

The University of New Mexico

Ray Tracing and Radiosity

Ed Angel

Professor of Computer Science, Electrical
and Computer Engineering, and Media Arts

Director, Arts Technology Center

University of New Mexico



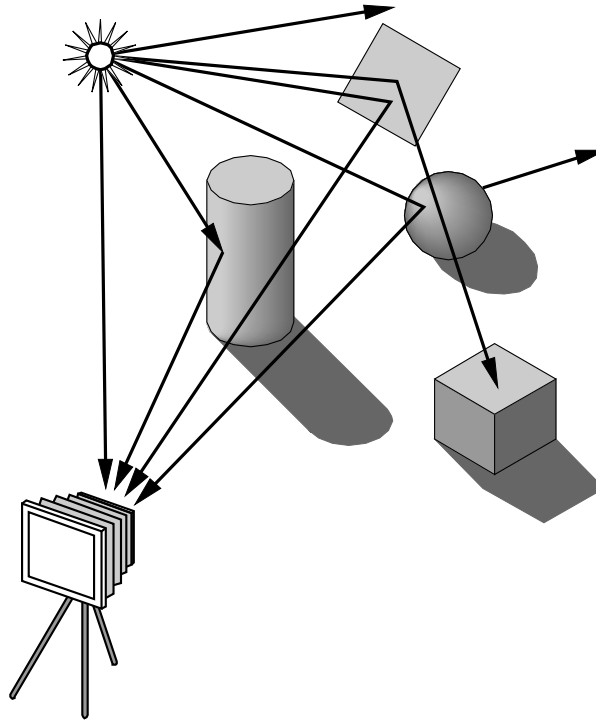
Introduction

- OpenGL is based on a pipeline model in which primitives are rendered one at time
 - No shadows (except by tricks or multiple renderings)
 - No multiple reflections
- Global approaches
 - Rendering equation
 - Ray tracing
 - Radiosity



Ray Tracing

- Follow rays of light from a point source
- Can account for reflection and transmission





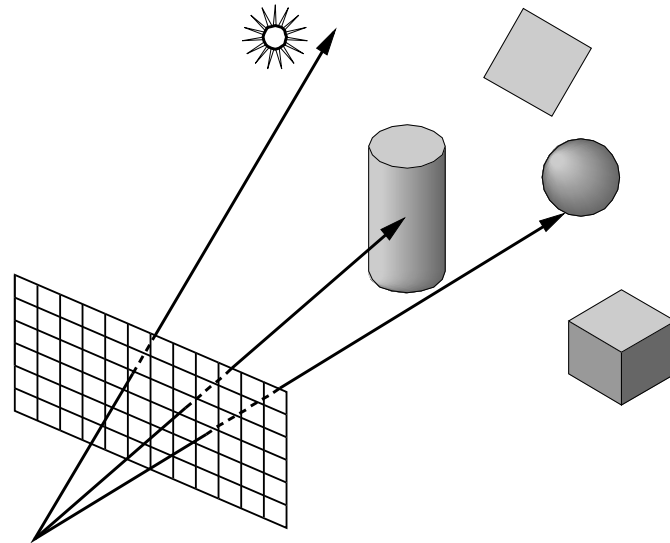
Computation

- Should be able to handle all physical interactions
- Ray tracing paradigm is not computational
- Most rays do not affect what we see
- Scattering produces many (infinite) additional rays
- Alternative: ray casting



Ray Casting

- Only rays that reach the eye matter
- Reverse direction and cast rays
- Need at least one ray per pixel
- Pipeline renderer works on a vertex-by-vertex basis; ray tracer works on a pixel-by-pixel basis





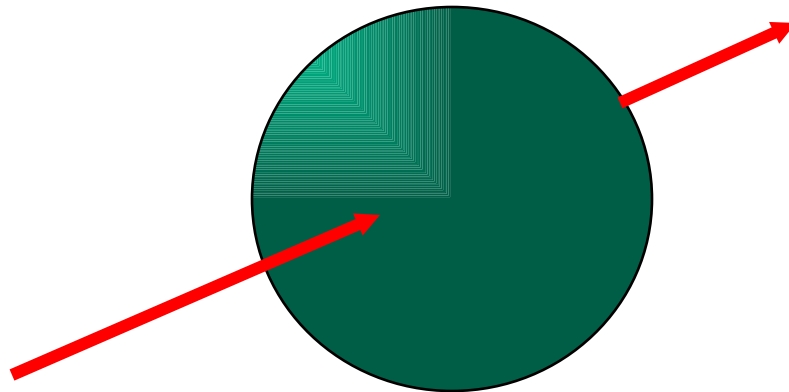
Ray Casting Quadrics

- 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

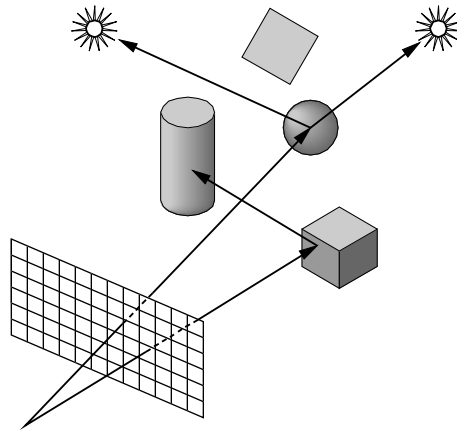
- Ray is parametric
- Sphere is quadric
- Resulting equation is a scalar quadratic equation which gives entry and exit points of ray (or no solution if ray misses)





Shadow Rays

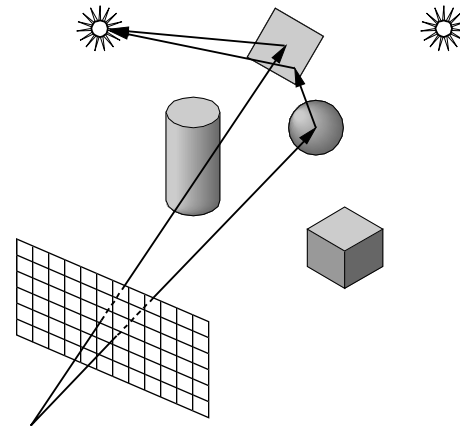
- Even if a point is visible, it will not be lit unless we can see a light source from that point
- Cast shadow or feeler rays
- Can find shadows at the cost of doing a type of hidden surface calculation for each point of intersection





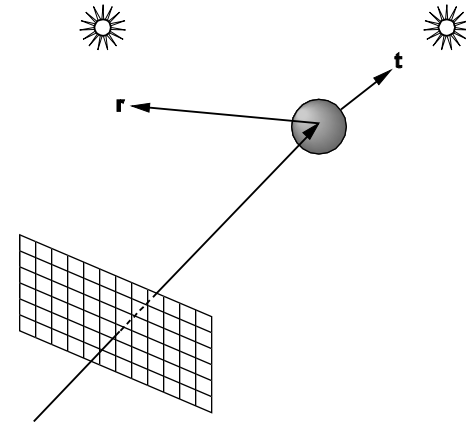
Reflection

- Ray tracing particularly suits perfectly reflecting and transmitting surfaces
- Follow shadow ray from surface to surface until it hits a light source or goes to infinity
- Process is recursive
- Account for absorption



Reflection and Transmission

- Light from source is partially absorbed and contributes towards the diffuse reflection
- The rest is transmitted ray + reflected ray

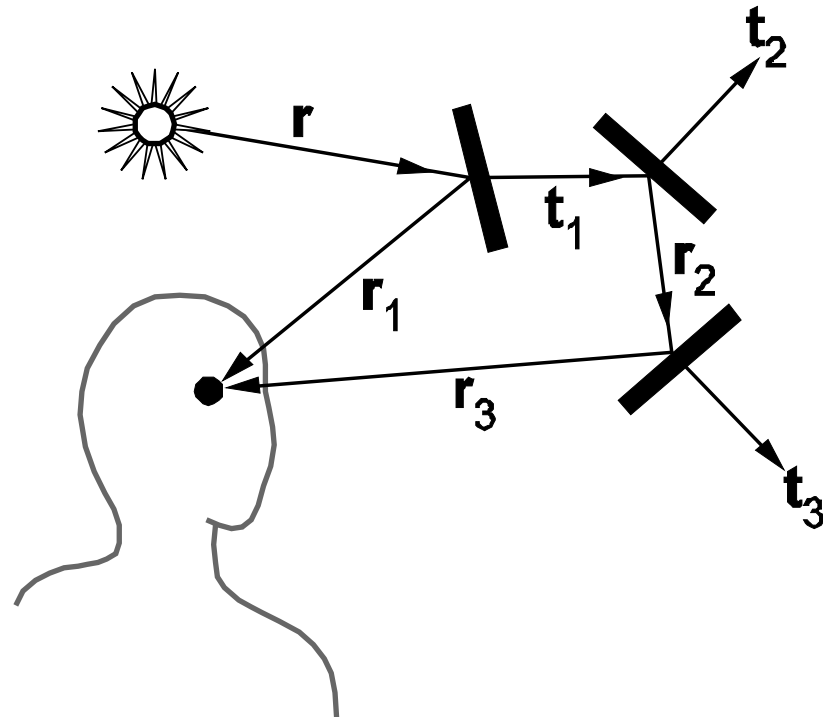


- From the perspective of cast ray, if a light source is visible at a point of intersection, we must
 1. Calculate contribution of the light source at the point
 2. Cast a ray in the direction of perfect reflection
 3. Cast a ray in the direction of the transmitted ray



Ray Trees

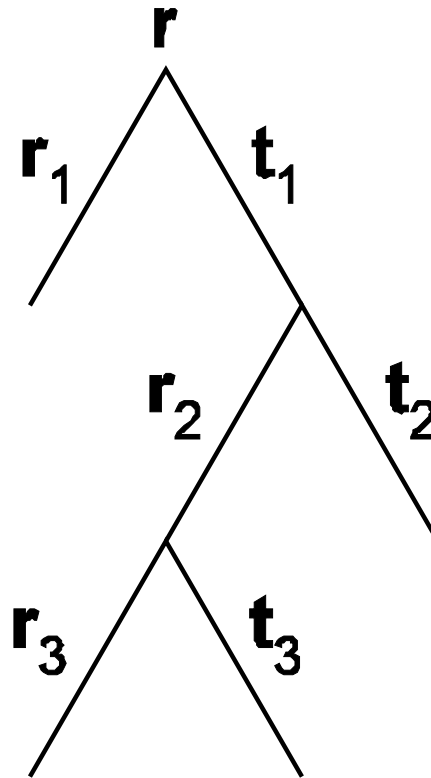
- Blinn-Phong model used to calculate the diffuse term at the point of intersection
- But light scattered diffusely is ignored





The University of New Mexico

Ray Tree





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



Building a Ray Tracer

- Best expressed recursively
- Can remove recursion later
- Image based approach
 - For each ray
- Find intersection with closest surface
 - Need whole object database available
 - Complexity of calculation limits object types
- Compute lighting at surface
- Trace reflected and transmitted rays



When to stop

-
- Some light will be absorbed at each intersection
 - Track amount left
 - Ignore rays that go off to infinity
 - Put large sphere around problem
 - Count steps



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);

    q = intersect(p, d, status);

    if(status==light_source)
        return(light_source_color);
```



Recursive Ray Tracer

```
if(status==no_intersection)
    return(background_color);

n = normal(q);
r = reflect(q, n);
t = transmit(q, n);

local = phong(q, n, r);
reflected = trace(q, r, step+1);
transmitted = trace(q, t, step+1);

return(local+reflected+transmitted);
```



The University of New Mexico

Computing Intersections

- Implicit Objects
 - Quadrics
- Planes
- Polyhedra
- Parametric Surfaces



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
(only a few cases do not require numerical methods)



Quadratics

General quadric can be written as

$$\mathbf{p}^T \mathbf{A} \mathbf{p} + \mathbf{b}^T \mathbf{p} + c = 0$$

Substitute equation of ray

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

to get quadratic equation

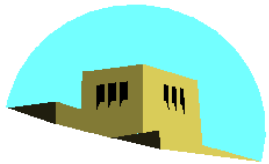


Sphere

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

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

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



The University of New Mexico

Planes

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

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

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



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



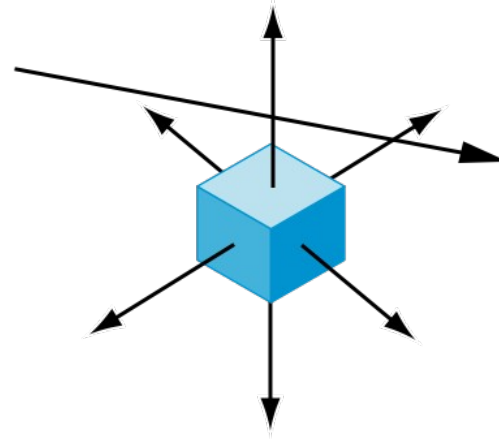
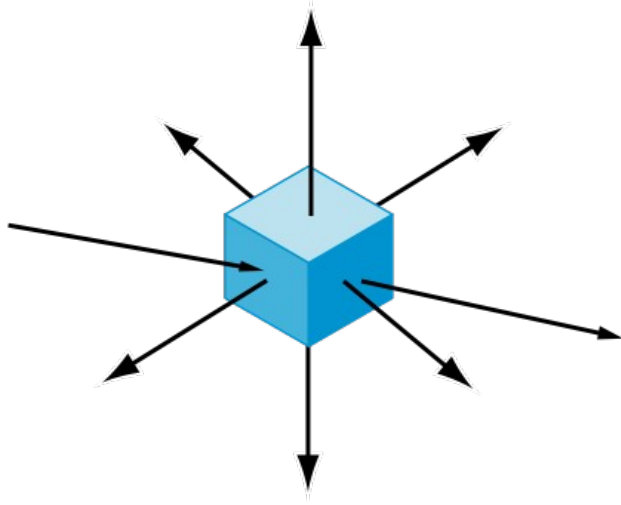
Ray Tracing Polyhedra

- 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



The University of New Mexico

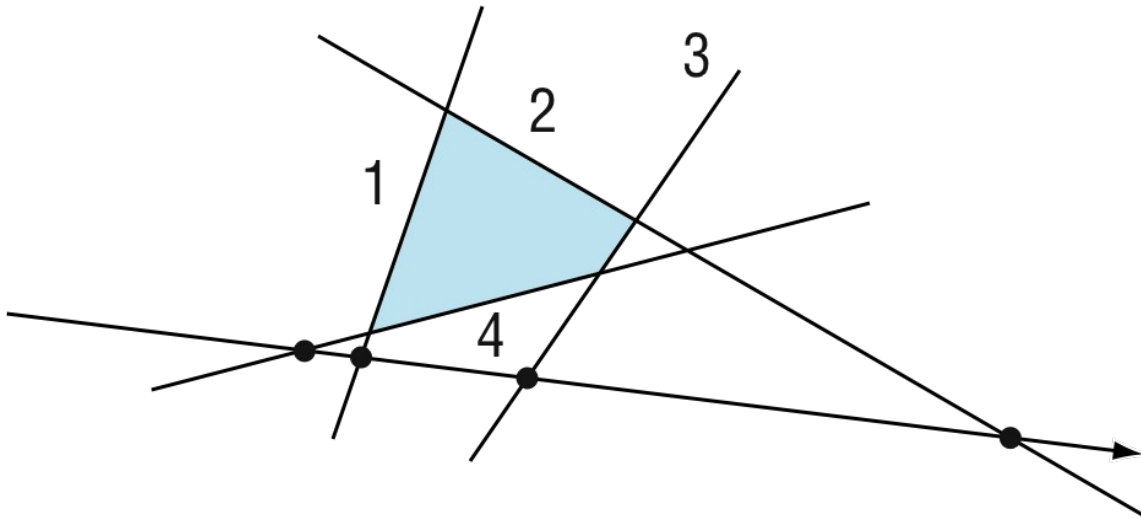
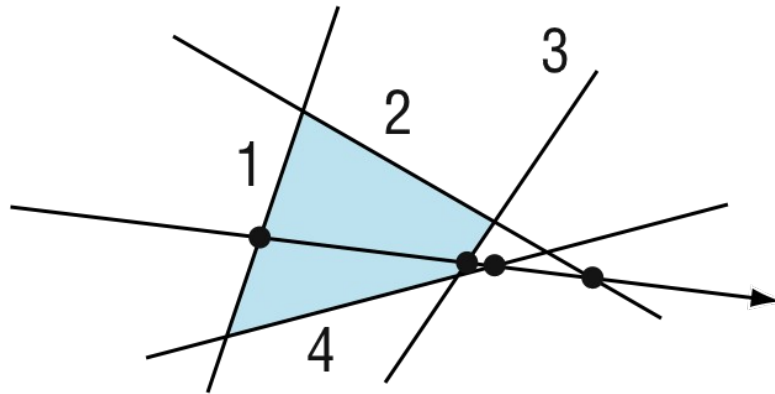
Ray Tracing Polyhedra





The University of New Mexico

Ray Tracing a Polygon





Ray Tracing Variations

- A quick check for no intersection can be done using bounding boxes or spheres because intersection with these can be done quickly.
- Ray tracing is a sampling method and subject to aliasing errors
- In a basic ray tracer, the amount of work is proportional to the number of rays
- Some ray tracers use a stochastic sampling method. Fewer rays are cast in a direction where former rays did not intersect an object and vice versa.
- Ray tracing is inherently a parallel process where the rays are independent of each other however each ray needs access to all objects



Radiosity

-
- Ray tracing is best with many highly specular surfaces
 - Not characteristic of real scenes
 - Rendering equation describes general shading problem
 - Radiosity solves rendering equation for perfectly diffuse surfaces



The Rendering Equation

- Based on the conservation of energy.
- We usually see a steady state scene and not rays bouncing around

Consider light at a point \mathbf{p} arriving from \mathbf{p}'

$$i(\mathbf{p}, \mathbf{p}') = v(\mathbf{p}, \mathbf{p}')(\epsilon(\mathbf{p}, \mathbf{p}') + \int \rho(\mathbf{p}, \mathbf{p}', \mathbf{p}'')i(\mathbf{p}', \mathbf{p}'')d\mathbf{p}'')$$

emission from \mathbf{p}' to \mathbf{p}

occlusion = 0 or $1/d^2$

light reflected at \mathbf{p}' from all points \mathbf{p}'' towards \mathbf{p}



Solving the Rendering Equation

- Rendering equation is simple but solution is complex
- High dimensionality
- Including the wavelength of light will further complicate the solution
- Numerical methods used to solve the general form of rendering equation
- Photon mapping – follow photons until absorbed
- Tracing rays is inefficient but photon mapping uses strategies to make the process feasible
- Special circumstances simplify the rendering equation (perfectly specular and perfectly diffuse surfaces)



Radiosity

- Diffuse-diffuse interactions
- Consider objects to be broken up into flat patches (may correspond to the polygons)
- Assume that patches are perfectly diffuse reflectors
- Find shade independent of the viewer
- Place viewer and render the scene in a conventional way, using a pipeline
- Radiosity = flux = energy/unit area/ unit time leaving patch (watts per square meter)



Notation

n patches numbered 1 to n

b_i = radiosity of patch i

a_i = area patch i

total intensity leaving patch $i = b_i a_i$

$e_i a_i$ = emitted intensity from patch i

ρ_i = reflectivity of patch i

f_{ij} = form factor = fraction of energy leaving patch j
that reaches patch i



Radiosity Equation

energy balance

$$b_i a_i = e_i a_i + \rho_i \sum f_{ji} b_j a_j$$

f_{ji} is form factor: the fraction of energy from patch j that reaches i

reciprocity

$$f_{ij} a_i = f_{ji} a_j$$

radiosity equation

$$b_i = e_i + \rho_i \sum f_{ij} b_j$$



Conclusion

- Global lighting can be determined through the rendering equation
- Rendering equation cannot be solved in general
- Numerical methods used to approximate solution
- Perfectly specular and perfectly diffuse surfaces simplify the rendering equation
- Ray tracing is suitable when the surfaces are perfectly specular
- Radiosity approach is suitable when the surfaces are perfectly diffuse