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

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:
**Area Light Sources**

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


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


**Integrating over Light Source**

**Rewrite the integration**
- Instead of integrating over directions
  \[ I_{\text{reflect}} = \int_{\text{light}} \rho(\omega) \cdot V(\omega) \cdot I_{\text{light}}(\omega) \cdot d\omega \]
- We can integrate over points on the light source
  \[ I_{\text{reflect}}(q) = \int_{s} \rho(p-q) \cdot V(p-q) \cdot I_{\text{light}}(q) \cdot ds \cdot dt \]
- Where \( q \) is a 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
  \[ I_{\text{approx}}(q) = \int_{p_0}^{p_f} f(p, q) \cdot dV \cdot dt \]
  \[ = A \sum_i^{N} \frac{V(p_i - q) \cdot f(p_i, q)}{p_i - q} \cdot I_{\text{light}}(p_i) \]

  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)

Monte Carlo Integration

**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
Global Illumination

Rendering Equation

Equation guiding global illumination:
\[ L_x(x,ω_x) = L_y(x,ω_y) + \int p(x,ω_x,ω_y) R(ω_x,ω_y) dω_y \]

Where
- \( p \) is the reflectance from \( ω_y \) at point \( x \)
- \( L_y \) is the outgoing (i.e., reflected) radiance at point \( x \) in direction \( ω_y \)
  - Radiance is a specific physical quantity describing the amount of light along a ray
  - Radiance is constant along a ray
- \( R \) 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 \( ω_y \)

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
- Systematically sample primary light

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

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

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

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

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

[Burke, Ghosh, Heidrich '05]
[Ghosh, Heidrich '06]
[Ghosh, Douté, Hellrich '06]

---

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

---

More on Global Illumination

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

---

Coming Up

**Tuesday:**
- Color

**Thursday:**
- Curves & surfaces