12. Advanced Rendering

Outline

- Global Rendering
- Ray Tracing
- What’s Next

Global Rendering

Introduction

- WebGL 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 based on the rendering equation
  - Ray tracing
  - Radiosity
  - Photon mapping
  - Path Tracing

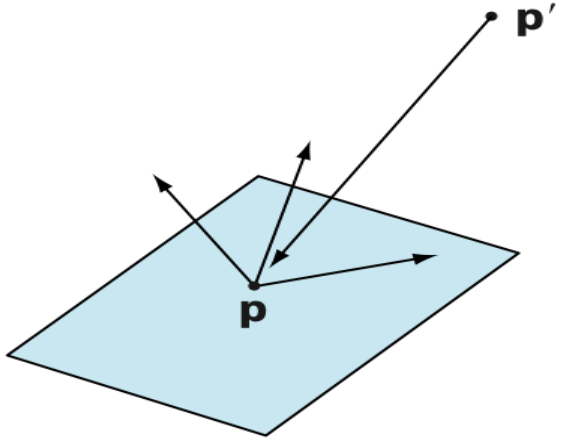
The Rendering Equation

- Use physical reasoning based on conservation of energy
- Within a closed environment, the energy entering a surface must equal the energy leaving
- Energy leaving in a given direction depends on the energy arriving from all directions

Rendering Equation (Kajia)

$$i(p,p') = \upsilon(p,p')(\varepsilon(p,p') + \int \rho(p,p',p'')i(p',p'')dp'')$$

$i(p,p')$	intensity from p arriving at p'
$\varepsilon(p,p')$	emission from p arriving at p'
$\upsilon(p,p')$	occlusion term = 0 or $1/r^2$
$\rho(p,p',p'')$	bidirectional reflection distribution function (BRDF)
	contribution from reflections from all other points

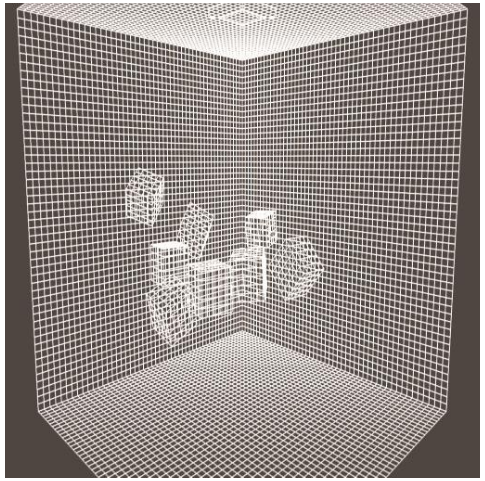


BRDF

- The BRDF characterizes the material
  - Generalization of reflection coefficient
- General case requires is a 9 variable function at given frequency
  - position
  - direction of three vectors
- Special cases lead to familiar methods

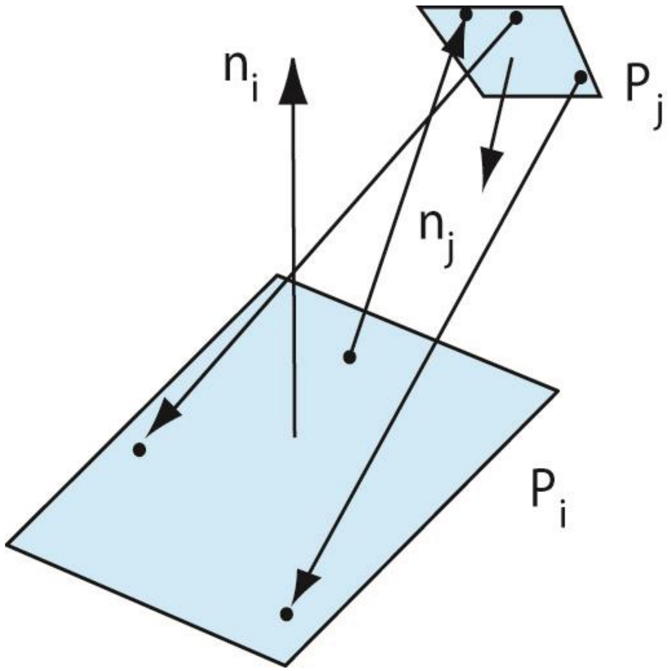
Radiosity

- Assume all surfaces are perfectly diffuse
- Light is scattered equally in all directions
- Divide space into small patches



Form Factors

- Need to compute the form factor between each pair of patches which describes effect of light from one patch onto the other



Radiosity Rendering

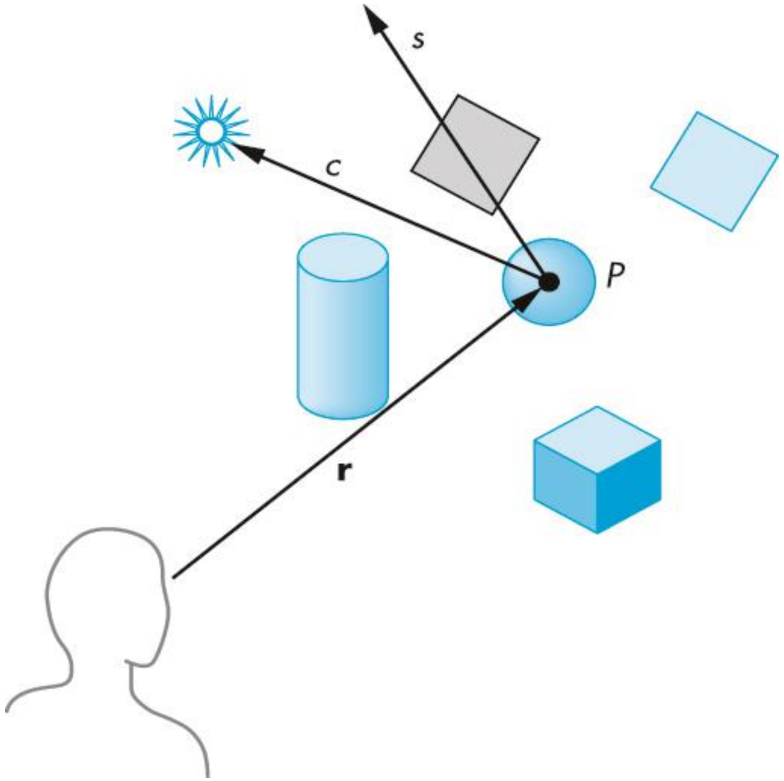
- Once the form factors are computer each patch is a small diffuse patch
- Final render now uses only diffuse term
- As long as geometry is unchanges, subsequent renderings use same form factors

Monte Carlo Methods

- Rendering equation cannot be solved analytically
- One approach to take a probabilistic (Monte Carlo) approach
  - Ray tracing
  - Photon mapping
  - Path tracing

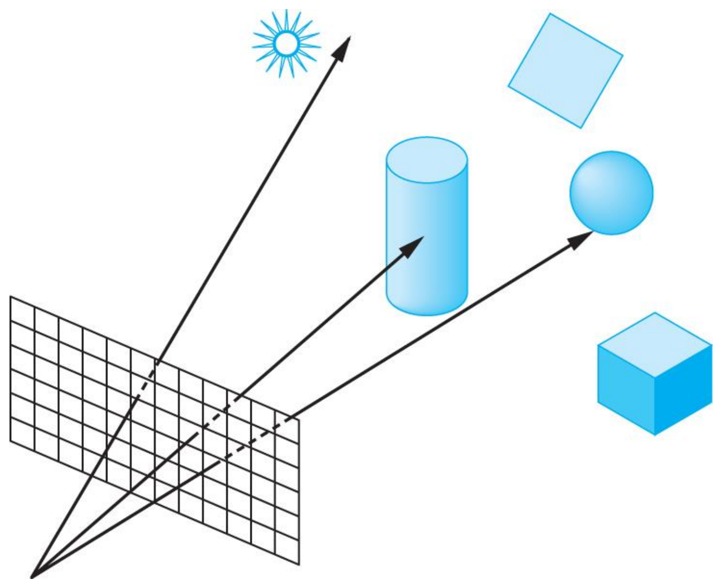
Ray Tracing

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

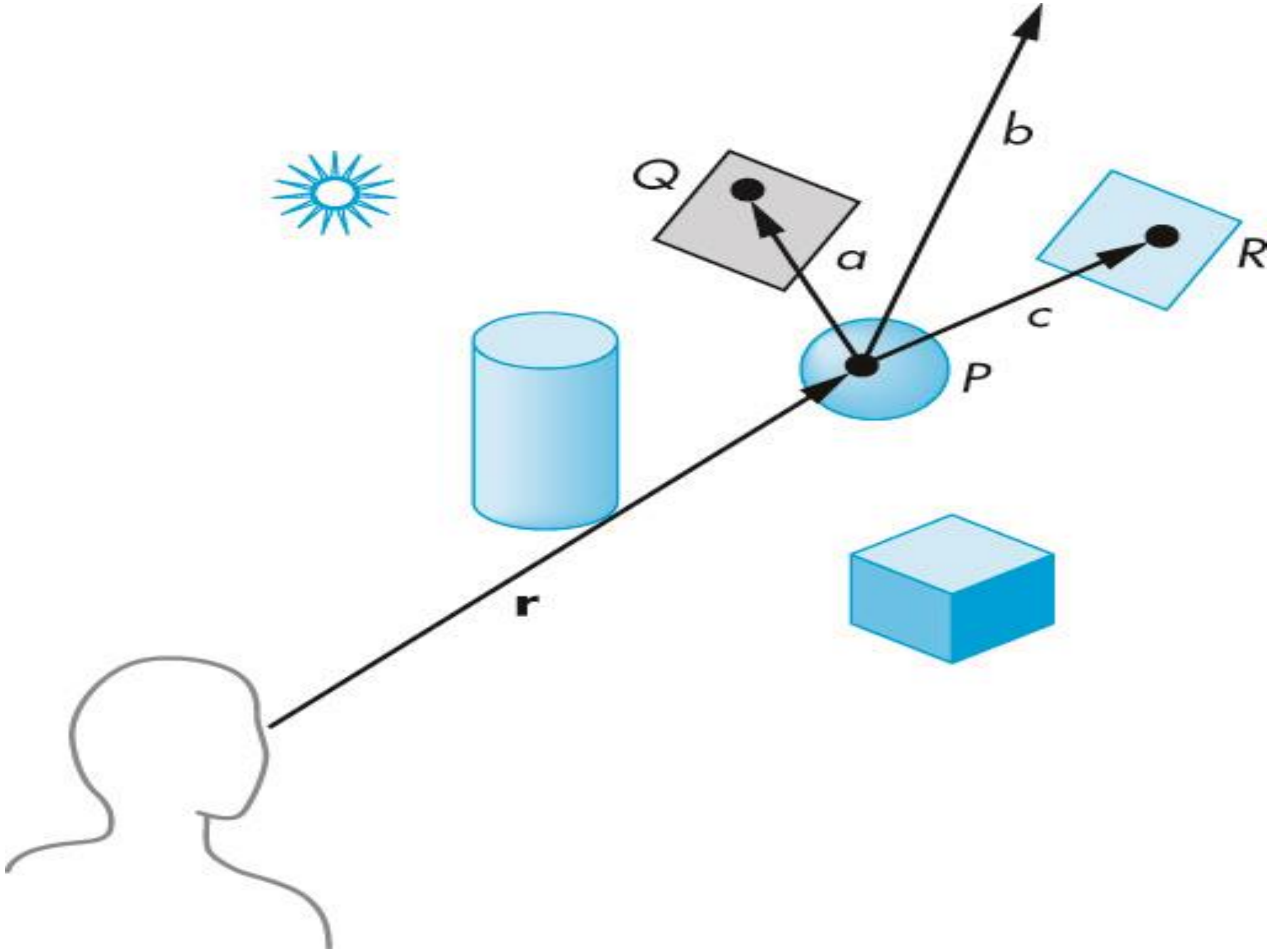


Ray Casting

- Only rays that reach the eye matter
- Reverse direction and cast rays
- Need at least one ray per pixel



Path Tracing



Adding Rays

- Basic limitation of ray tracing is the number of rays we can trace
- Add more rays
  - multitisampling
  - stochastically
  - adaptively

Photon Mapping

- Instead of tracing rays trace photons
- Send out stream of photons
- When photons strike surface, they can be
  - absorbed
  - go off in multiple directions
- Total number of photons limited by compute time required
- Off line and stochastic

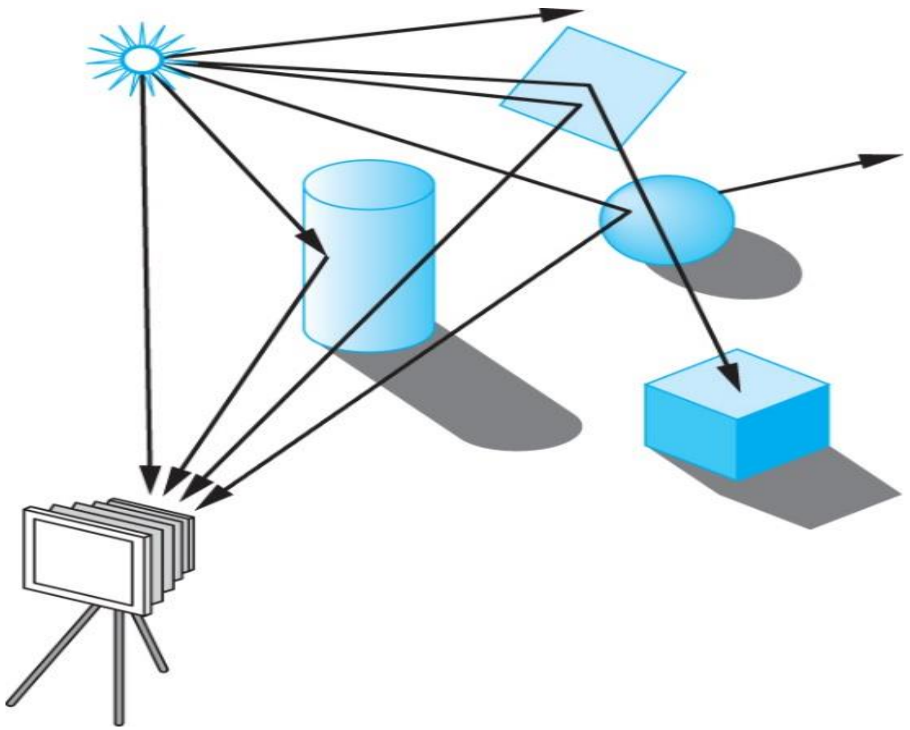
Ray Tracing

Objectives

- Develop a basic recursive ray tracer
- Computer intersections for quadrics and polygons

Ray Tracing

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

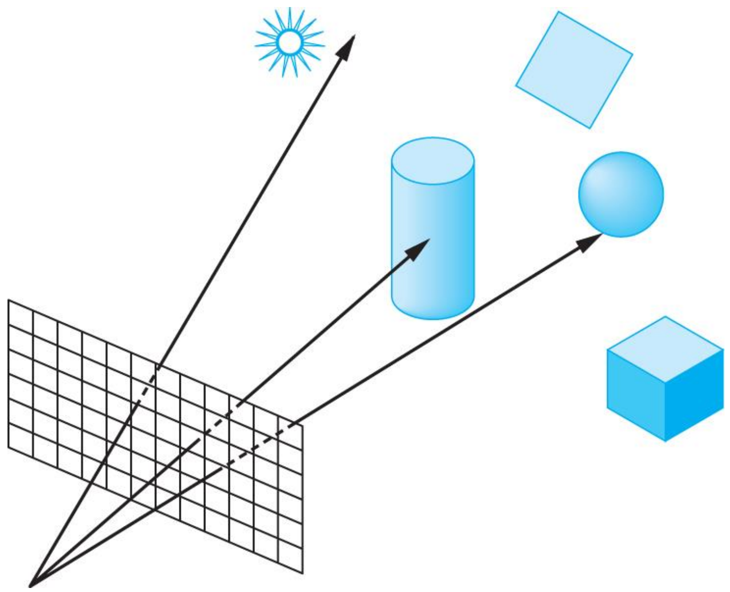


Computation

- 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

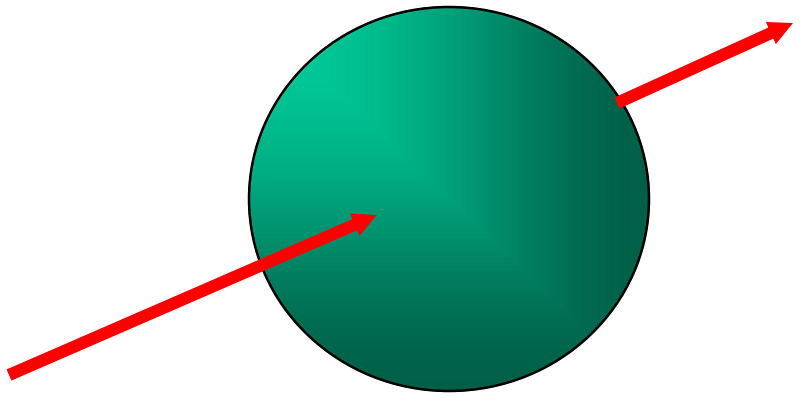


Ray Casting Quadrics

- 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

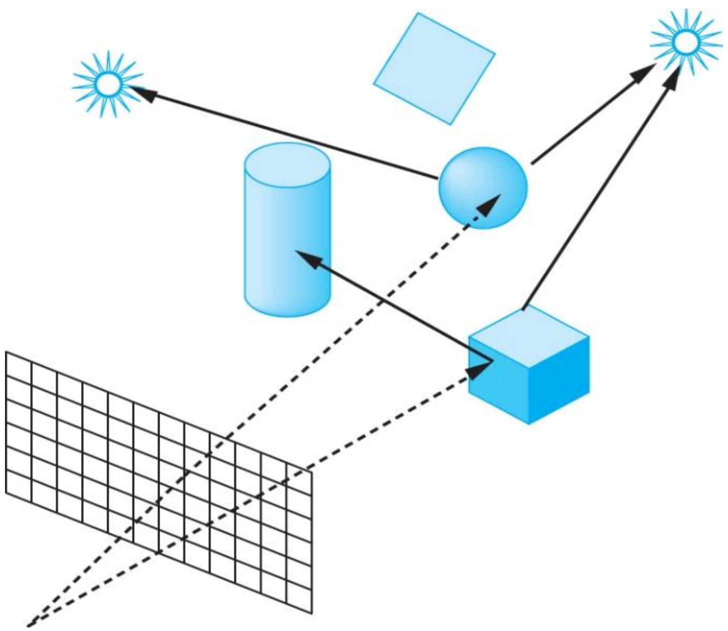
Ray Casting a Sphere

- 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)



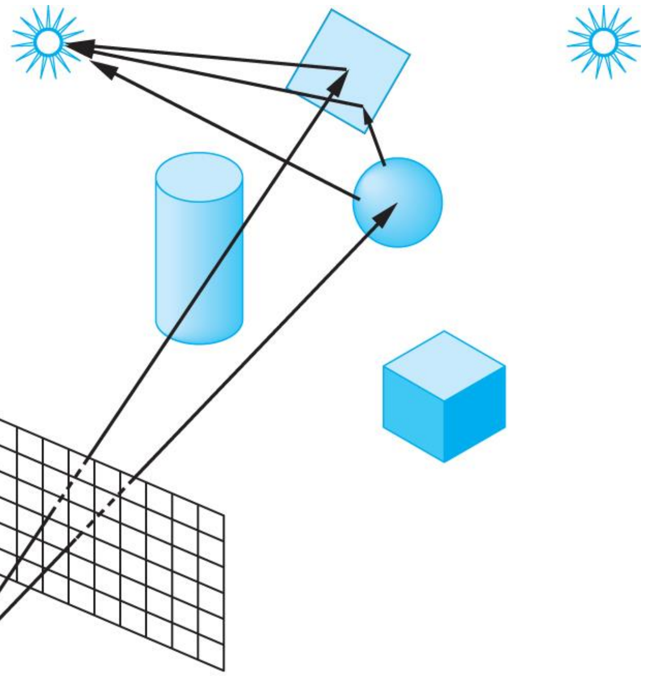
Shadow Rays

- 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

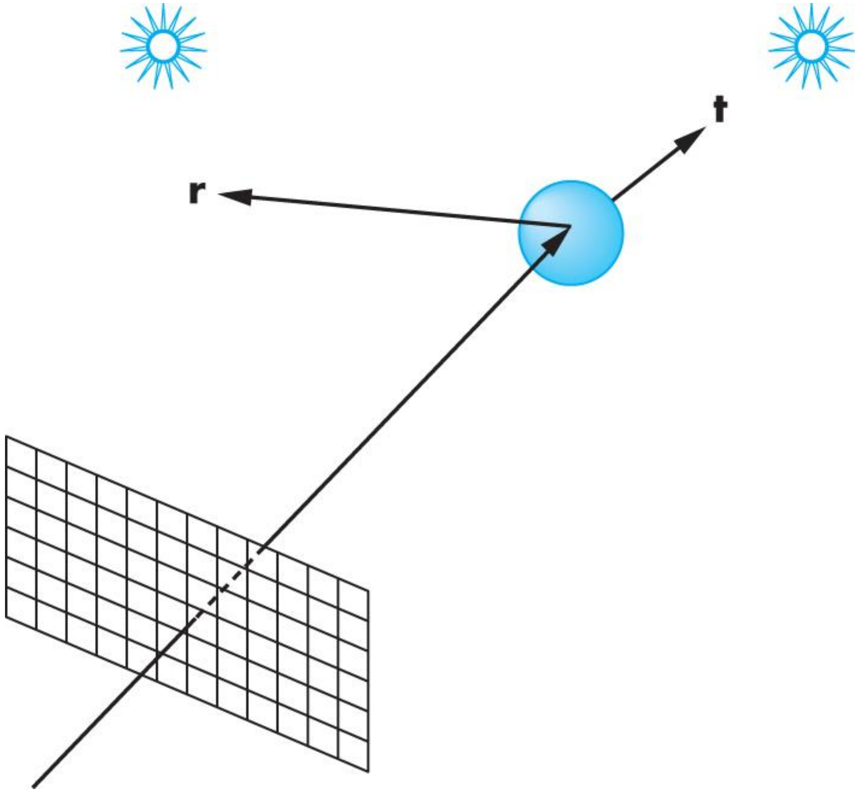


Reflection

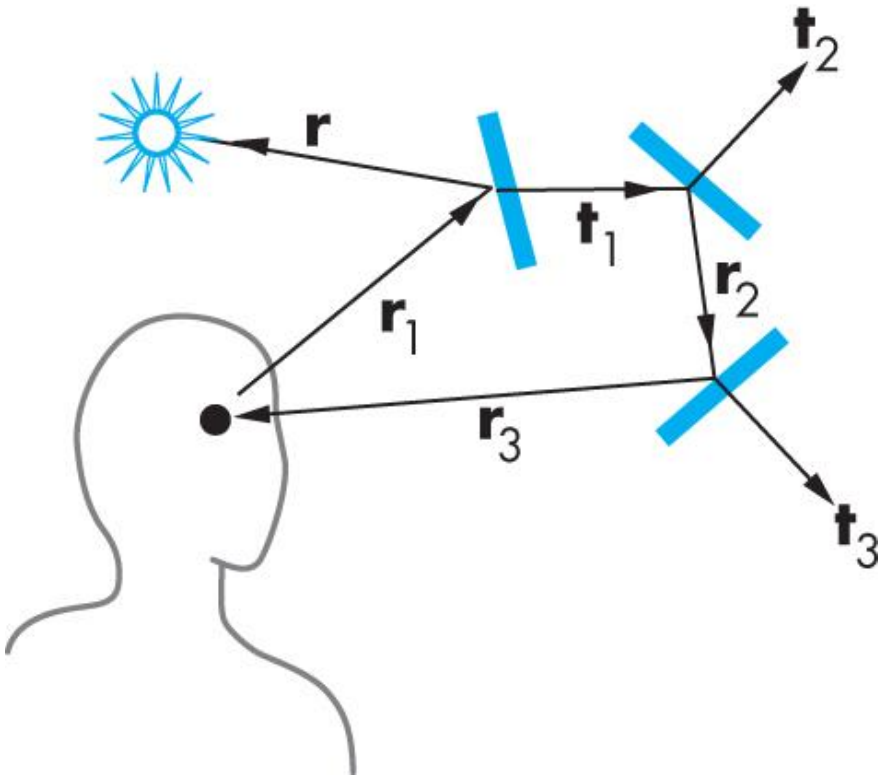
- Must follow shadow rays off reflecting or transmitting surfaces
- Process is recursive



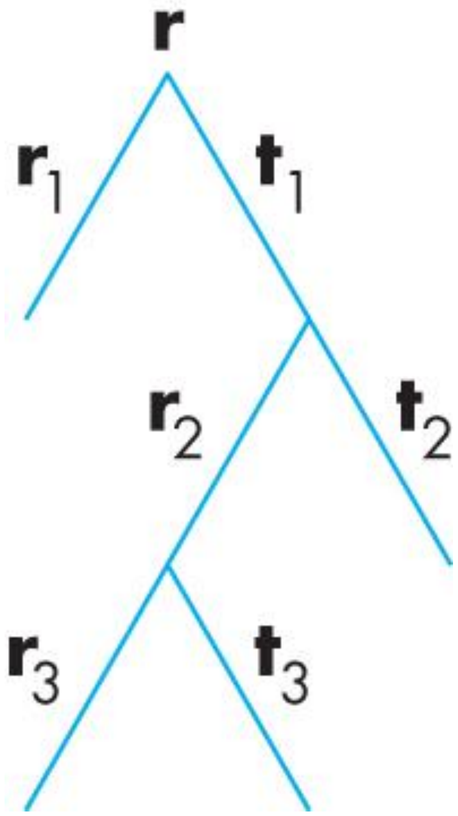
Reflection and Transmission



Ray Trees



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 modified 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);
```

Recursive Ray Tracer

```
q = intersect(p, d, status);
if(status==light_source)
 return(light_source_color);
if(status==no_intersection)
 return(background_color);

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

Recursive Ray Tracer

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

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

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

Quadrics

General quadric can be written as

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

Substitute equation of ray

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

to get quadratic equation

Sphere

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

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

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

Planes

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

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

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

Polyhedra

- Generally we want to intersect with closed convex 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

What's Next

What we can do now

- Create basic 3D web applications
  - can integrate with other HTML5 packages
- Work with event-driven input
- Use a variety of texture-based methods
- Make use of off-screen rendering

What we haven't covered

- More OpenGL capabilities
- Modeling
- Alternate renderers
- Integration with Web
- Where are things going

What's in desktop OpenGL

- Much more control and many more options
  - Geometry,Tessellation and Compute Shaders
  - Level of Detail (LOD)
  - 1-4 Dimensional Textures
  - Many more texture options
- Vertex Array Buffers
- Occlusion Queries

What should we expect soon in APIs

- Movement of more desktop OpenGL features to ES and WebGL
- WebCL 1.0 released March 2014
- ES 3.0 and ES 3.1
- ECMA 6 Script draft (new version of JS)
- Many JS variants such as Coffee Script

The Players have Changed

- Originally hardware and software was dominated by the scientific and CAD communities
  - SGI key for both hardware and software
  - OpenGL developed by SGI
- With PCs and graphics cards leadership moved to Microsoft and game users
  - DirectX
  - Video Toaster

Players Have Changed

- Development of GPUs
  - Nvidia, AMD and Intel dominate
  - OpenGL makes a comeback
  - Cg (Nvidia) leads to GLSL
  - Interactive games control direction of hw and sw
- Web and smart phones
  - Google, Mozilla, Nokia and others dominate software
  - ARM dominates smart phone chips

Advanced Topics

- Level of Detail (LOD)
- Image based rendering
- Light field rendering
- Ray Tracing
- Volume Rendering
- Point Clouds
- Particle Systems
- Information Visualization

What about Games?

- Need more courses
  - Digital Storytelling
  - Game AI
  - HCI
  - Real-time graphics

Supercomputing

- Fastest supercomputers use GPUs for floating point operations
  - GPGPU
  - OpenCL/WebGL
  - Compute shaders
- Low power is a major issue
  - Exascale machine will require 20MW
  - Intel vs ARM?