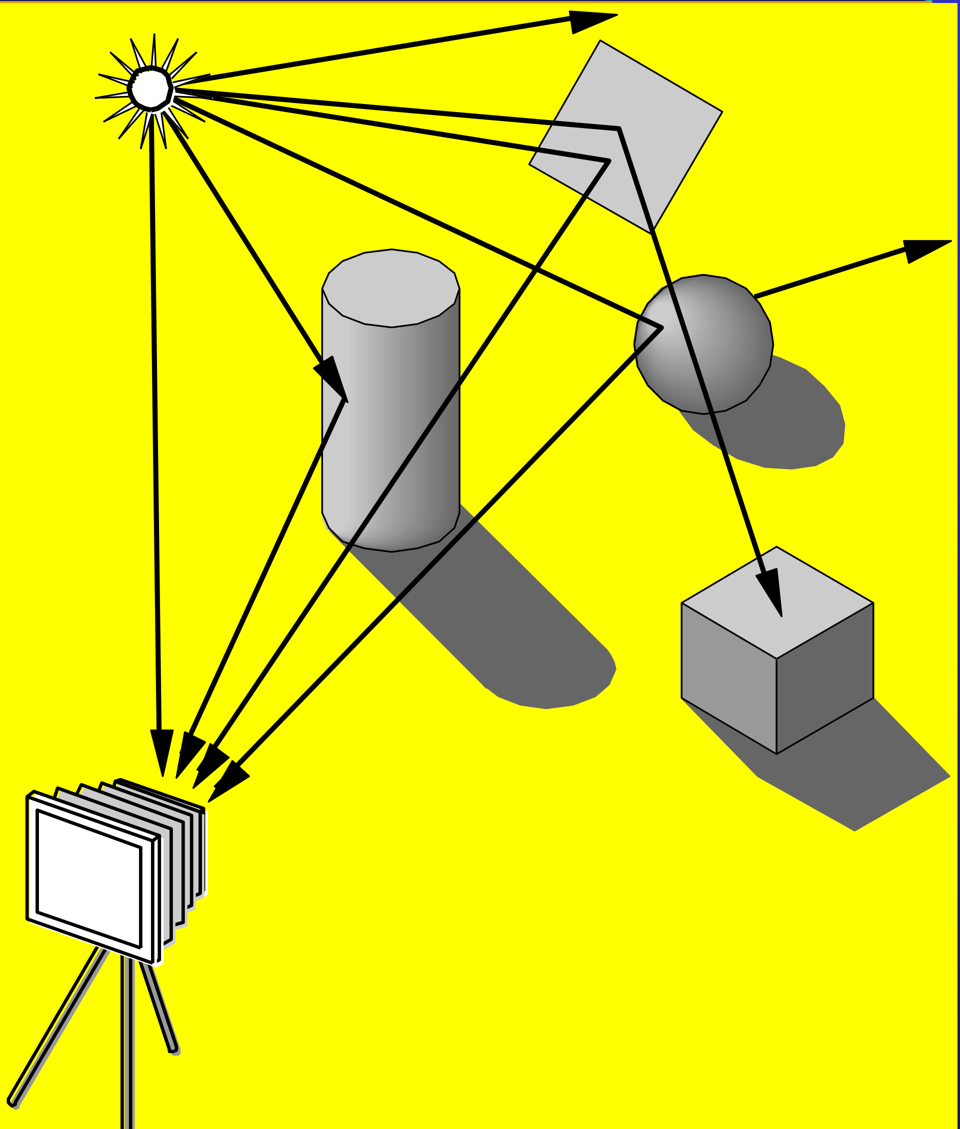
Global Approaches

- Ray Tracing
- Radiosity
- Rendering equation

Photo-realistic Rendering

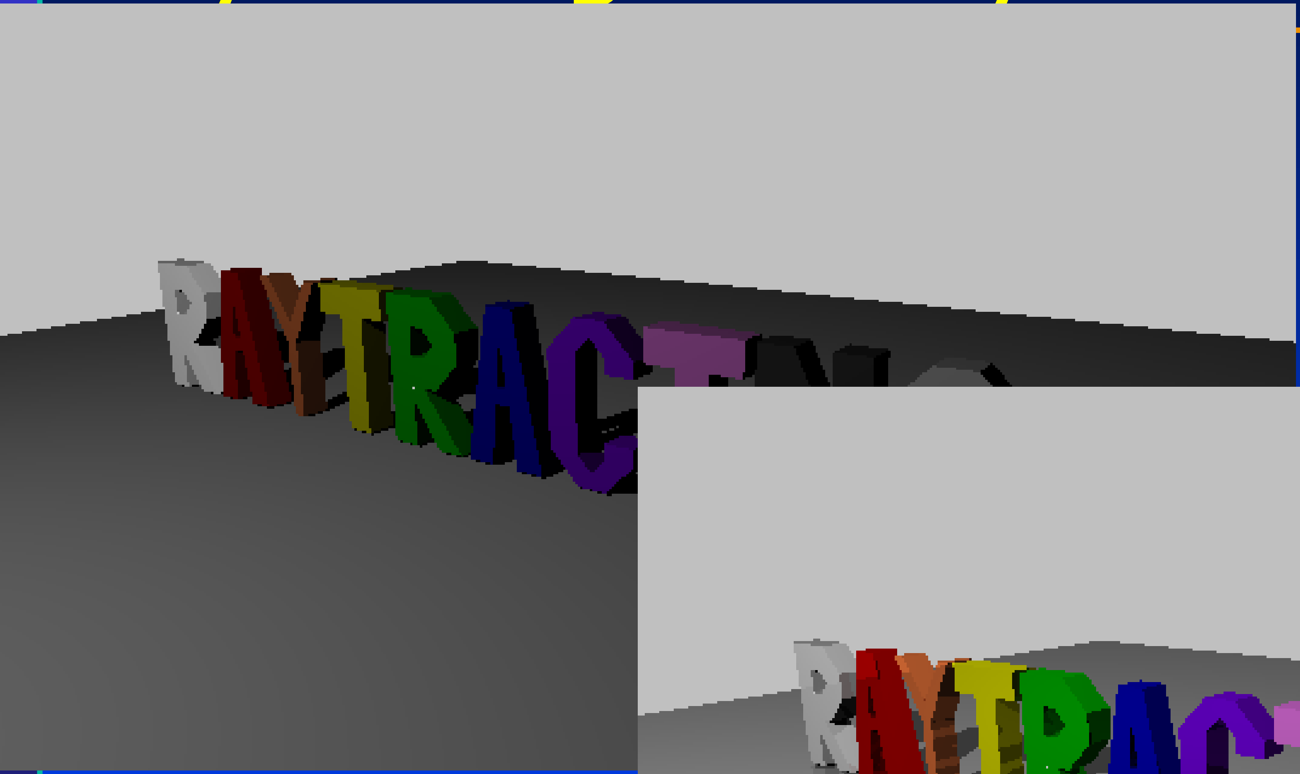
- Simple forward approach: Follow light rays from a point light source
- Can account for reflection and transmission (refraction) during ray transmission from a light source to image plane

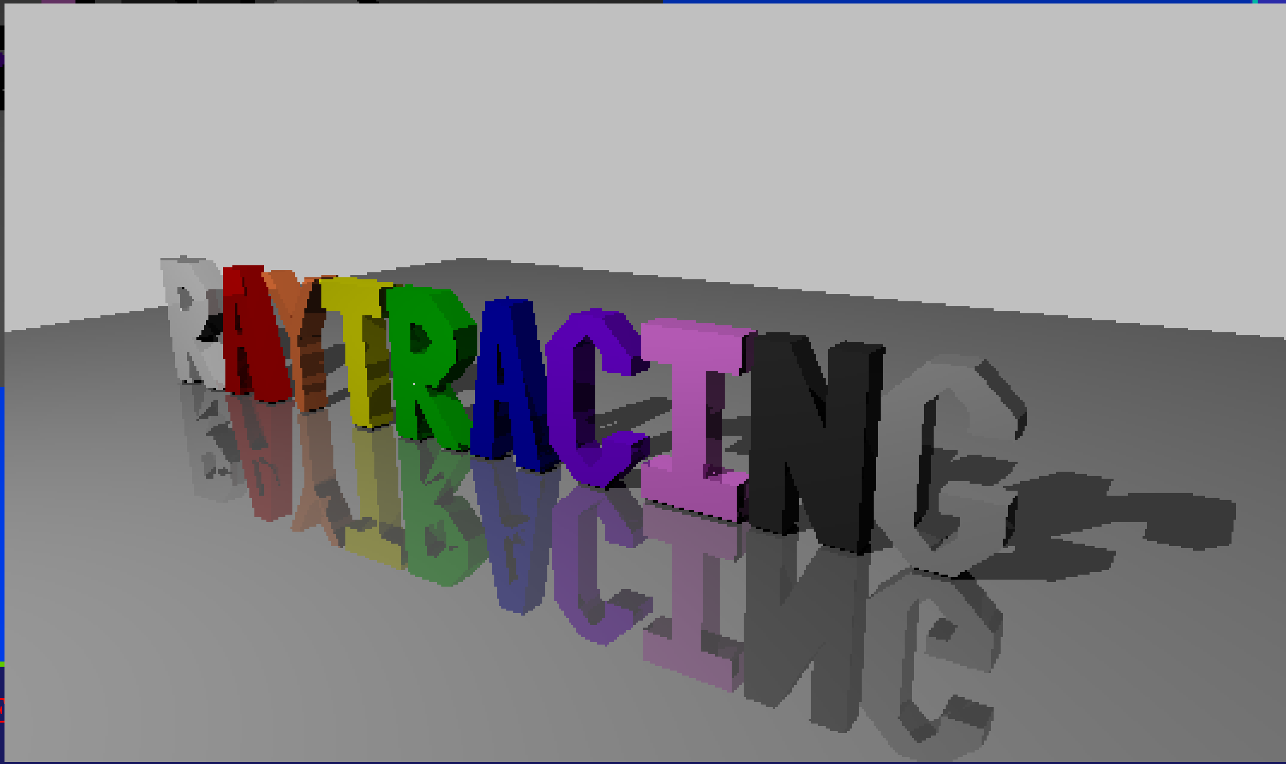


Computation

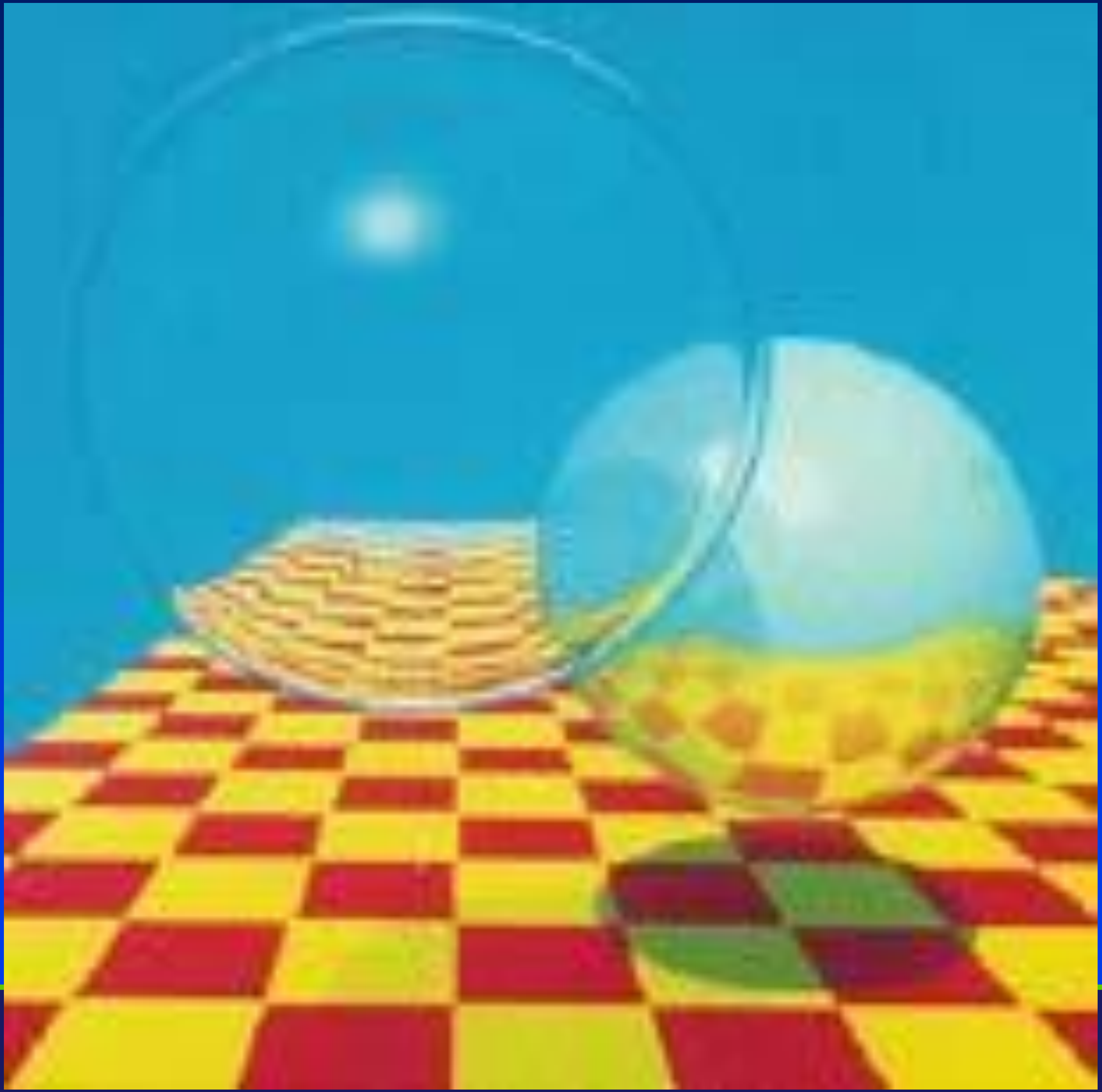
- Should be able to handle all physical interactions between objects and light rays
- Unfortunately, the direct, forward paradigm is not computational tractable at all
- Most rays do not affect what we see on the image plane, because those rays do not penetrate through the image plane at all
- Scattering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

Raycasting vs. Ray Tracing





Ray Tracing



Ray Tracing

Ray-Tracing Examples

Global Illumination

Global Illumination

Global Illumination

- Lighting based on the full scene
- Lighting based on physics
- Traditionally represented by two algorithms
  - Ray Tracing – 1980
  - Radiosity – 1984
- More modern techniques include *photon mapping* and many variations of raytracing and radiosity ideas

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 Examples

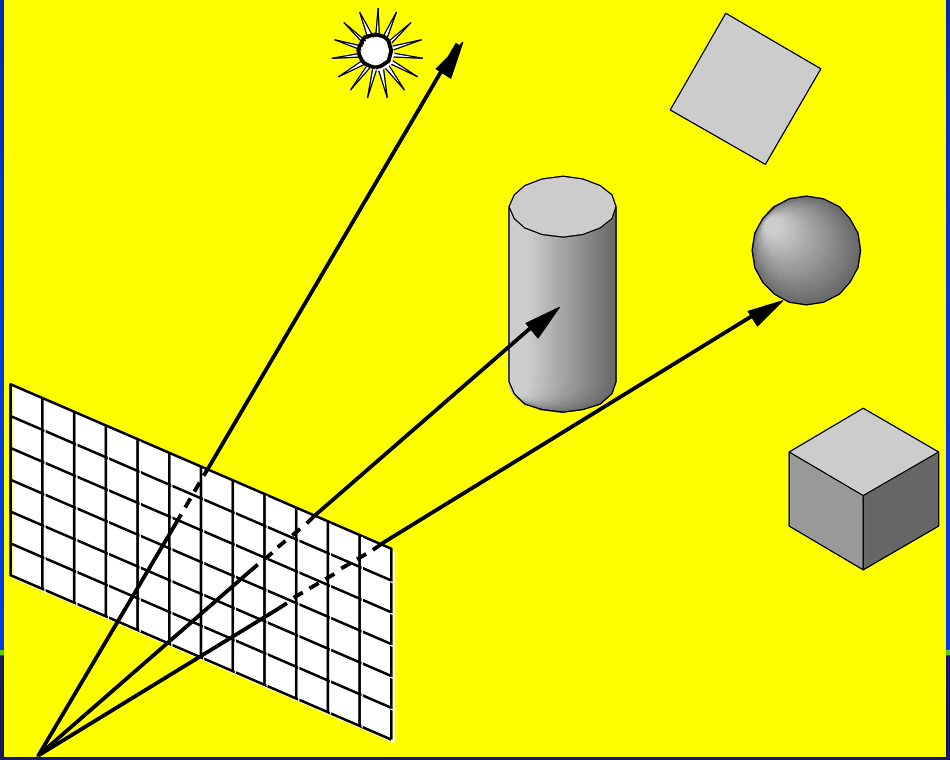
Ray-Tracing Examples

Ray-Tracing Examples

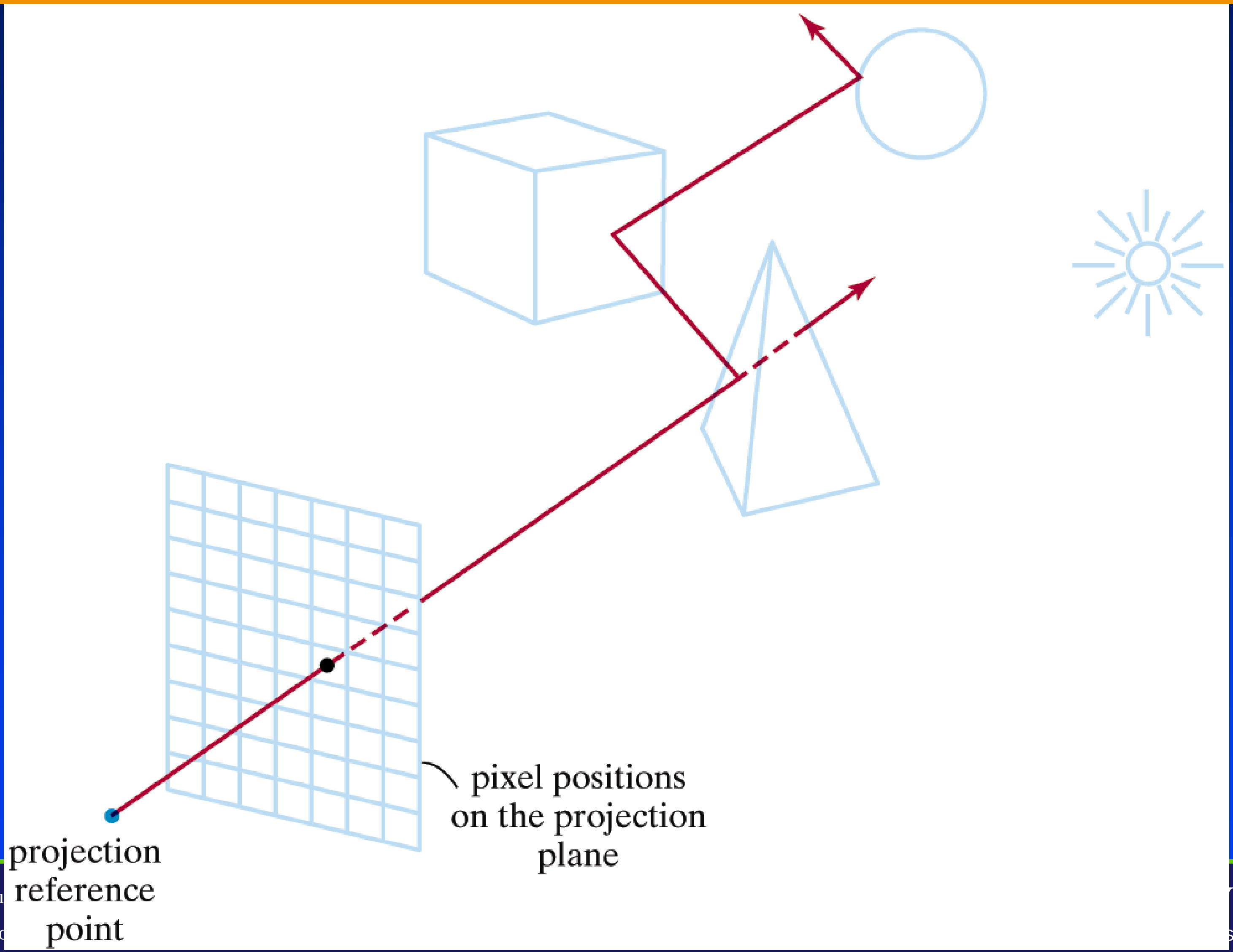
Ray-Tracing Examples

Ray Casting: Basic Principle

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



Ray-Tracing: Basic Principles



Ray Tracing

- Highly realistic images
  - Ray tracing enables correct simulation of light transport

Internet Ray Tracing Competition, June 2002

Ray Tracing

- 3D image rendering
- Calculate the paths of light rays
- Nice-looking reflections, refractions, shadows

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: Basic Setup

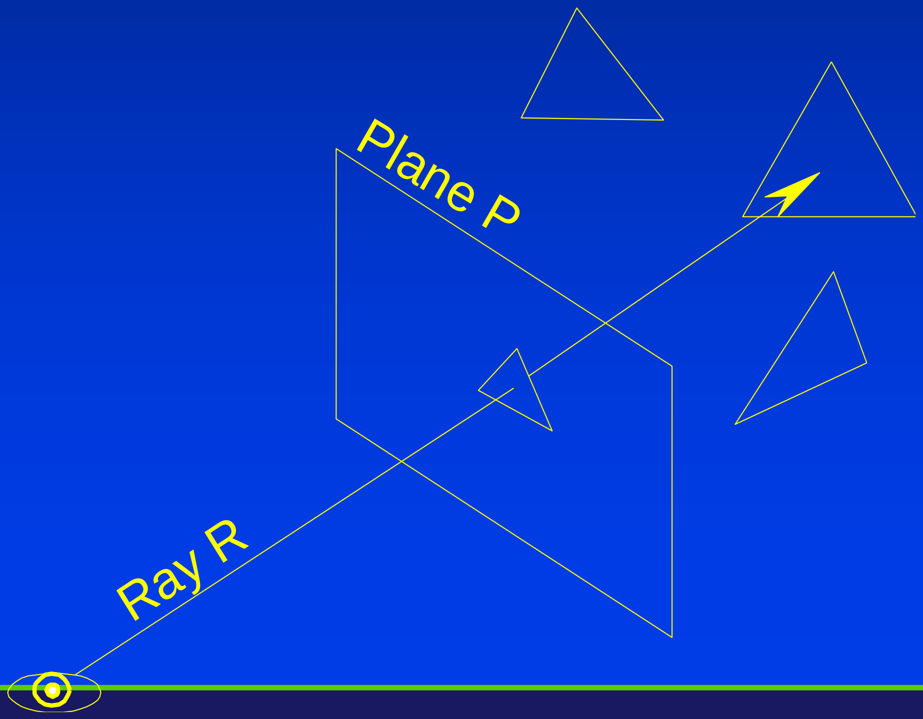
- Assumption: empty space totally transparent
- Surfaces (geometric objects)
  - 3D geometric models of objects
- Optical surface characteristics (appearance)
  - Absorption, reflection, transparency, color, …
- Illumination
  - Position, characteristics of light sources

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

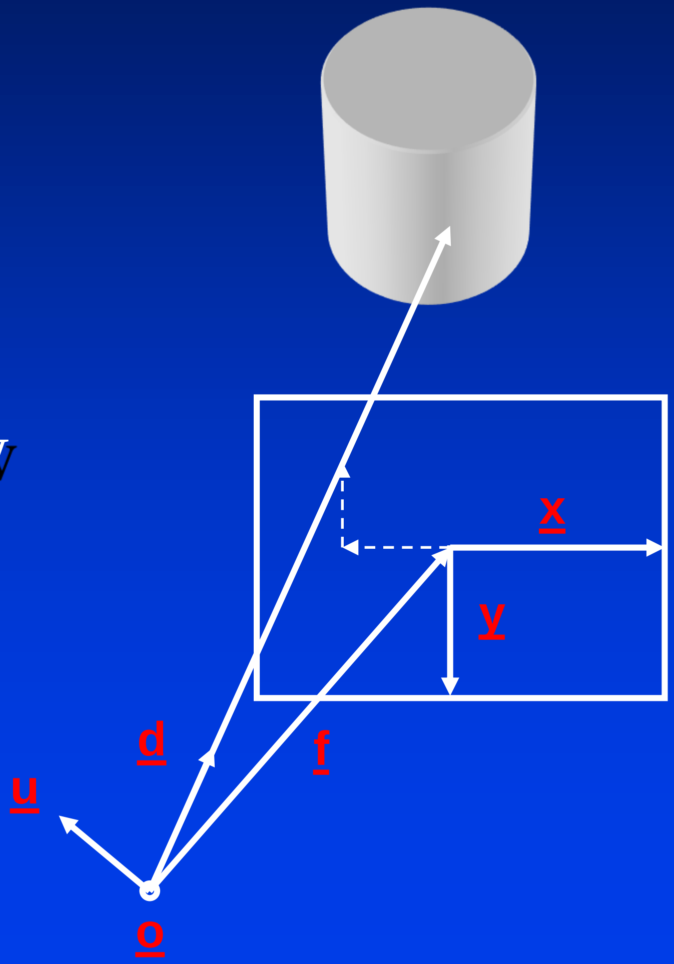
Ray Tracing Algorithm: First Step

- 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



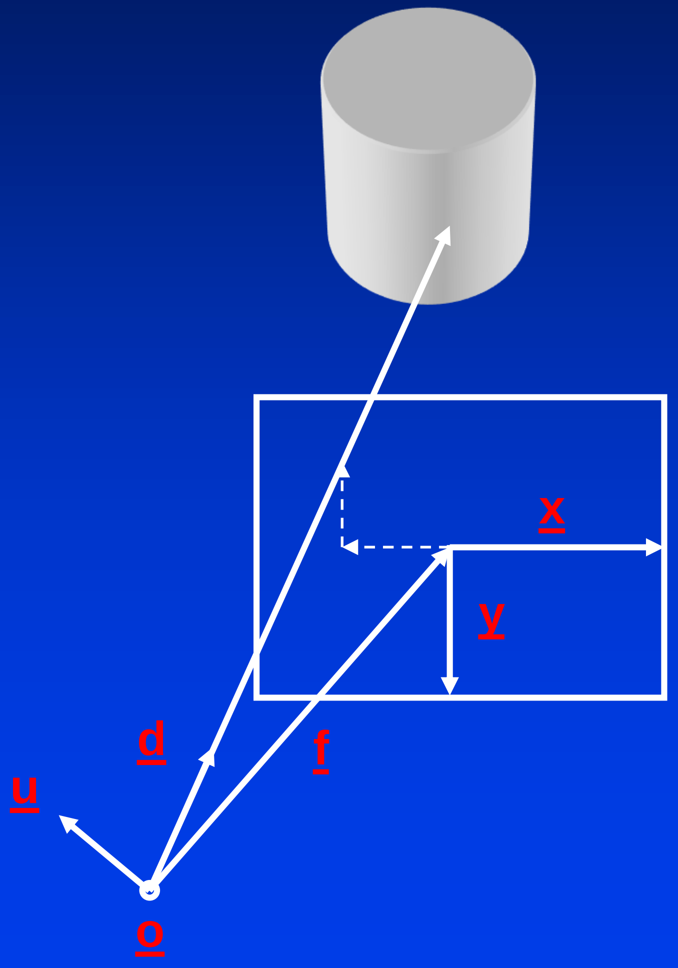
Ray Generation

- 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



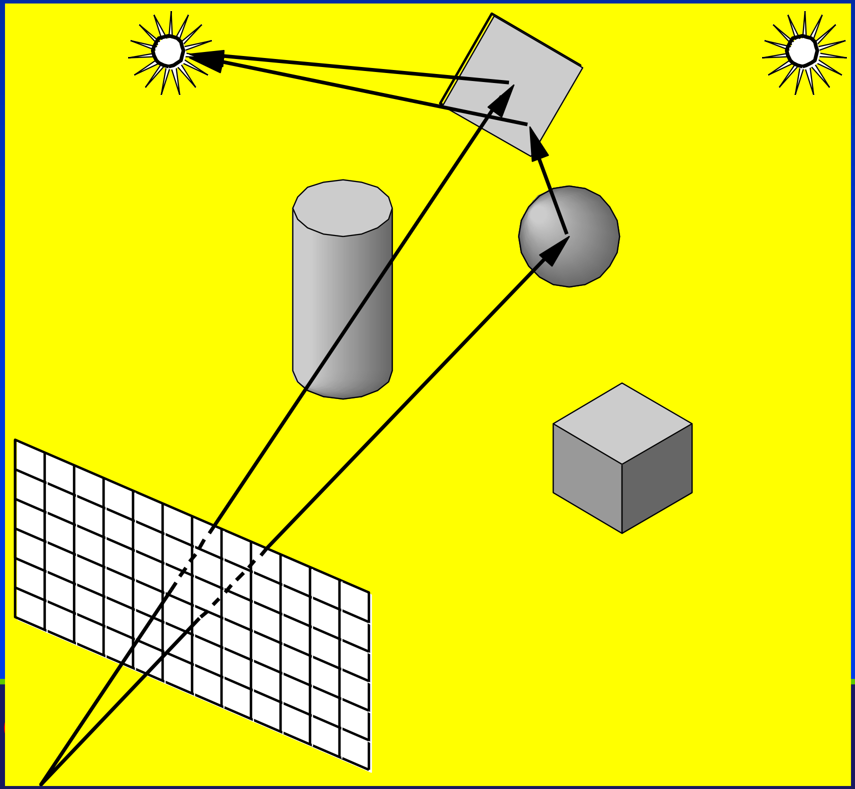
Algorithm

```
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);
  }
```

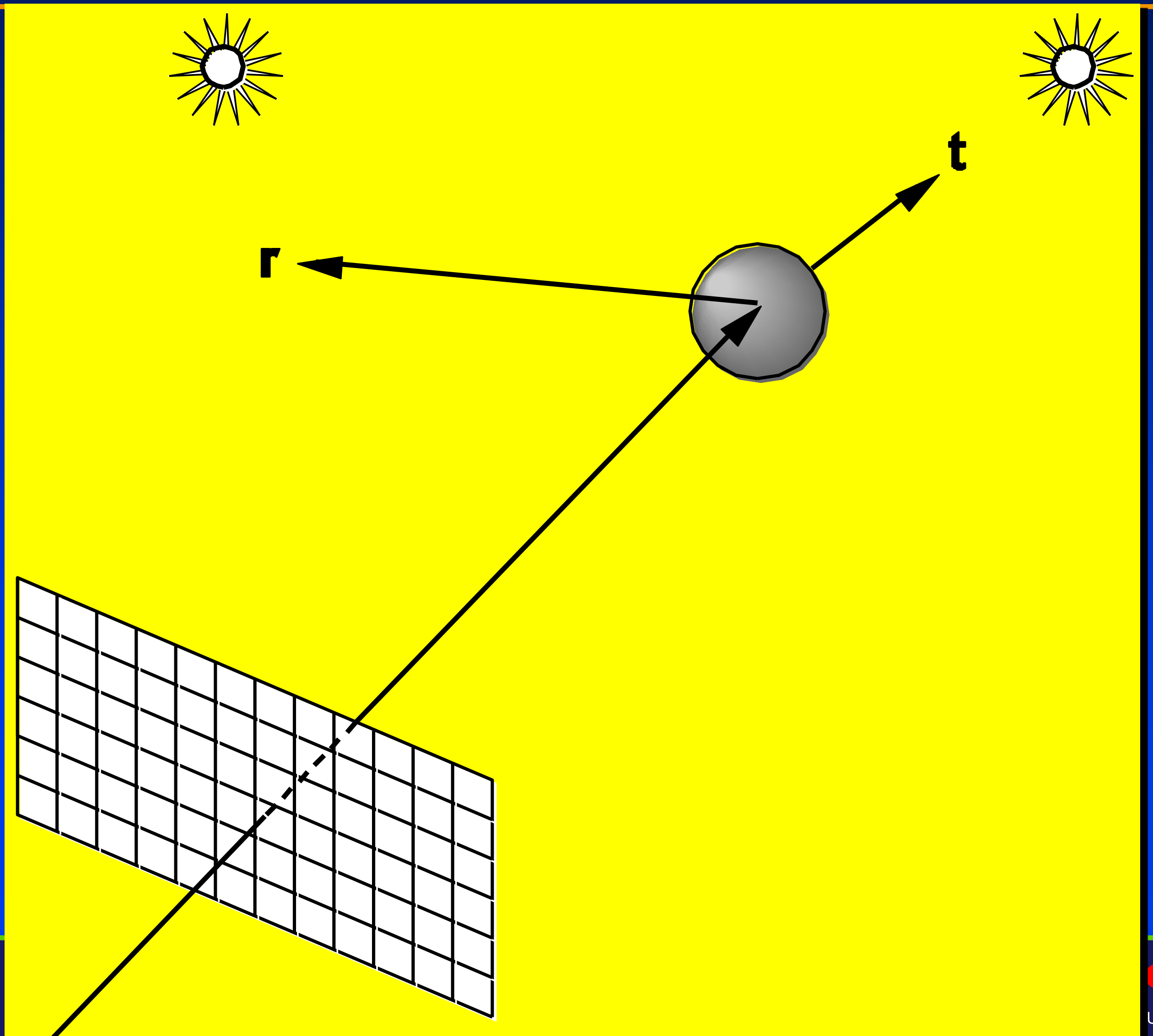


Reflection

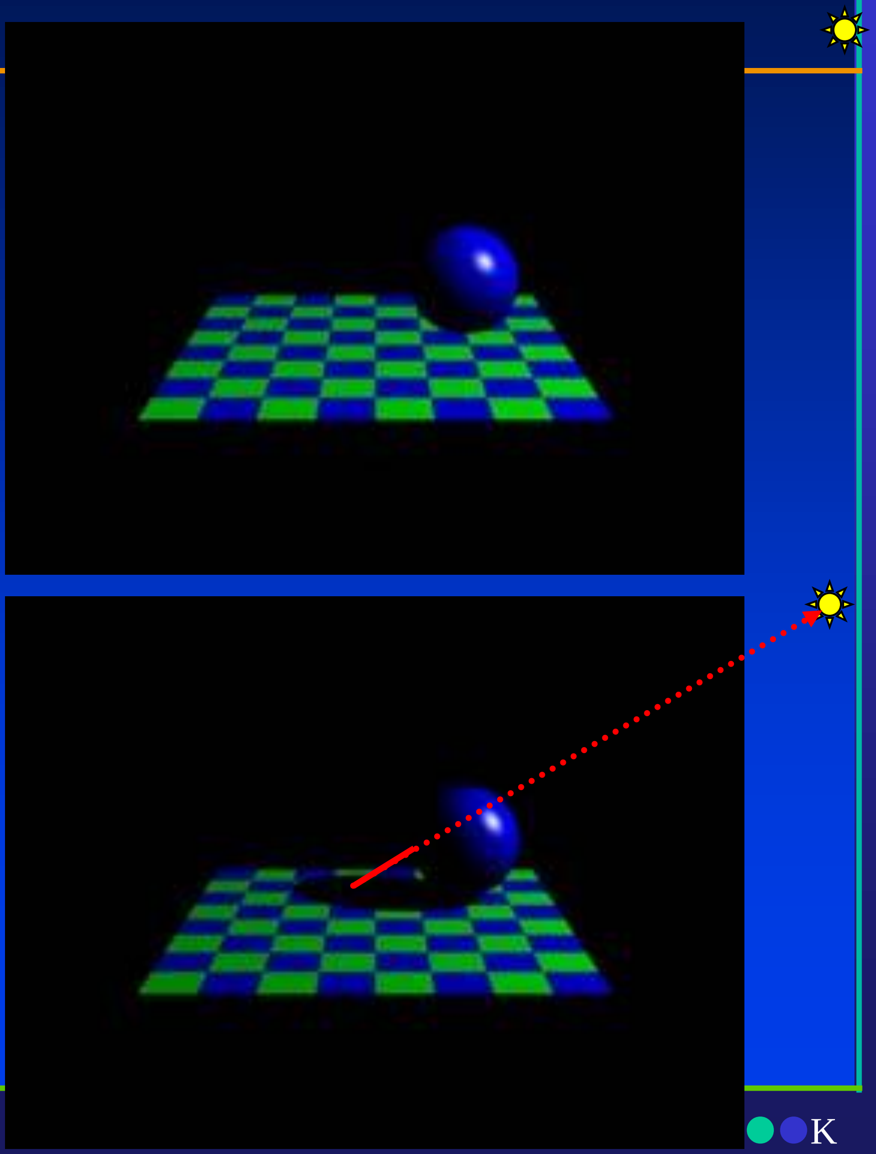
- Must follow shadow rays off reflecting or transmitting surfaces
- Process is recursive



Reflection and Transmission



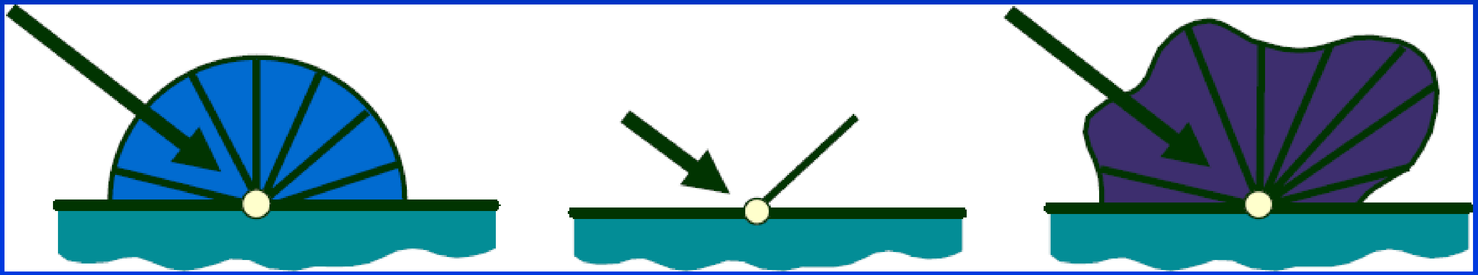
Shadow Ray



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

Ray Tracing



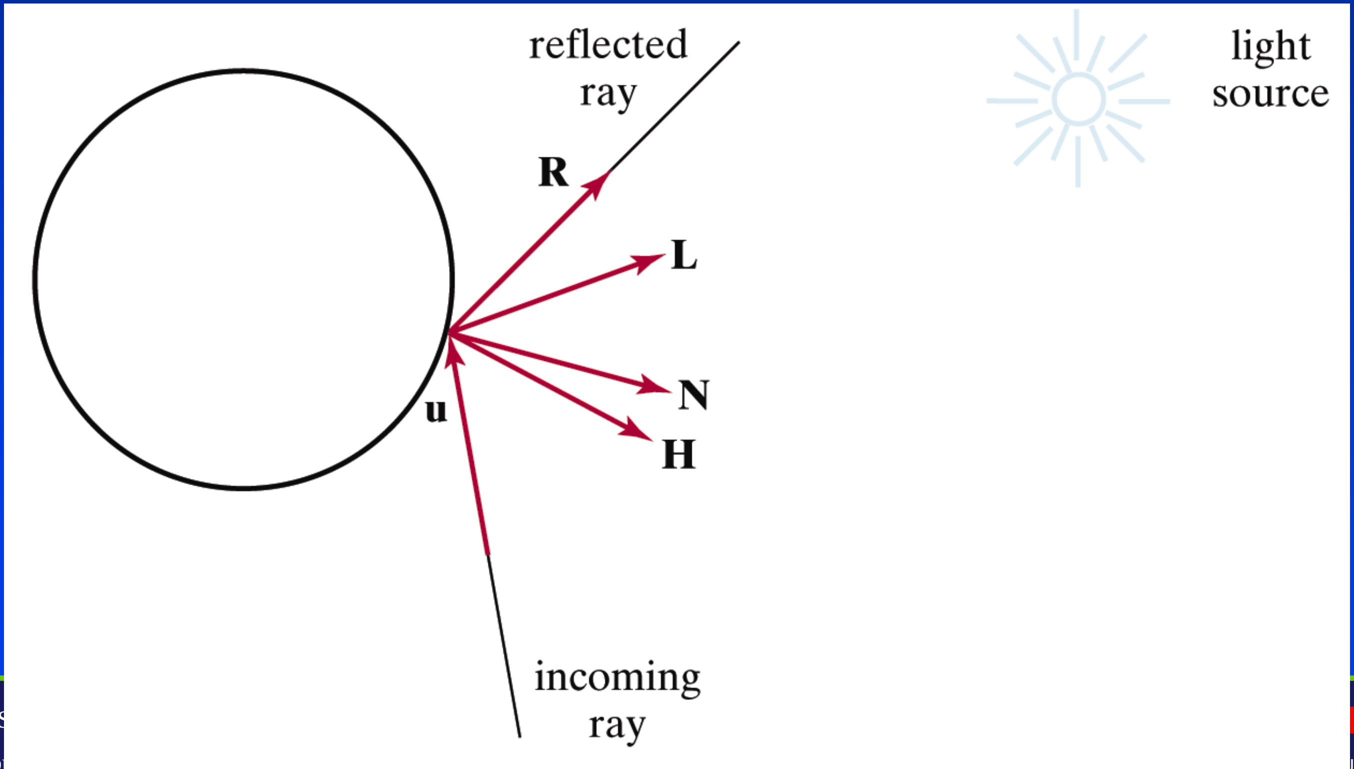
- Diffuse
- Cos (N.L)

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

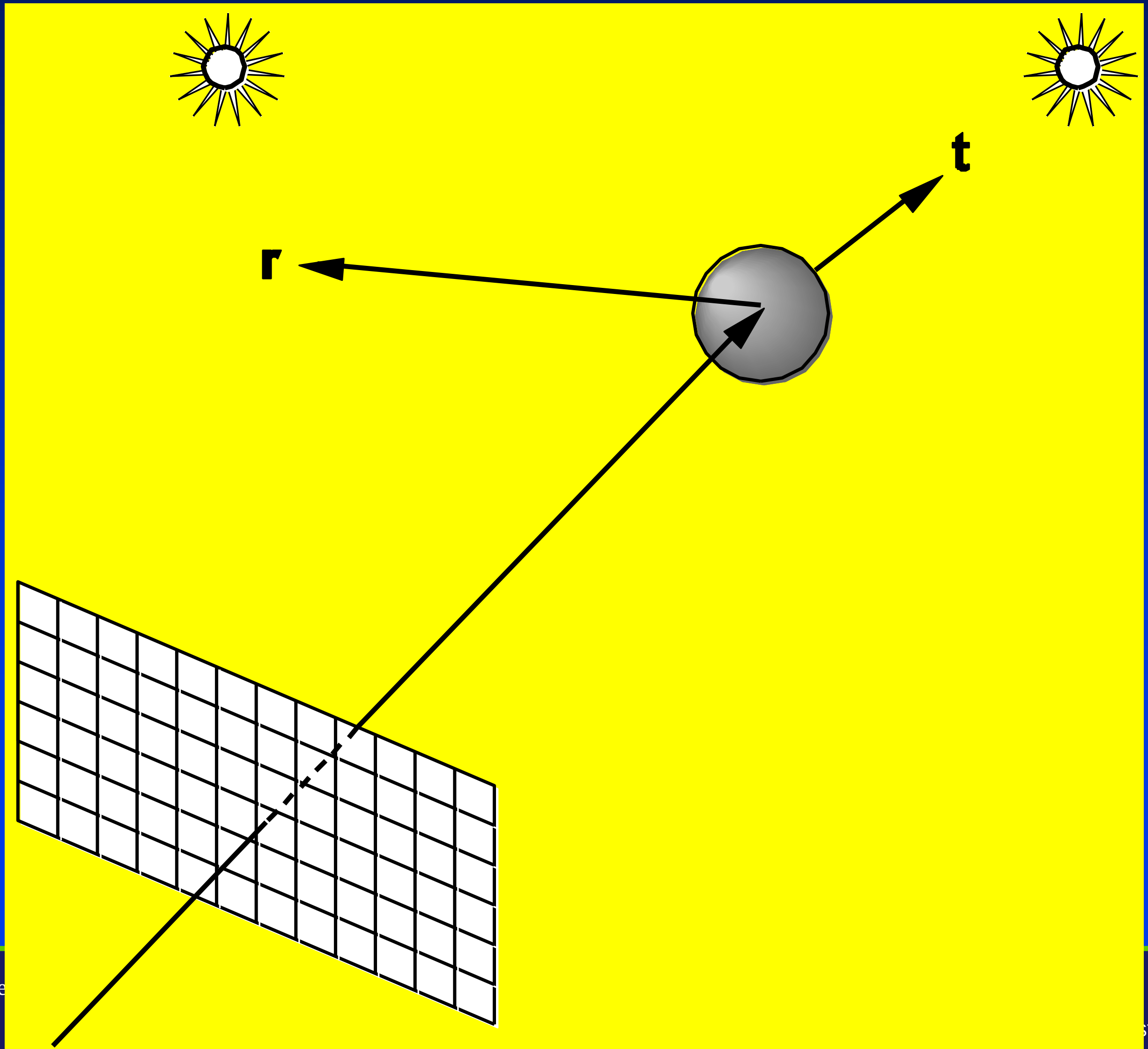
- Phong shading
- Cos (R.V) of (N.H)
- Exponential n

Ray-Tracing & Illumination Models

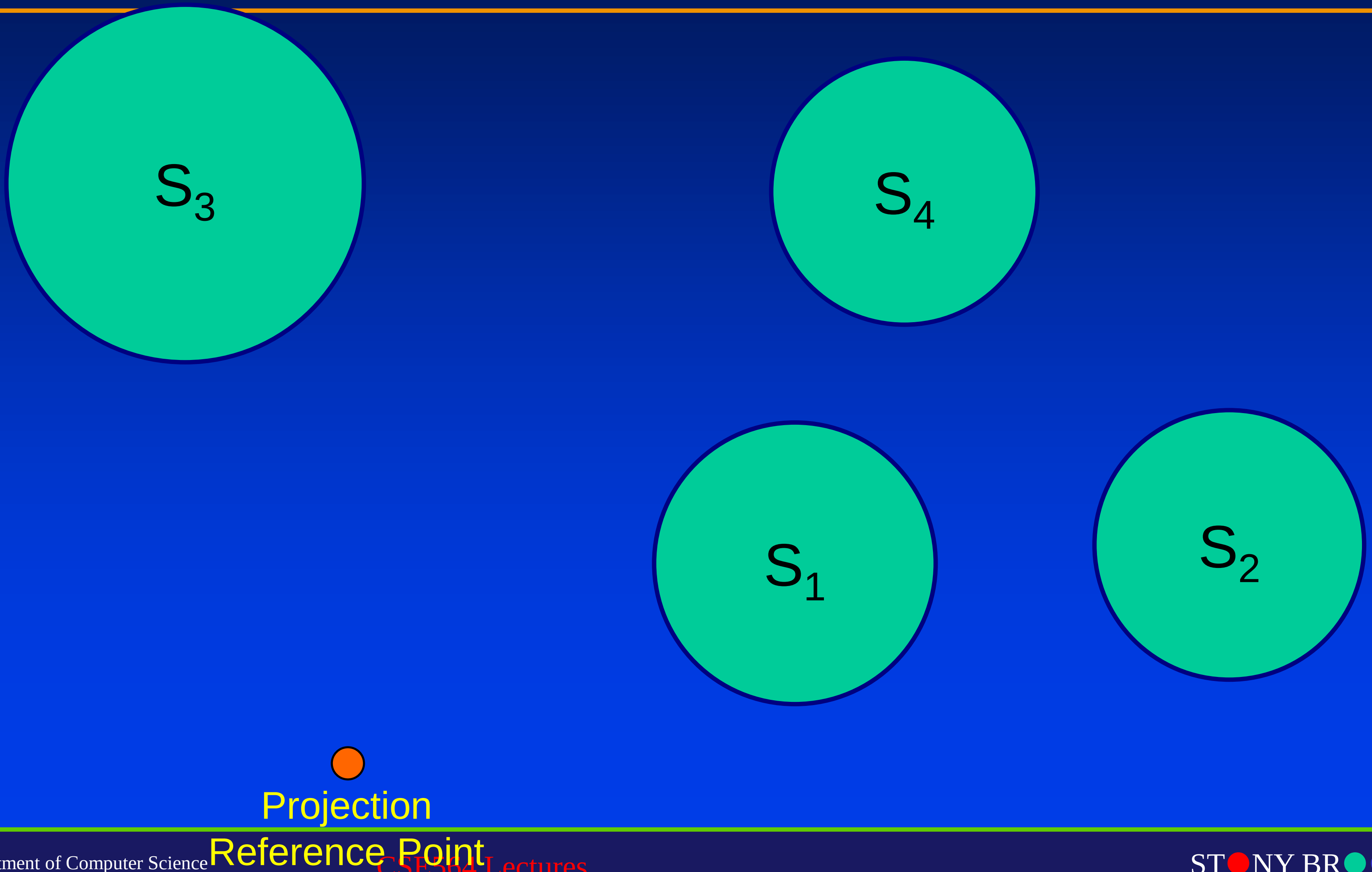
- At each surface intersection the illumination model is invoked to determine the surface intensity contribution



Reflection and Transmission



Ray-Tracing Tree Example



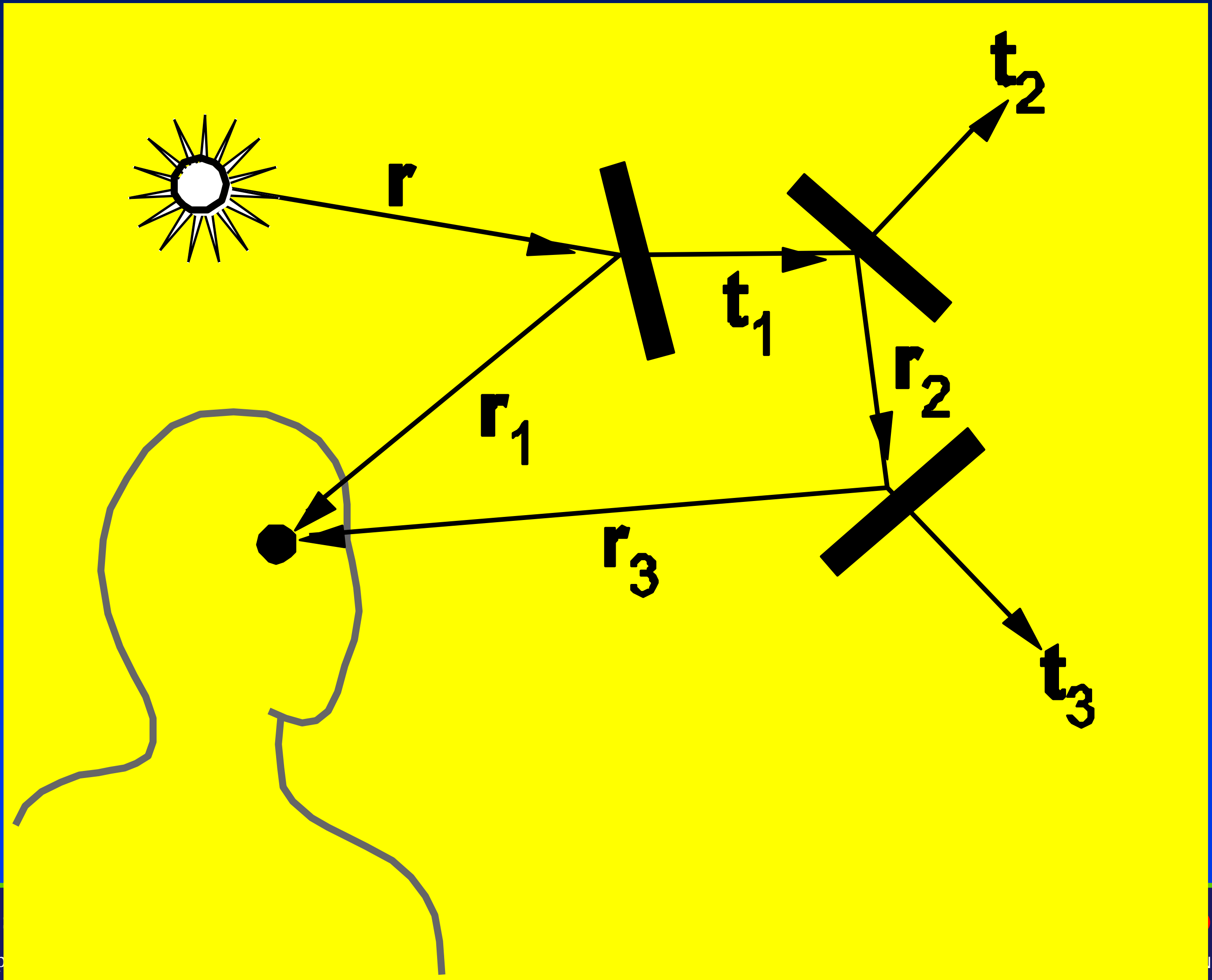
Basic Ray-Tracing

- 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

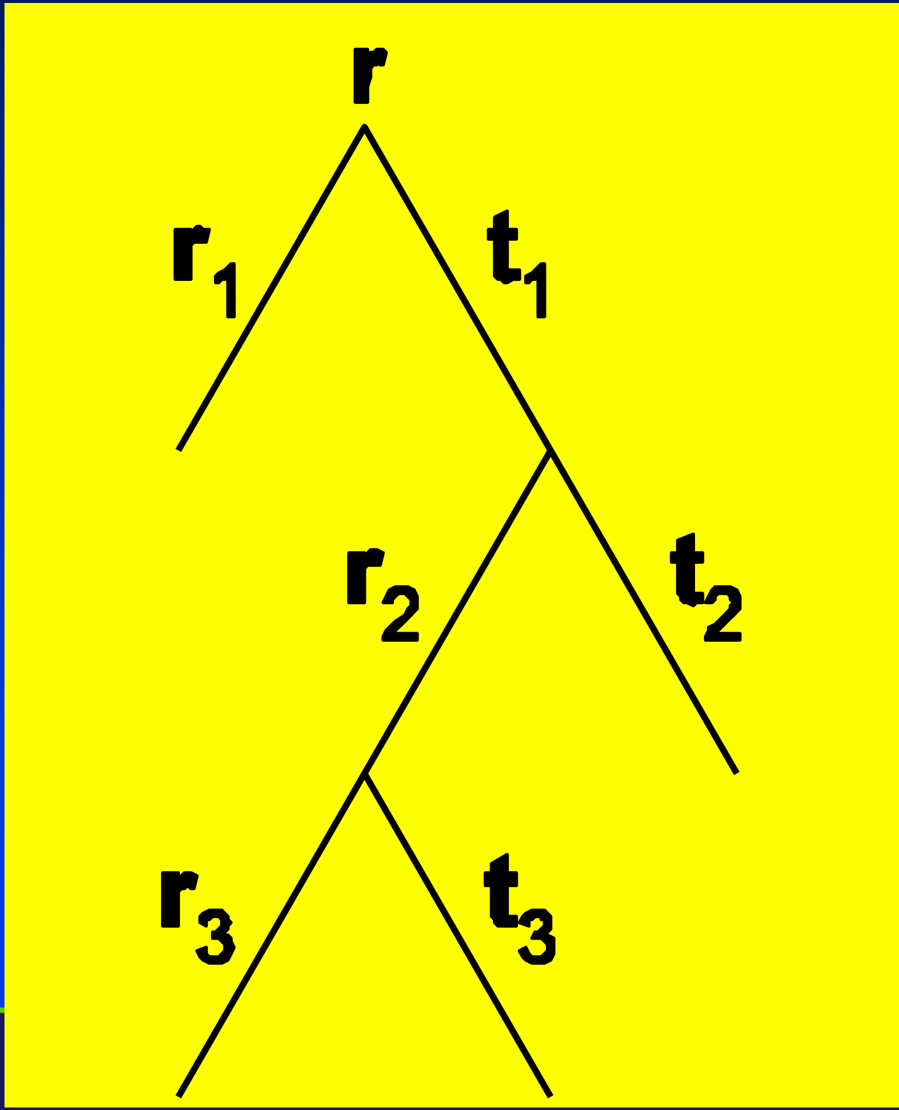
Ray-Tracing Tree

- 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

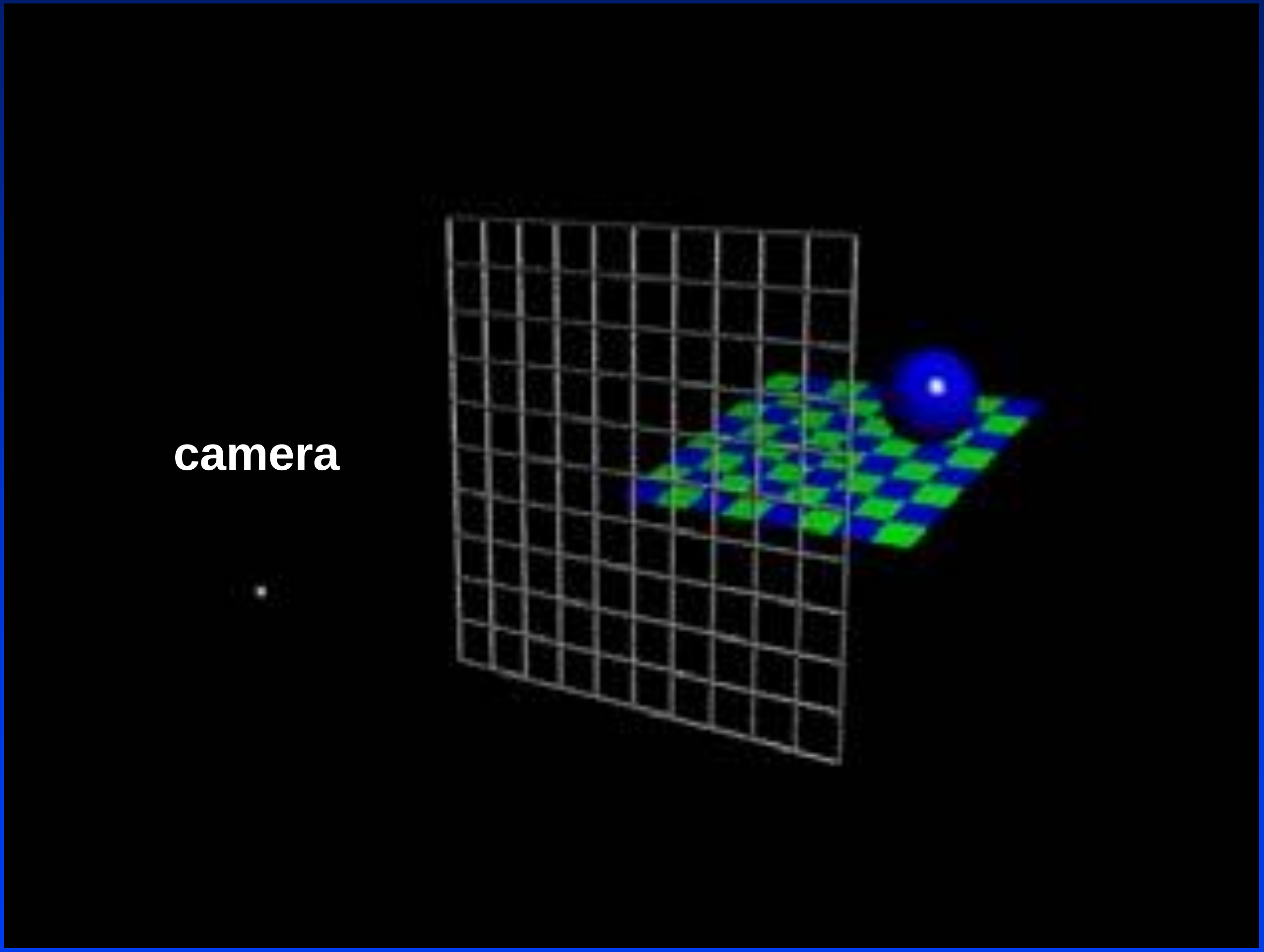


Ray-Tracing Tree

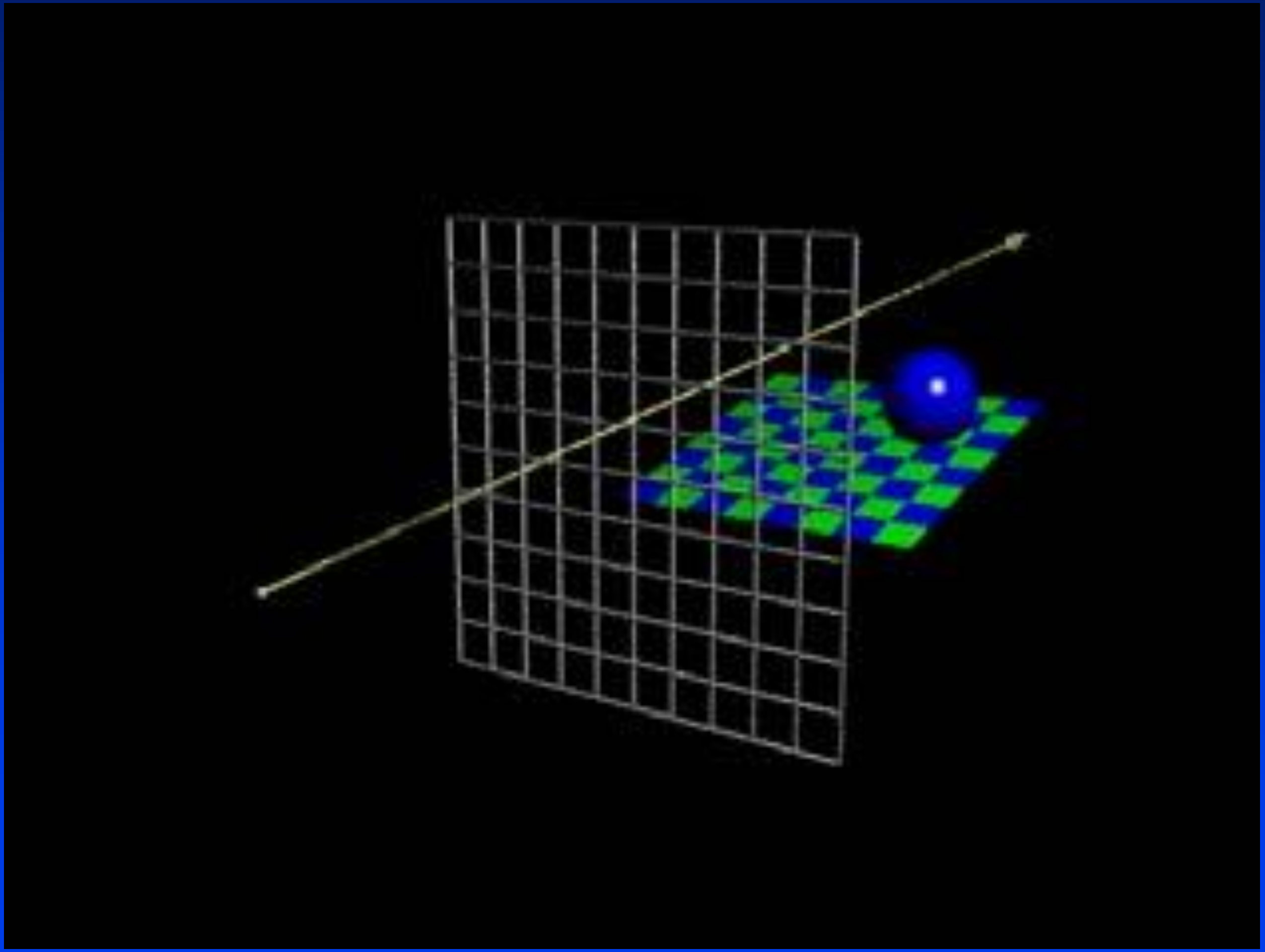


Ray Casting: Basic Principles

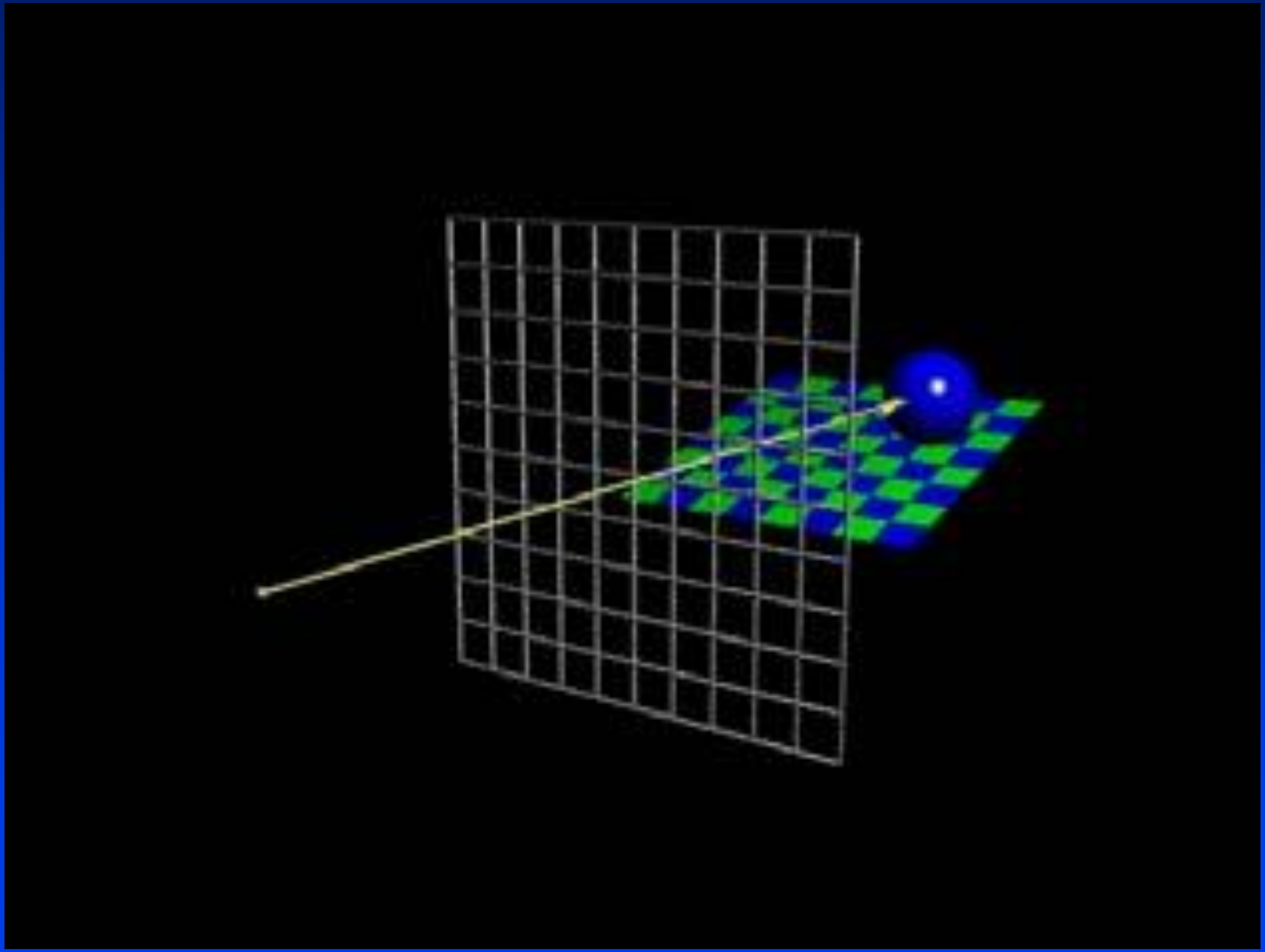
- Camera
- Pixel plane
- Scene



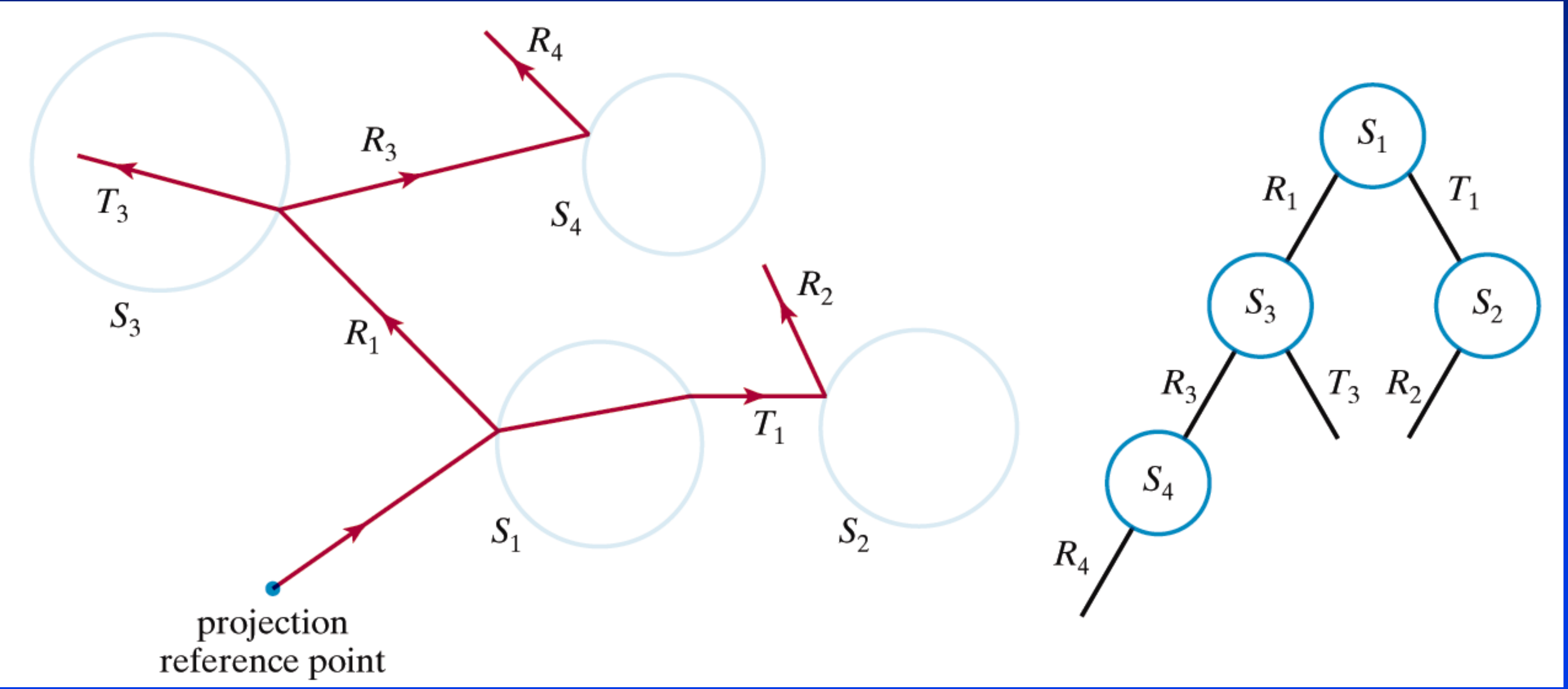
Ray Casting: Basic Principles



Ray Casting: Basic Principles



Ray-Tracing Tree Example



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

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

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

Computing Intensities using Ray-Tracing Tree

- After the ray-tracing tree has been completed for a pixel, the intensity contributions shall be accumulated
- We start at the terminal nodes (leaves) 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

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);
```

Computing Intersections

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

Planes

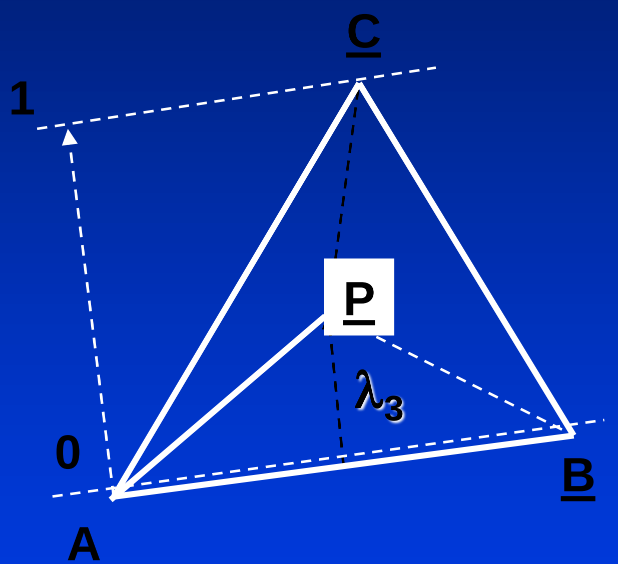
$$\mathbf{p} \bullet \mathbf{n} + c = 0$$

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

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

Intersection Ray - Triangle

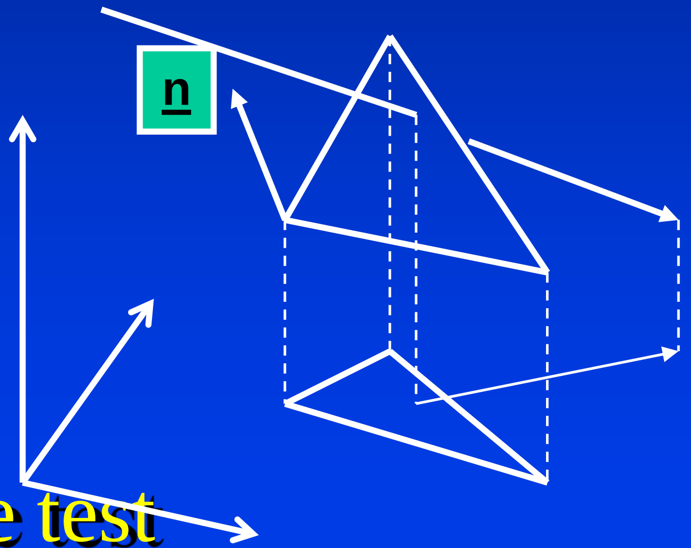
- Barycentric coordinates
  - Non-degenerate triangle ABC
    $\underline{P} = \lambda_1 \underline{A} + \lambda_2 \underline{B} + \lambda_3 \underline{C}$
  - $\lambda_1 + \lambda_2 + \lambda_3 = 1$
  - $\lambda_3 = \angle(APB) / \angle(ACB)$ etc
    - Relative area



- Hit iff all λ_i greater or equal than zero

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



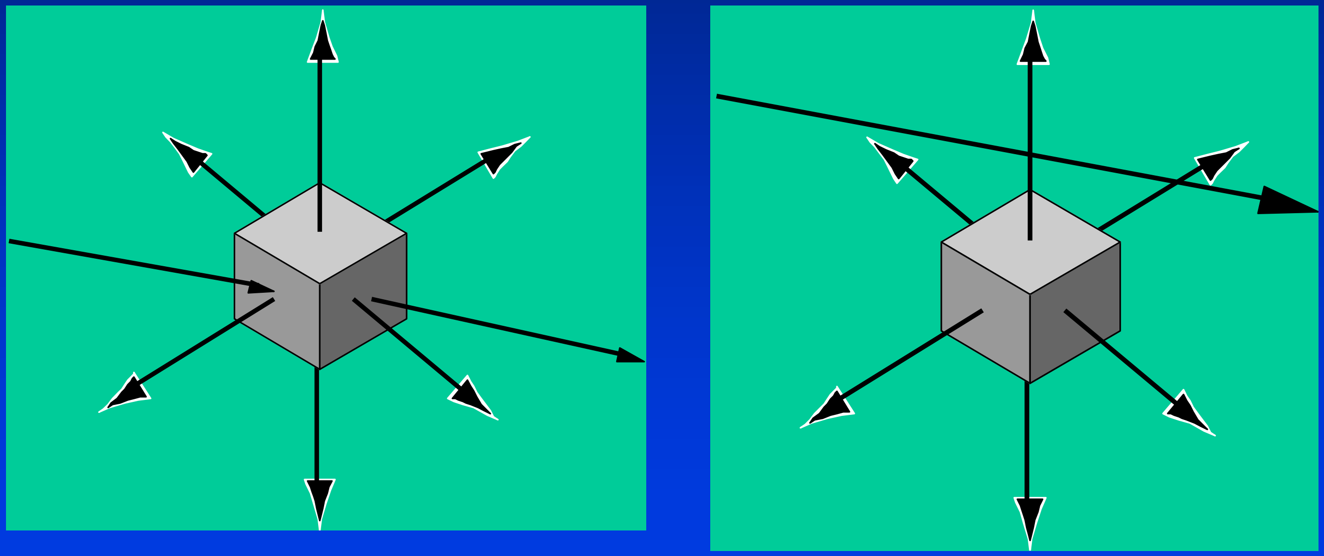
Polyhedra

- 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

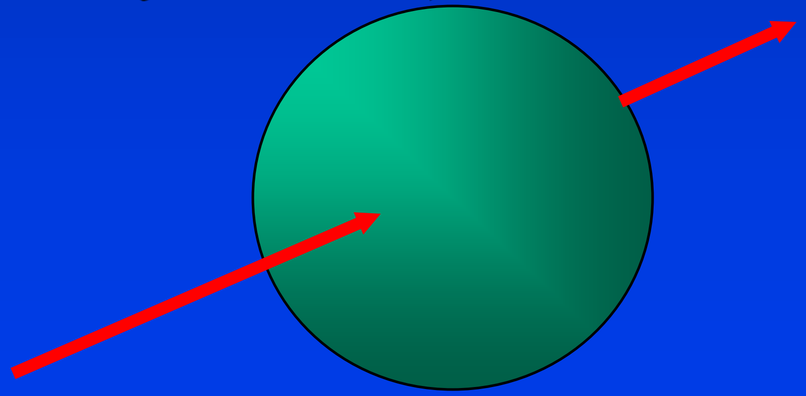


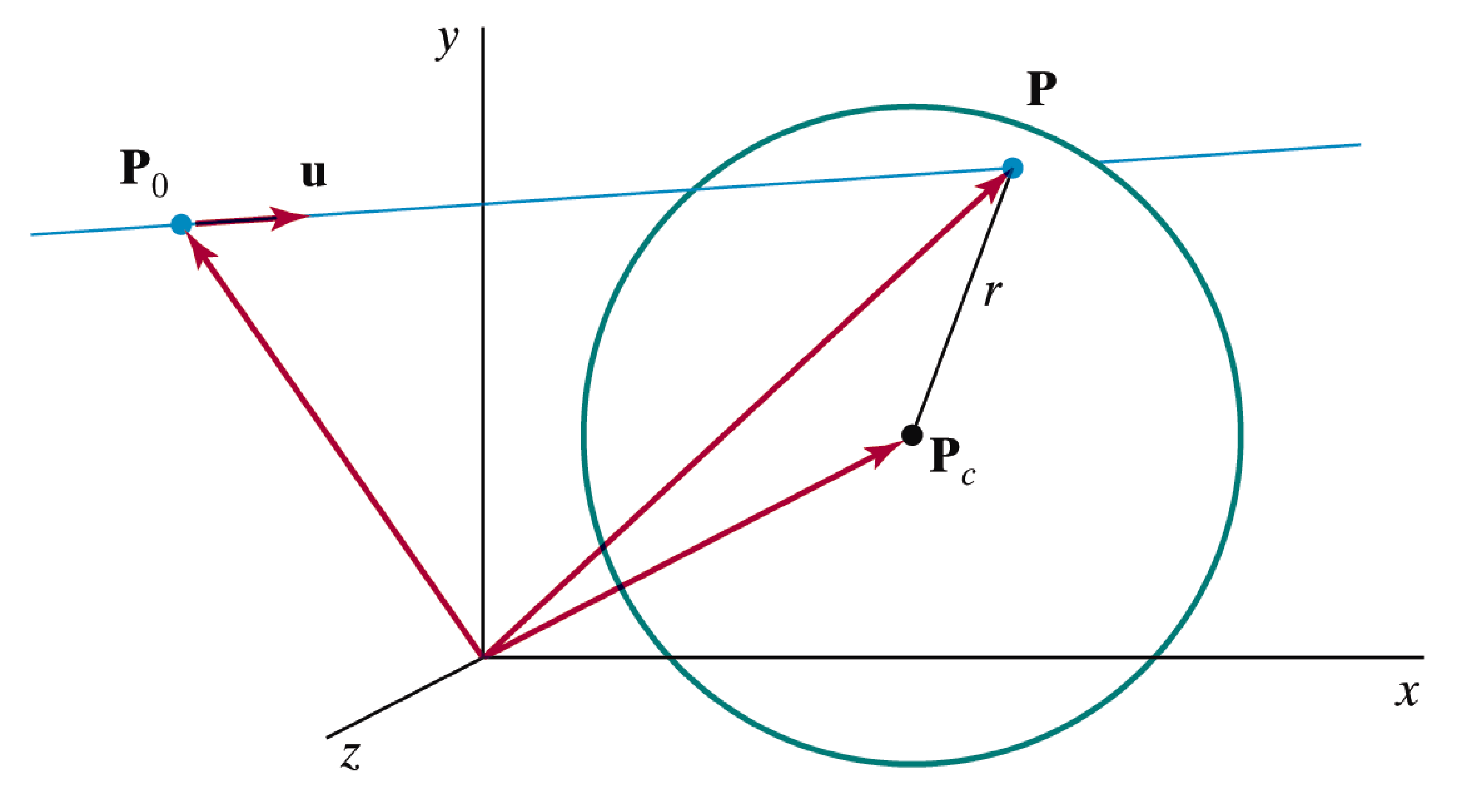
Ray Casting Quadrics

- 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

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)




y
P
P_0
u
r
P_c
x
z

$$P = P_0 + su$$

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

Sphere Equation

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

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

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

Ray Casting for 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

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)

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

Grid: Issues

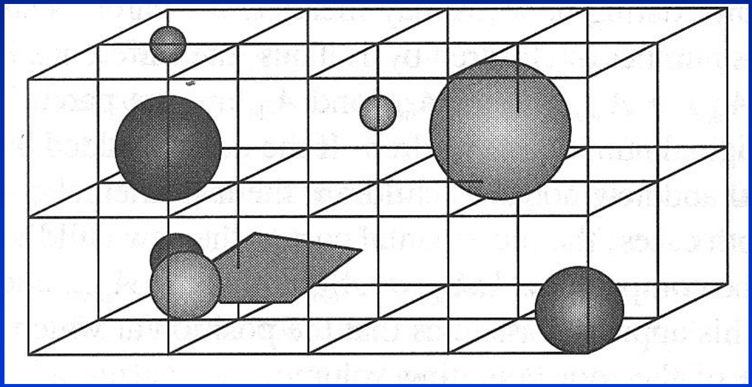
- Objects in multiple voxels
  - Store only references
  - Use mailboxing to avoid multiple intersection computations
    - Store (ray, object)-tuple in small cache (e.g. with hashing)
    - Do not intersect if found in cache
  - Original mailbox uses ray-id stored with each triangle
    - Simple, but likely to destroy CPU caches

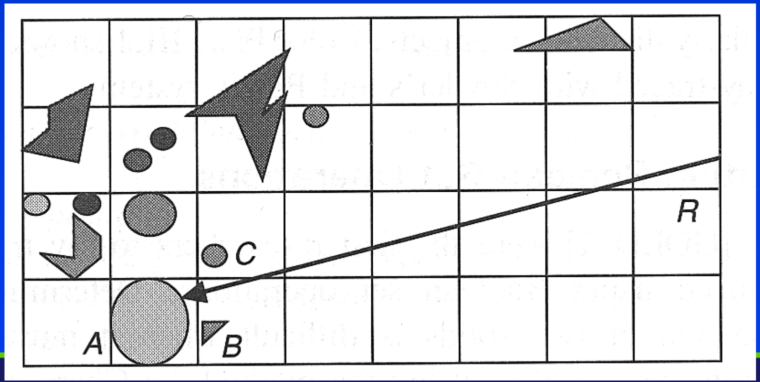
History of Intersection Algorithms

- 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]

Spatial Partitioning: Grid Structure

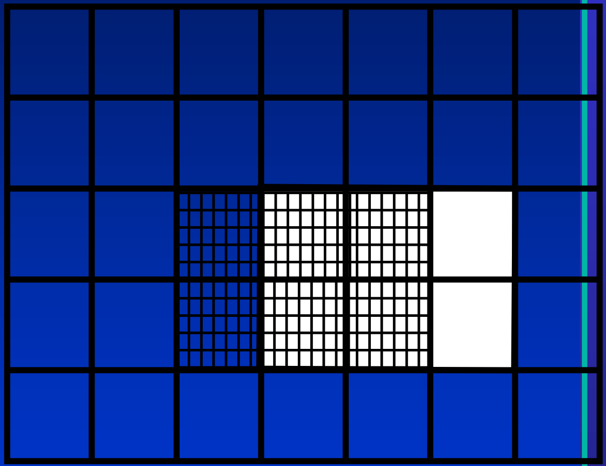
- Building a grid structure
  - Start with bounding box
  - Resolution: often ~ $\sqrt[3]{n}$
  - Overlap or intersection test
- Traversal
  - 3D-DDA
  - Stop if intersection found in current voxel



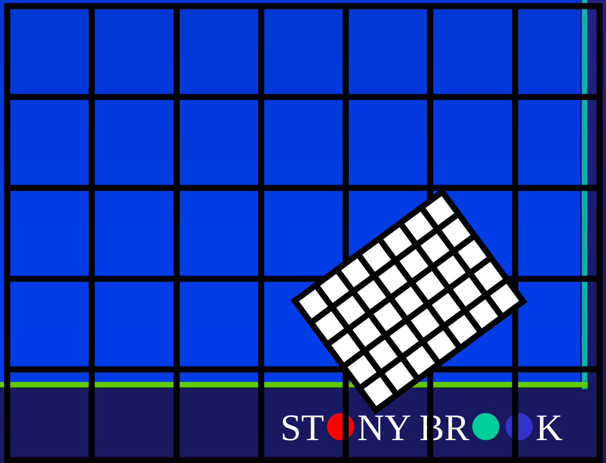


Hierarchical Grids

- Simple building algorithm
  - Recursively create grids in high-density voxels
  - Problem: What is the right resolution for each level?

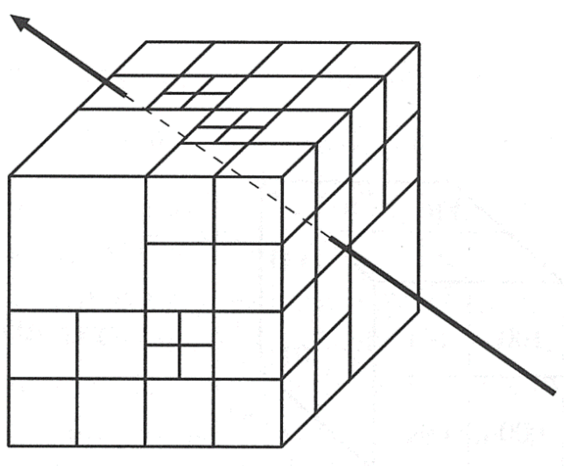


- Advanced algorithm
  - Separate grids for object clusters
  - Problem: What are good clusters?



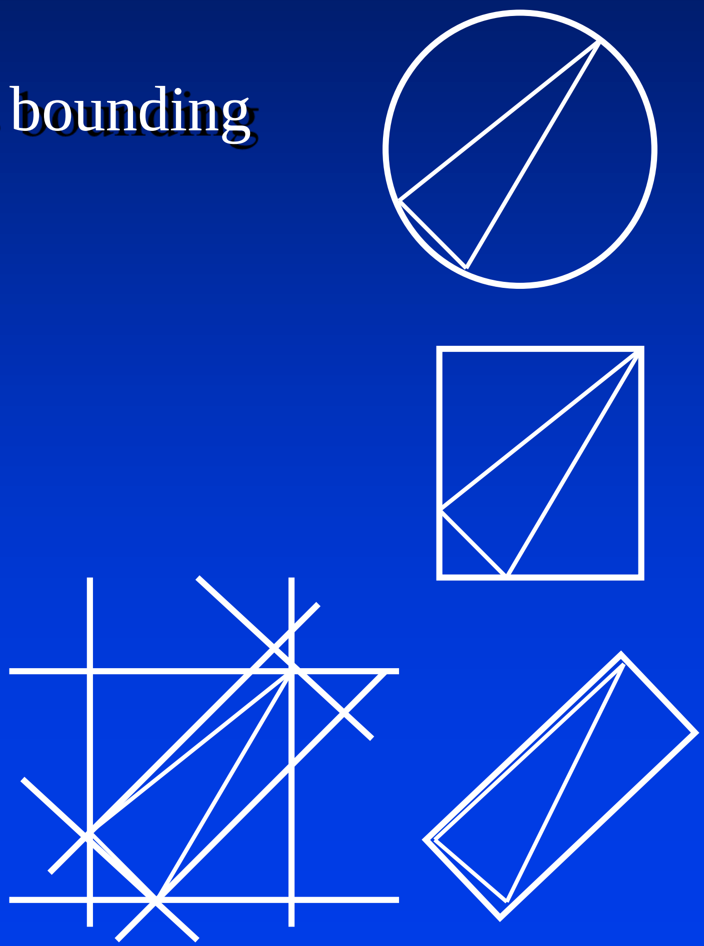
Octree

- 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



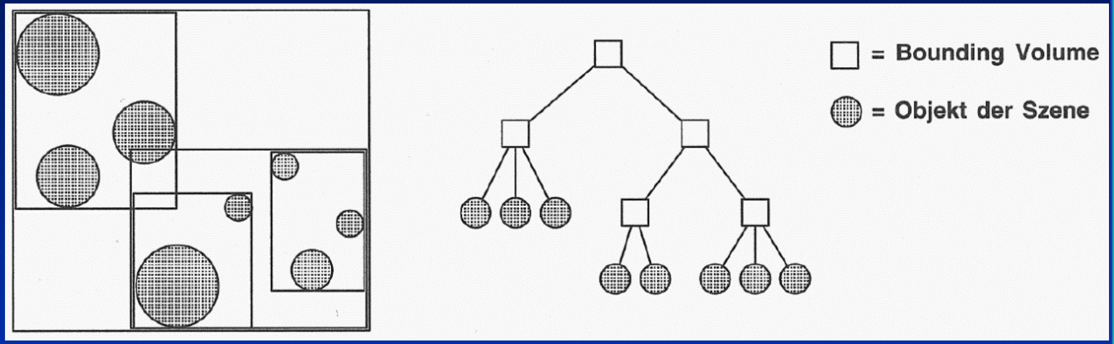
Bounding Volumes

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



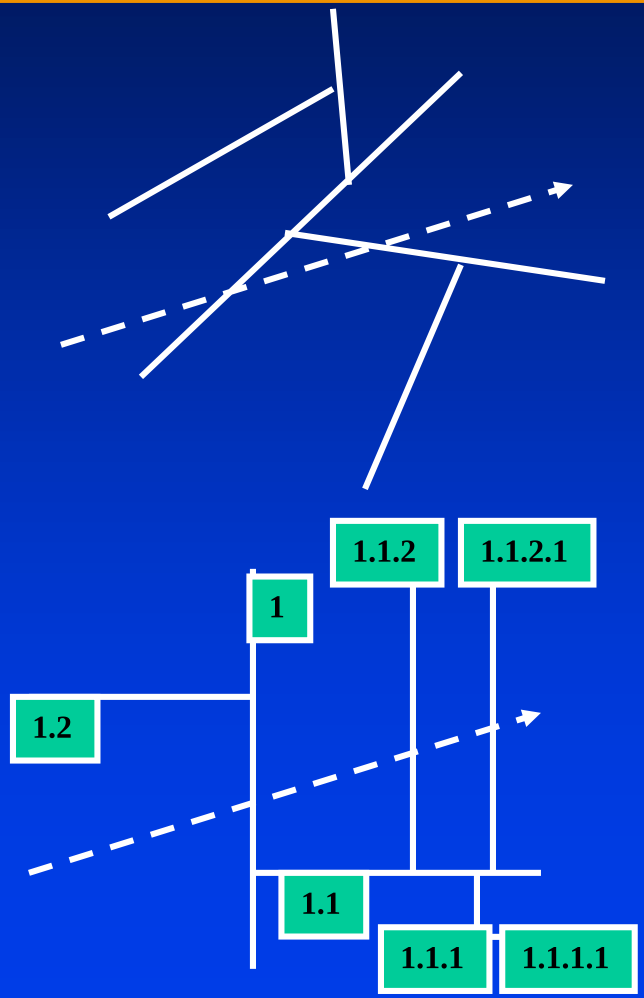
Bounding Volume Hierarchies

- Idea:
  - Apply recursively
- Advantages:
  - Very good adaptivity
  - Efficient traversal O(log N)
- Problems
  - How to arrange bounding volumes?

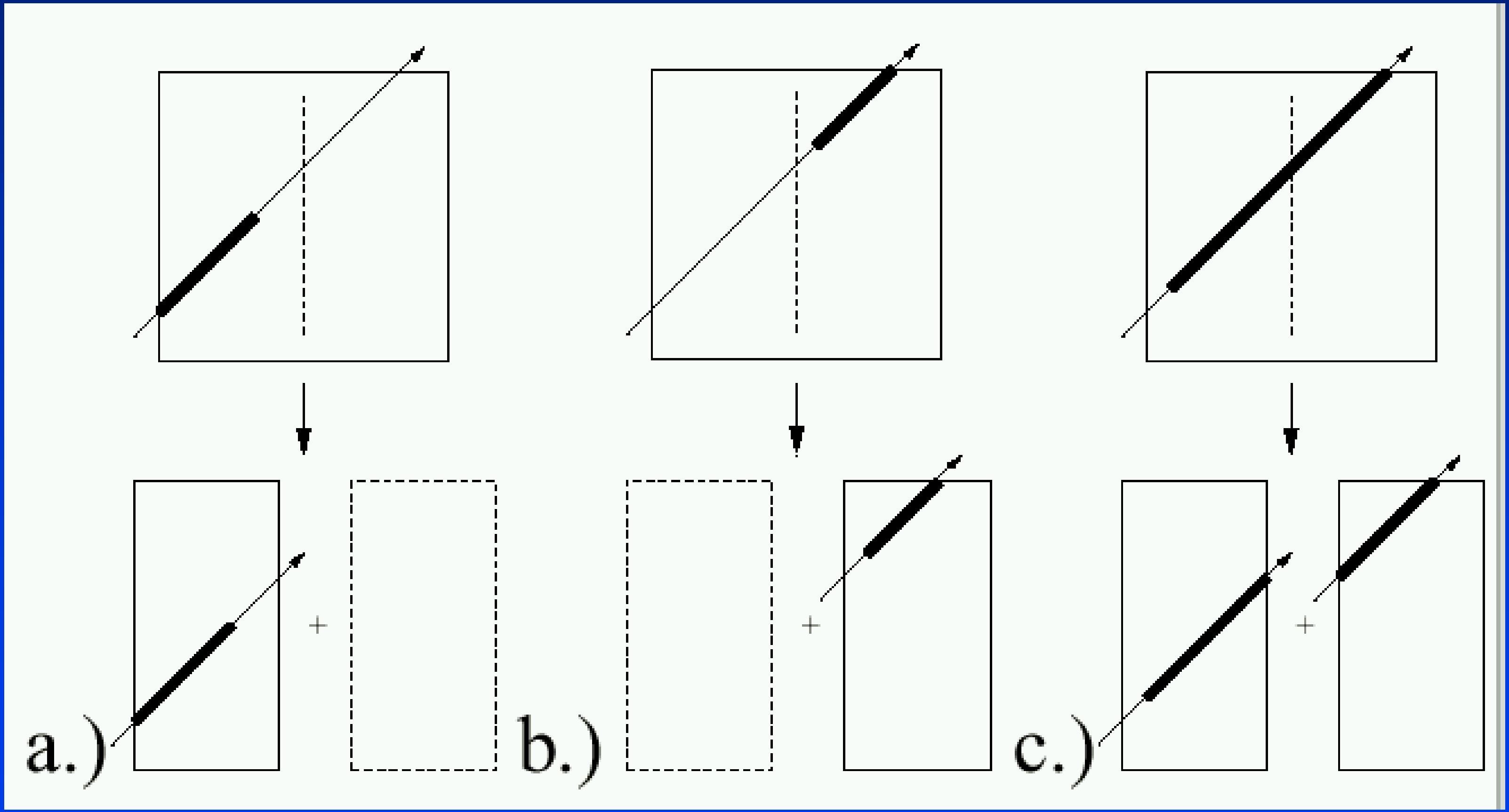


BSP- and Kd-Trees

- 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

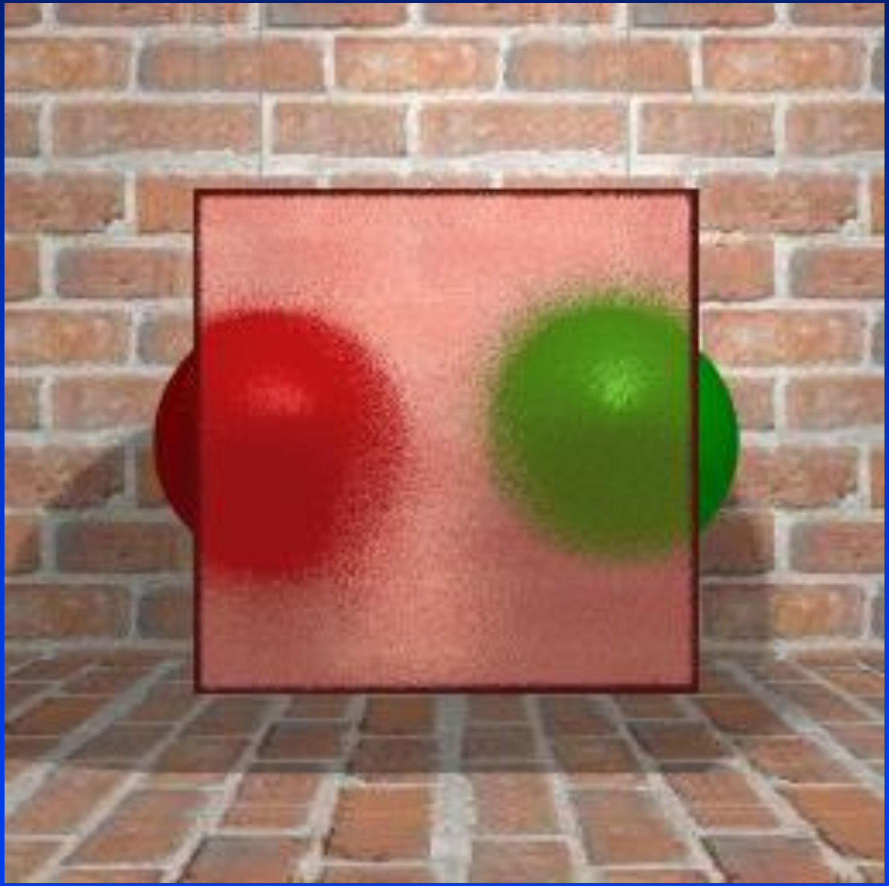


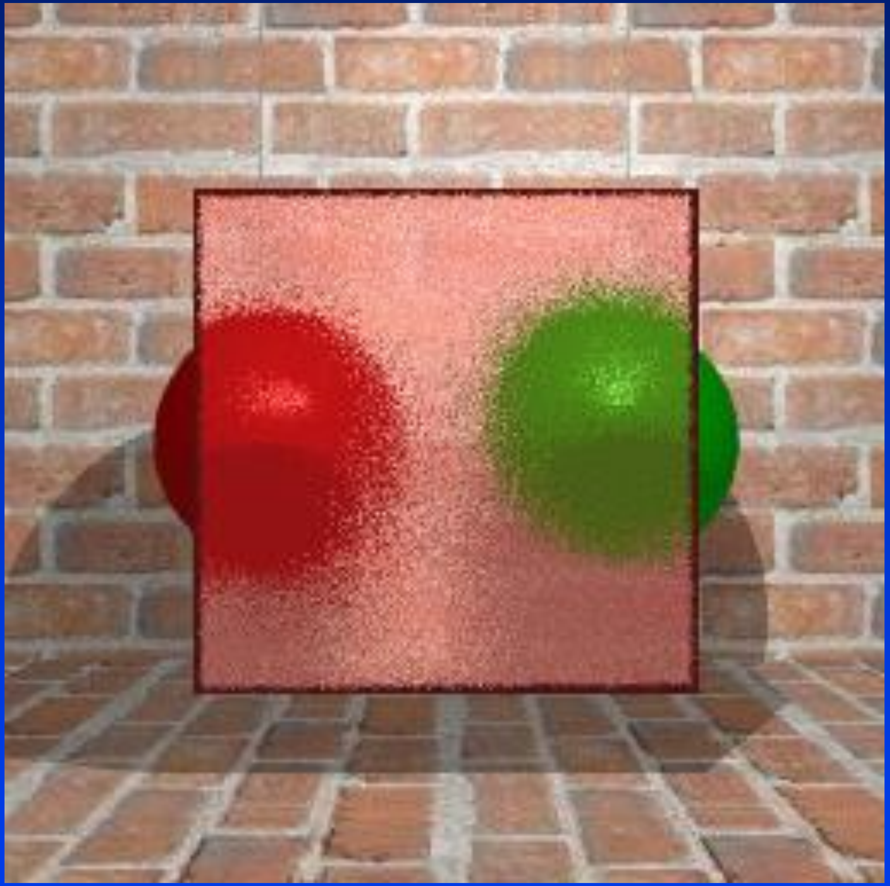
Kd-Tree Traversal



Other Visual Effects

Transparency



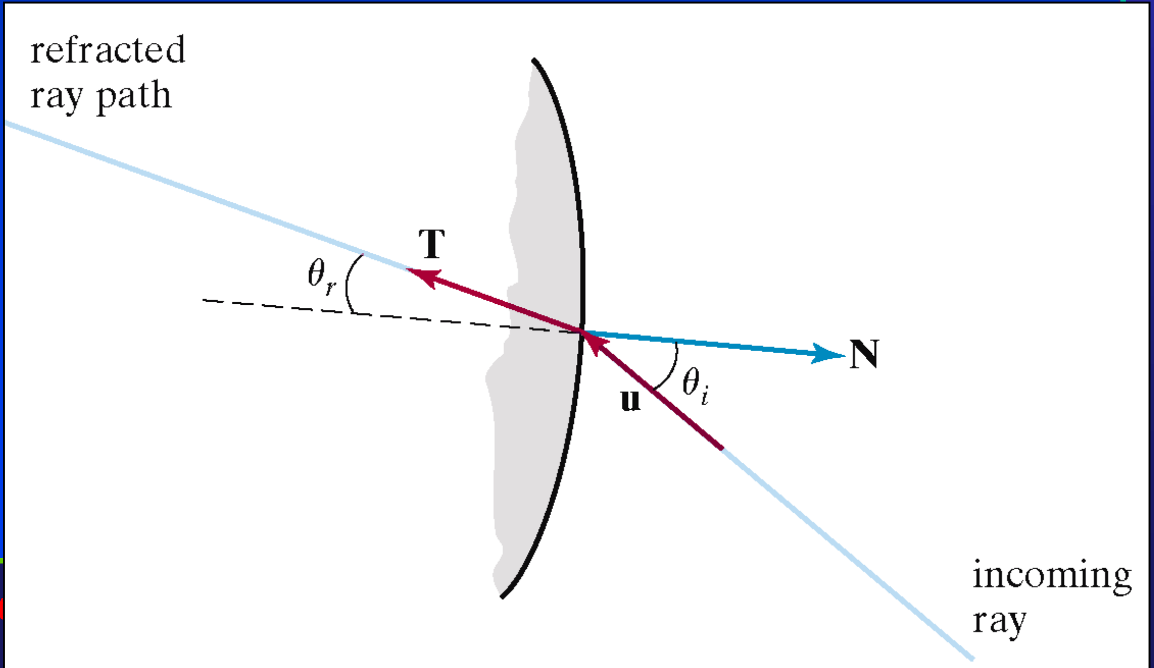


4 rays

16 rays

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



Geometric Optics

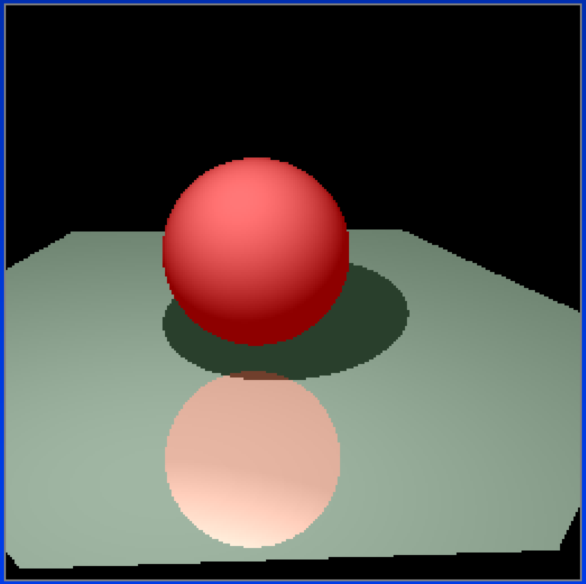
$$R = u - (2u \cdot N)N$$

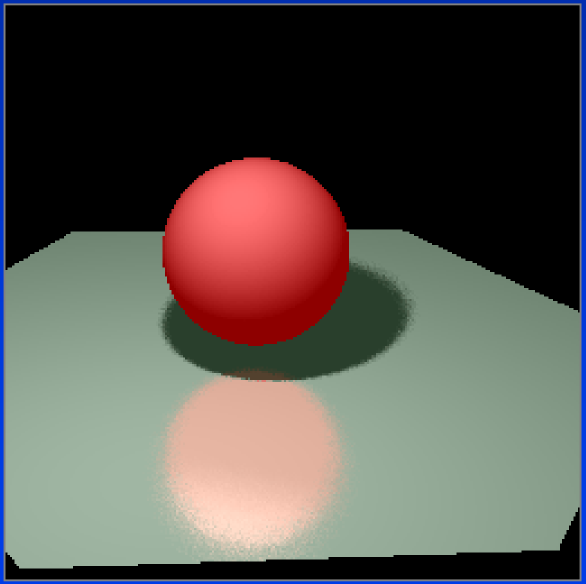
$$T = \frac{\eta_i}{\eta_r} u - \left(\cos\theta_r - \frac{\eta_i}{\eta_r} \cos\theta_i \right) N$$

$$\cos\theta_r = \sqrt{1 - \left(\frac{\eta_i}{\eta_r} \right)^2 \left(1 - \cos^2\theta_i \right)}$$

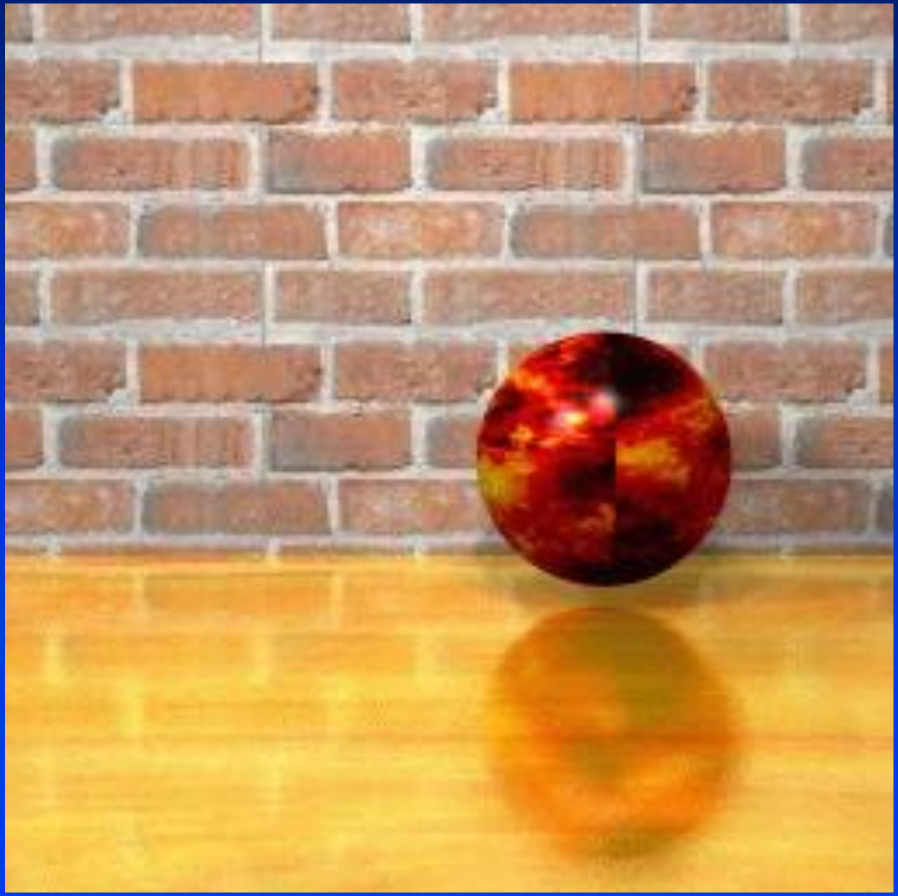
Gloss/Translucency

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

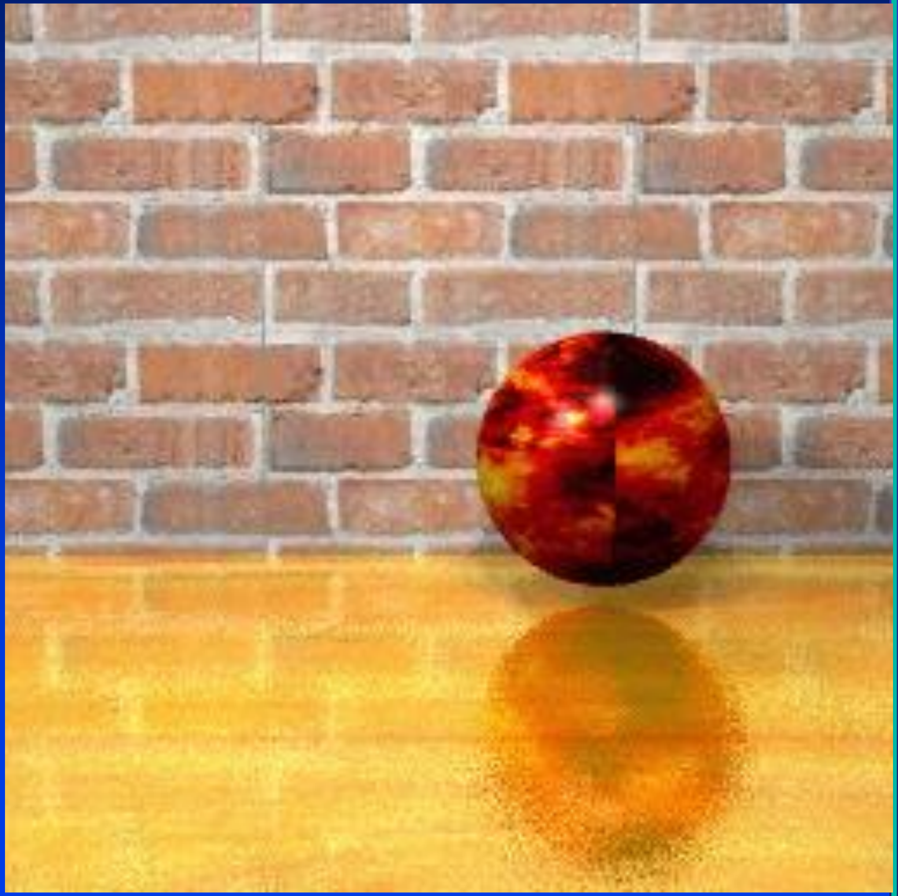




Reflection

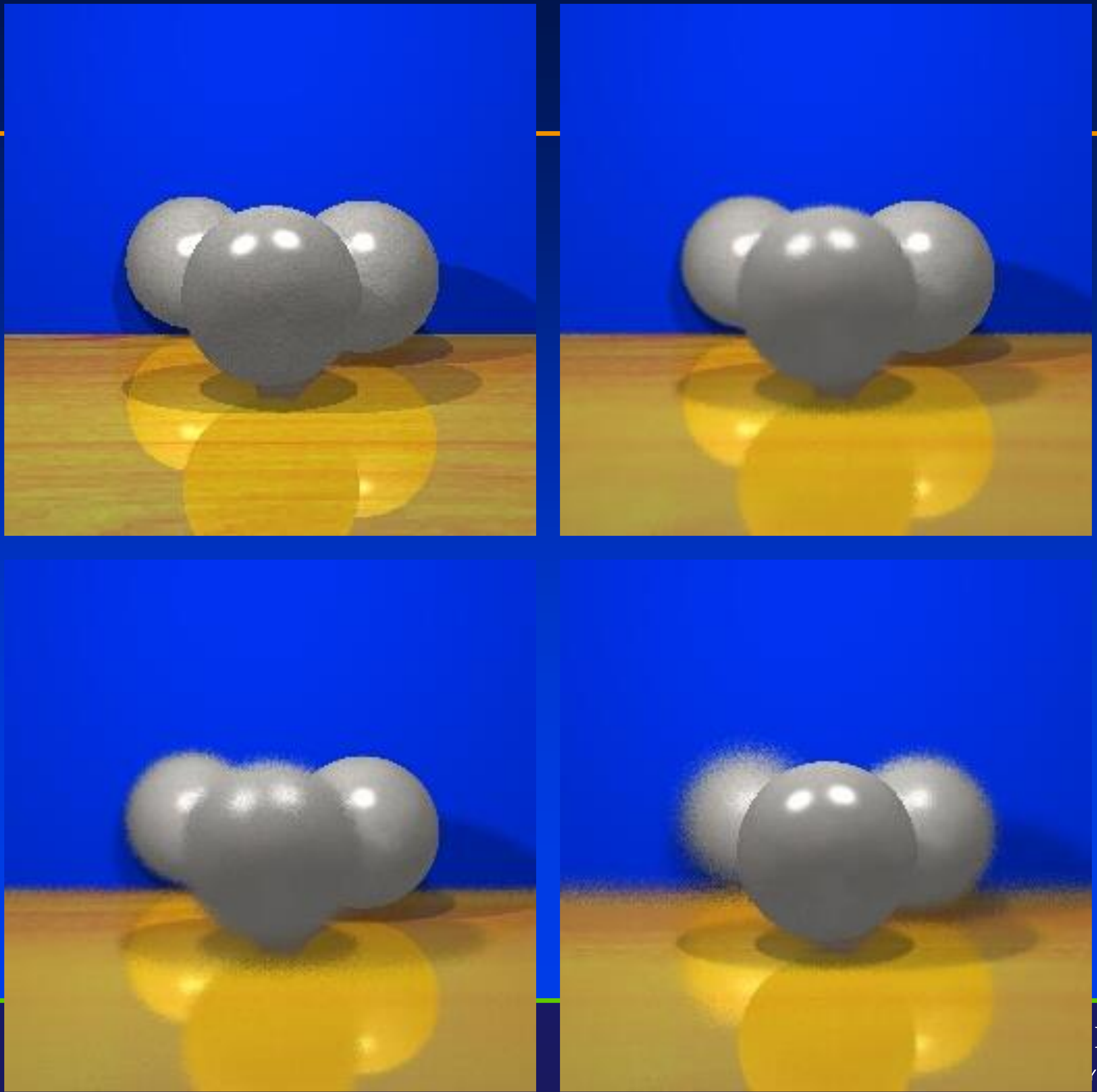


4 rays



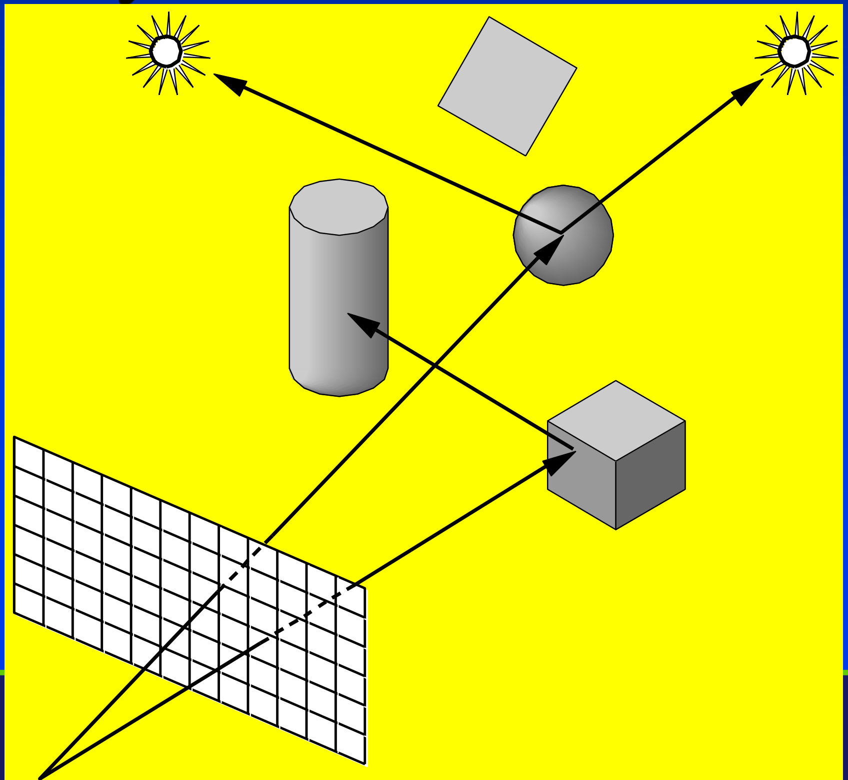
64 rays

Depth of Field



Shadow Rays

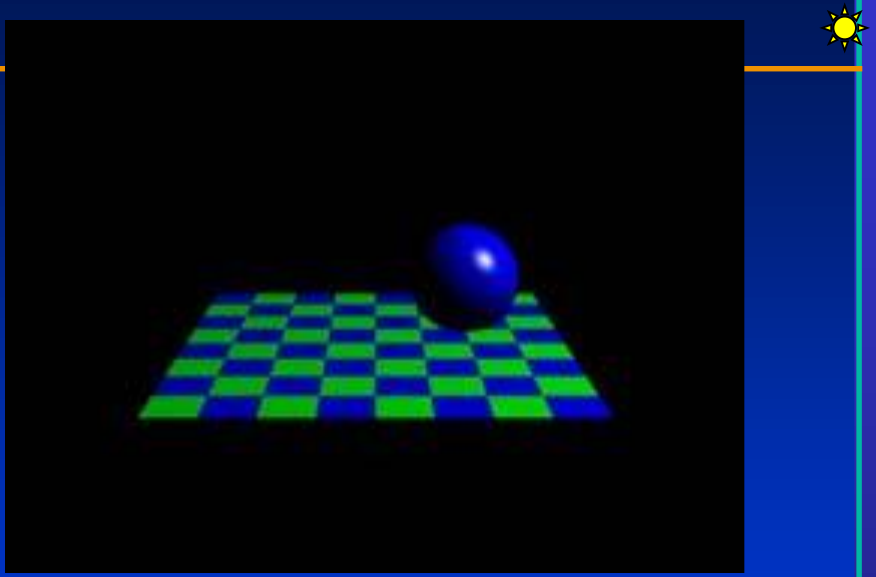
- Even if a point is visible, it will not be lit unless we can see a light source from that point
- Cast shadow rays

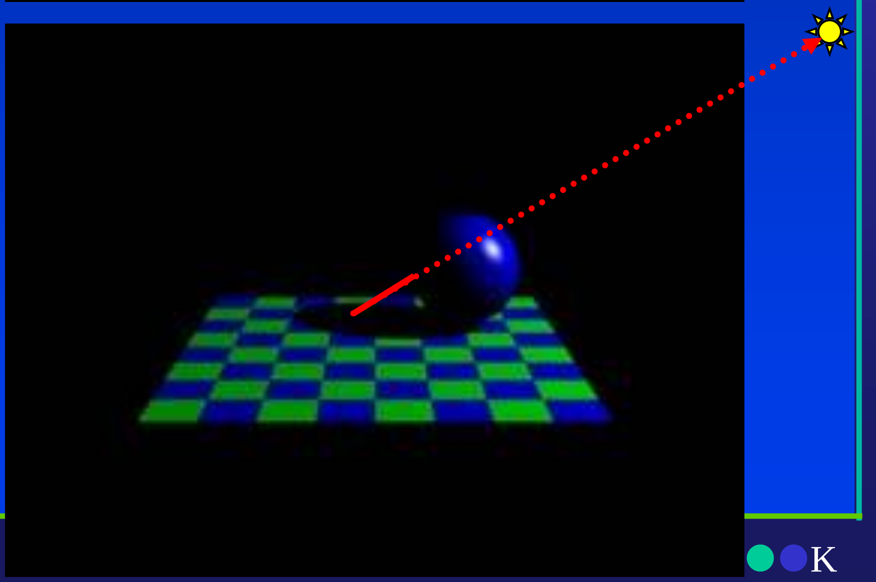


The Shadow Ray

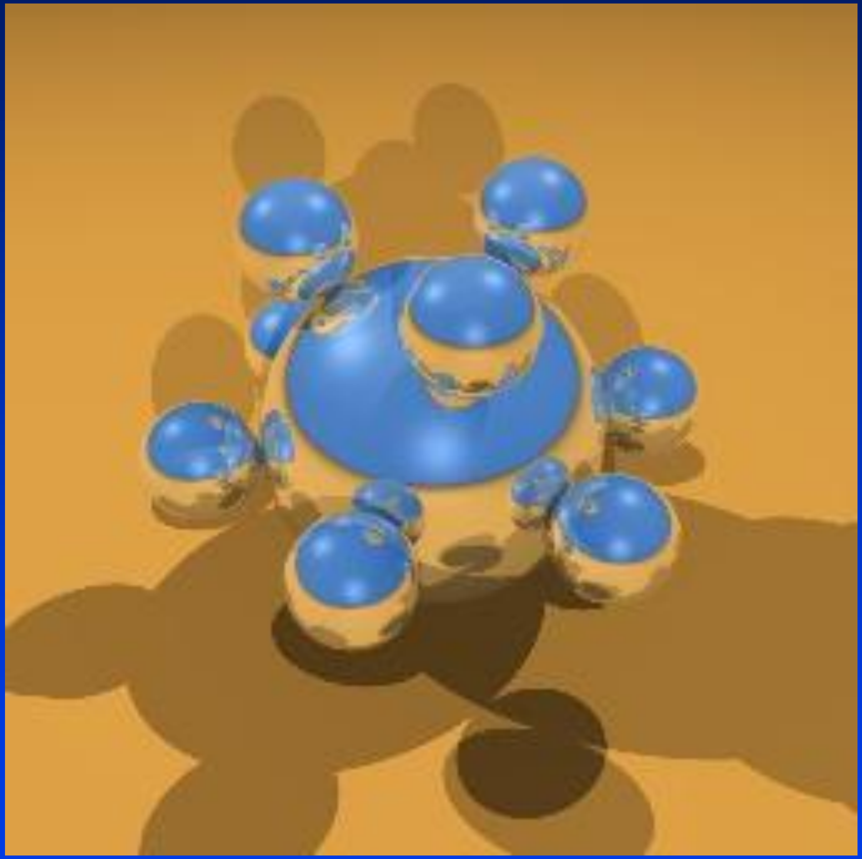
- The path from the intersection to the light source is known as the **shadow ray**
- If any object intersects the shadow ray between the surface and the light source then the surface is in shadow with respect to that source

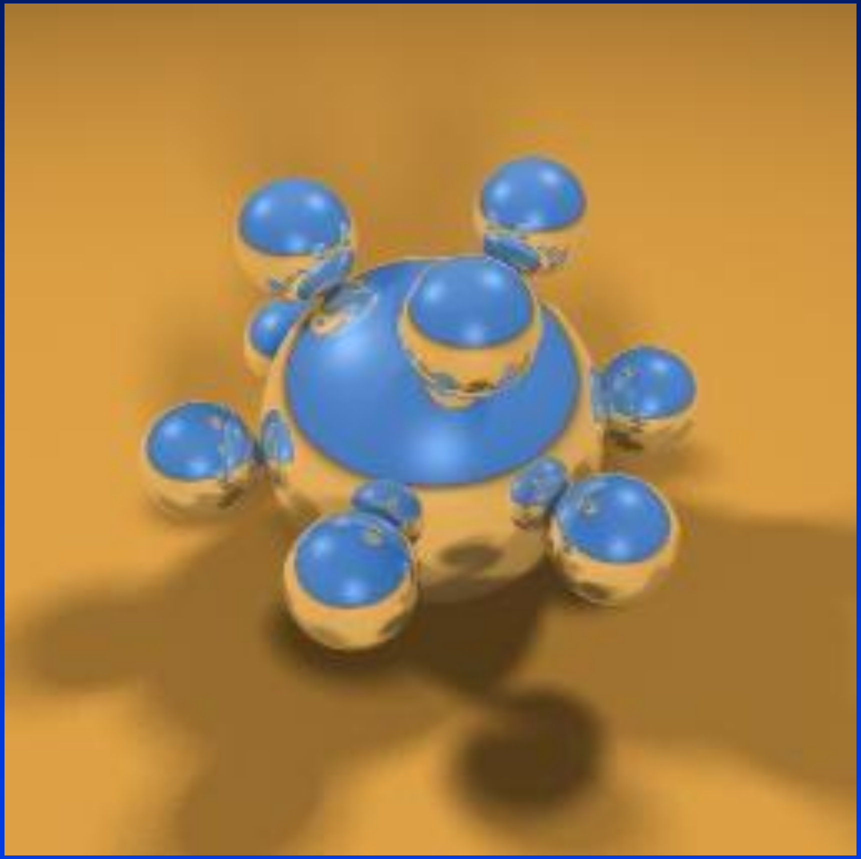
Shadow Ray





Shadow Examples





Shadow Examples

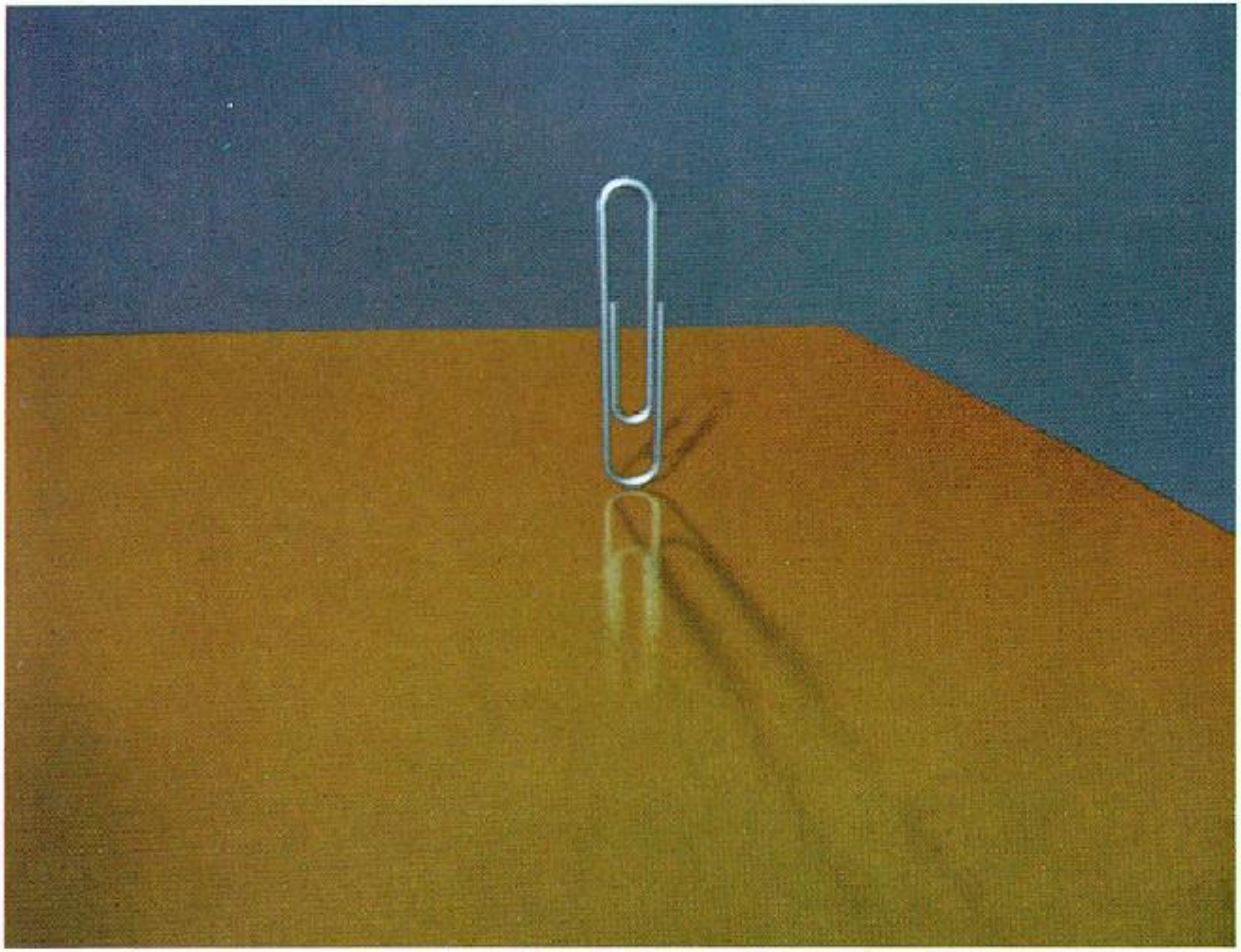
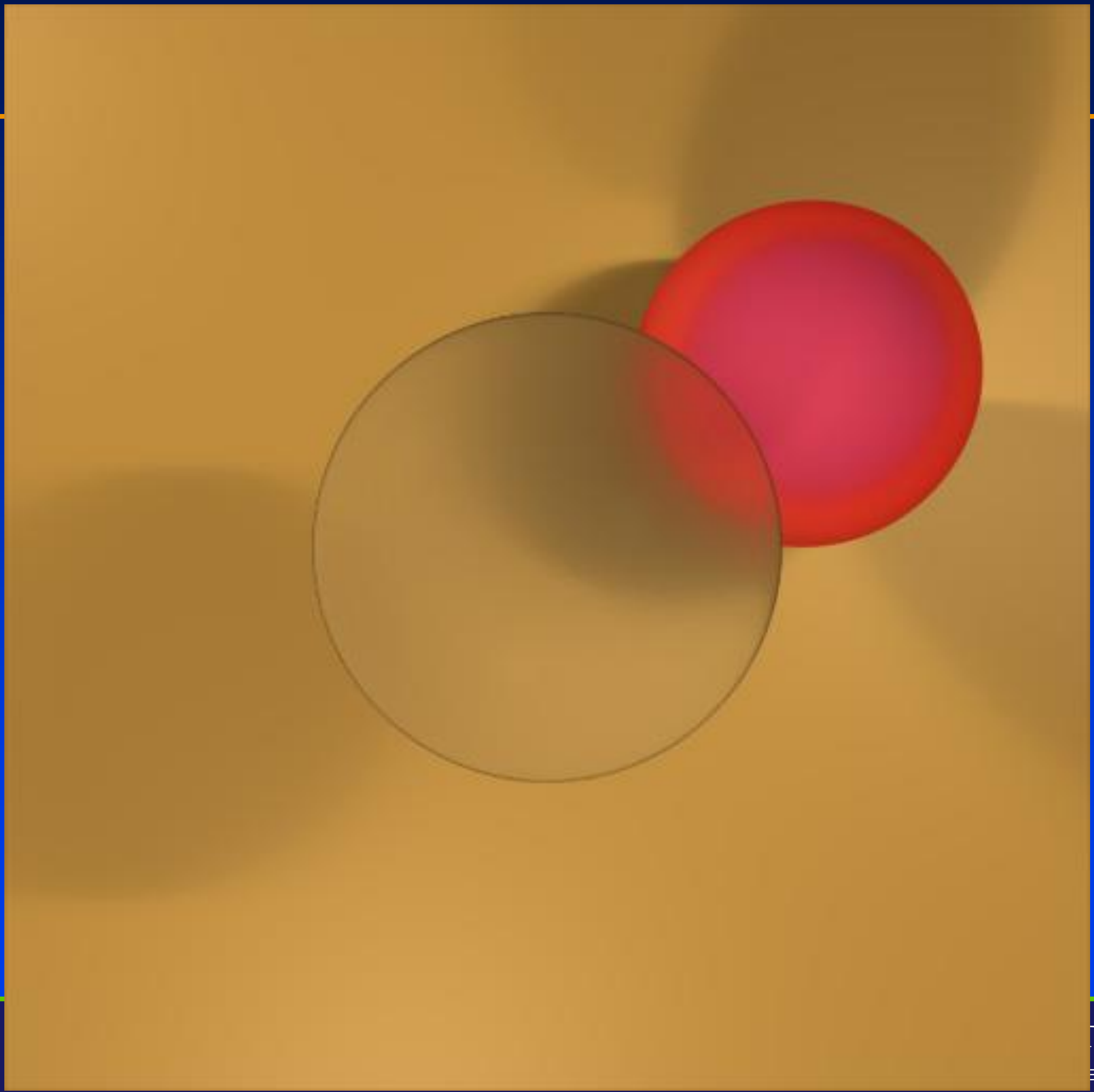
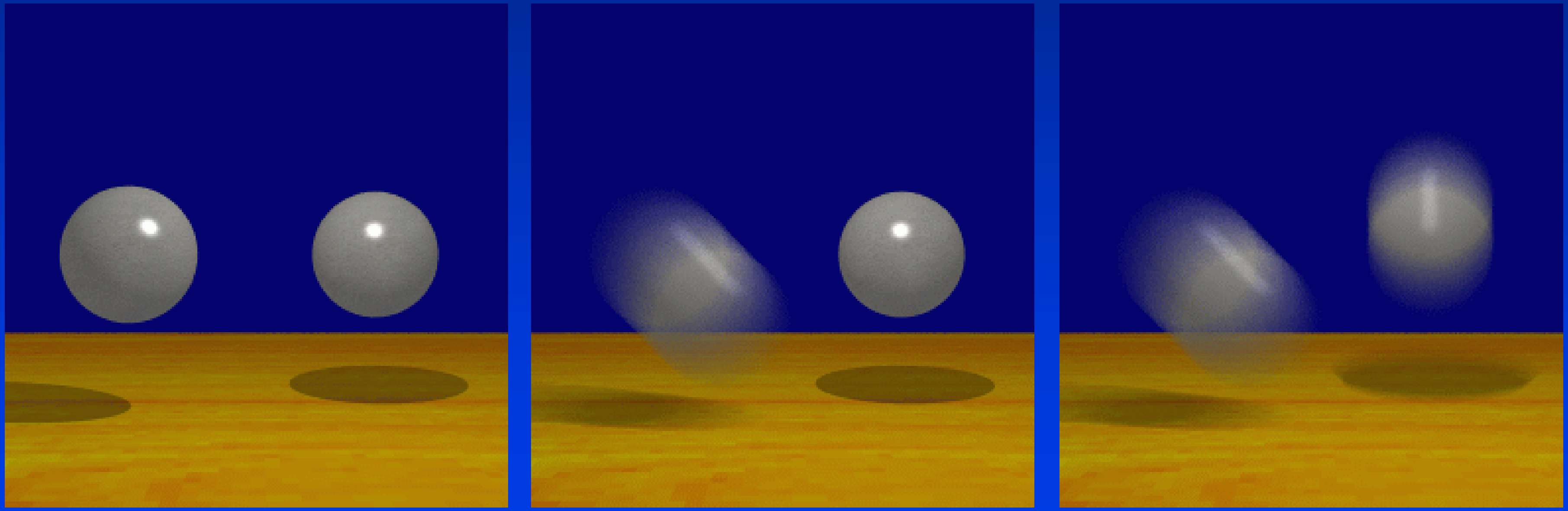


Fig. 17. Example of penumbrae and blurry reflection.



Motion Blurring




1
4

Cook (

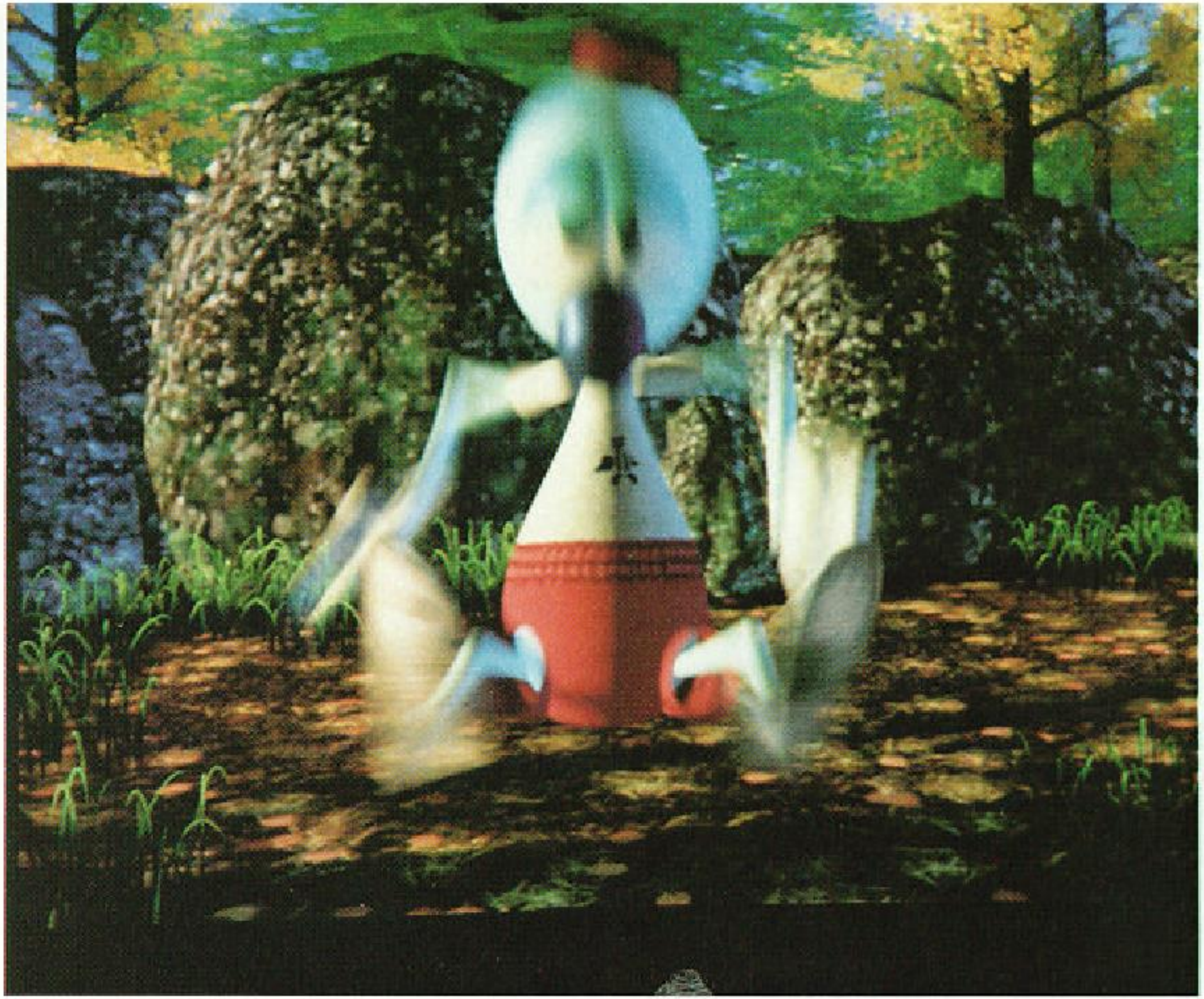
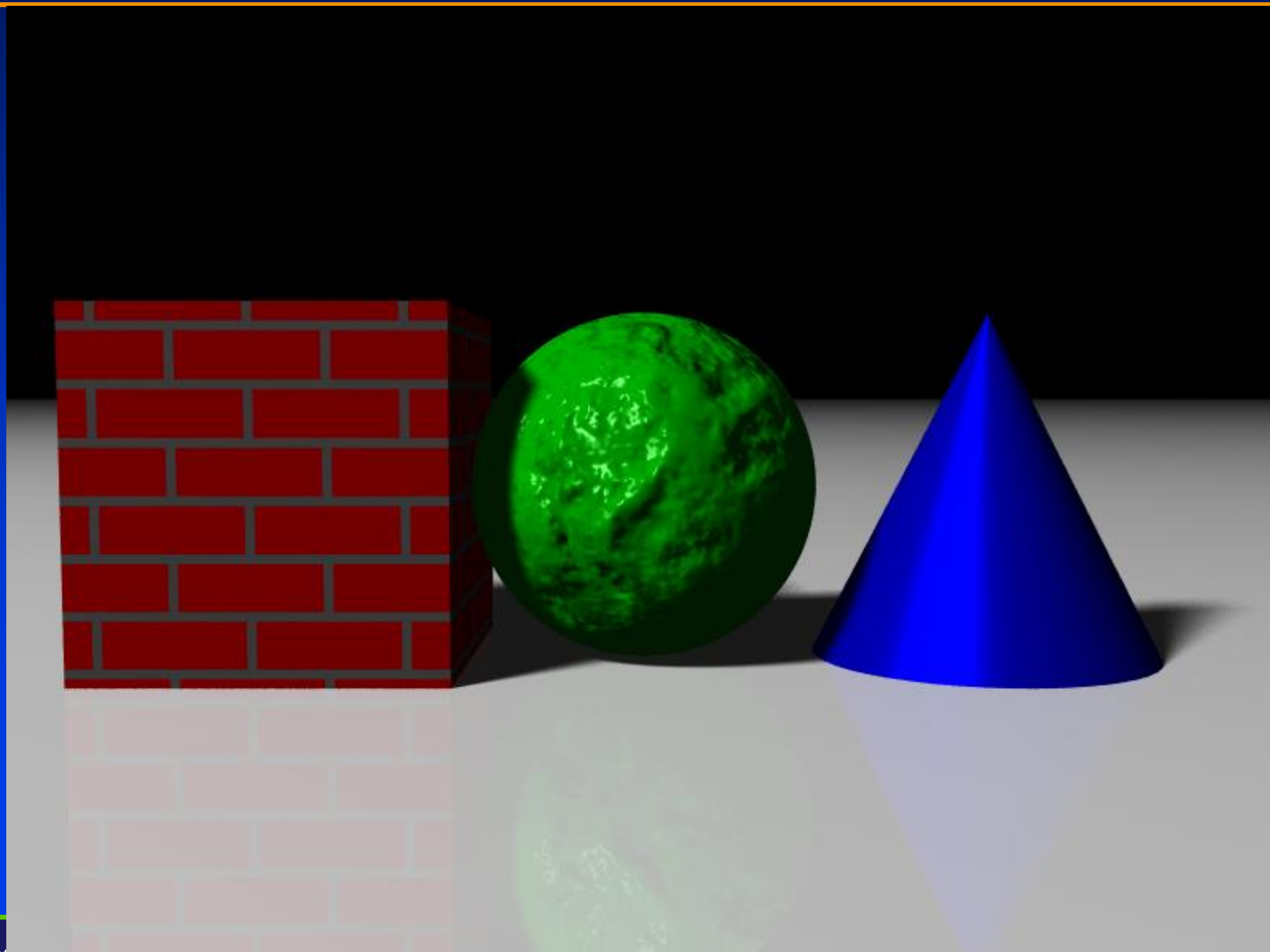


Fig. 15a. Example of motion blur from *The Adventures of André & Wally B.*

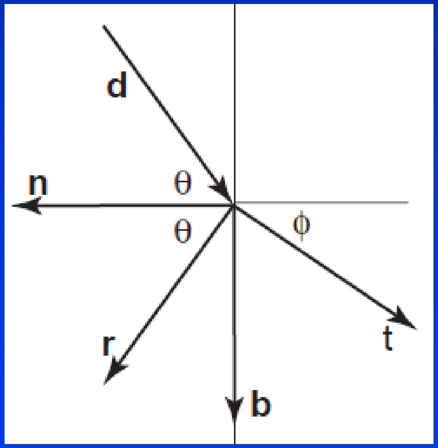
Cook (1986)

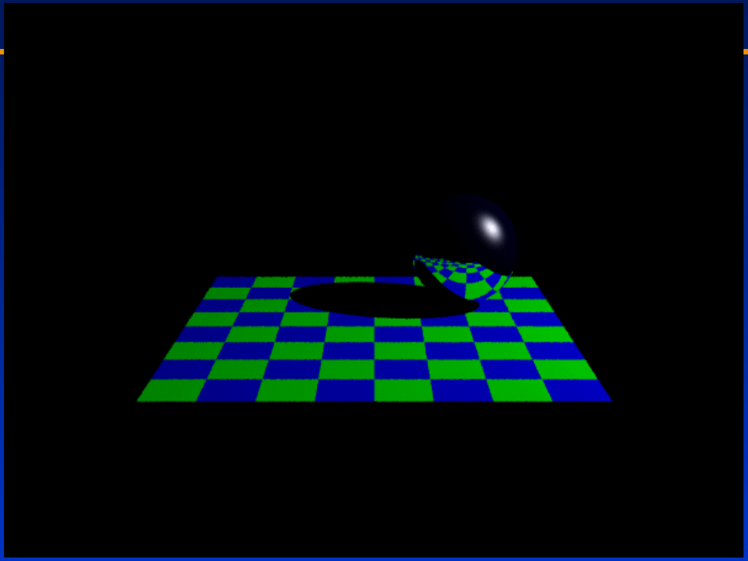
Fig. 15b. Example of motion blur from *The Adventures of André & Wally B.*

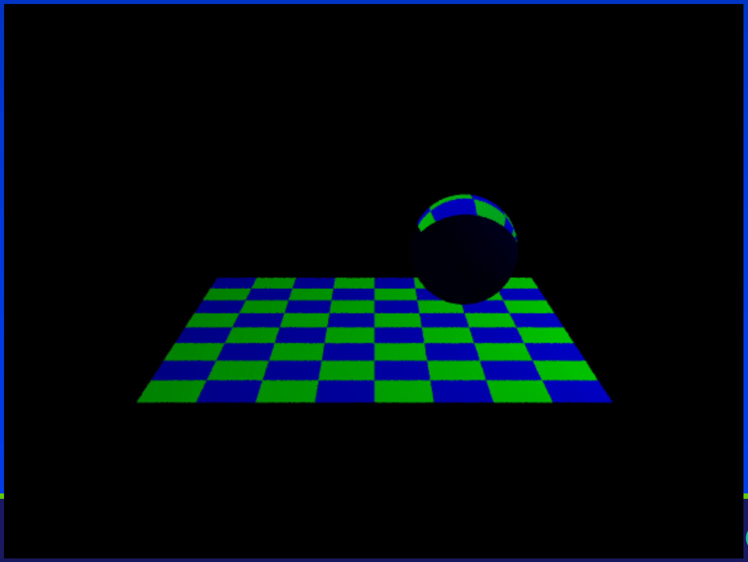
POV-Ray



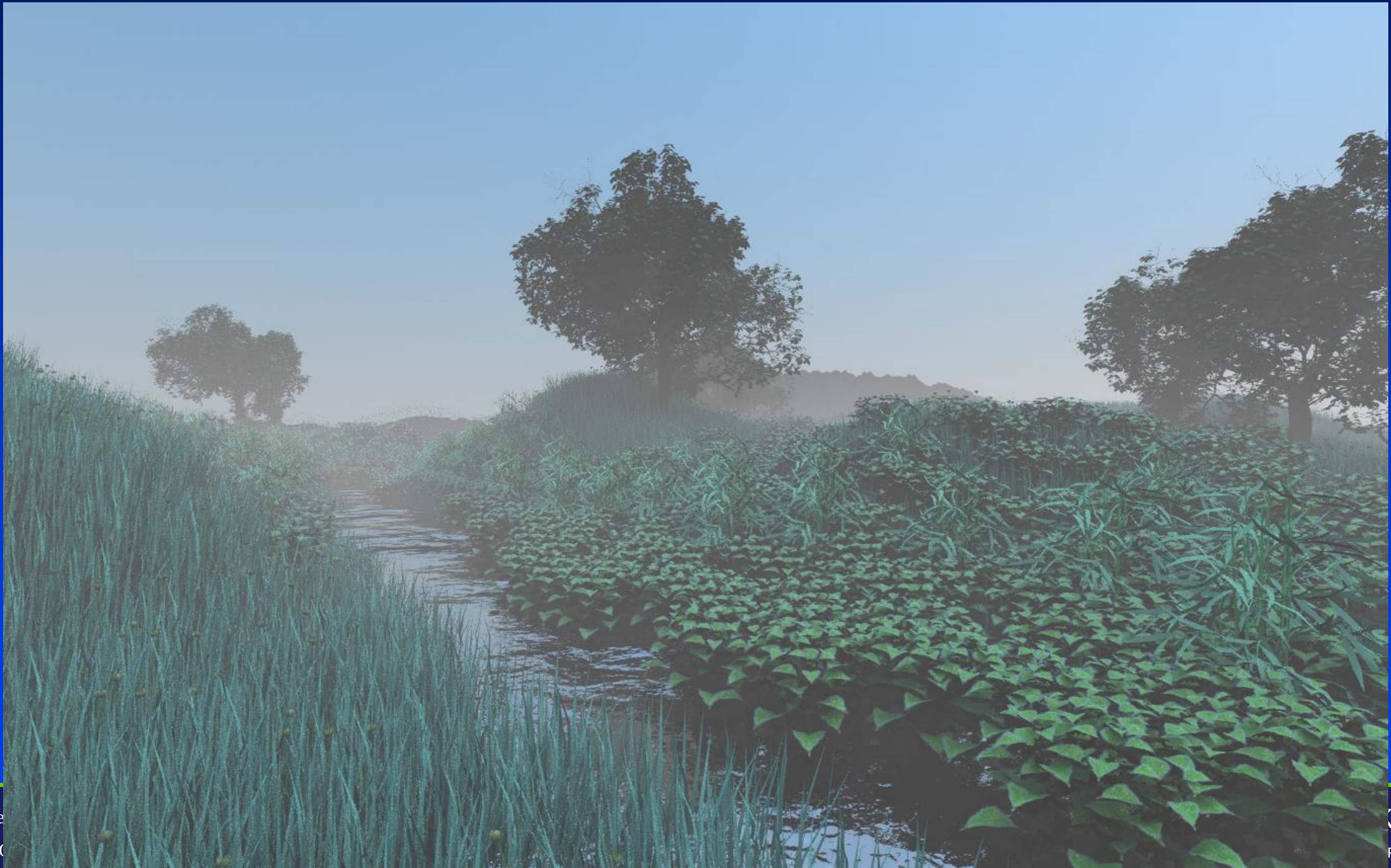
Ray-Tracing







Fog



Diffuse Surfaces

- Theoretically the scattering at each point of intersection generates an infinite number of new rays that should be traced
- In practice, we only trace the transmitted and reflected rays but use the Phong model to compute shade at point of intersection
- Radiosity works best for perfectly diffuse (Lambertian) surfaces

Radiosity

- Ray tracing:
  - Models specular reflection easily
  - Diffuse lighting is more difficult
- *Radiosity* methods explicitly model light as an energy-transfer problem
  - Models diffuse inter-reflection easily
  - Shiny, specular surfaces more difficult

Introduction: Radiosity

- First lighting model: Phong
  - Still used in interactive graphics
  - Major shortcoming: local illumination!
- After Phong, two major approaches:
  - Ray Tracing
  - Radiosity

Introduction: Radiosity

- Ray Tracing: ad hoc approach to simulating optics
  - Deals well with specular reflection
  - Trouble with diffuse illumination
- Radiosity: theoretically rigorous simulation of light transfer
  - Very realistic images
  - But makes simplifying assumption: *only* diffuse interaction!

Introduction: Radiosity

- Ray Tracing:
  - Computes a *view-dependent* solution
  - End result: a picture
- Radiosity:
  - Models only diffuse interaction, so can compute a *view-independent* solution
  - End result: a 3-D model

Fundamentals of Radiosity

- Theoretical foundation: heat transfer
- Need system of equations that describes surface interreflections
- Simplifying assumptions:
  - Environment is closed
  - All surfaces are *Lambertian* reflectors

Radiosity

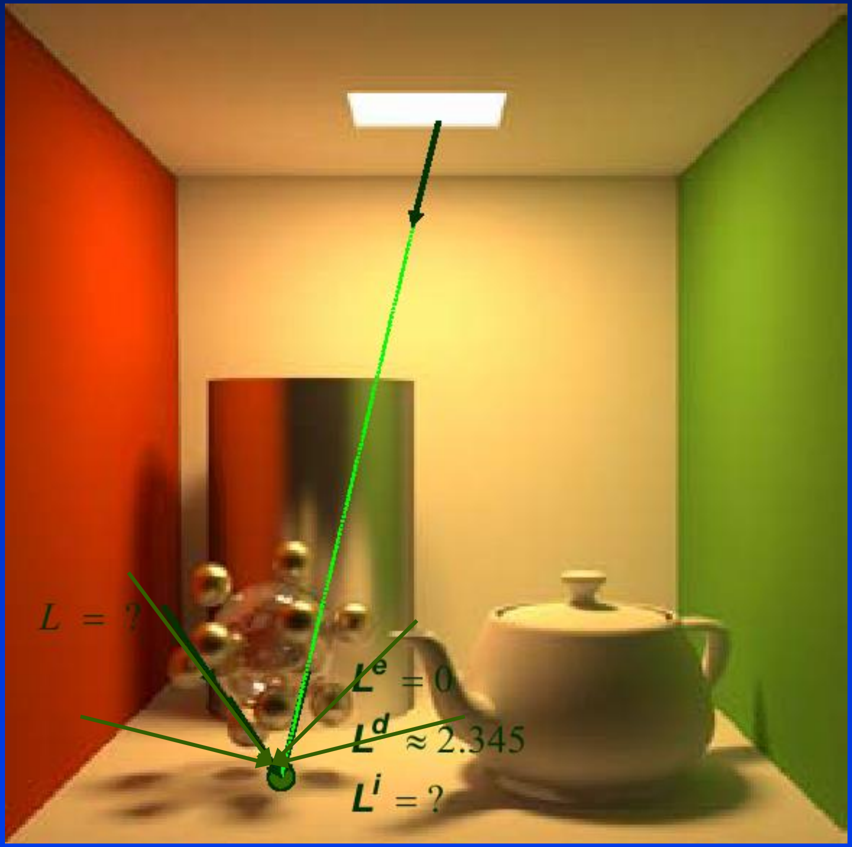
- Basic idea: represent surfaces in environment as many discrete *patches*
- A patch, or *element*, is a polygon over which light intensity is constant

Radiosity

- The *radiosity* of a surface is the rate at which energy leaves the surface
- Radiosity = rate at which the surface *emits* energy + rate at which the surface *reflects* energy
  - Notice: previous methods distinguish light sources from surfaces
  - In radiosity all surfaces can emit light
  - Thus: all emitters inherently have area

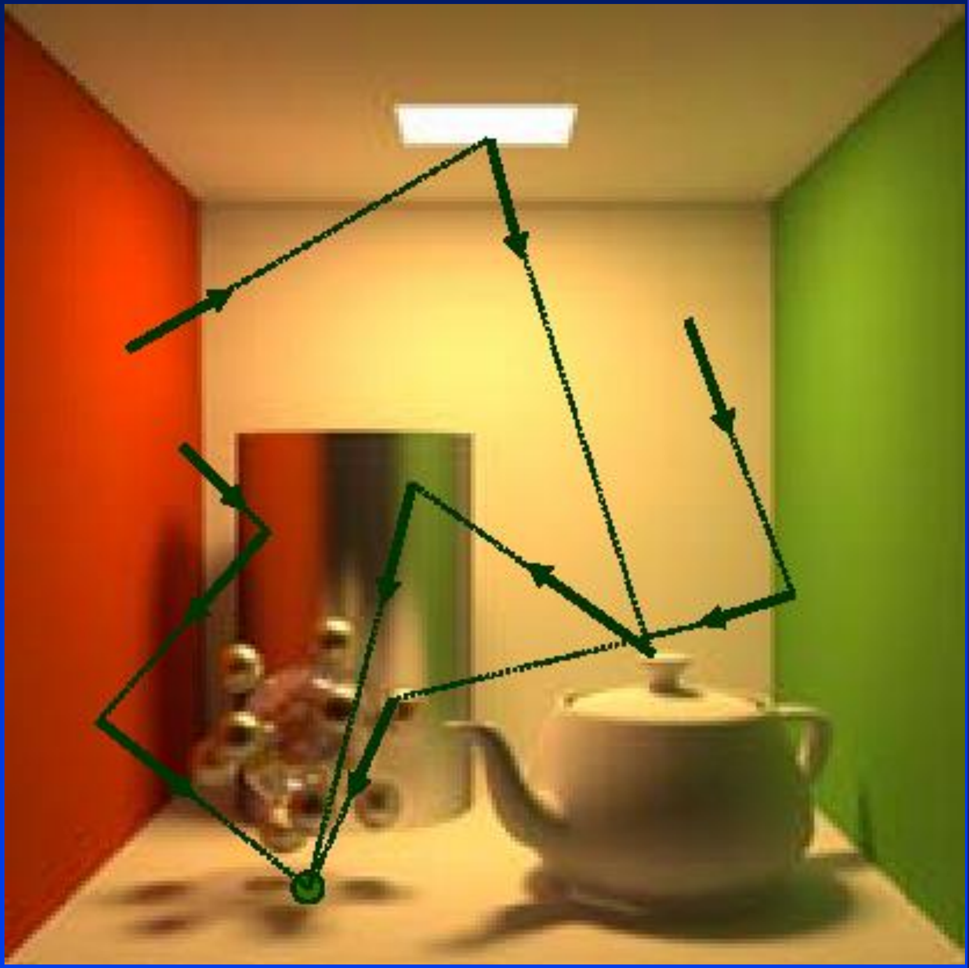
Global Illumination

- Path tracing



Global Illumination

- path tracing



Questions?

- Bezier Patches?

  or

- Triangle Mesh?