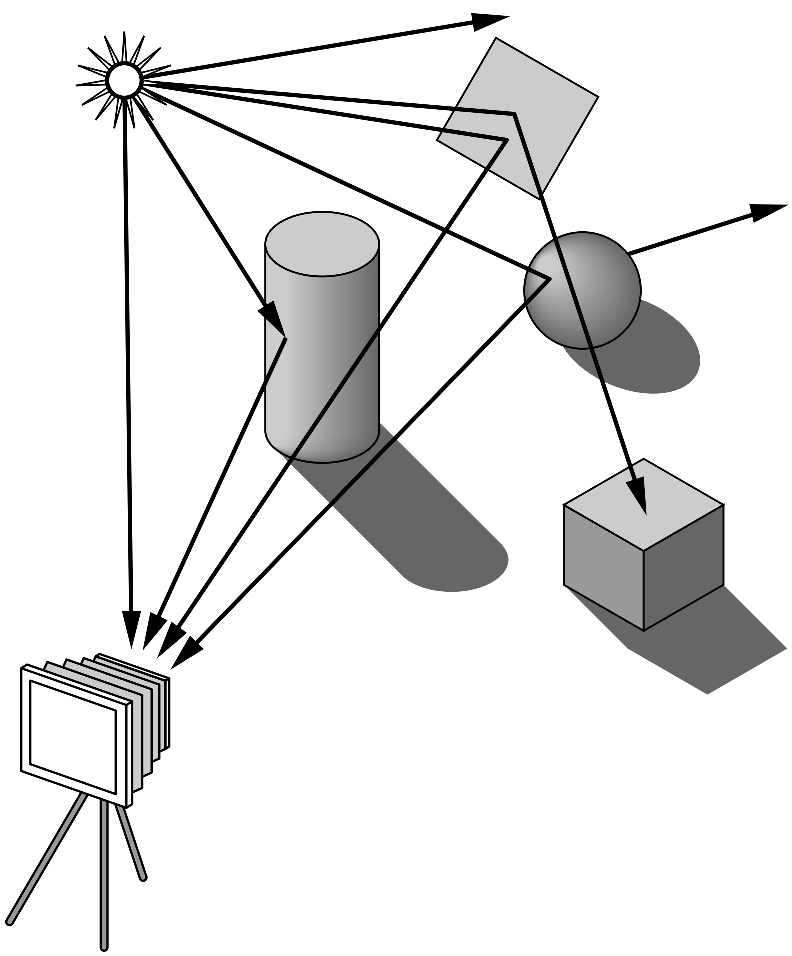
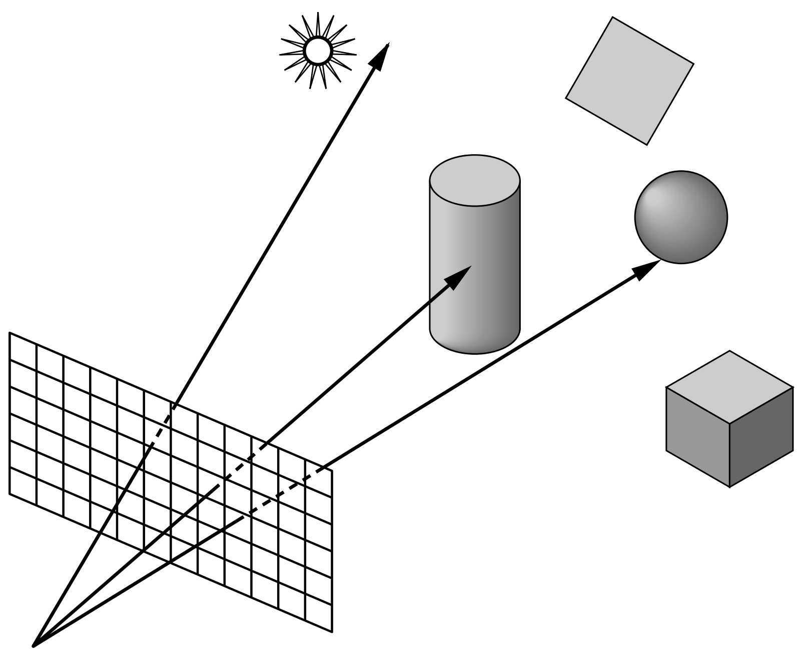
The problem



A solution

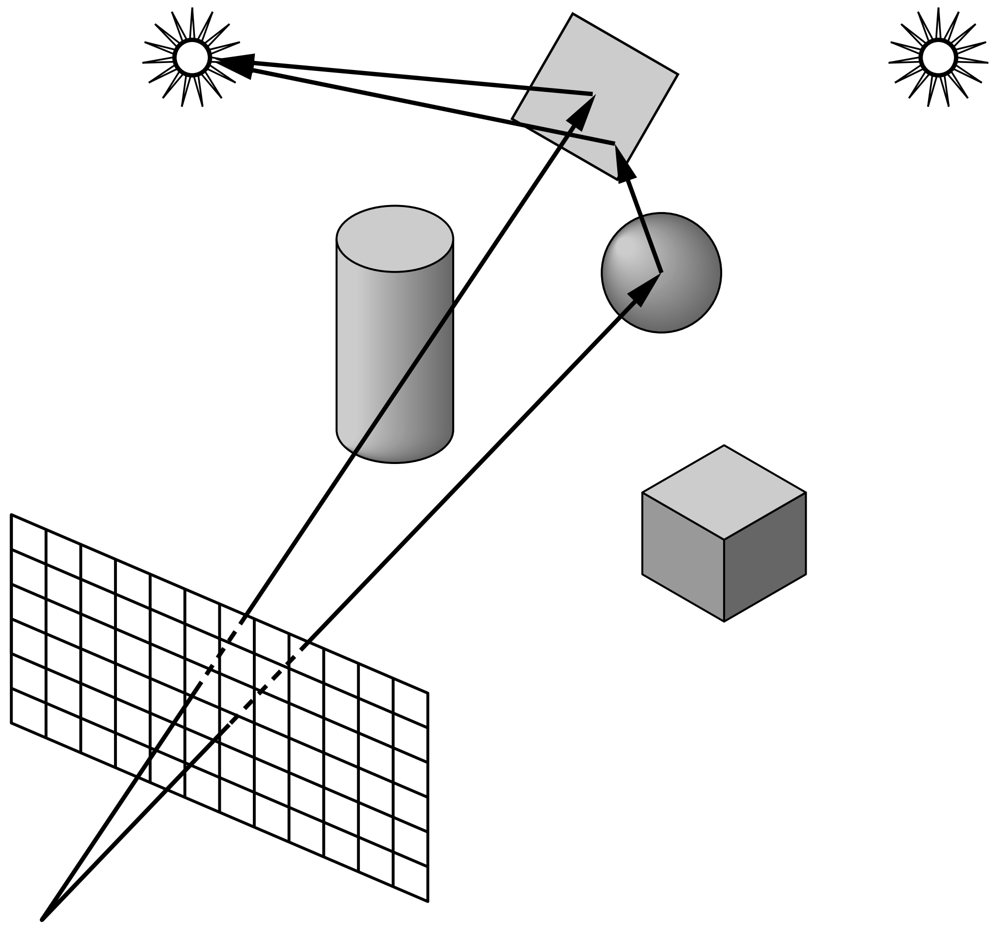


Ray Tracing

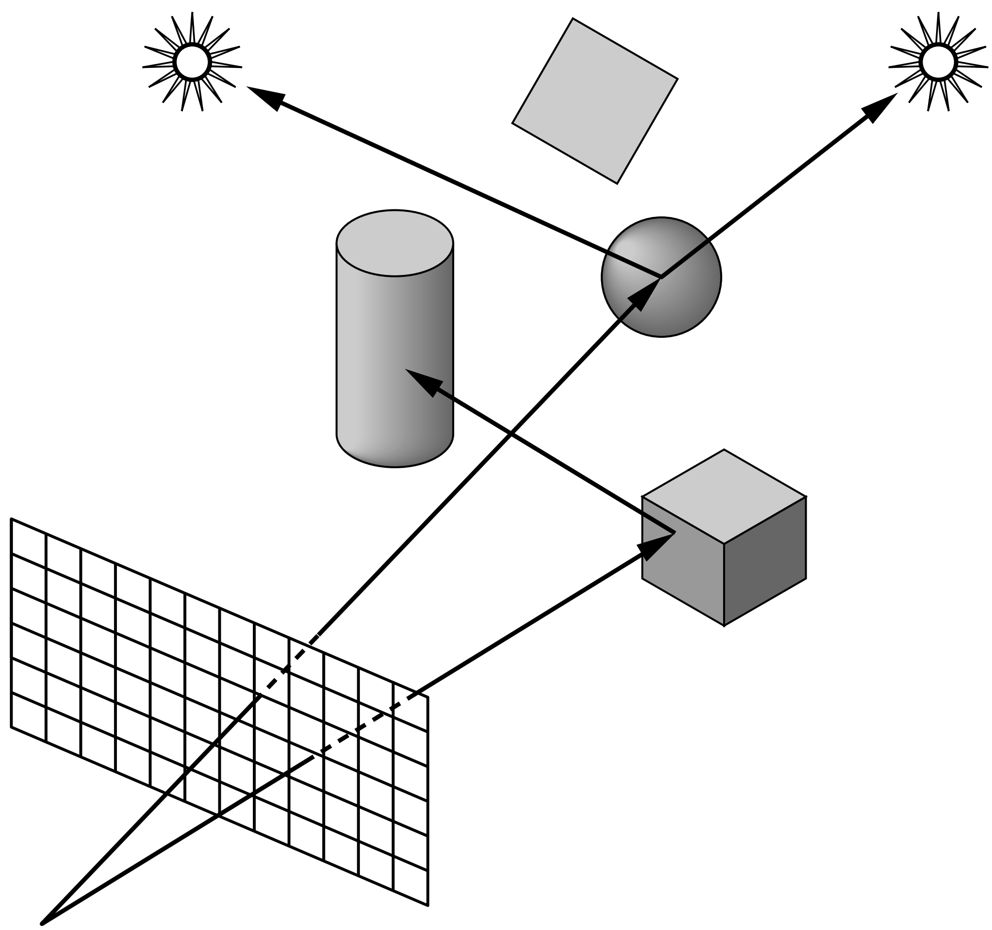
Hidden surface removal

- ☆ For each pixel π, shoot a ray ρ from the view point to the center of π.
- ☆ If ρ does not intersect any object, color π with the background color.
- ☆ Otherwise, compute the first object O intersected by ρ and the first intersection point σ.
- ☆ Compute the color at σ using the reflection model.
- ☆ Draw π with the computed color.
- ☆ Each pixel is colored only once.
- ☆ Computing σ is expensive!

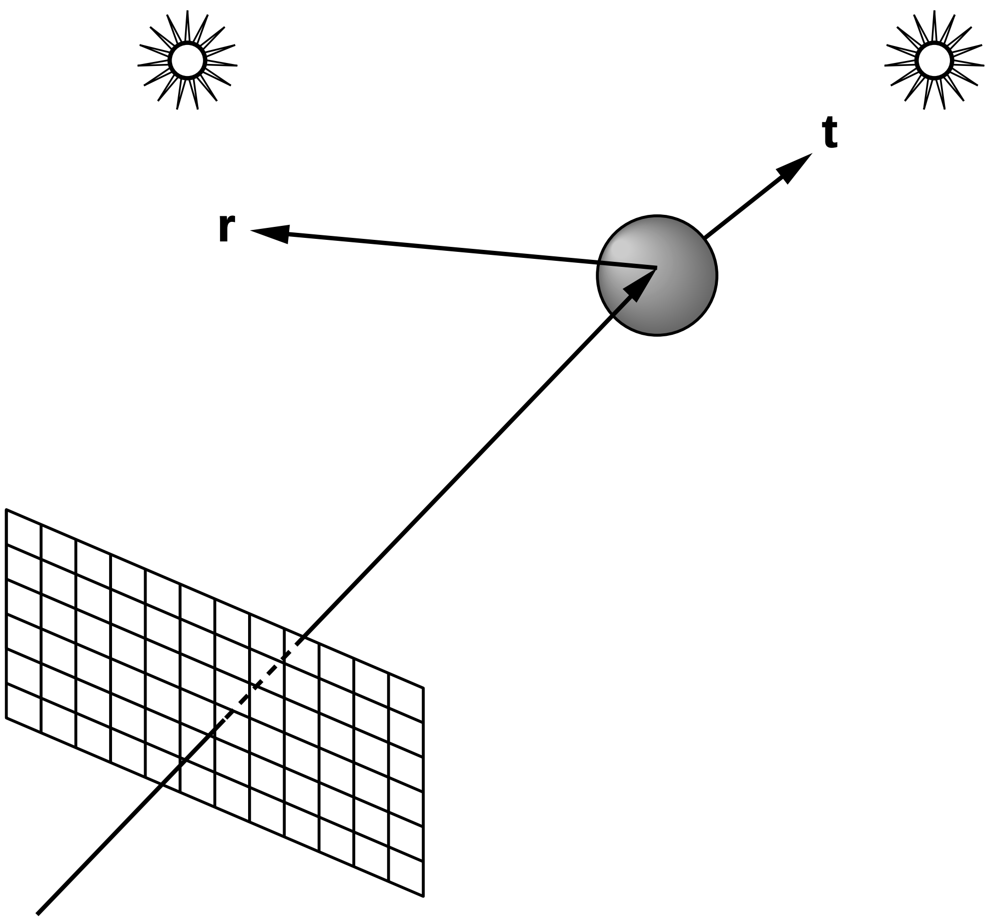
Ray Tracing: Reflection



Ray Tracing: Shadow Generation



Ray Tracing: Refraction

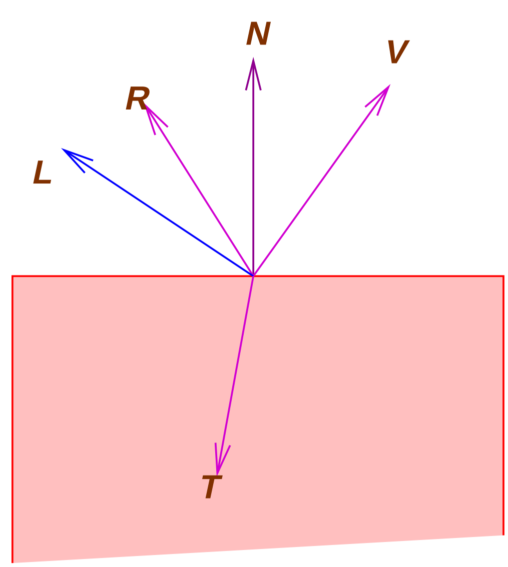


Recursive Ray Tracing

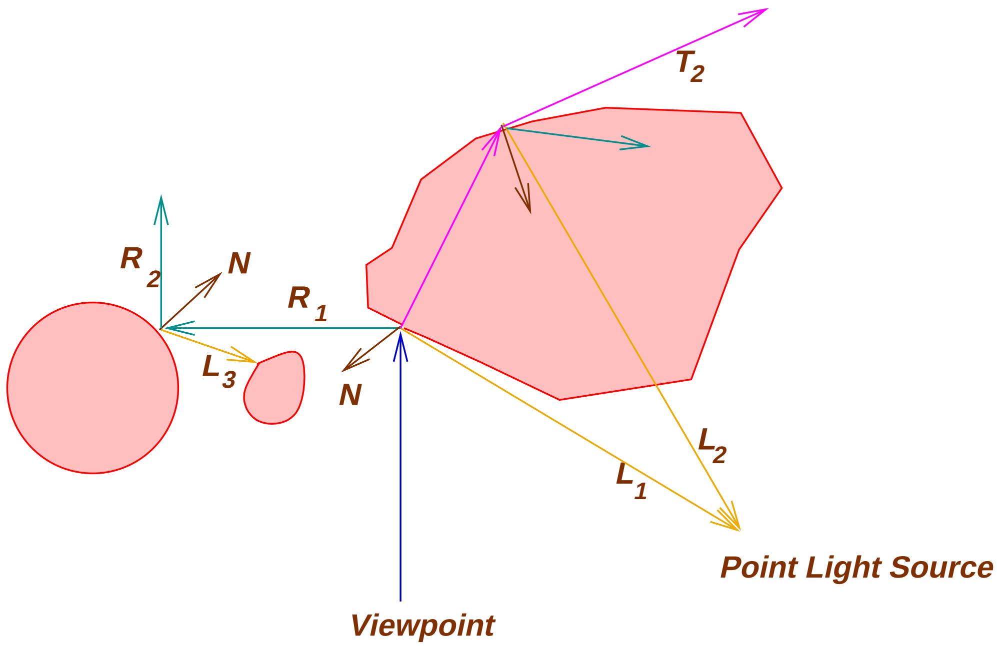
- ☆ Extend the standard ray tracing to handle shadows, reflection, and refraction.
- ☆ Shoot secondary rays recursively to calculate shadows, reflection, and refraction.

For each pixel π on the screen, do the following:

- ☆ *Primary ray (ρ_P):* Ray emanating from the viewer to the center of π.
- ☆ If ρ_P doesn't hit any object, render π with the background color.
- ☆ Suppose the first intersect point of ρ and an object is p.



Secondary Rays



- Shoot the following *secondary* rays from p:

  Shadow ray (ρ_L): Shoot a ray along $\vec{pL}$.

  Reflection ray (ρ_R): If the object has reflectance property (e.g., mirror), shoot a ray in direction R.

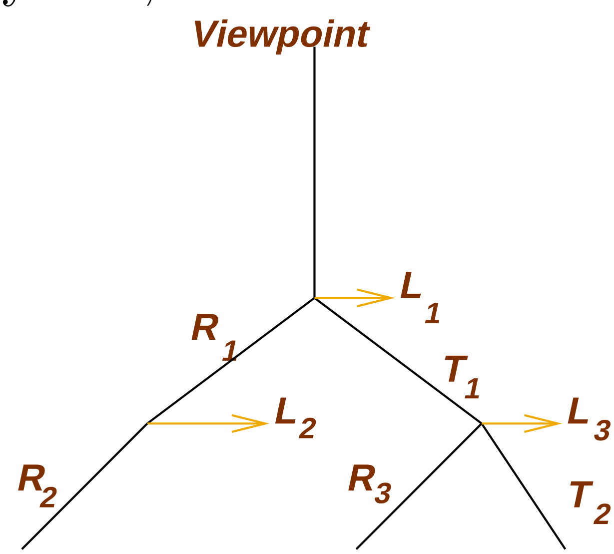
  Refraction ray (ρ_T): If the object is transparent, shoot a ray in direction T.

- If ρ_R, ρ_T hit an object, shoot secondary rays from there as above.
- Apply distance attenuation to the intensity of secondary rays.

Stopping Criteria

- ☆ No object is hit.
- ☆ Light source is hit.
- ☆ Reached a cut-off depth.

Creates a ray tree; evaluate in bottom-up fashion.



$$
\begin{aligned}
I_\lambda &= k_a I_{a\lambda} O_{a\lambda} + \\
&\sum_{i=1}^{k} S_i f_{att} I_{L_i\lambda} \left[k_d \cdot O_{d\lambda}(\mathbf{N} \cdot \mathbf{L}_i) + k_s (\mathbf{R} \cdot \mathbf{V})^n\right] \\
&+ k_R I_{R\lambda} + k_T I_{t\lambda}
\end{aligned}
$$

Pros and Cons

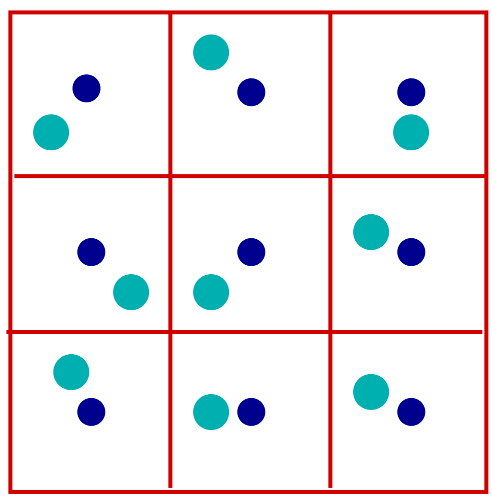
- ⋆ Better illumination model.
- ⋆ Prone to numerical instability.
- ⋆ Very expensive.

Efficiency Issues:

- ⋆ Ray object intersection: Use object hierarchy, spatial decomposition techniques (oct trees, BSP's).
- ⋆ Adaptive tree depth
- ⋆ Reflection maps
- ⋆ Light buffer

Distributed Ray Tracing

- ☆ Handles antialiasing.
- ☆ Divide pixel into subpixels.
- ☆ Choose pixels at random (under some given distribution).
- ☆ Divide each pixel into a grid; *jitter* the centers of the grid randomly within the grid cell.



- ☆ Instead of uniform sampling, use weighted sampling, e.g., distribution of subpixel depends on light intensity.
- ☆ Shoot different rays at slightly different times.