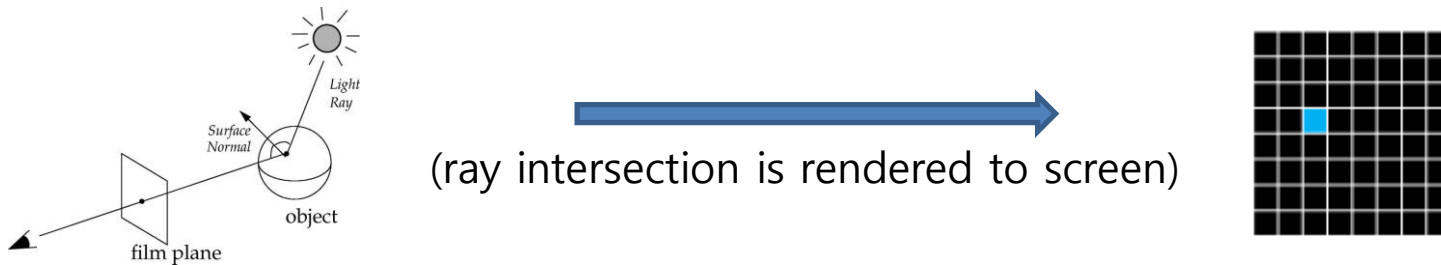


Topics Covered

- Ray Tracing
 - Ray Types
 - Recursion in Ray Tracing
 - Ray-Object Intersection
- Discussion on Ray Tracing
 - As Hidden Surface Removal
 - As Global Illumination
 - Acceleration Techniques
 - Path Tracing
- Recent Trend in Ray Tracing

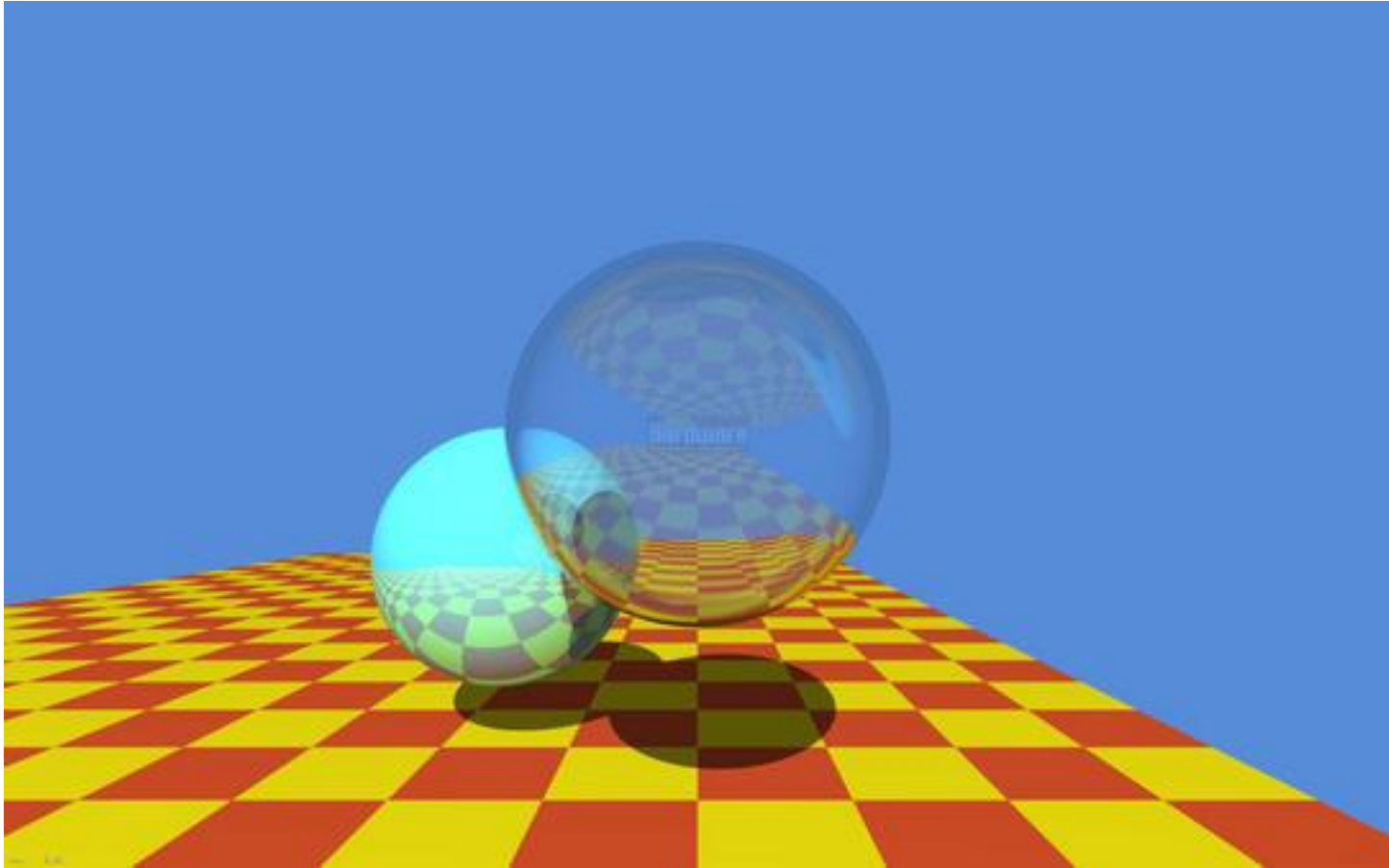
Recall: Two Approaches to Rendering - 2

for **each pixel** in image(film plane)
determine which object should be shown at the pixel
set color of the pixel based on texture and lighting model



- Called **image-oriented rendering** or **ray tracing**

Ray Tracing

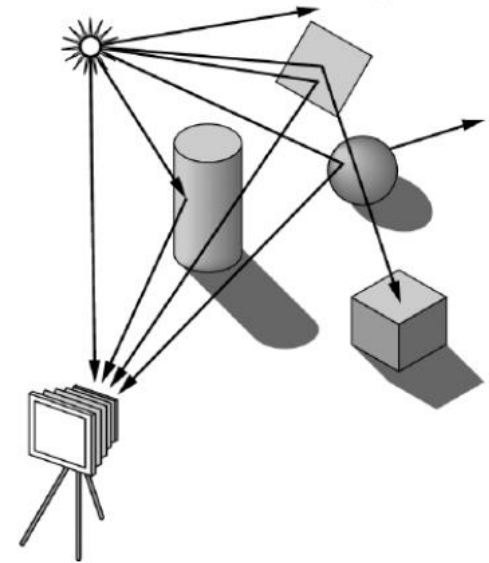


Turner Whitted, 1980



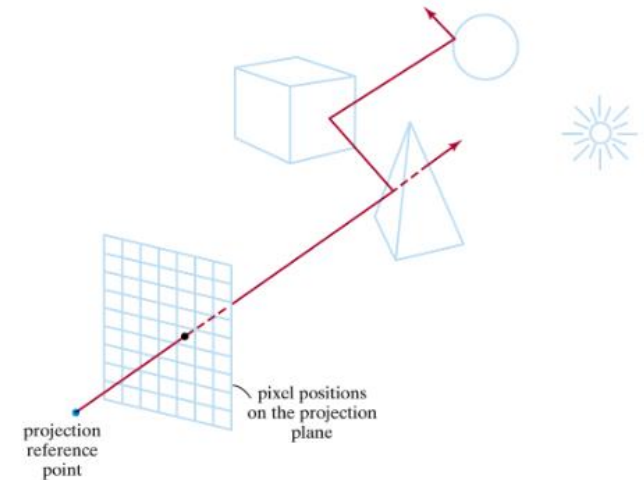
Forward Ray Tracing

- Simulate lights in nature
- A light source emits *photons*
- Follow photons using rays – as paths of photons
- Problem: Many rays will not contribute to image
- Highly inefficient!



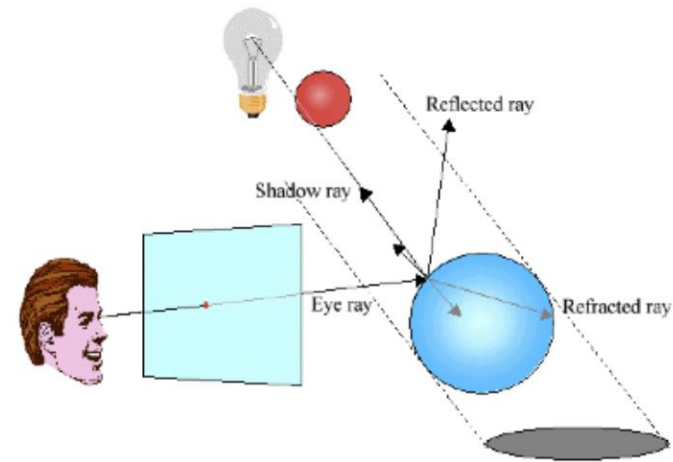
Backward Ray Tracing

- Trace rays backward from viewer to light sources
- Each ray goes through each pixel in image plane from eye position
- Color of each pixel is determined based on which object the ray intersects with
- Much more efficient than forward approach - we can only consider rays that contribute image!
- Actually, this approach is what we call “Ray Tracing”



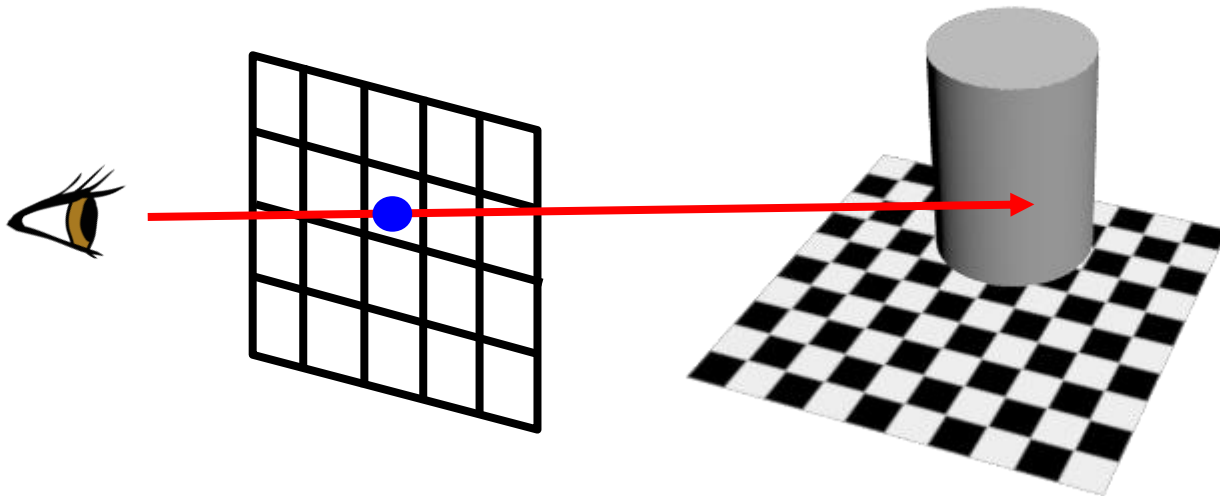
Types of Rays

- Eye rays
 - from eye to surface, passing through each pixel
- Shadow (Illumination) rays
 - from surface point to light source
- Reflection rays
 - from surface point in mirror direction
- Refraction rays
 - from surface point in refracted direction



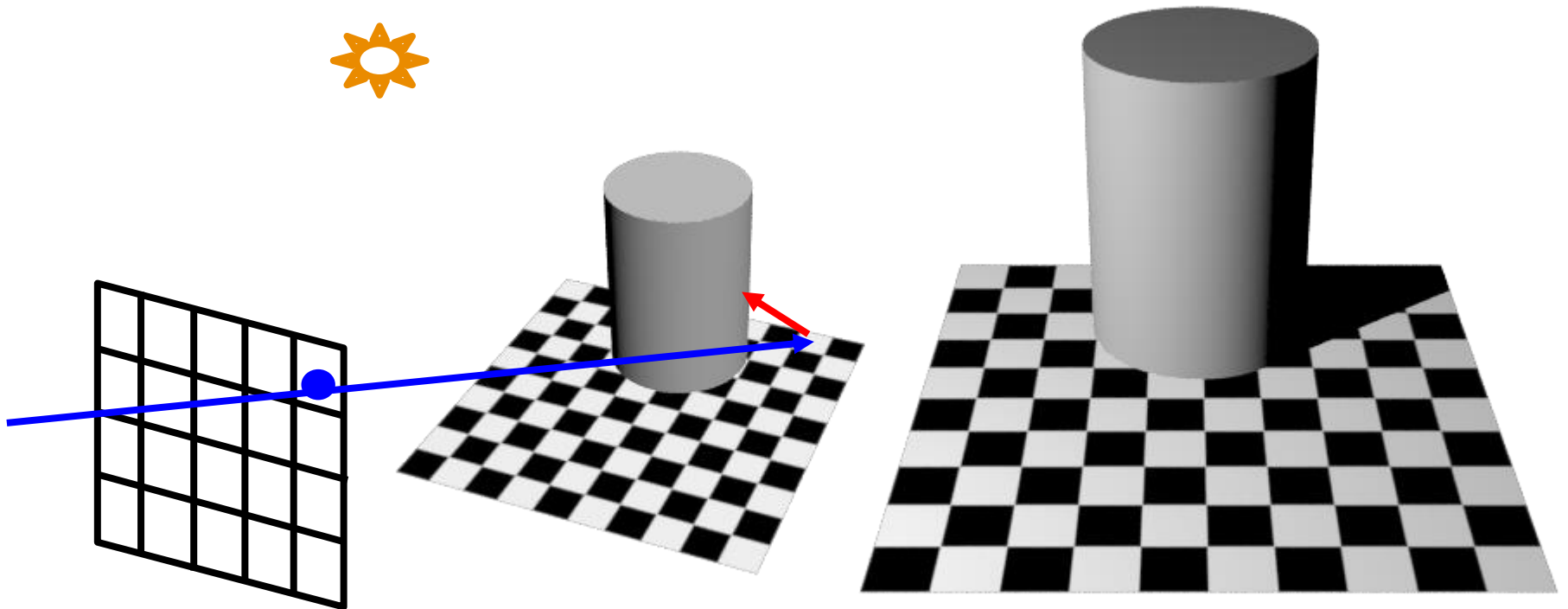
Eye Rays

- Casted from eye (or camera) to surface, passing through a pixel
- Find closest surface point hit by the ray



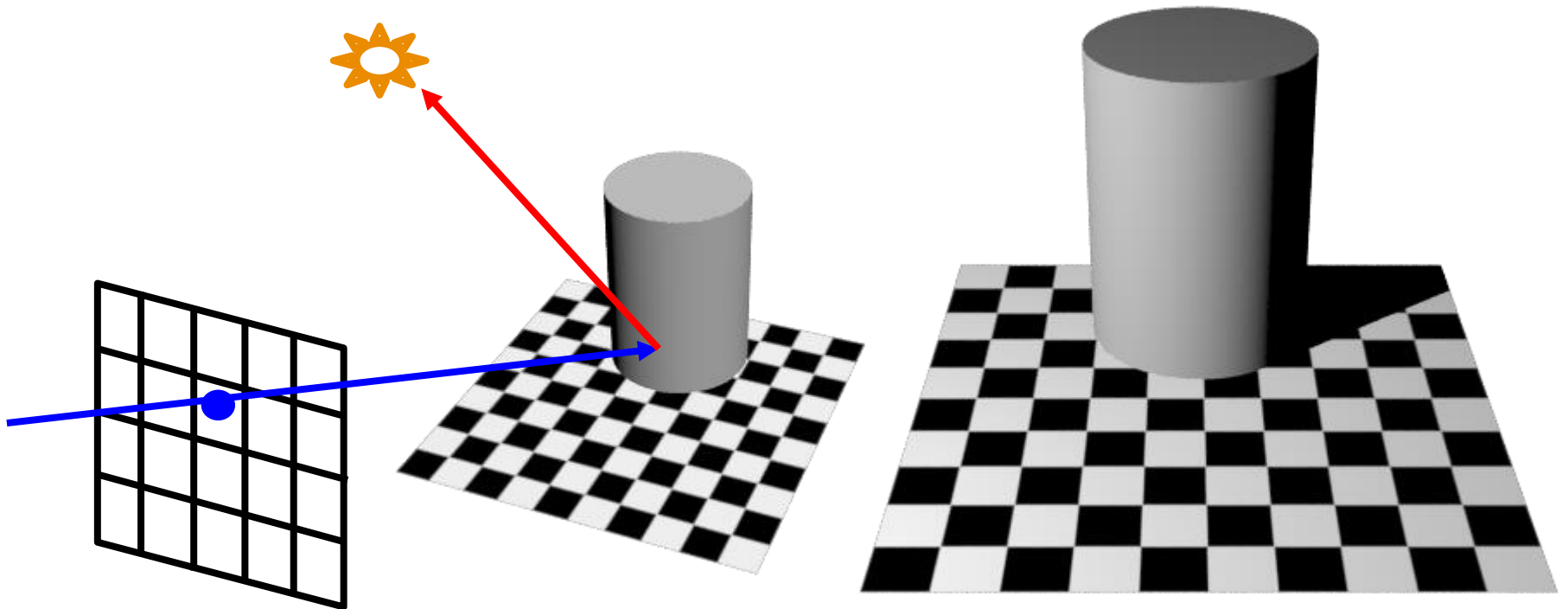
Shadow (Illumination) Rays

- Casted from surface point to *each* light source
 - If the ray is **blocked** by an opaque object, no contribution of the light for the pixel color (shadow)



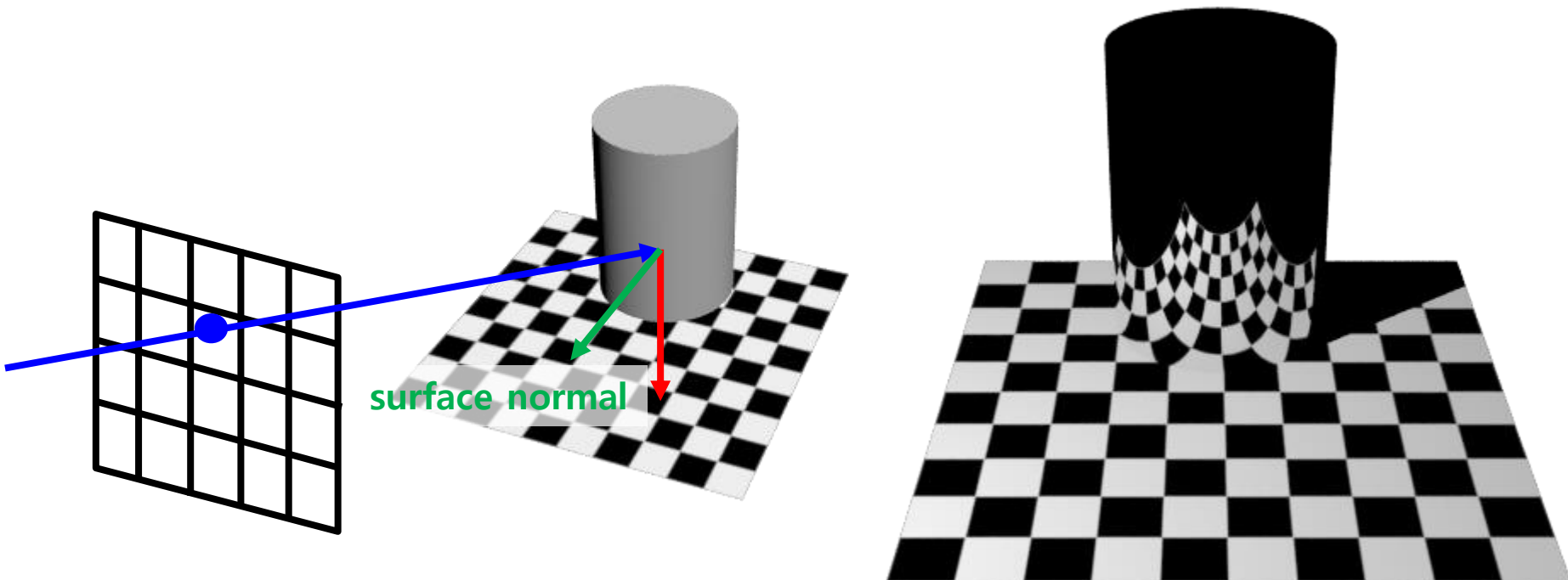
Shadow (or Illumination) Rays

- Casted from surface point to *each* light source
 - If the ray is **blocked** by an opaque object, no contribution of the light for the pixel color (shadow)
 - If the ray **reaches the light**, compute the contribution of the light for the pixel color using local illumination model (such as Phong model)



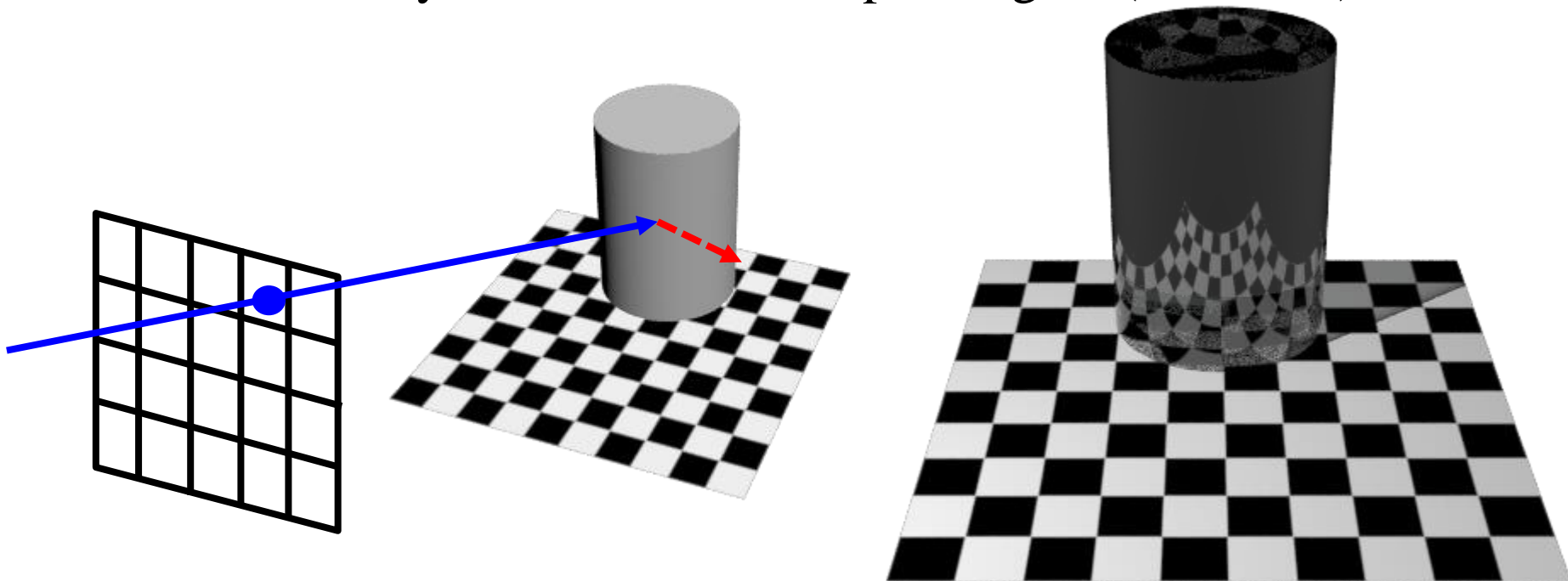
Reflection Rays

- Casted from surface point in mirror direction if the surface is specular (following the laws of reflection)
- If this ray reaches other surfaces, cast shadow / reflection / refraction rays from that surface point again (recursive)



Refraction Rays

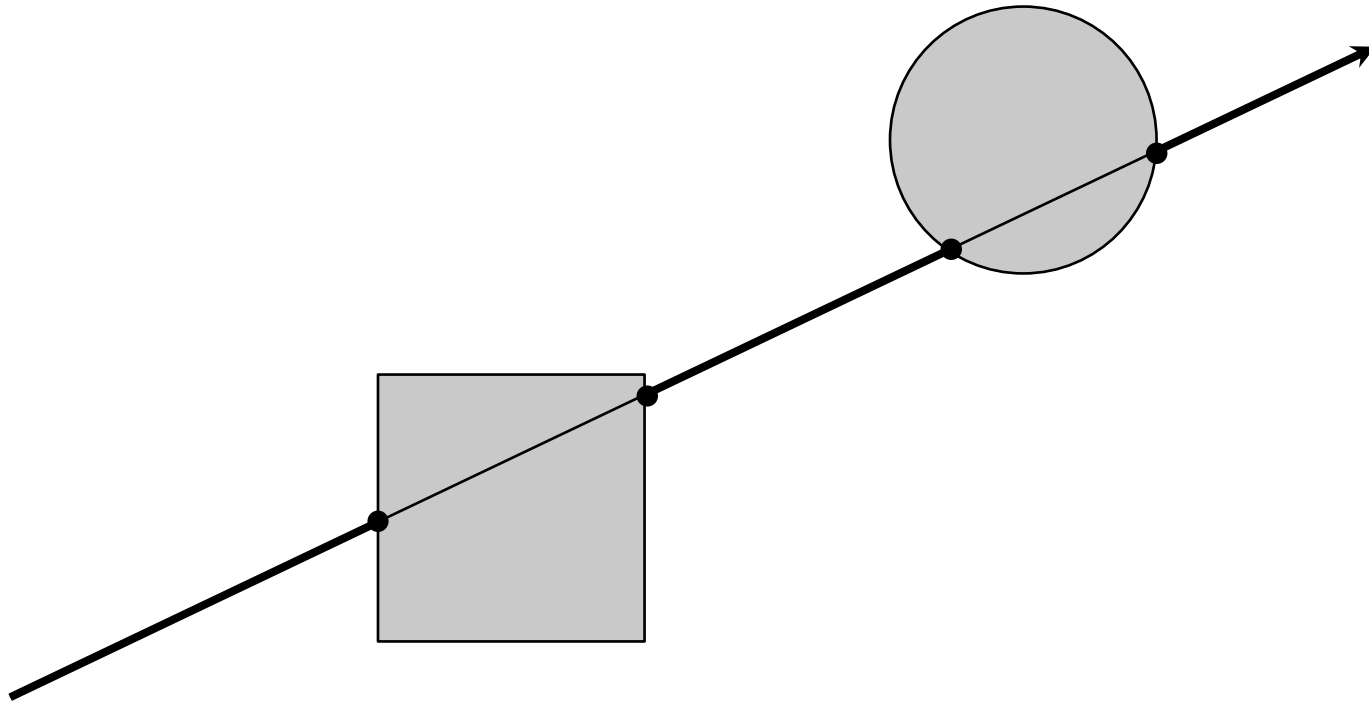
- Casted from surface point in refracted direction if the surface is transparent (following Snell's law)
- If this ray reaches other surfaces, cast shadow / reflection / refraction rays from that surface point again (recursive)



Recursion in Ray Tracing

- Reflection & refraction rays recursively spawn new shadow, reflection, refraction rays at each intersection with object surface until
 - contribution is negligible
 - or some max recursion depth is reached
- Rays are attenuated with
 - Specular reflectivity of object surface
 - Opacity of transparent material
 - Distance traveled through transparent material

Ray intersection

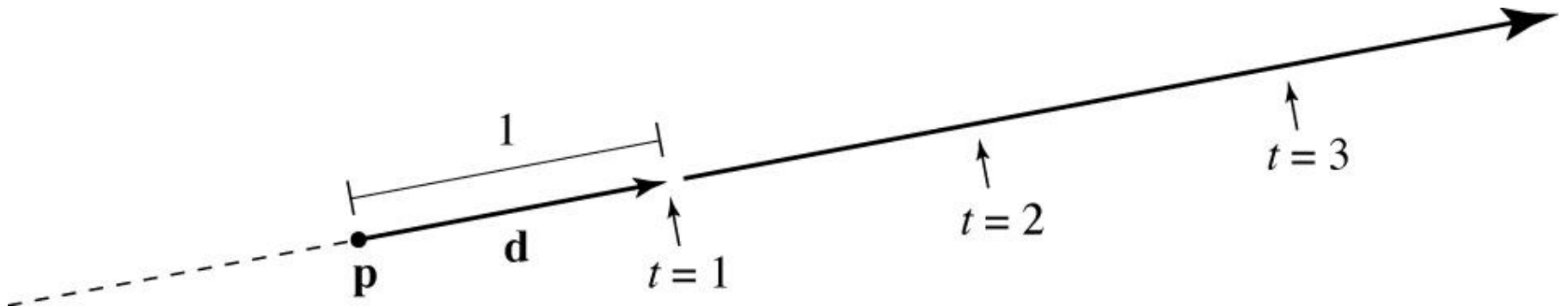


Ray: a half line

- Standard representation: point \mathbf{p} and direction \mathbf{d}

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

- this is a *parametric equation* for the line



Simple Strategy

- **Parametric ray equation**

- Gives all points along the ray as a function of the parameter

$$\vec{p}(t) = \vec{o} + t \vec{d}$$

- **Implicit surface equation**

- Describes all points on the surface as the zero set of a function

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

- **Substitute ray equation into surface function and solve for t**

$$f(\vec{o} + t \vec{d}) = 0$$

Ray-sphere intersection: algebraic

- Condition 1: intersection point r is on ray

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

- Condition 2: point is on sphere
 - assume unit sphere;

$$\|\mathbf{x}\| = 1 \Leftrightarrow \|\mathbf{x}\|^2 = 1$$

$$f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{x} - 1 = 0$$

- Substitute:

$$(\mathbf{p} + t\mathbf{d}) \cdot (\mathbf{p} + t\mathbf{d}) - 1 = 0$$

- this is a quadratic equation in t

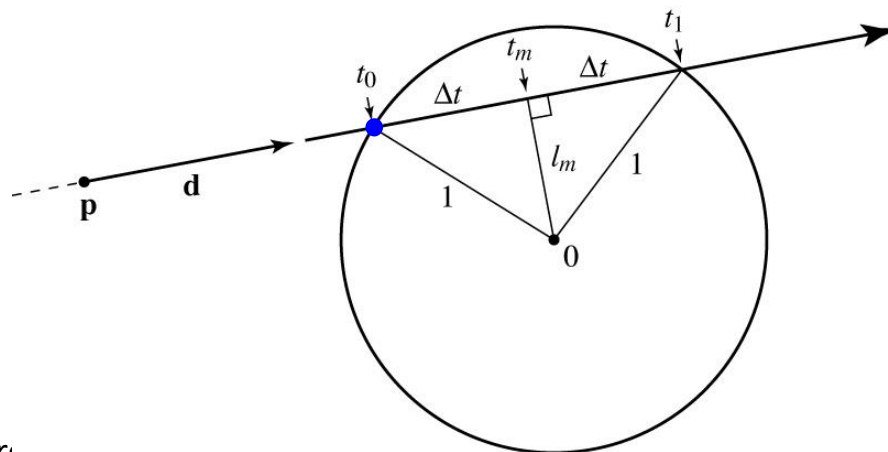
Ray-sphere intersection: algebraic

- Solution for t by quadratic formula:

$$t = \frac{-\mathbf{d} \cdot \mathbf{p} \pm \sqrt{(\mathbf{d} \cdot \mathbf{p})^2 - (\mathbf{d} \cdot \mathbf{d})(\mathbf{p} \cdot \mathbf{p} - 1)}}{\mathbf{d} \cdot \mathbf{d}}$$

$$t = -\mathbf{d} \cdot \mathbf{p} \pm \sqrt{(\mathbf{d} \cdot \mathbf{p})^2 - \mathbf{p} \cdot \mathbf{p} + 1}$$

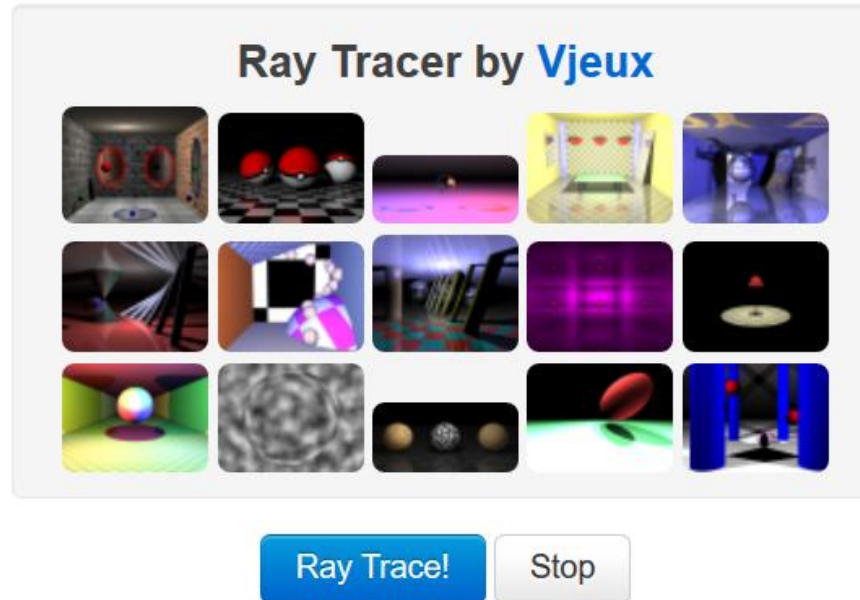
- Choose the smallest positive t



Ray-Object Intersection

- Similarly, we can find ray intersection with
 - Box
 - Plane
 - Triangle (which means we can use not only implicit surfaces but also polygon meshes in ray tracing)
- I'll just skip more details of ray-object intersection

[Practice] Ray Tracing Online Demo

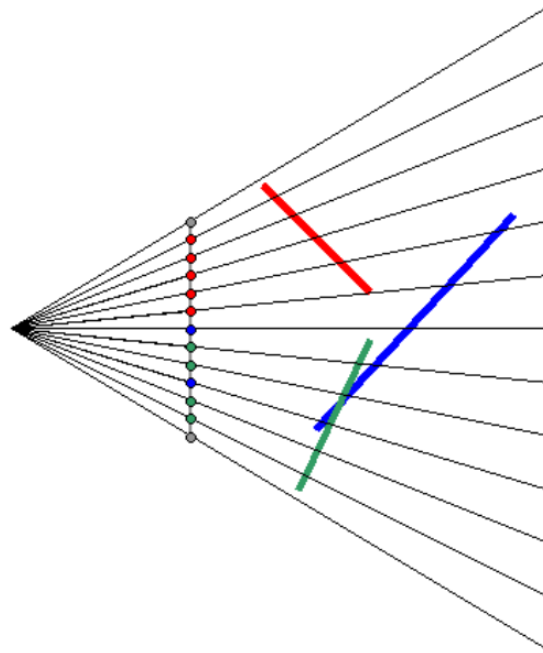


<http://foo.fr/~vjeux/epita/raytracer/raytracer.html#portal>

- Change the scene and click “Ray Trace!” button

Discussion: Ray Tracing as Hidden Surface Removal

- Ray-object intersection solves “Hidden Surface Removal” problem
 - Finding closest surface point hit by the ray



Discussion: Ray Tracing as Global Illumination

- Computing each pixel color (lighting) in ray tracing
 - Use local illumination model to calculate **direct** contribution from light sources (with shadow rays)
 - Recursively compute **indirect** contribution from reflection / refraction (with reflection / refraction rays)
- Final pixel color is the sum of these contributions
 - So ray tracing can be considered as global (indirect) illumination model
- No diffuse reflection rays → Ray tracing is *limited approximation* to global illumination

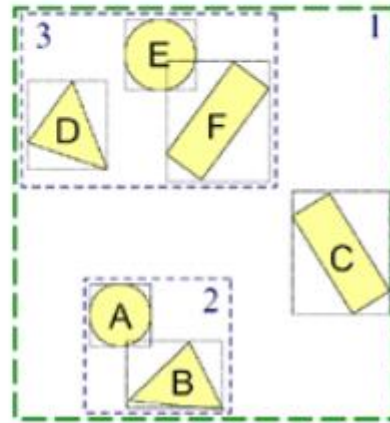
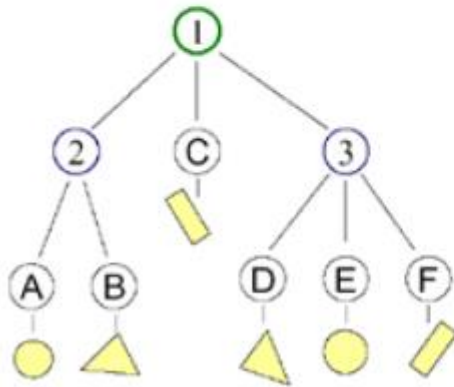
Discussion: Acceleration Techniques for Ray Tracing

- Ray tracing is slow!
 - Spend most of time in ray-object intersection
 - Proportional to (the number of pixels) \times (the number of primitives in the scene)
- To reduce the number of ray-object test,
 - Bounding volume hierarchies
 - Spatial subdivision

Discussion: Acceleration Techniques for Ray Tracing

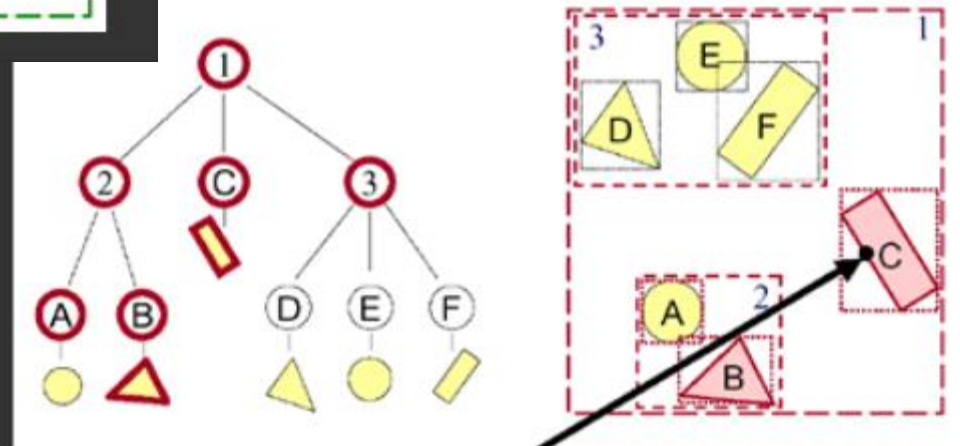
Bounding Volume Hierarchies 1

- Build hierarchy of bounding volumes
 - Bounding volume of interior node contains all children



Bounding Volume Hierarchies 2

Use hierarchy to accelerate ray intersections
Intersect node contents only if hit bounding volume



Traditional Use of Rasterization

- Generally, ray-tracing is slower than rasterization, but generates better quality results
- Thus, rasterization have been dominant for real-time application
 - A single frame should be rendered in a few tens of milliseconds
 - e.g. OpenGL or Direct3D for computer games

Traditional Use of Ray Tracing

- But moviemakers can take as long as they like to render a single frame
- So ray-tracing-based rendering is dominant for non-realtime applications
 - e.g. movies, animations
 - Generally rendered in offline using large CPU cluster, called *render farm*
 - Pixar, ILM, Weta Digital, ...
- Similar but much more improved technique, *path tracing*, captures diffuse scattering (with hundreds of thousands rays in a pixel), so it can generate photo-realistic images
 - Used not only by film makers, but also by architects





Recent Trend in Ray Tracing

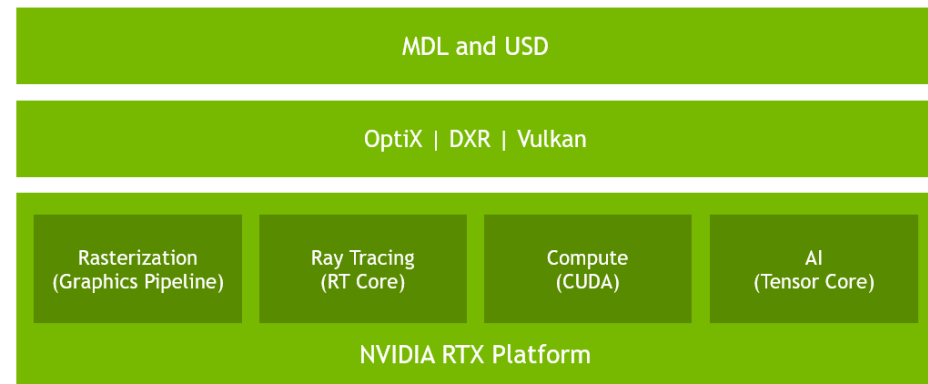
- Modern ray tracers have started to use GPUs to speed up the computation
 - Arnold(Autodesk), V-Ray(Chaos Group), Renderman(Pixar), ...
- Even real-time ray tracing engines have been announced
- Perhaps, ray tracing based games will become popular..?

Recent Trend in Ray Tracing

- NVIDIA OptiX
 - A software development kit for achieving high performance ray tracing on the GPU
 - Announced at SIGGRAPH 2009
 - <https://developer.nvidia.com/optix>
- Microsoft DirectX® Raytracing (DXR)
 - Fully integrates ray tracing into DirectX, and makes it a companion to rasterization
 - Announced on March 19, 2018
 - <https://blogs.msdn.microsoft.com/directx/2018/03/19/announcing-microsoft-directx-raytracing/>

Recent Trend in Ray Tracing

- NVIDIA RTX™ Platform
 - Ray Tracing (OptiX, Microsoft DXR, Vulkan)
 - AI-Accelerated Features (NGX)
 - Rasterization (Advanced Shaders)
 - Simulation (CUDA 10, PhysX, Flex)
 - Asset Interchange Formats (USD, MDL)
 - <https://developer.nvidia.com/rtx>
 - <https://youtu.be/tjf-1BxpR9c>



Acknowledgement

- Acknowledgement: Some materials come from the lecture slides of
 - Prof. Andy van Dam, Brown Univ., <http://cs.brown.edu/courses/csci1230/lectures.shtml>
 - Prof. Jehee Lee, SNU, http://mrl.snu.ac.kr/courses/CourseGraphics/index_2017spring.html
 - Prof. Sung-eui Yoon, KAIST, <https://sglab.kaist.ac.kr/~sungeui/CG/>
 - Prof. Taesoo Kwon, Hanyang Univ., <http://calab.hanyang.ac.kr/cgi-bin/cg.cgi>
 - Prof. Steve Marschner, Cornell Univ., <http://www.cs.cornell.edu/courses/cs4620/2014fa/index.shtml>