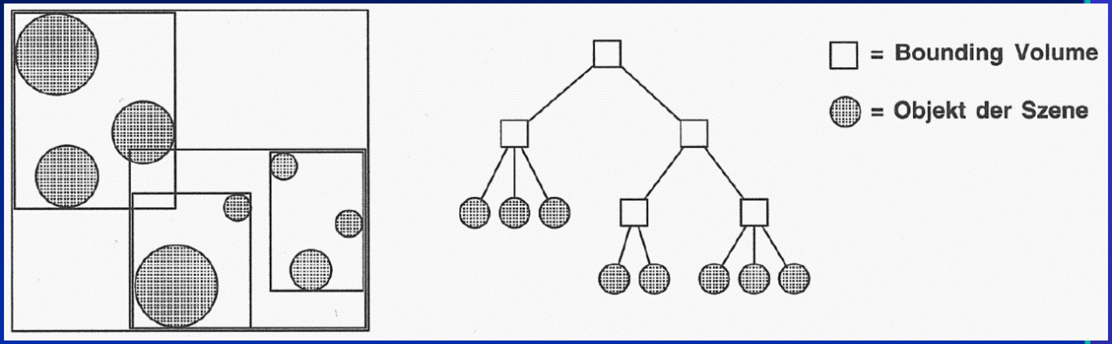






Bounding Volume Hierarchies

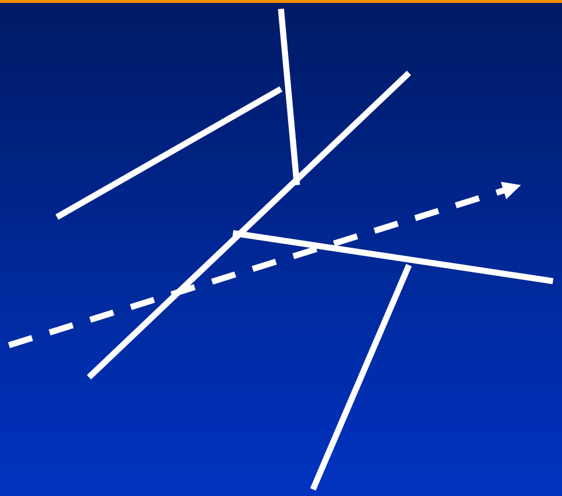
- Idea:
  – Apply recursively

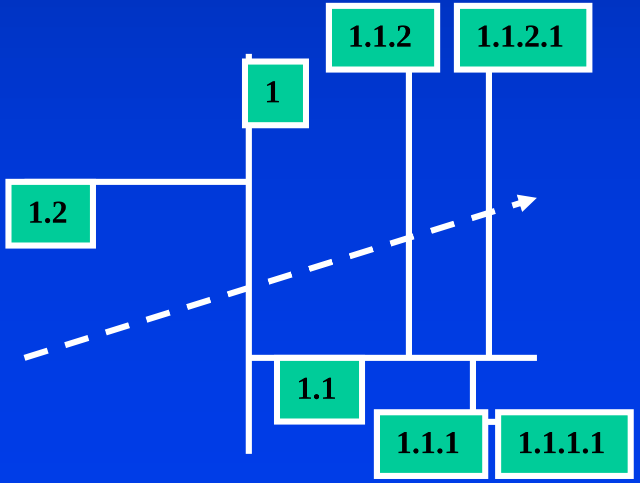


- Advantages:
  – Very good adaptivity
  – Efficient traversal O(log N)
- Problems
  – How to arrange Bounding volumes?

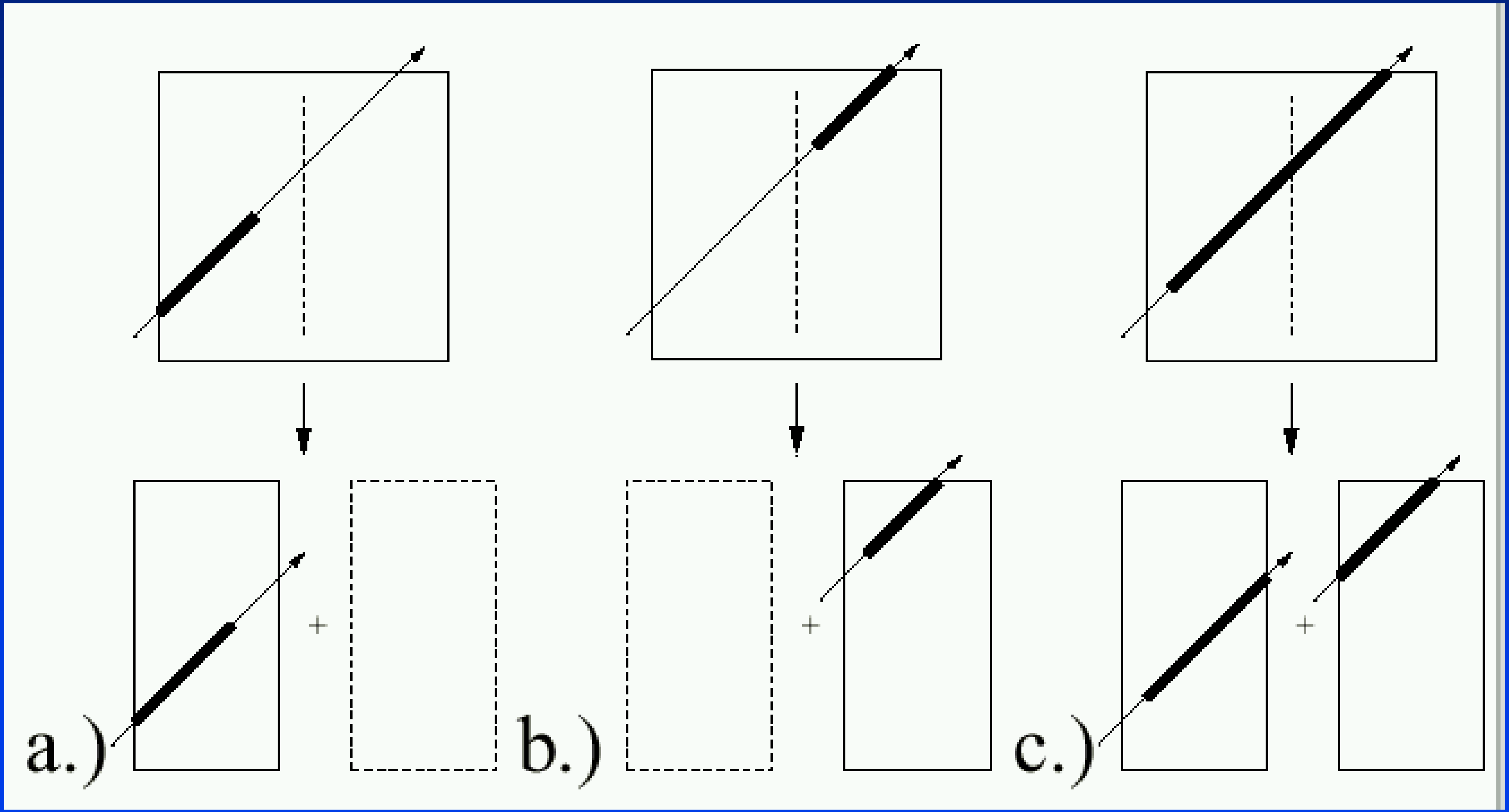
BSP- and Kd-Trees

- Recursive space partitioning with half-spaces
- Binary Space Partition (BSP):
  - Splitting with half-spaces in arbitrary position
- Kd-Tree
  - Splitting with axis-aligned half-spaces





Kd-Tree Traversal



History of Intersection Algorithms

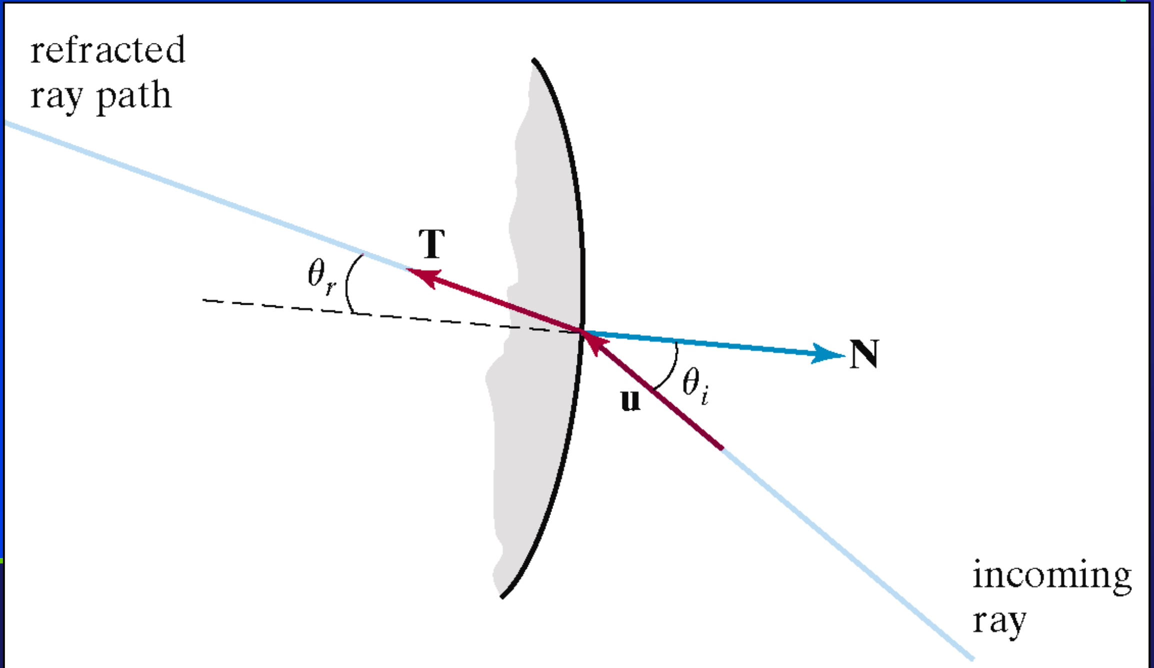
- Ray-geometry intersection algorithms

– Polygons:	[Appel '68]
– Quadrics, CSG:	[Goldstein & Nagel '71]
– Recursive Ray Tracing:	[Whitted '79]
– Tori:	[Roth '82]
– Bicubic patches:	[Whitted '80, Kajiya '82, Benthin '04]
– Algebraic surfaces:	[Hanrahan '82]
– Swept surfaces:	[Kajiya '83, van Wijk '84]
– Fractals:	[Kajiya '83]
– Deformations:	[Barr '86]
– NURBS:	[Stürzlinger '98]
– Subdivision surfaces:	[Kobbelt et al '98, Benthin '04]
– Points	[Schaufler et al.'00, Wald '05]

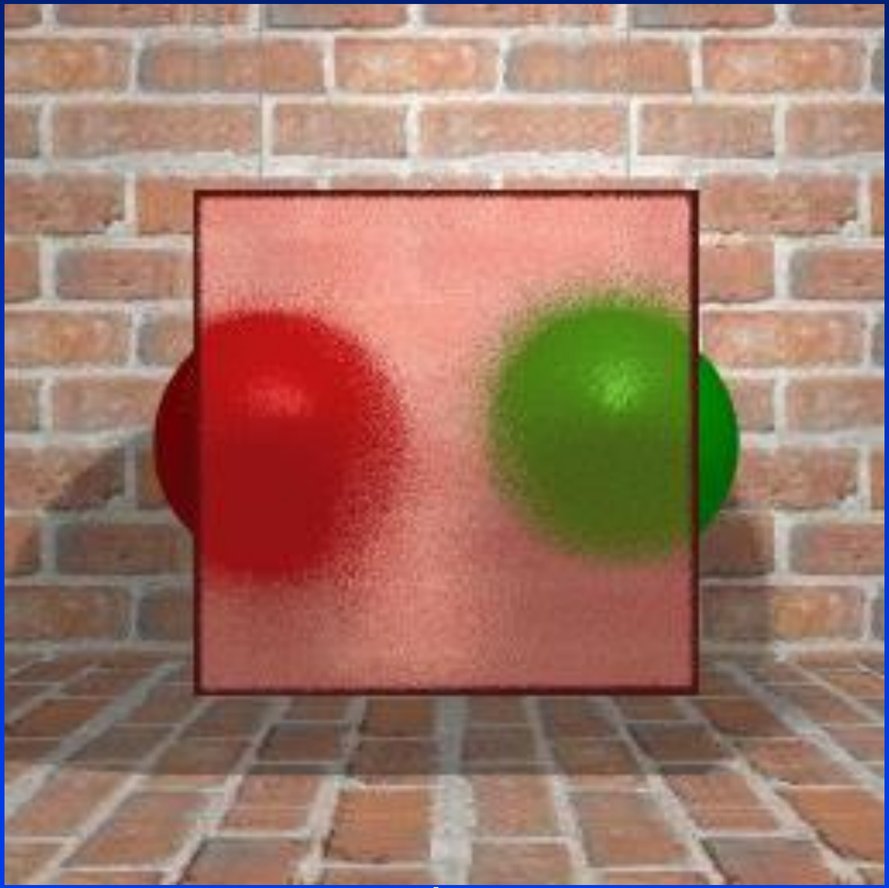
Other Visual Effects

Ray-Tracing & Transparent Surfaces

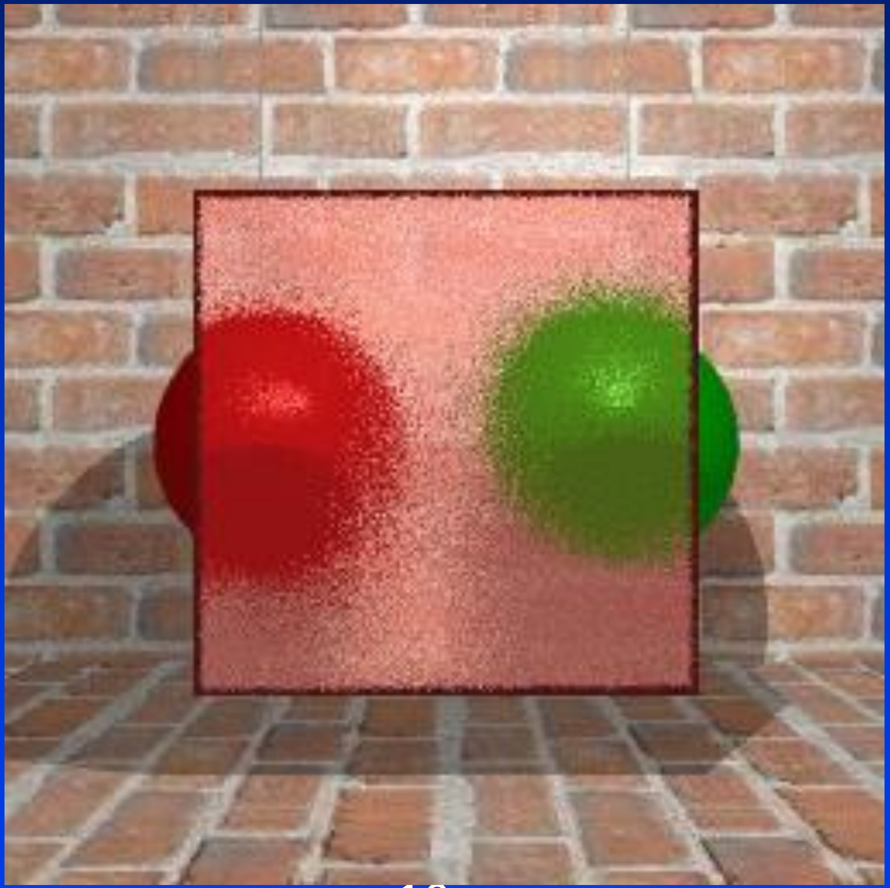
- For transparent surfaces we need to calculate a ray to represent the light refracted through the material
- The direction of the refracted ray is determined by the refractive index of the material



Transparency



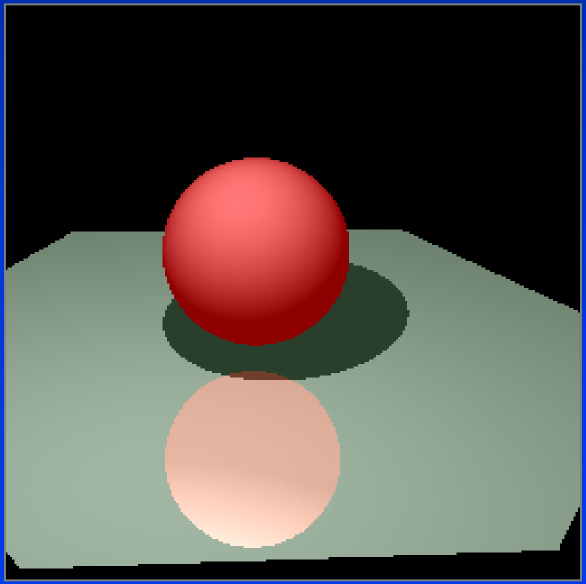
4 rays

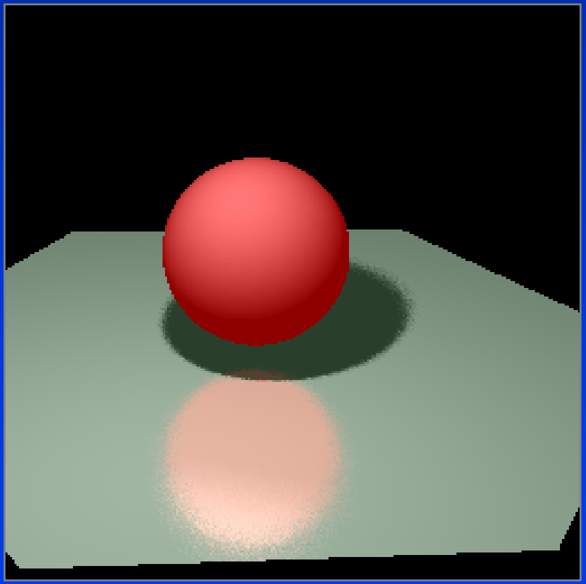


16 rays

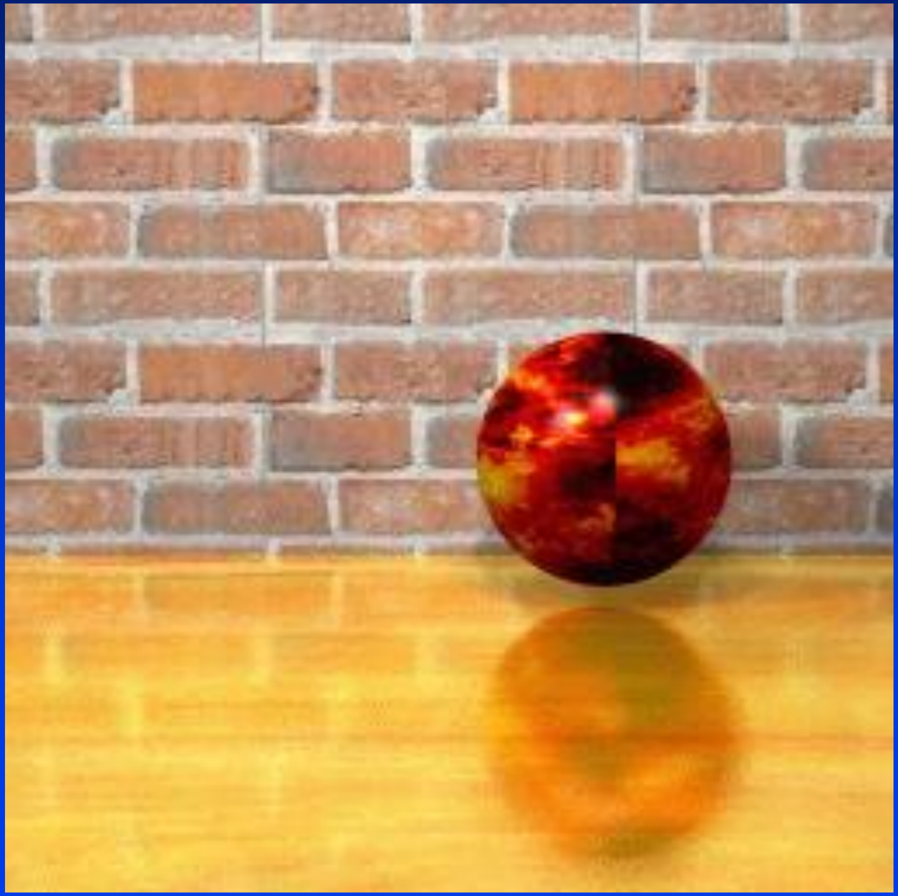
Gloss/Translucency

- Blurry reflections and transmissions are produced by randomly perturbing the reflection and transmission rays from their "true" directions.

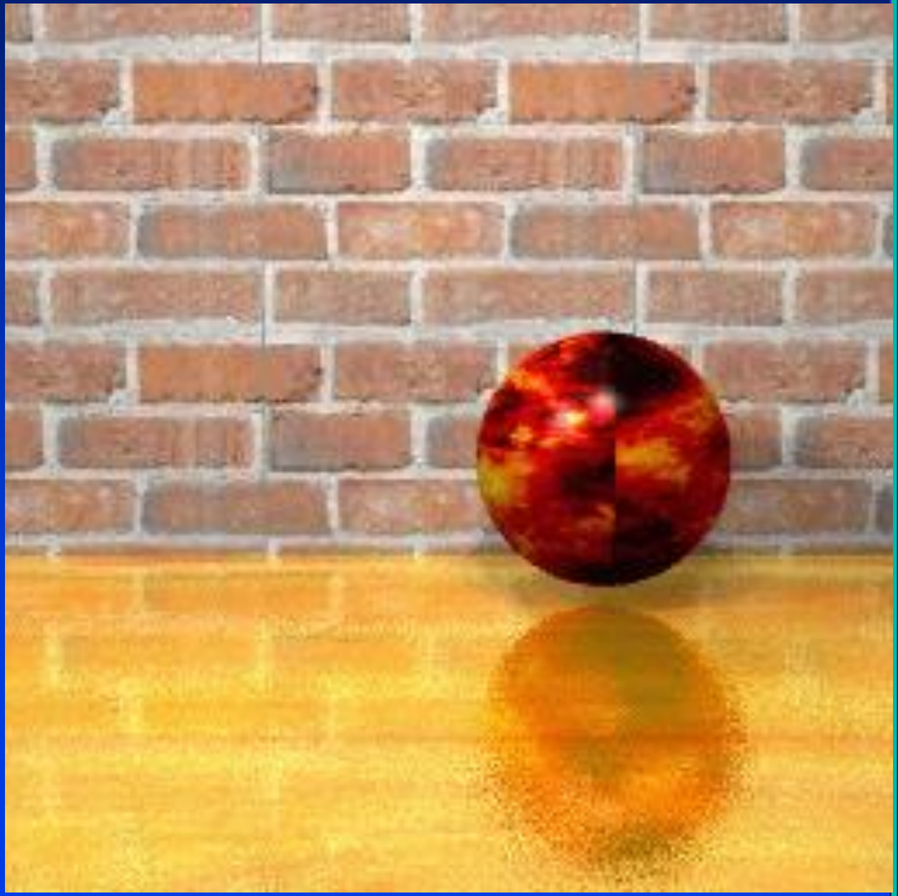




Reflection

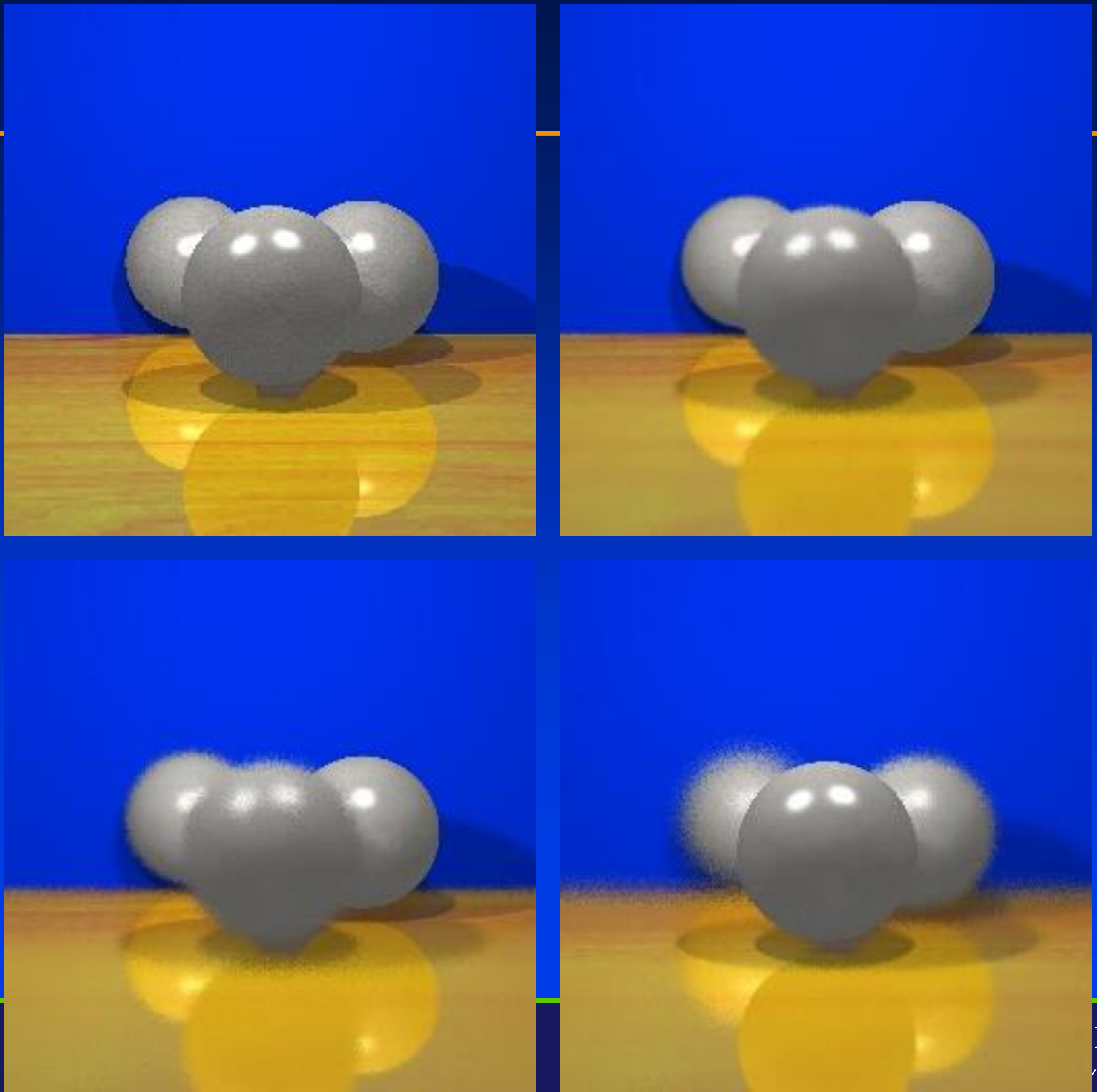


4 rays



64 rays

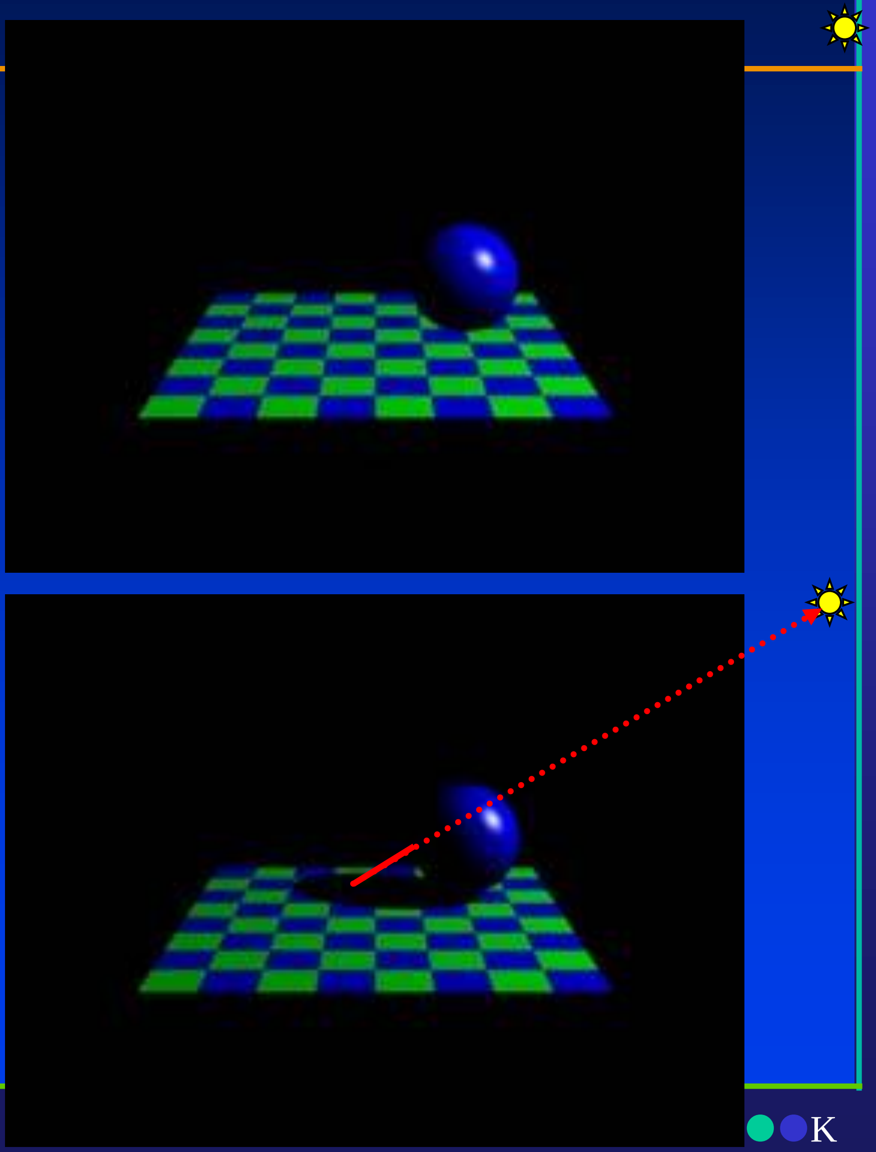
Depth of Field



The Shadow Ray

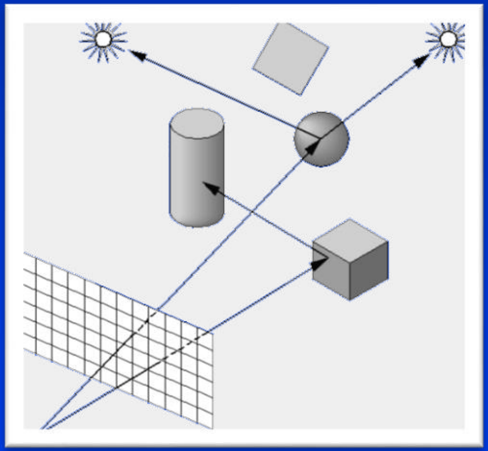
- The path from the intersection to the light source is known as the **shadow ray**
- If any object intersects the shadow ray between the surface and the light source then the surface is in shadow with respect to that source

Shadow Ray

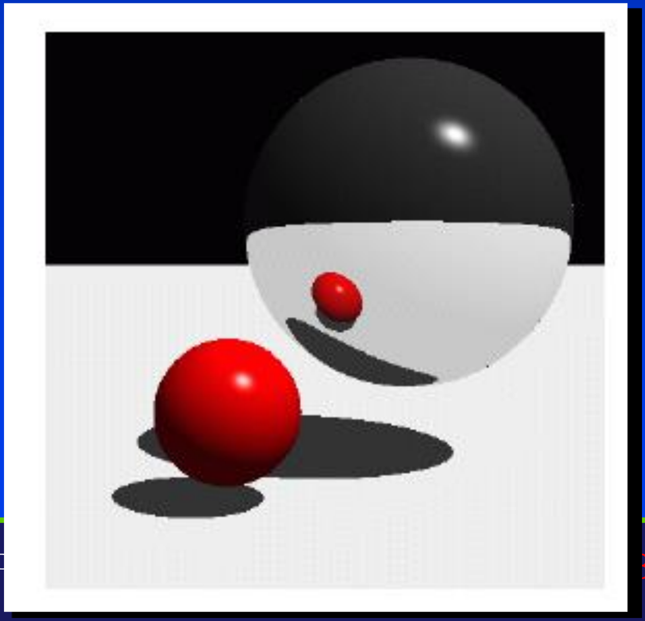


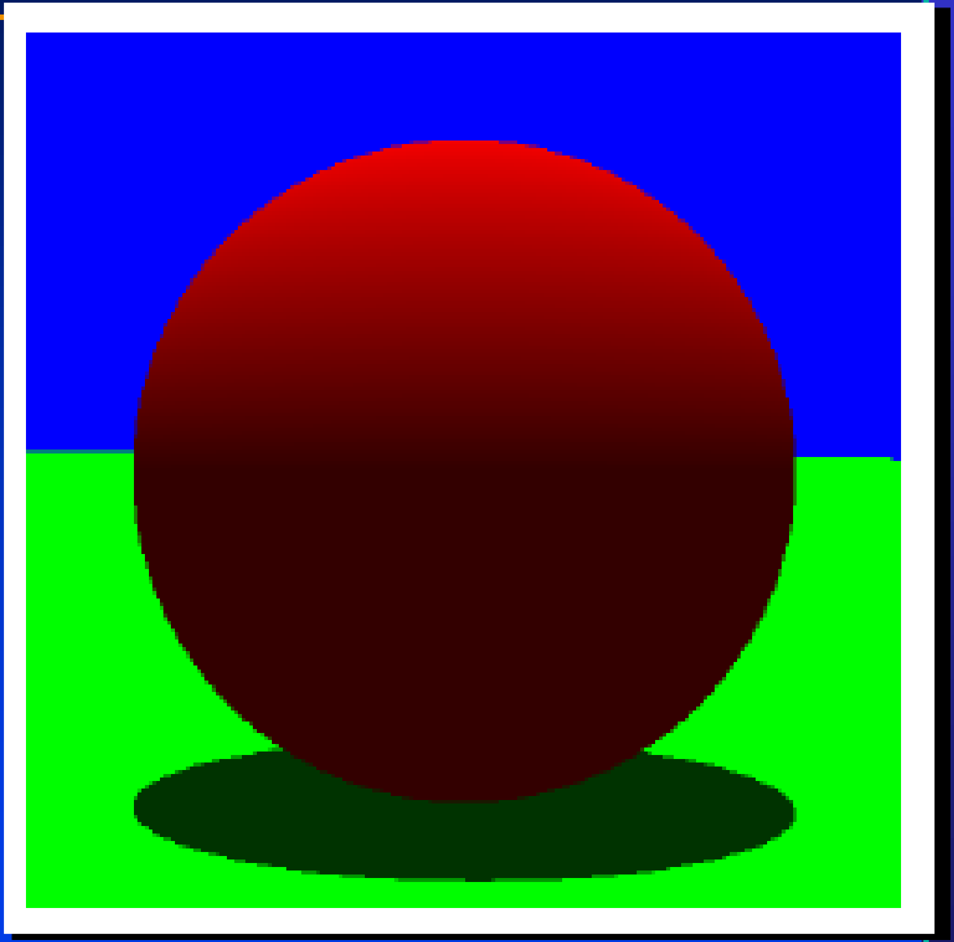
Shadow Rays

- Even if a point is visible, it will not be lit unless we can see a light source from that point
- Cast shadow rays

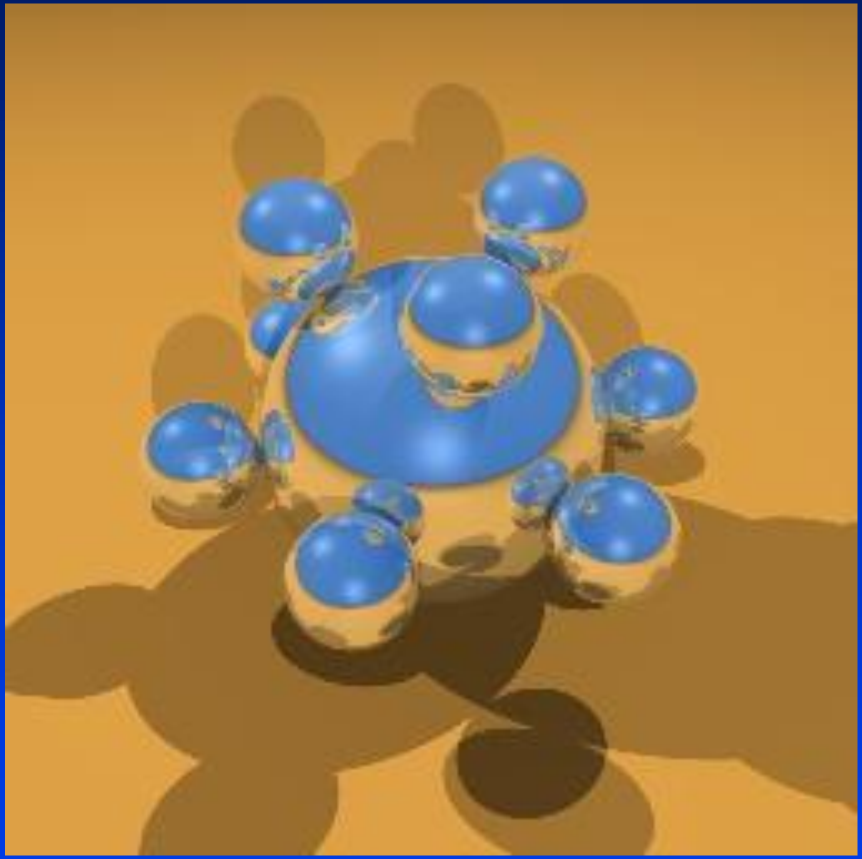


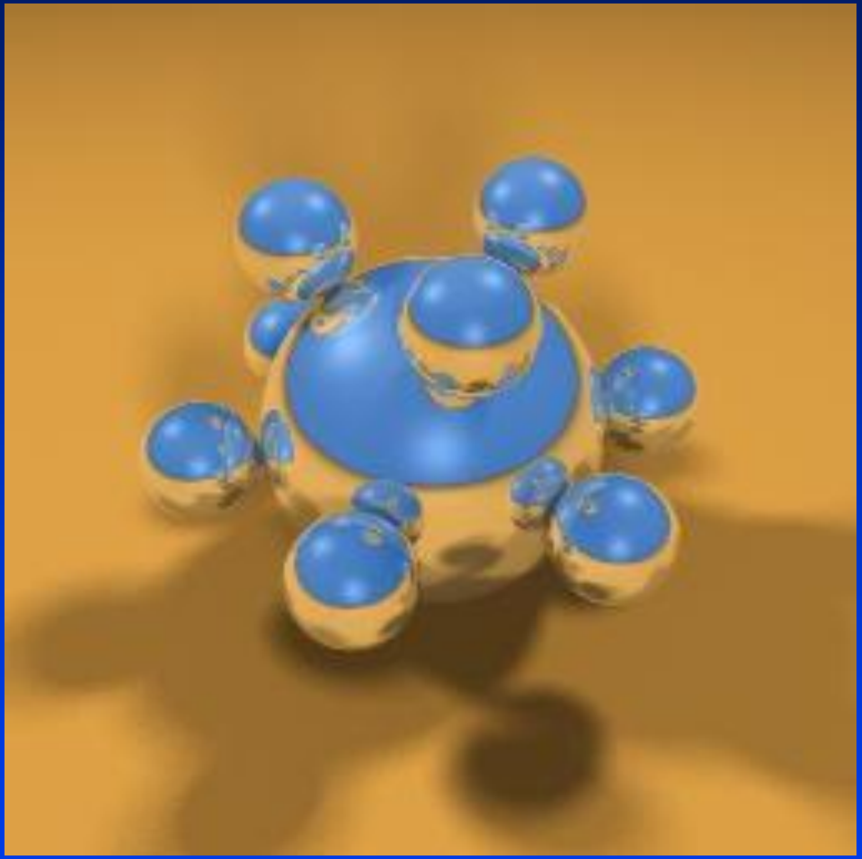
More Examples on Shadow

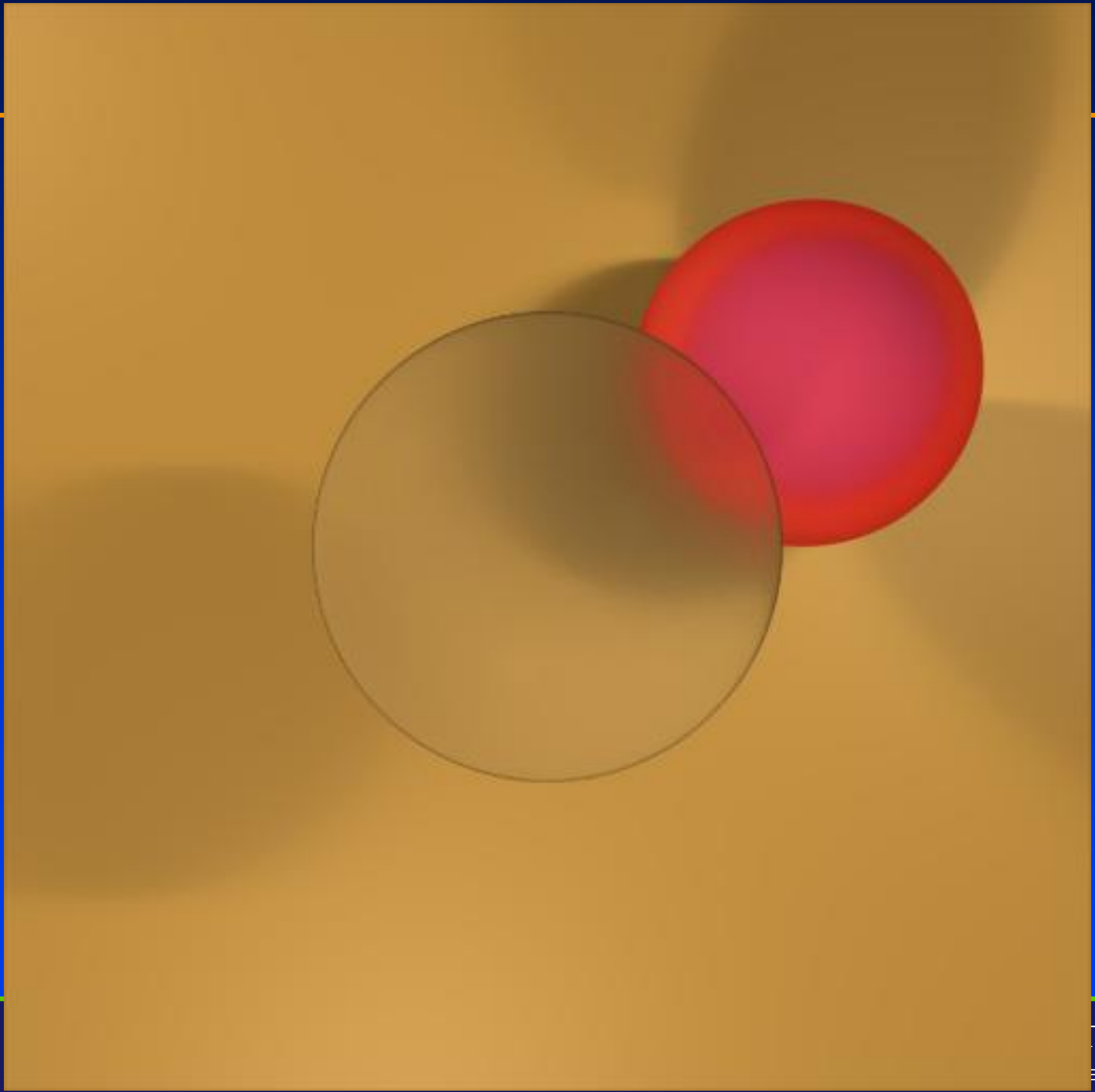




Shadow Examples







Shadow Examples

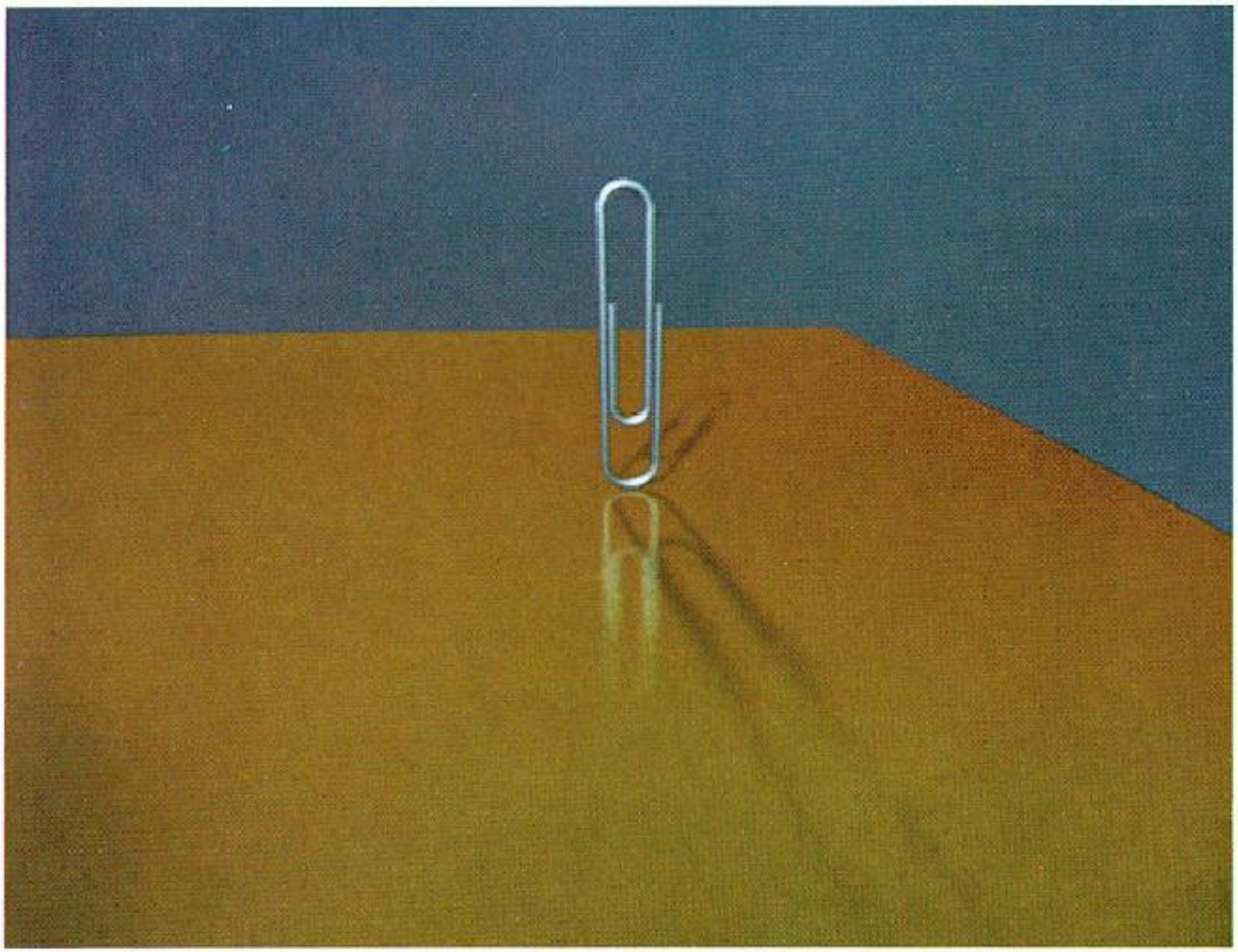
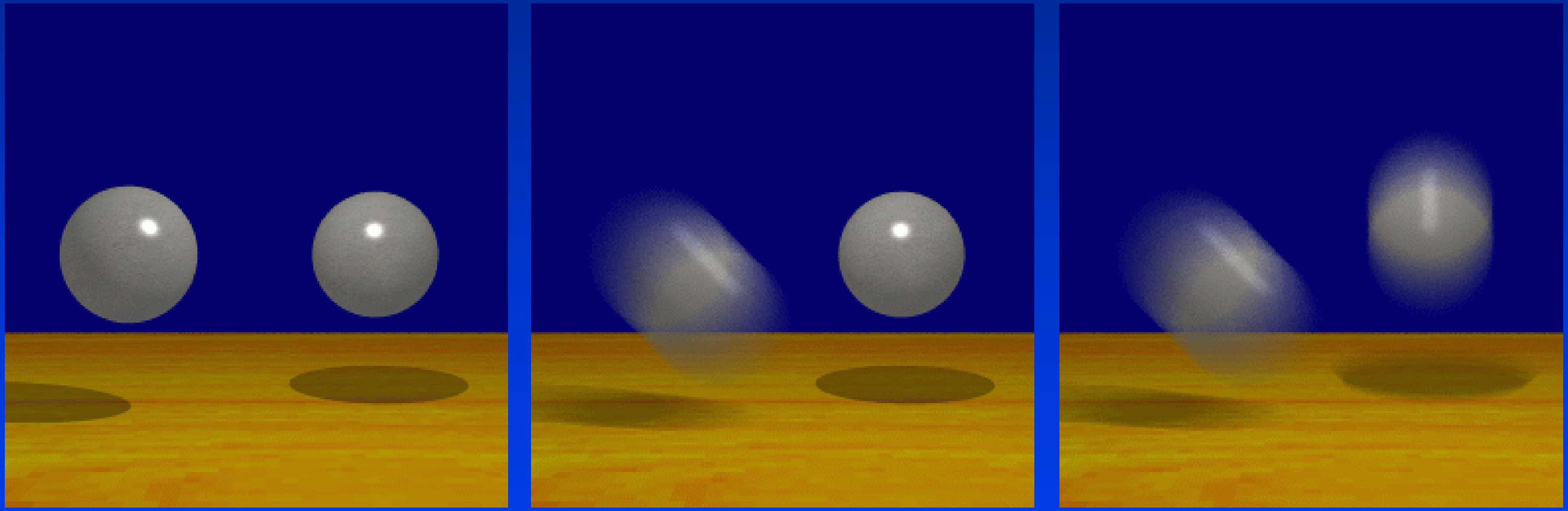
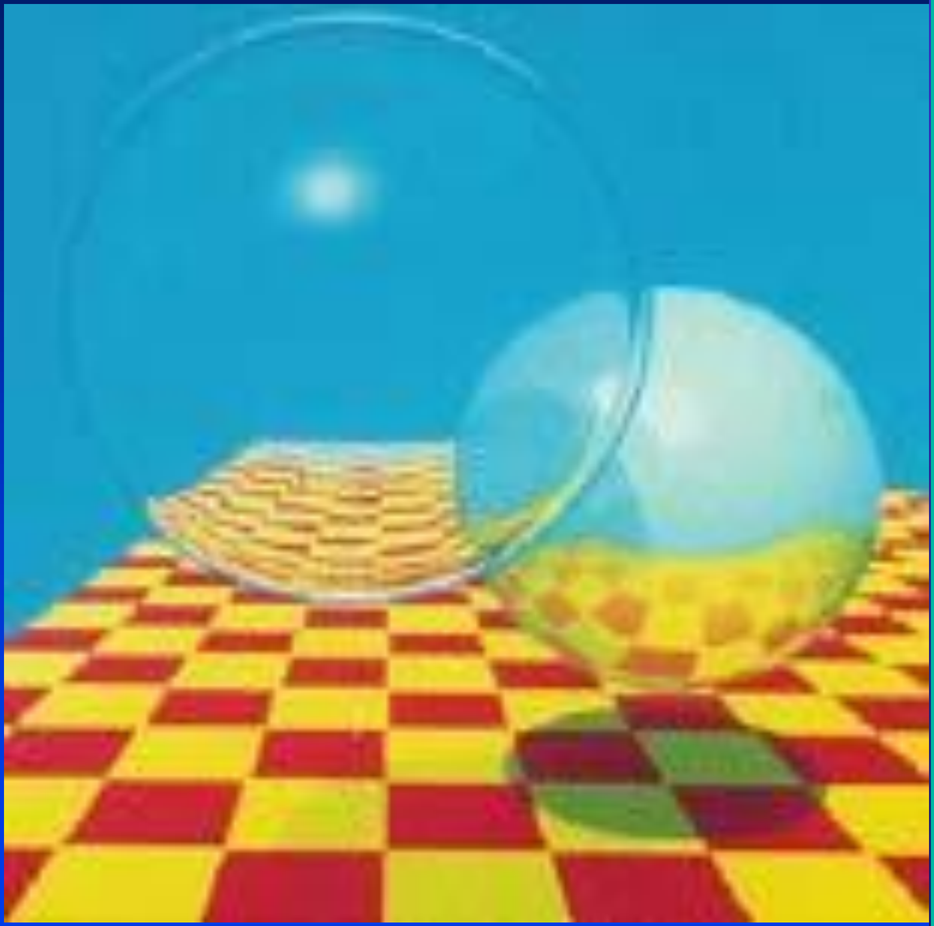


Fig. 17. Example of penumbrae and blurry reflection.

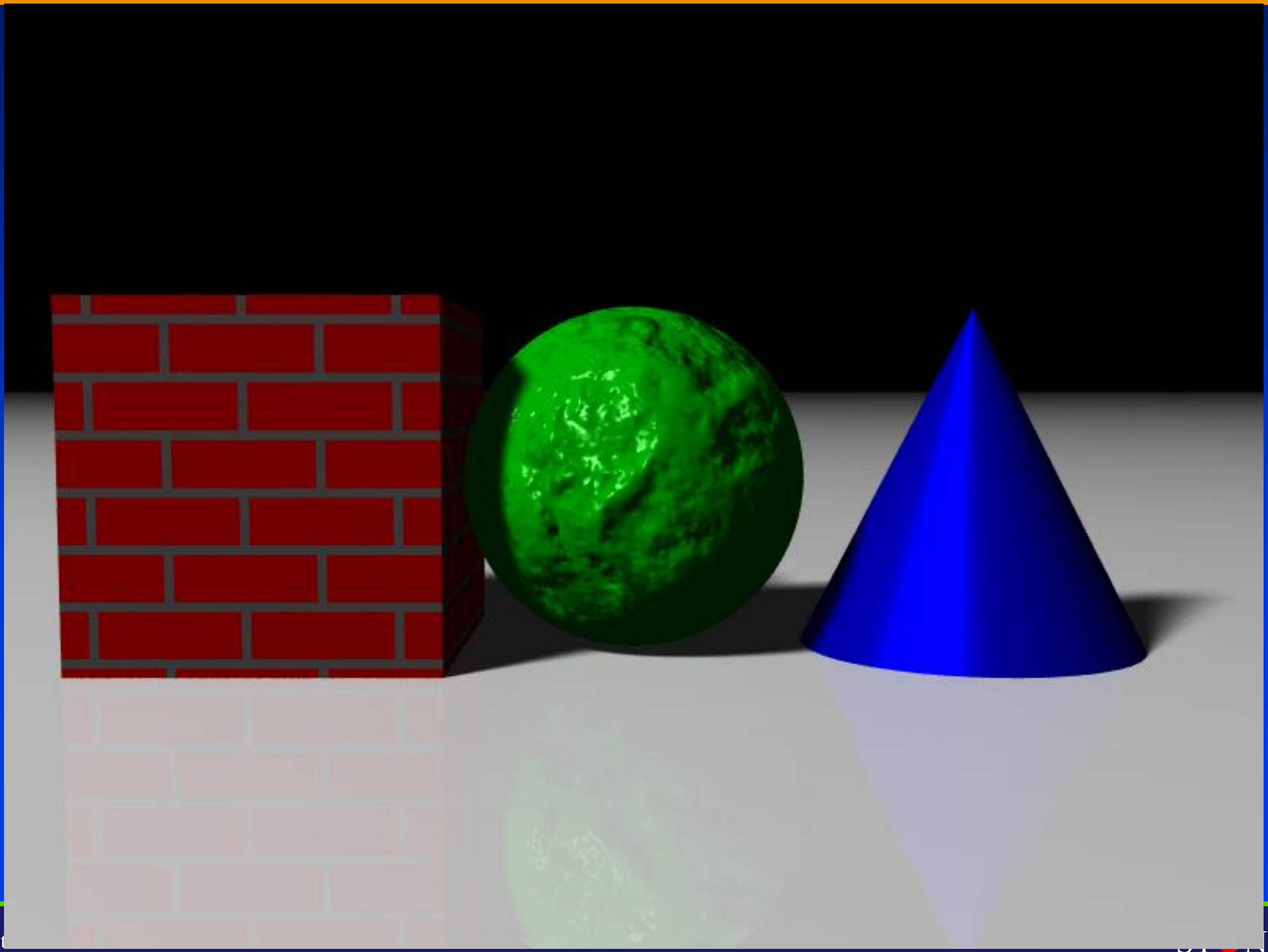
Motion Blurring



Ray Tracing



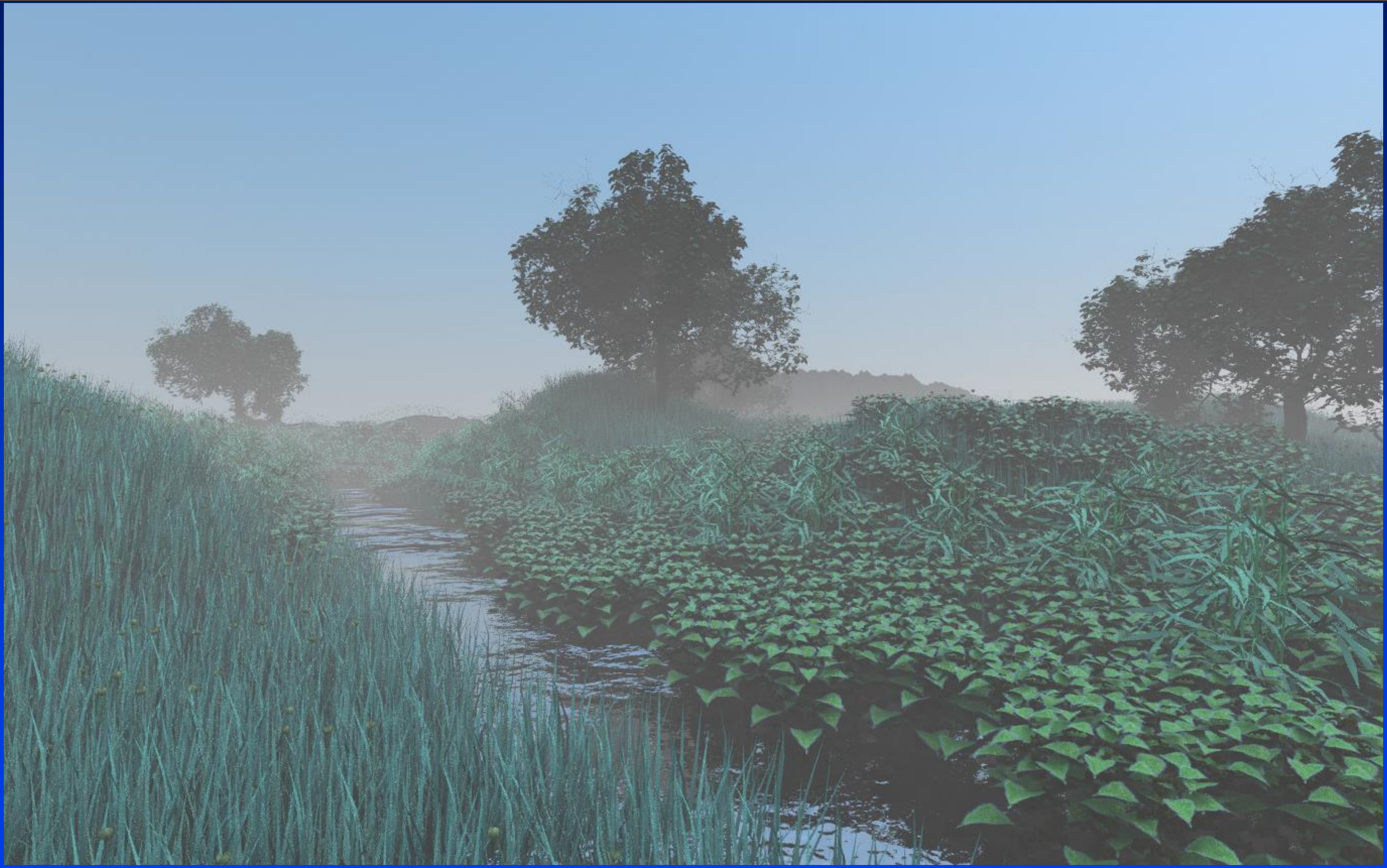
POV-Ray



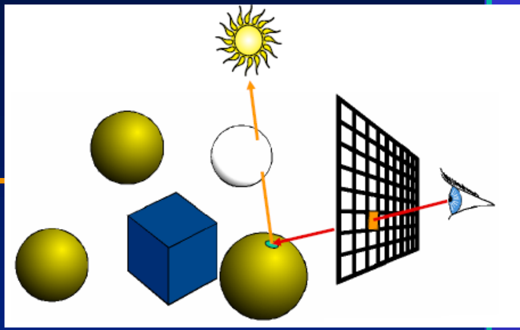
Global Illumination

Global Illumination

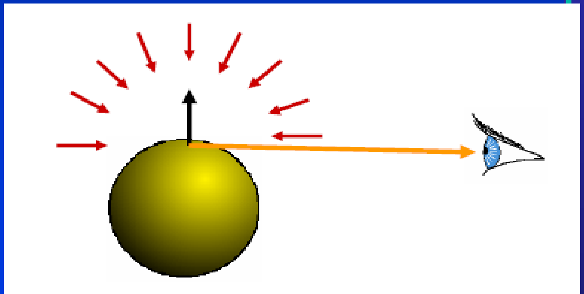
Fog



Summary



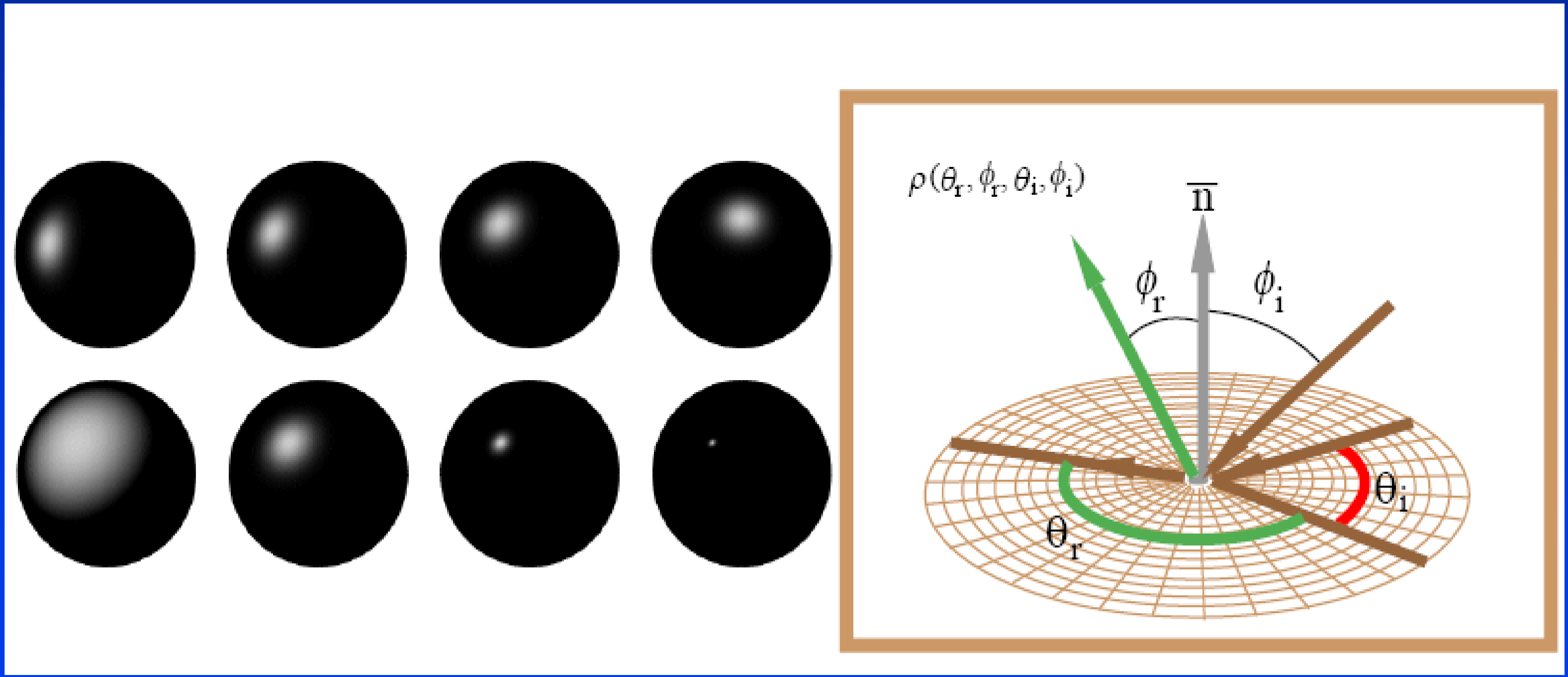
- Does Ray Tracing simulate Physics?
- Ray Tracing is full of (graphics) tricks
  - For example, shadows of transparent objects
    - Possible solutions: opaque, multiply by transparency color, then no refraction at all



- The rendering equation
  - Physics-correct
  - Math. Framework for light-transport simulation
  - Outgoing light in one direction is the integral of incoming light in all directions multiplied by reflectance property

Summary

- Reflectance properties, shading, and BRDF



Questions?