Ray Tracing

- Forward & Backward Ray Tracing
- Ray Tracing
- Ray-Surface Intersection Testing
- Shadows
- Reflections
- Transmission

Next time: efficient ray tracing
Ray Tracing

Real-time Ray traced Bugatti Veyron with reflected reflections, adaptive FSAA and realistic shadows. [Source: Nvidia]
Object-oriented vs. Pixel-oriented Rendering

OpenGL rendering:
walk through objects, transforming and then drawing each one unless the z buffer says that it is not in front

Ray tracing
walk through each pixel looking for what object (if any) should be shown there
Light is Bouncing Photons

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

If we can model photon bounces we can generate images

Technique: follow each photon from the light source until:
  All of its energy is absorbed (after too many bounces)
  It departs the known universe (not just the part of the world that is within the viewing volume!)
  It strikes the image and its contribution is added to appropriate pixel
Forward Ray Tracing

Rays are the paths of these photons
This method of rendering by following photon paths is called *ray tracing*

*Forward* ray tracing follows the photon in direction that light travels (from the source)

BIG problem with this approach:
  Only a tiny fraction of rays reach the image
  Many, many rays are required to get a value for each pixel

Ideal Scenario:
  We'd like to magically know which rays will eventually contribute to the image, and trace only those
Backward Ray Tracing

The solution is to start from the image and trace backwards—*backward* ray tracing

Start from the image and follow the ray until the ray finds (or fails to find) a light source
Backward Ray Tracing

Basic idea:

Each pixel gets light from just one direction—the line through the image point and focal point

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 hit a light source—done
  If we find nothing—done
  If we hit a surface—see where that surface is lit from

At the end we’ve done forward ray tracing, but ONLY for the rays that contribute to the image
Ray Tracing

The basic algorithm is
compute u, v, w basis vectors
for each pixel do
shoot ray from eye point through pixel (x,y) into scene
intersect with all surfaces, find first one the ray hits
shade that point to compute pixel (x,y)’s color
Ray Tracing
Computing Rays

\[ p(t) = e + t(s - e) \]

\[ t = 0 \text{ origin of the ray} \]
\[ t > 0 \text{ in positive direction of ray} \]
\[ t < 0 \Rightarrow \text{then } p(t) \text{ is behind the eye} \]
\[ t_1 < t_2 \Rightarrow p(t_1) \text{ is closer to the eye than } p(t_2) \]
Computing Rays

Where is s? (x,y of image)
Intersection of ray with image plane

Details in book.
Derived using viewing transformations
Ray Object Intersection

Sphere
Triangle
Polygon

blackboard
Ray Object Intersection

Sphere
Triangle
Polygon

blackboard
Ray Object Intersection

Sphere
Triangle
Polygon

Ray-polygon—in book
Intersection with plane of polygon
in/outside of polygon determination

Ray-triangle—3D models composed of triangles

Ray-sphere—early models for raytracing, and now bounding volumes
Global vs. Local Rendering Models

Local rendering models: the color of one object is independent of its neighbors (except for shadows)
Missing scattering of light between objects, real shadowing

Global Rendering Models
  - Raytracing—specular highlights
  - Radiosity—diffuse surfaces, closed environments
Recursive Ray Tracing

Four ray types:

- Eye rays: originate at the eye
- Shadow rays: from surface point toward light source
- Reflection rays: from surface point in mirror direction
- Transmission rays: from surface point in refracted direction
Writing a Simple Ray Caster (no bounces)

Raycast() // generate a picture
    for each pixel x, y
        color(pixel) = Trace(ray_through_pixel(x, y))

Trace(ray) // fire a ray, return RGB radiance
    // of light traveling backward along it
    object_point = Closest_intersection(ray)
    if object_point return Shade(object_point, ray)
    else return Background_Color

Closest_intersection(ray)
    for each surface in scene
        calc_intersection(ray, surface)
    return the closest point of intersection to viewer
    (also return other info about that point, e.g., surface normal, material properties, etc.)

Shade(point, ray) // return radiance of light leaving
    // point in opposite of ray direction
    calculate surface normal vector
    use Phong illumination formula (or something similar)
to calculate contributions of each light source
Shadow Rays

$p + t l$ does not hit any objects
$q + t l$ does hit an object and is shadowed

In the same for both points because this is a directional light (infinitely far away)
Specular Reflection Rays

\[ \mathbf{r} \quad \mathbf{n} \quad \theta \quad \theta \quad \mathbf{d} \]
Transmission Rays

Dielectrics—transparent material that refracts (and filters) light. Diamonds, glass, water, and air.

Light bends by the physics *principle of least time*
  - light travels from point A to point B by the fastest path
  - when passing from a material of one index to another *Snell’s law* gives the angle of refraction

When traveling into a denser material (larger \( n \)), light bends to be more perpendicular (eg air to water) and vice versa

<table>
<thead>
<tr>
<th>MATERIAL</th>
<th>INDEX OF REFRACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>air/vacuum</td>
<td>1</td>
</tr>
<tr>
<td>water</td>
<td>1.33</td>
</tr>
<tr>
<td>glass</td>
<td>about 1.5</td>
</tr>
<tr>
<td>diamond</td>
<td>2.4</td>
</tr>
</tbody>
</table>
Refraction
Transmission Rays

Dielelectrics—transparent material that refracts (and filters) light. Diamonds, glass, water, and air.

Snell’s law:

\[ n \sin \theta = n_t \sin \phi \]

\( n \) is the refractive index of the first material.

\( n_t \) is the refractive index of the second material.
Transmission Rays

Dielectrics—transparent material that refracts (and filters) light. Diamonds, glass, water, and air.

Snell’s law:

\[ n \sin \theta = n_t \sin \phi \]

\( n \) is the refractive index of the first material.

\( n_t \) is the refractive index of the second material.
Transmission Rays
Total Internal Reflection

Practice Problems

In a backwards ray tracing algorithm that follows rays through a single bounce only, the color rendered at a pixel is a sum of what components? Write an equation for computing pixel color and label the terms. Include direct lighting, specular reflection, and specular refraction.

Now suppose you follow rays through two bounces (i.e., a ray can bounce off two objects in succession). What new terms are added? What effects can we capture in two bounces that cannot be captured in one?
Practice Problems

Read about the algorithm for ray-triangle intersection in the Shirley text. Draw up an example and work it out to test your understanding of how to solve for the intersection in this manner.

Create a numerical example problem where total internal reflection is demonstrated.