Ray-Tracing

**Basic Algorithm (Whithead):**

for every pixel $p_i$ {

Generate ray $r$ from camera position through pixel $p_i$

for every object $o$ in scene {

if ($r$ intersects $o$)

Compute lighting at intersection point, using local
normal and material properties; store result in $p_i$

else

$p_i$ = background color

} 

}
Ray-Tracing

Issues:
- Generation of rays
- Intersection of rays with geometric primitives
- Geometric transformations
- Lighting and shading
- Efficient data structures so we don’t have to test intersection with every object

Ray-Tracing

Shadows

Approach:
- To test whether point is in shadow, send out shadow rays to all light sources
  - If ray hits another object, the point lies in shadow
**Ray-Tracing Reflections/Refractions**

**Approach:**
- Send rays out in reflected and refracted direction to gather incoming light
- That light is multiplied by local surface color and Fresnel term, and added to result of local shading

---

**Ray Tracing**

**Data Structures**
- Goal: reduce number of intersection tests per ray
- Lots of different approaches:
  - (Hierarchical) bounding volumes
  - Hierarchical space subdivision
    - Oct-tree, k-D tree, BSP tree
Bounding Volumes

Idea:
- Rather than testing every ray against a potentially very complex object (e.g. triangle mesh), do a quick conservative test first which eliminates most of the rays
  - Surround complex object by very simple, easy to test geometry (typically sphere or axis-aligned box)
    - Want to make bounding volume as tight as possible!

Hierarchical Bounding Volumes

Extension of previous idea:
- Use bounding volumes for groups of objects
Regular Grid

Subdivide space into rectangular grid:

- Associate every object with the cell(s) that it overlaps with
- Find intersection: traverse grid

In 3D: regular grid of cubes (voxels):

Soft Shadows & Area Light Sources

CPSC 314
Area Light Sources

**So far:**
- All lights were either point-shaped or directional
  - *Both for ray-tracing and the rendering pipeline*
- Thus, at every point, we only need to compute lighting formula and shadowing for **ONE** light direction

**In reality:**
- All lights have a finite area
- Instead of just dealing with one direction, we now have to integrate over all directions that go to the light source

---

**Area Light Sources**

**Area lights produce soft shadows:**
- In 2D:

![Diagram](attachment:image.png)

- **Occluding surface**
- **Receiving surface**
- **Umbra** (core shadow)
- **Penumbra** (partial shadow)
Area Light Sources

**Point lights:**
- Only one light direction:
  \[ I_{\text{reflected}} = \rho \cdot V \cdot I_{\text{light}} \]
- \( V \) is visibility of light (0 or 1)
- \( \rho \) is lighting model (e.g., diffuse or Phong)

Area Light Sources

**Area Lights:**
- Infinitely many light rays
- Need to integrate over all of them:
  \[ I_{\text{reflected}} = \int_{\text{light directions}} \rho(\omega) \cdot V(\omega) \cdot I_{\text{light}}(\omega) \cdot d\omega \]
- Lighting model visibility and light intensity can now be different for every ray!
Rewrite the integration

- Instead of integrating over directions

\[ I_{\text{reflected}} = \int_{\text{light directions}} \rho(\omega) \cdot V(\omega) \cdot I_{\text{light}}(\omega) \cdot d\omega \]

we can integrate over points on the light source

\[ I_{\text{reflected}}(q) = \int_{s,t} \frac{\rho(p - q) \cdot V(p - q)}{\lVert p - q \rVert^2} \cdot I_{\text{light}}(p) \cdot ds \cdot dt \]

where q: point on reflecting surface, p = F(s,t) is a point on the area light
- We are integrating over p
- Denominator: quadratic falloff!

Integration

Problem:
- Except for the simplest of scenes, either integral is **not solvable analytically**!
- This is mostly due to the visibility term, which could be arbitrarily complex depending on the scene

So:
- Use numerical integration
- Effectively: approximate the light with a whole number of point lights
Numerical Integration

Regular grid of point lights

- Problem: will see 4 hard shadows rather than as soft shadow
- Need LOTS of points to avoid this problem

Monte Carlo Integration

Better:
- Randomly choose the points
- Use different points on light for computing the lighting in different points on reflecting surface
- This produces random noise
- Visually preferable to structured artifacts
Monte Carlo Integration

Formally:

Approximate integral with finite sum

\[ I_{\text{reflected}}(q) = \int_{s,t} \frac{p(p - q) \cdot V(p - q) \cdot I_{\text{light}}(p) \cdot ds \cdot dt}{1 - \|p - q\|^2} \]

\[ = \frac{A}{N} \sum_{i=1}^{N} \rho(p_i - q) \cdot V(p_i - q) \cdot I_{\text{light}}(p_i) \]

where

- The \( p_i \) are randomly chosen on the light source
  - With equal probability!
- \( A \) is the total area of the light
- \( N \) is the number of samples (rays)
Sampling

Sample directions vs. sample light source

- Most directions do not correspond to points on the light source
  - Thus, variance will be higher than sampling light directly

Monte Carlo Integration

Note:

- This approach of approximating lighting integrals with sums over randomly chosen points is much more flexible than this!
- In particular, it can be used for global illumination
  - Light bouncing off multiple surfaces before hitting the eye
**Global Illumination**

**So far:**
- Have considered only light directly coming from the light sources
  - As well as mirror reflections, refraction

**In reality:**
- Light bouncing off diffuse and/or glossy surfaces also illuminates other surfaces
  - *This is called global illumination*

---

**Direct Illumination**

Image by Henrik Wann Jensen
Global Illumination

Rendering Equation

Equation guiding global illumination:

\[ L_e(x, \omega_o) = L_e(x, \omega_o) + \int_{\Omega} \rho(x, \omega_i, \omega_0) L_r(\omega_i) d\omega_i \]

Where

- \( \rho \) is the reflectance from \( \omega_i \) to \( \omega_o \) at point \( x \)
- \( L_e \) is the outgoing (i.e. reflected) radiance at point \( x \) in direction \( \omega_i \)
  - Radiance is a specific physical quantity describing the amount of light along a ray
  - Radiance is constant along a ray
- \( L_o \) is the emitted radiance (=0 unless point \( x \) is on a light source)
- \( R \) is the “ray-tracing function”. It describes what point is visible from \( x \) in direction \( \omega_i \)
Rendering Equation

**Equation guiding global illumination:**

\[ L_e(x,\omega_o) = L_e(x,\omega_o) + \int_{\Omega} \rho(x,\omega_i,\omega_o) L_i(\omega_i) d\omega_i \]

\[ = L_e(x,\omega_o) + \int_{\Omega} \rho(x,\omega_i,\omega_o) L_o(R(x,\omega_i),-\omega_i) d\omega_i \]

**Note:**

- The rendering equation is an integral equation
- This equation cannot be solved directly
  - Ray-tracing function is complicated!
  - Similar to the problem we had computing illumination from area light sources!

Ray Casting

- Cast a ray from the eye through each pixel
- The following few slides are from Fred Durand (MIT)
Ray Tracing

- Cast a ray from the eye through each pixel
- Trace secondary rays (light, reflection, refraction)

Monte Carlo Ray Tracing

- Cast a ray from the eye through each pixel
- Cast random rays from the visible point
  - Accumulate radiance contribution
Monte Carlo Ray Tracing

- Cast a ray from the eye through each pixel
- Cast random rays from the visible point
- Recurse

Monte Carlo

- Cast a ray from the eye through each pixel
- Cast random rays from the visible point
- Recurse
Monte Carlo

- Systematically sample primary light

Monte Carlo Path Tracing

**In practice:**

- Do not branch at every intersection point
  - *This would have exponential complexity in the ray depth!*
- Instead:
  - *Shoot some number of primary rays through the pixel (10s-1000s, depending on scene!)*
  - *For each pixel and each intersection point, make a single, random decision in which direction to go next*
Monte Carlo Path Tracing

- Trace only one secondary ray per recursion
- But send many primary rays per pixel
- (performs antialiasing as well)

How to Sample?

Simple sampling strategy:
- At every point, choose between all possible reflection directions with equal probability
- This will produce very high variance/noise if the materials are specular or glossy
- Lots of rays are required to reduce noise!

Better strategy: importance sampling
- Focus rays in areas where most of the reflected light contribution will be found
- For example: if the surface is a mirror, then only light from the mirror direction will contribute!
- Glossy materials: prefer rays near the mirror direction
How to Sample?

- Images by Veach & Guibas

Naive sampling strategy

Multiple importance sampling

How to Sample?

**Sampling strategies are still an active research area!**

- Recent years have seen drastic advances in performance
- Lots of excellent sampling strategies have been developed in statistics and machine learning
  - *Many are useful for graphics*
How to Sample?

Objective:
• Compute light transport in scenes using stochastic ray tracing
  – Monte Carlo, Sequential Monte Carlo
  – Metropolis

[Burks, Ghosh, Heidrich ‘05]
[Ghosh, Heidrich ‘06]
[Ghosh, Doucet, Heidrich ‘06]

How to Sample?

• E.g: importance sampling (left) vs. Sequential Monte Carlo (right)
More on Global Illumination

This was a (very) quick overview
• More details in CPSC 514 (Computer Graphics: Rendering)
• Not offered this year, but in 2008/9

Coming Up

Tuesday:
• Color

Thursday:
• Curves & surfaces