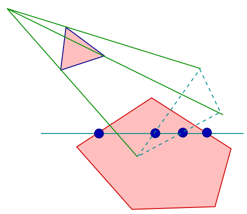
Shadow Generation

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

$$S_i = \begin{cases} 0 & \text{if light } i \text{ is blocked,} \\ 1 & \text{if light } i \text{ is not blocked.} \end{cases}$$



Scan-line method

- ☆ Use light source as the center of projection.
- ☆ Project the edges of polygons that cast shadows on the polygons intersecting the scan line.
- ☆ Whenever the scan line visits one of the projected points, change the intensity.

Slide 1

Shadow Generation

Two-Pass Object Precision Algorithm:

- ☆ Find the portion of each polygon visible from the light source.
- ☆ Decompose each polygon into subpolygons, each being either completely lit or completely under dark.
- ☆ Render each polygon as folllows:
  - If the polygon is in dark, set intensity to the ambient light
    $$I_\lambda = k_a I_{a\lambda} O_{a\lambda}.$$
  - If the polygon is lit, then use ambient, diffuse, and specular reflection.
- ☆ Repeat the first two steps for each light source.

Slide 2

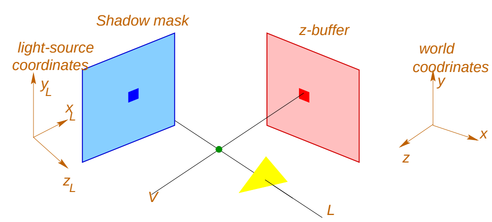
Shadow Generation

Two-Pass Z-Buffer Algorithm:

- ☆ Two passes
  - W.r.t light source
  - W.r.t. view point
- ☆ Compute depth information w.r.t. the light source.
  light buffer (LB) or *shadow mask*
- ☆ Compute the value of the frame buffer at each pixel π w.r.t. the viewpoint as follows:
  - Suppose the point p in the world coordinate is drawn at pixel π.
  - Determine if p is under shadow.
  - If under shadow, use ambient light.
    Otherwise compute the lighting information at π.
- ☆ For multiple light sources, maintain a shadow mask for each light source.

Slide 3

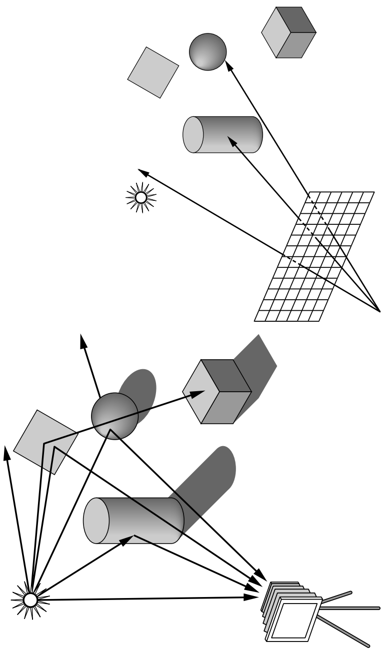
Shadow Generation



- ☆ Compute the pixel (a, b) in the shadow mask corresponding to point p.
- ☆ Compute the distance c of p from the light source.
- ☆ Compare c with $z_L = LB[a, b]$.
- ☆ If $z_L < c$, p is under shadow; otherwise p is lit.

Slide 4

Ray Tracing



Slide 5

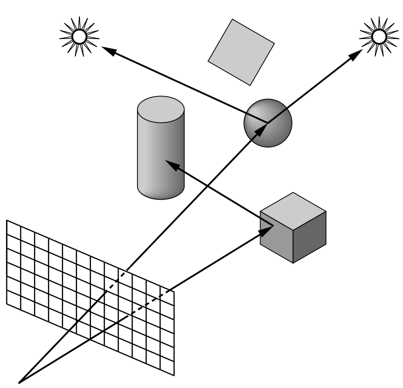
Ray Tracing

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!

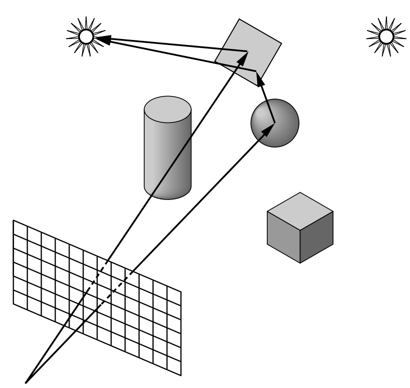
Slide 6

Ray Tracing: Shadow Generation



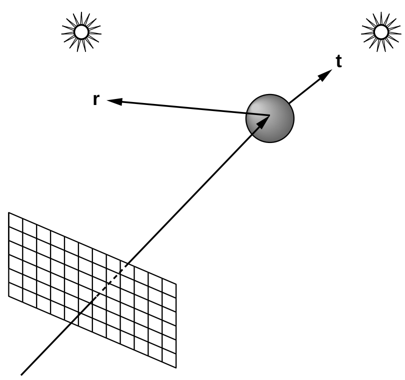
Slide 7

Ray Tracing: Reflection



Slide 8

Ray Tracing: Refraction



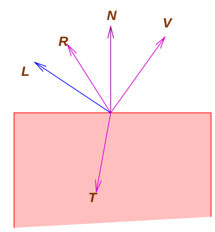
Slide 9

Recursive Ray Tracing

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

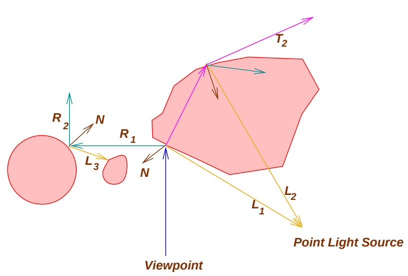
For each pixel π on the screen, do the following:

- ☆ *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.



Slide 10

Ray Tracing



- ☆ Shoot the following secondary rays from p:
  - ***Shadow ray (ρ_L):*** Shoot a ray along $p\vec{L}$.
  - ***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.

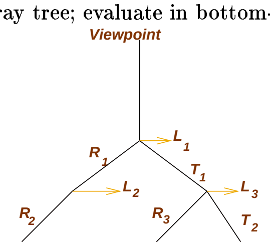
Slide 11

Ray Tracing

Stopping criteria:

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

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



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

Slide 12

Ray Tracing

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

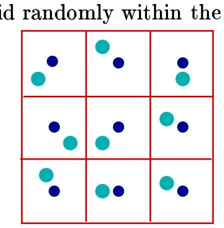
Efficiency Issues:

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

Slide 13

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.

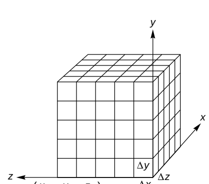


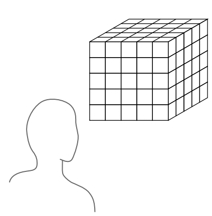
- ☆ Instead of uniform sampling, use weighted sampling, e.g., distribution of subpixel depends on light intensity.
- ☆ Shoot different rays at slightly different times.

Slide 14

Volume Rendering

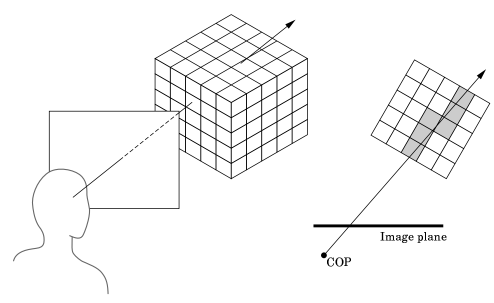
- ☆ 3D model is constructed a a set of small cubes, called *voxels*.
- ☆ Each voxel v is assigned a value $f(v)$.
- ☆ $f(v)$ determines the color of the voxel.
- ☆ Volume data is typically generated in slices and put together.
- ☆ Applications
  - Medical imaging
  - Molecular modeling
  - Atmospheric modeling
  - Scientific visualization





Slide 15

Volume Ray Tracing



Repeat the following for each pixel π on the plane

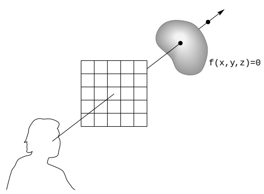
- ☆ Draw the ray ρ in the viewing from π
- ☆ Find all voxels $v_1, v_2, \ldots, v_m$ that intersect ρ
- ☆ Assign a weight w_i to each voxel v_i
- ☆ Compute the weight sum

$$F(\pi) = \sum_{i=1}^{m} w_i f(v_i)$$

Slide 16

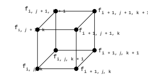
Viewing Implicit Functions

- ☆ Regard the scene as a 3-dimensional function
- ☆ Each point $(x, y, z) \in \mathbb{R}^3$ has a value $f(x, y, z)$
- ☆ Values are sampled at voxel vertices
- ☆ Linearly interpolate the values inside a voxel
- ☆ For a given $c \in \mathbb{R}$, define
$$\Sigma(c) = \{(x, y, z) \mid f(x, y, z) = c\}$$
- ☆ $\Sigma(c)$ is a polyhedral surface
- ☆ Render $\Sigma(c)$
- ☆ Useful for *gel* like objects.



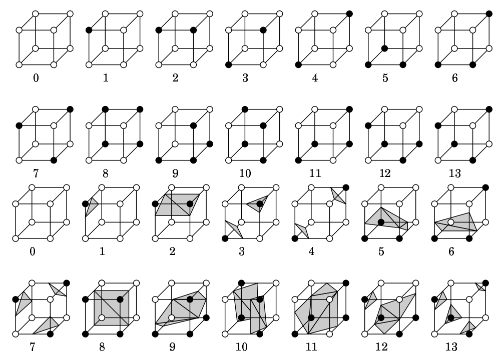
Slide 17

Isosurface Extraction



$\Sigma(c)$ intersects a voxel if $f(x, y, z) > c$ for some vertices and $< c$ for some otehrs

- ☆ Color a vertex z black if $f(x, y, z) > c$ and if $f(x, y, z) < c$
- ☆ Interpolate Σ within a voxel by triangles
- ☆ Visit voxels in a consistent manner, e.g., row by row, then plane by plane.
- ☆ Easy to parallelize; faster than ray tracing.



Slide 18