EECS 487: Interactive Computer Graphics

Lecture 27:

- Introduction to Global Illumination and Ray Tracing

Ray Tracing

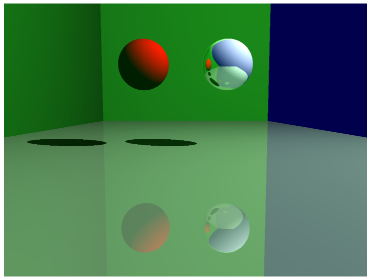
Introduction and context
- ray casting

Recursive ray tracing
- shadows
- reflection
- refraction

Ray tracing implementation

Distributed Ray Tracing
- anti-aliasing
- soft-shadows
- motion blur
- depth-of-field
- glossy surface
- translucency

Global Illumination



Computes the color at a point in terms of light directly emitted by light sources and of light indirectly reflected by and transmitted through other objects, allowing for computation of shadows, reflection, refraction, caustics, and color bleed

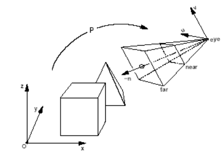
Light paths are complex, not light → triangle → pixel

Nature finds equilibrium efficiently

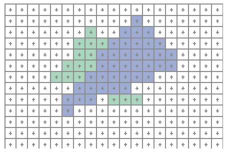
Computers struggle ☹

Akeley

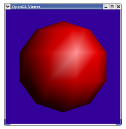
Pipelined Rasterization



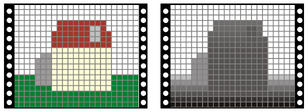
Perform projection of vertices



Rasterize triangle: find which pixels should be lit



Compute per-pixel color



Test visibility, update frame buffer

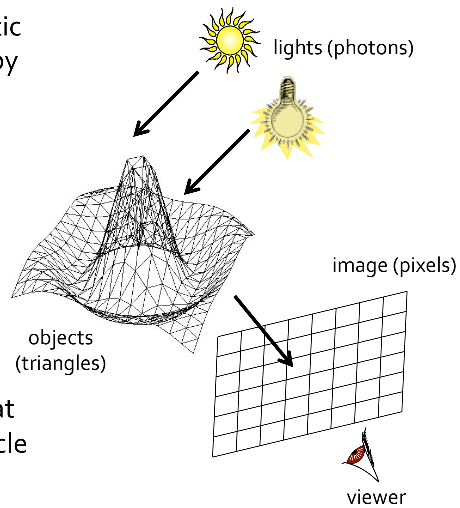
Durand

Ray Tracing

Can we produce more realistic results if we render a scene by simulating physical light transport?

The Greeks questioned the nature of light: do light rays proceed from the eye to the light or from the light to the eye?

Modern theories of light treat it as both a wave and a particle



Hanrahan,Yu,Curless,Akeley

Geometric Optics

We will take a combined and somewhat simpler view of light–the view of geometric optics

Light sources send off photons in all directions

- model these as particles that bounce off objects in the scene
- each photon has a wavelength and energy (color and intensity)
- when photons bounce, some energy is absorbed, some reflected, some transmitted
- photons bounce until:
  - all of its energy is absorbed (after too many bounces)
  - it departs the known universe (not just the view volume!)
  - it strikes the image plane and its contribution is added to appropriate pixel

We call the path of these photons "light rays"

If we can model light rays we can generate images

Curless, Hodgins

Geometric Optics

Light rays follow these rules:

- travel in straight lines in free space
- do not interfere with each other if they cross (light is invisible!)
- travel from the light sources to the eye, but the physics is invariant under path reversal (reciprocity)
- obey the laws of reflection and refraction

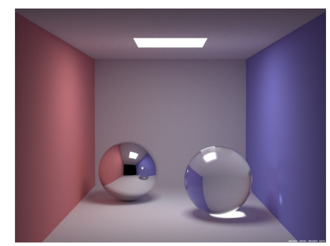
Curless, Hanrahan

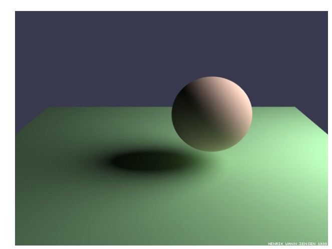
Why Trace Rays?

More elegant than pipelined rasterization, especially for sophisticated physics:

- modeling light reflectance, e.g., from skin
- modeling light transport, e.g., inter-reflection, caustics
- rendering, e.g., soft shadows

Easiest photorealistic global illumination renderer to implement





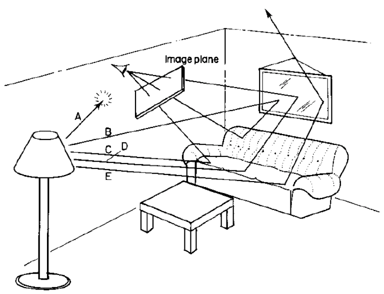
Jensen, Harto8

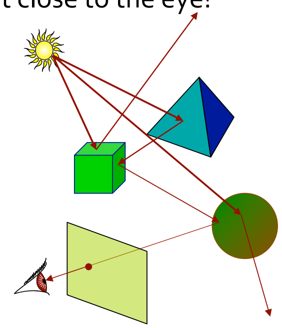
Light-Ray Tracing

Rays emanate from light sources and bounce around

Rays that pass through the image plane and enter the eye contribute to the final image

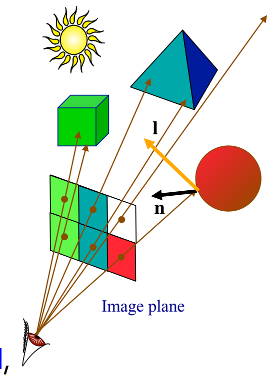
Very inefficient since it computes many rays that are never seen, most rays will never even get close to the eye!





Merrell08

Ray Casting



For each pixel shoot a ray (a 3D line) from the eye into the view volume through a point on the screen

- find the nearest polygon that intersects with the ray
- shade that intersection according to light, e.g., by using the Phong illumination model, or other physically-based BRDFs, to compute pixel color

Computes only visible rays (since we start at the eye): more efficient than light tracing

Introduced by Appel in 1968 for local illumination (on pen plotter)

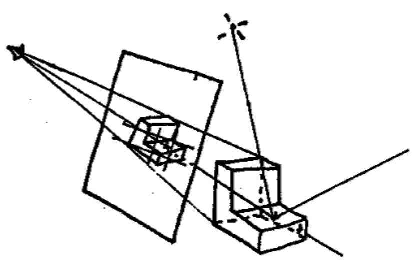
Merrell, Curless

Ray Casting Illumination Model

So far it's still light → triangle → pixel

With ray casting, we additionally shoot a ray from each point toward each light in the scene and ask:

- is the light visible from the intersection point?
  - does the ray intersect any objects on the way to the light?
- if light is not visible, it doesn't lit the point (point in shadow)



Ray Casting vs. Pipelined Rasterization

Ray Casting (Image Order)

Image → Output Image

Image ⇅ Cast rays ⇅ Find first intersection ⇅ Shading

Find first intersection ↔ Scene Graph

Shading ↔ Scene Graph

Pipelined Rasterization (Object Order)

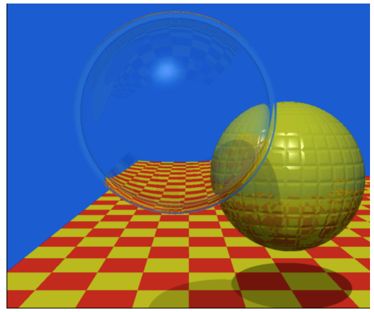
Command → Geometry → Rasterization → Texture → Display

Zwicker06

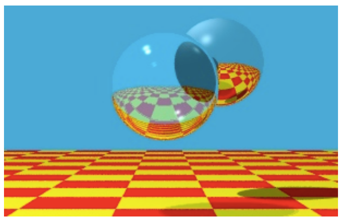
Ray Tracing

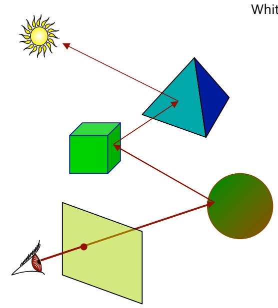
Whitted introduced ray tracing to the graphics community in 1980:

- recursive ray casting
- first global illumination model:
  - an object's color is influenced by lights and other objects in the scene → shadows
  - simulates specular reflection and refractive transmission



Whitted80





Merrell08

Ray Tracing vs. Pipelined Rasterization

Ray Tracing

+ no computation for hidden parts
– usually implemented in software
– slow, batch rendering
– no dominant standard for scene description (RenderMan format, POVray, PBRT, ...)
+ photo-realistic images: more complex shading and lighting effects possible

Pipelined Rasterization

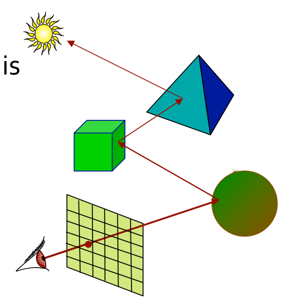
+ implemented in GPU
+ standardized APIs
+ interactive rendering, games
– limited photo-realism: harder to get global illumination (but getting closer)

Zwicker06

Recursive Ray Tracing

Basic idea:

- Each pixel gets light from just one direction—the line through the screen and the eye
- Any photon contributing to that pixel's color has to come from this direction
- So head in that direction and see what is sending light
  - if we find nothing—done
  - if we hit a light source—done
  - if we hit a surface—see where that surface is lit from, recurse

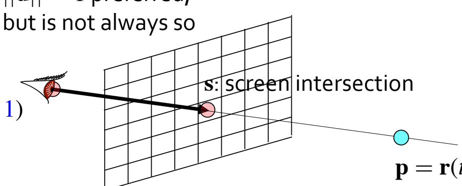


Hodgins, Merrell

Ray: a Half Line

Direction:
$\mathbf{d} = \mathbf{s} - \mathbf{e} = (x_d, y_d, z_d, 0)$
$||\mathbf{d}|| = 1$ preferred,
but is not always so

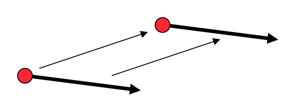
Anchor point:
$\mathbf{e} = (x_e, y_e, z_e, 1)$



$\mathbf{p} = \mathbf{r}(t) = \mathbf{e} + t\,\mathbf{d}$,
$t > 0$

Ray: $\mathbf{r} = (\mathbf{e}, \mathbf{d})$

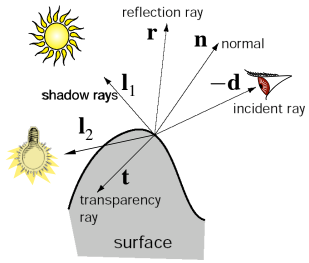
Translating rays:
points translate,
directions don't
$\mathbf{Tr} = (\mathbf{Te}, \mathbf{Td}) = (\mathbf{Te}, \mathbf{d})$



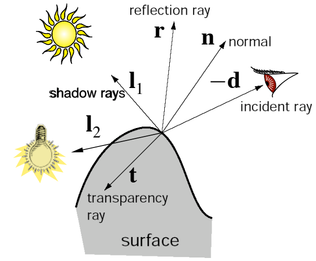
Hart08

Recursive Ray Tracing Algorithm

1. For each pixel ($\mathbf{s}$), trace a primary ray from the eye ($\mathbf{e}$), in the direction $\mathbf{d} = (\mathbf{s} - \mathbf{e})$ to the first visible surface
2. For each intersection, trace secondary rays, to collect light:
   - shadow rays in directions $\mathbf{l}_i$ to light source i
   - reflective surface: reflected ray in direction $\mathbf{r}$, recurse
   - transparent surface: refracted/transmitted/transparency ray in direction $\mathbf{t}$, recurse



Shadow Ray



At every ray-object intersection, we shoot a shadow ray towards each light source

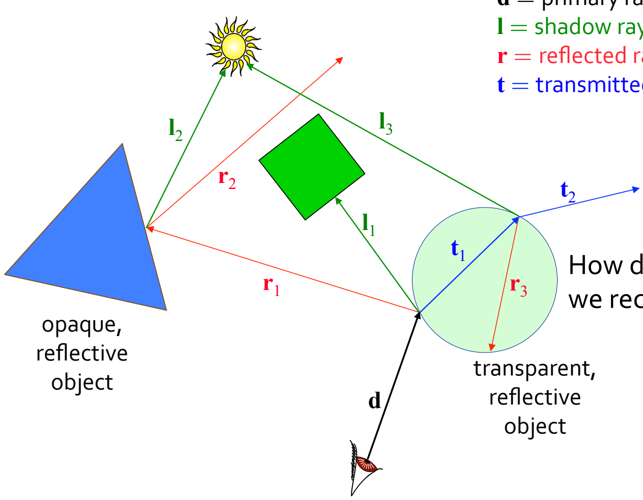
If the ray hits the light source ($\mathbf{l}_1$), the light is used in lighting calculation, with the shadow ray as the light direction

If the ray hits an object ($\mathbf{l}_2$), the intersection is in shadow and the light is not used in lighting calculation

Shadow rays do not spawn additional rays (no transparency through transparent object!)

Recursive Ray Tracing

$\mathbf{d}$ = primary ray
$\mathbf{l}$ = shadow rays
$\mathbf{r}$ = reflected rays
$\mathbf{t}$ = transmitted rays



How deep do we recurse?

Ray Tree

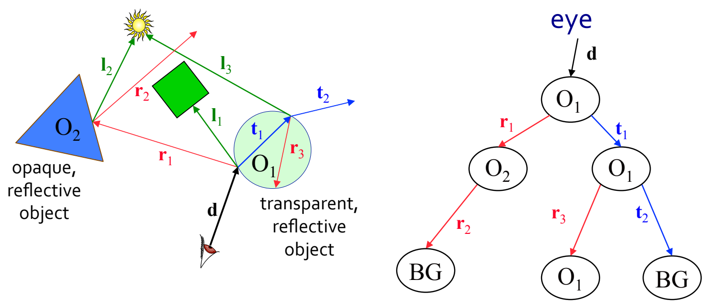
Each intersection may spawn secondary rays:

- reflected and transmitted rays form a ray tree
  - nodes are the intersection points
  - edges are the reflected and refracted rays
- shadow rays are sent from every intersection point (to determine if point is in shadow), but they do not spawn additional rays

Rays are recursively spawned until:

- ray does not intersect any object
- tree reaches a maximum depth
- light reaches some minimum value (reflected/ refracted contribution to color becomes too small)

Ray Tree Example

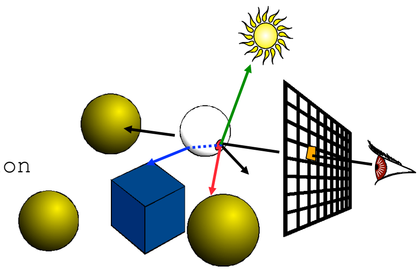


Ray tree is evaluated bottom up:

- depth-first traversal (by recursion)
- the node color is computed based on its children's colors (BG: background, ambient color)
- don't forget to negate the normal when inside an object!

Merrello8

raytrace()



```
raytrace(ray r)
  find first intersection
  color = ambient term
  for every light
     cast shadow ray
     if (not in shadow)
        color += local diffuse+specular terms
        // Phong illumination model
  if reflective surface
    color += reflectedContrib // constant
     * raytrace(reflected ray)*local specular term
  if transparent object
    color += refractedContrib // constant
     * raytrace(refracted ray)*local diffuse term
```

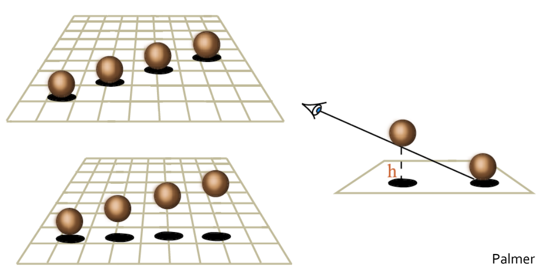
Durand

Shadows

Increase realism by adding spatial relations

- provide contact points, stop "floating" objects
- provide depth cue
- emphasize illumination direction

Provide "atmosphere"



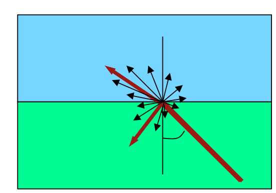
Palmer

Light Hitting a Surface

Some of it is absorbed (heat, vibration)
Some of it is reflected (bounces back)
Some of it is refracted (goes inside the material)

The proportion of absorbed, reflected, and refracted light depend on the medium, the frequency of light, and on the angle between the direction of incident light and the surface normal

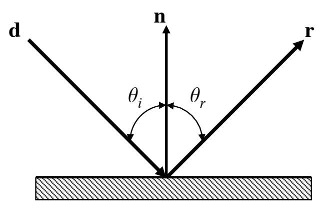
Light may also be scattered by the medium it traverses



Rossignac

Perfect Specular Reflection

Reflection: arriving energy from one direction goes out in only one reflection direction



$$\mathbf{r} = \mathbf{d} - 2(\mathbf{d} \bullet \mathbf{n})\mathbf{n}$$

$$\theta_i = \theta_r$$

Refraction

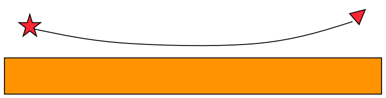
Light transmits through transparent objects

Light entering a new medium is refracted

- its trajectory bends inwards when entering a denser medium
  - think of the wheel on one side of a cart slowing down first
  - similarly, sound bends towards cooler air

Why does hot road appear wet?

- light is bent: light travels faster through the hot air near the ground

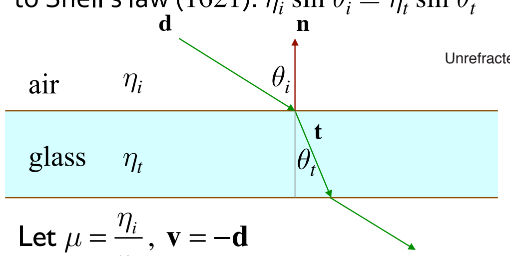


Rossignac

Refraction

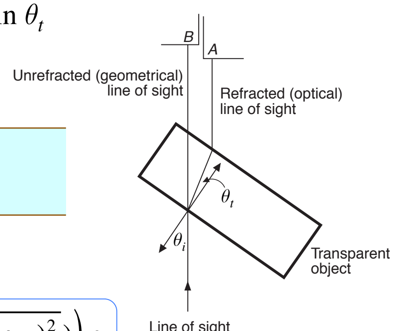
Assume that $k_i c$ is the speed of light in medium M_i and $k_t c$ is the speed of light in medium M_t:

- index of refraction of a material M is: $\eta = 1/k$
- light bends when moving from one medium to another according to Snell's law (1621): $\eta_i \sin\theta_i = \eta_t \sin\theta_t$



Let $\mu = \dfrac{\eta_i}{\eta_t}$, $\mathbf{v} = -\mathbf{d}$

$$\mathbf{t} = \mu\mathbf{v} - \left(\mu(\mathbf{n} \bullet \mathbf{v}) + \sqrt{1 - \mu^2(1 - (\mathbf{n} \bullet \mathbf{v})^2)}\right)\mathbf{n}$$



Merrell, Rossignac,FvD

Fresnel Coefficient

Captures how much light reflects from a smooth interface between two materials (F, fraction of light reflected):

$$\mathbf{c} = F^*\mathbf{c}_{reflection} + (1-F)\,\mathbf{c}_{refraction}$$

- reflectance depends on angle of incidence
  - not so much for metal (conductor material)
  - but dramatically for water/glass (dielectric/insulator material): 4% at normal, 100% at grazing angle

Schlick's approximation of Fresnel coefficient:

$$F(\theta) = F(0) + (1-F(0))(1-(\mathbf{n}\bullet\mathbf{v}))^5$$

where $F(0) = \left(\dfrac{\eta_t - \eta_i}{\eta_t + \eta_i}\right)^2$

is the Fresnel coefficient at normal (0°)

Gold: $F(0) = 0.82$
Silver: $F(0) = 0.95$
Glass: $F(0) = 0.04$
Diamond: $F(0) = 0.15$

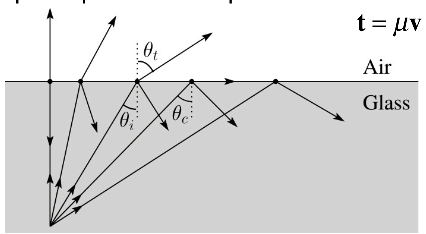
Zwicker06, Gillies09

Total Internal Reflection

When going from a dense to a less dense medium, the angle of refraction becomes larger than the angle of incidence ($\theta_t > \theta_i$)

If the angle of incidence is too large ($\geq \theta_c$), light can get trapped inside the dense material

- diamond $\rightarrow$ air: $\theta_c = 24.6^o$
- water $\rightarrow$ air: $\theta_c = 48.6^o$
- principle behind optical fiber



$$\mathbf{t} = \mu\mathbf{v} - \left(\mu(\mathbf{n}\bullet\mathbf{v}) + \sqrt{1-\mu^2(1-(\mathbf{n}\bullet\mathbf{v})^2)}\right)\mathbf{n}$$

Can be negative for grazing angles when $\eta > 1$, e.g., when going from glass to air, resulting in total internal reflection (no refraction)

Hart