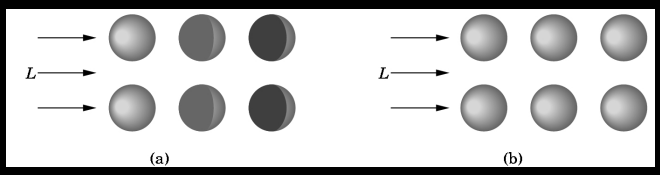
Ray Tracing

Ray Casting
Ray-Surface Intersections
Barycentric Coordinates
Reflection and Transmission
[Angel, Ch 13.2-13.3]
Ray Tracing Handouts

Local vs. Global Rendering Models

- Local rendering models (graphics pipeline)
  - Object illuminations are independent
  - No light scattering between objects
  - No real shadows, reflection, transmission
- Global rendering models
  - Ray tracing (highlights, reflection, transmission)
  - Radiosity (surface interreflections)

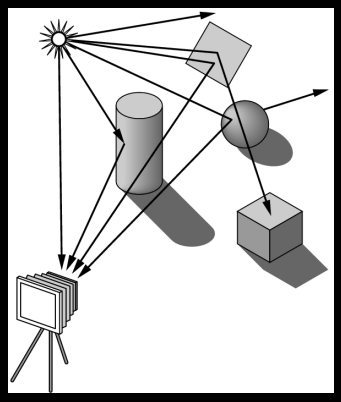


Object Space vs. Image Space

- Graphics pipeline: for each object, render
  - Efficient pipeline architecture, on-line
  - Difficulty: object interactions
- Ray tracing: for each pixel, determine color
  - Pixel-level parallelism, off-line
  - Difficulty: efficiency, light scattering
- Radiosity: for each two surface patches, determine diffuse interreflections
  - Solving integral equations, off-line
  - Difficulty: efficiency, reflection

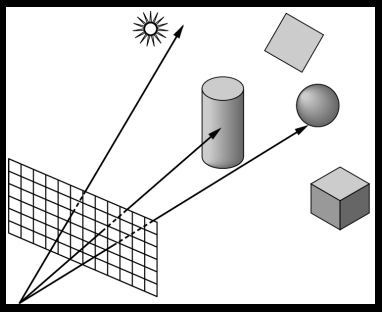
Forward Ray Tracing

- Rays as paths of photons in world space
- Forward ray tracing: follow photon from light sources to viewer
- Problem: many rays will not contribute to image!



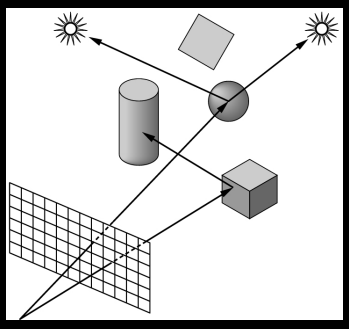
Backward Ray Tracing

- Ray-casting: one ray from center of projection through each pixel in image plane
- Illumination
  1. Phong (local as before)
  2. Shadow rays
  3. Specular reflection
  4. Specular transmission
- (3) and (4) are recursive



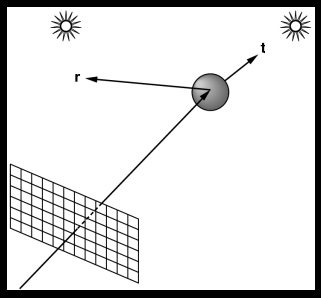
Shadow Rays

- Determine if light “really” hits surface point
- Cast shadow ray from surface point to light
- If shadow ray hits opaque object,no contribution
- Improved diffuse reflection



Reflection Rays

- Calculate specular component of illumination
- Compute reflection ray (recall: backward!)
- Call ray tracer recursively to determine color
- Add contributions
- Transmission ray
  - Analogue for transparent or translucent surface
  - Use Snell's laws for refraction
- Later:
  - Optimizations, stopping criteria

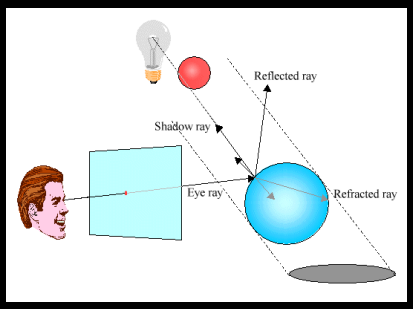


Ray Casting

- Simplest case of ray tracing
- Required as first step of recursive ray tracing
- Basic ray-casting algorithm
  - For each pixel (x,y) fire a ray from COP through (x,y)
  - For each ray & object calculate closest intersection
  - For closest intersection point **p**
    - Calculate surface normal
    - For each light source, calculate and add contributions
- Critical operations
  - Ray-surface intersections
  - Illumination calculation

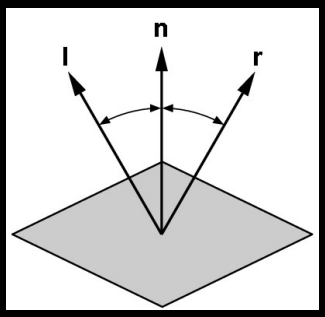
Recursive Ray Tracing

- Calculate specular component
  - Reflect ray from eye on specular surface
  - Transmit ray from eye through transparent surface
- Determine color of incoming ray by recursion
- Trace to fixed depth
- Cut off if contribution below threshold



Angle of Reflection

- Recall: incoming angle = outgoing angle
- $\mathbf{r} = 2(\mathbf{l} \cdot \mathbf{n})\,\mathbf{n} - \mathbf{l}$
- For incoming/outgoing ray negate **l** !
- Compute only for surfaces with actual reflection
- Use specular coefficient
- Add specular and diffuse components



Refraction

- Index of refraction is relative speed of light
- Snell's law
  - η_l = index of refraction for upper material
  - η_t = index of refraction for lower material

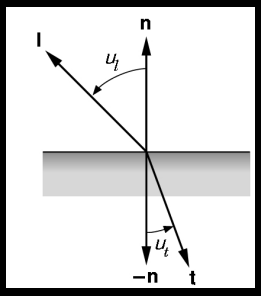
[U = θ]

$$\frac{\sin(\theta_l)}{\sin(\theta_t)} = \frac{\eta_t}{\eta_l} = \eta$$

$$\mathbf{t} = -\frac{1}{\eta}\mathbf{l} - (\cos(\theta_t) - \frac{1}{\eta}\cos(\theta_l))\mathbf{n}$$

where $\cos(\theta_l) = \mathbf{l} \cdot \mathbf{n}$

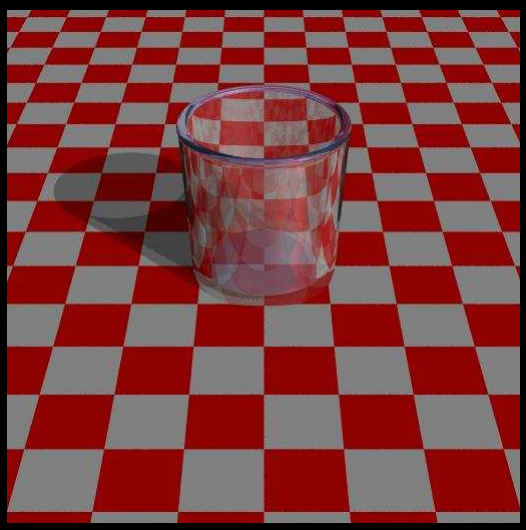
and $\cos^2(\theta_t) = 1 - \frac{1}{\eta^2}(1 - \mathbf{l} \cdot \mathbf{n})$



Raytracing Example

www.povray.org

Raytracing Example



rayshade gallery

Raytracing Example

rayshade gallery

Raytracing Example

www.povray.org

Raytracing Example

Saito, Saturn Ring

Raytracing Example

www.povray.org

Raytracing Example

www.povray.org

Raytracing Example

rayshade gallery

Intersections

Ray-Surface Intersections

- General implicit surfaces
- General parametric surfaces
- Specialized analysis for special surfaces
  - Spheres
  - Planes
  - Polygons
  - Quadrics
- Do not decompose objects into triangles!
- CSG is also a good possibility

Rays and Parametric Surfaces

- Ray in parametric form
  - Origin $\mathbf{p}_0 = [x_0 \;\; y_0 \;\; z_0 \;\; 1]^T$
  - Direction $\mathbf{d} = [x_d \;\; y_d \;\; z_d \;\; 0]^t$
  - Assume $\mathbf{d}$ normalized ($x_d^2 + y_d^2 + z_d^2 = 1$)
  - Ray $\mathbf{p}(t) = \mathbf{p}_0 + \mathbf{d}\, t$ for $t > 0$
- Surface in parametric form
  - Point $\mathbf{q} = g(u, v)$, possible bounds on u, v
  - Solve $\mathbf{p} + \mathbf{d}\, t = g(u, v)$
  - Three equations in three unknowns (t, u, v)

Rays and Implicit Surfaces

- Ray in parametric form
  - Origin $\mathbf{p}_0 = [x_0 \;\; y_0 \;\; z_0 \;\; 1]^T$
  - Direction $\mathbf{d} = [x_d \;\; y_d \;\; z_d \;\; 0]^t$
  - Assume $\mathbf{d}$ normalized ($x_d^2 + y_d^2 + z_d^2 = 1$)
  - Ray $\mathbf{p}(t) = \mathbf{p}_0 + \mathbf{d}\, t$ for $t > 0$
- Implicit surface
  - Given by $f(\mathbf{q}) = 0$
  - Consists of all points $\mathbf{q}$ such that $f(\mathbf{q}) = 0$
  - Substitute ray equation for $\mathbf{q}$: $f(\mathbf{p}_0 + \mathbf{d}\, t) = 0$
  - Solve for t (univariate root finding)
  - Closed form (if possible) or numerical approximation

Ray-Sphere Intersection I

- Common and easy case
- Define sphere by
  - Center $\mathbf{c} = [x_c \;\; y_c \;\; z_c \;\; 1]^T$
  - Radius r
  - Surface $f(\mathbf{q}) = (x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 - r^2 = 0$
- Plug in ray equations for x, y, z:

$$\begin{aligned} x &= x_0 + x_d\, t \\ y &= y_0 + y_d\, t \\ z &= z_0 + z_d\, t \end{aligned}$$

$$\begin{aligned} & (x_0 + x_d\, t - x_c)^2 \\ + \; & (y_0 + y_d\, t - y_c)^2 \\ + \; & (z_0 + z_d\, t - z_c)^2 = r^2 \end{aligned}$$

Ray-Sphere Intersection II

- Simplify to

$$a\,t^2 + b\,t + c = 0$$

where

$$a = x_d^2 + y_d^2 + z_d^2 = 1 \quad \text{since } |\mathbf{d}| = 1$$
$$b = 2\,(x_d(x_0 - x_c) + y_d(y_0 - y_c) + z_d(z_0 - z_c))$$
$$c = (x_0 - x_c)^2 + (y_0 - y_c)^2 + (z_0 - z_c)^2 - r^2$$

- Solve to obtain t_0 and t_1

$$t_{0,1} = \frac{-b \pm \sqrt{b^2 - 4c}}{2}$$

Check if t_0, $t_1 > 0$ (ray)
Return min(t_0, t_1)

Ray-Sphere Intersection III

- For lighting, calculate unit normal

$$\mathbf{n} = \frac{1}{r}\,[(x_i - x_c)\ (y_i - y_c)\ (z_i - z_c)\ 0]^T$$

- Negate if ray originates inside the sphere!

Simple Optimizations

- Factor common subexpressions
- Compute only what is necessary
  - Calculate $b^2 - 4c$, abort if negative
  - Compute normal only for closest intersection
  - Other similar optimizations [Handout]

Ray-Polygon Intersection I

- Assume planar polygon
  1. Intersect ray with plane containing polygon
  2. Check if intersection point is inside polygon
- Plane
  - Implicit form: $ax + by + cz + d = 0$
  - Unit normal: $\mathbf{n} = [a \;\; b \;\; c \;\; 0]^T$ with $a^2 + b^2 + c^2 = 1$
- Substitute:

$$a(x_0 + x_d t) + b(y_0 + y_d t) + c(z_0 + z_d t) + d = 0$$

- Solve:

$$t = \frac{-(ax_0 + by_0 + cz_0 + d)}{ax_d + by_d + cz_d}$$

Ray-Polygon Intersection II

- Substitute t to obtain intersection point in plane
- Test if point inside polygon [see Handout]

Ray-Quadric Intersection

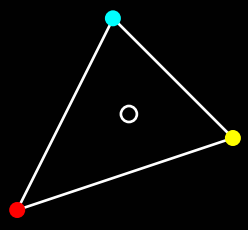
- Quadric f(**p**) = f(x, y, z) = 0, where f is polynomial of order 2
- Sphere, ellipsoid, paraboloid, hyperboloid, cone, cylinder
- Closed form solution as for sphere
- Important case for modelling in ray tracing
- Combine with CSG

[see Handout]

Barycentric Coordinates

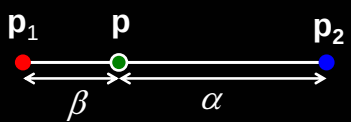
Interpolated Shading for Ray Tracing

- Assume we know normals at vertices
- How do we compute normal of interior point?
- Need linear interpolation between 3 points
- Barycentric coordinates
- Yields same answer as scan conversion



Barycentric Coordinates in 1D

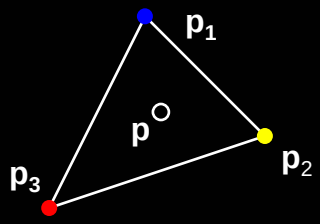
- Linear interpolation
  - $\mathbf{p}(t) = (1 - t)\mathbf{p}_1 + t\,\mathbf{p}_2$, $0 \leq t \leq 1$
  - $\mathbf{p}(t) = \alpha\,\mathbf{p}_1 + \beta\,\mathbf{p}_2$ where $\alpha + \beta = 1$
  - $\mathbf{p}$ is between $\mathbf{p}_1$ and $\mathbf{p}_2$ iff $0 \leq \alpha, \beta \leq 1$
- Geometric intuition
  - Weigh each vertex by ratio of distances from ends



- α, β are called barycentric coordinates

Barycentric Coordinates in 2D

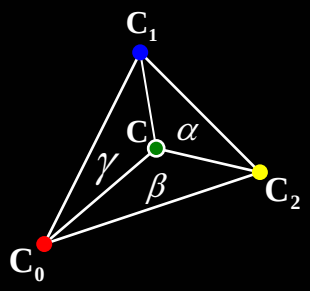
- Given 3 points instead of 2



- Define 3 barycentric coordinates, α, β, γ
- $\mathbf{p} = \alpha\,\mathbf{p}_1 + \beta\,\mathbf{p}_2 + \gamma\,\mathbf{p}_3$
- $\mathbf{p}$ inside triangle iff $0 \leq \alpha, \beta, \gamma \leq 1$, $\alpha + \beta + \gamma = 1$
- How do we calculate α, β, γ given $\mathbf{p}$?

Barycentric Coordinates for Triangle

- Coordinates are ratios of triangle areas



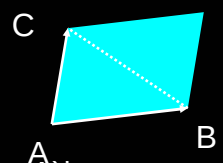
$$\alpha = \frac{Area(\mathbf{C}\mathbf{C}_1\mathbf{C}_2)}{Area(\mathbf{C}_0\mathbf{C}_1\mathbf{C}_2)}$$

$$\beta = \frac{Area(\mathbf{C}_0\mathbf{C}\mathbf{C}_2)}{Area(\mathbf{C}_0\mathbf{C}_1\mathbf{C}_2)}$$

$$\gamma = \frac{Area(\mathbf{C}_0\mathbf{C}_1\mathbf{C})}{Area(\mathbf{C}_0\mathbf{C}_1\mathbf{C}_2)} = 1 - \alpha - \beta$$

Computing Triangle Area

- In 3 dimensions
  - Use cross product
  - Parallelogram formula
  - Area(ABC) = (1/2)|(B – A) × (C – A)|
  - Optimization: project, use 2D formula
- In 2 dimensions
  - Area(x-y-proj(ABC)) = (1/2)(($b_x - a_x$)($c_y - a_y$) – ($c_x - a_x$) ($b_y - a_y$))



Ray Tracing Preliminary Assessment

- Global illumination method
- Image-based
- Pros:
  - Relatively accurate shadows, reflections, refractions
- Cons:
  - Slow (per pixel parallelism, not pipeline parallelism)
  - Aliasing
  - Inter-object diffuse reflections

Ray Tracing Acceleration

- Faster intersections
  - Faster ray-object intersections
    - Object bounding volume
    - Efficient intersectors
  - Fewer ray-object intersections
    - Hierarchical bounding volumes (boxes, spheres)
    - Spatial data structures
    - Directional techniques
- Fewer rays
  - Adaptive tree-depth control
  - Stochastic sampling
- Generalized rays (beams, cones)