Ray-Tracing Soft Shadows
Global Illumination

Wolfgang Heidrich

Course News

Homework 8
- Ray-tracing, global illumination
- Discussed today, tomorrow in labs

Assignment 3 (project)
- Due Friday!!
- Demos in labs starting this Friday
- Demos are MANDATORY(!)

Reading
- Chapter 10 (ray tracing), except 10.8-10.10
- Chapter 14 (global illumination)

Ray-Tracing

Basic Algorithm (Whithead):
for every pixel $p_i$
  { Generate ray $r$ from camera position through pixel $p_i$
    $p_i$ = background color
    for every object $o$ in scene {
      if ($r$ intersects $o$ && intersection is closer than previously
        found intersections)
        Compute lighting at intersection point, using local
        normal and material properties, store result in $p_i$
    }
  }

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

Recursive Ray Tracing

Ray tracing can handle
- Reflection (chrome)
- Refraction (glass)
- Shadows

Spawn secondary rays
- Reflection, refraction
- If another object is hit, recurse to
  find its color
- Shadow
  - Cast ray from intersection point to
  - light source, check if intersects
  - another object
Recursive Ray-Tracing

Algorithm Termination Criteria

- **Termination criteria**
  - No intersection
  - Reach maximal depth
  - **Number of bounces**
  - Contribution of secondary ray attenuated below threshold
  - Each reflection/refraction attenuates ray

Reflection

```
RayTrace(r, scene)
  obj := FirstIntersection(r, scene)
  if (no obj) return BackgroundColor;
  else begin
    if (Reflect(obj)) then
      reflect_color := RayTrace(ReflectRay(r, obj));
    else
      reflect_color := Black;
    if (Transparent(obj)) then
      refract_color := RayTrace(RefractRay(r, obj));
    else
      refract_color := Black;
    return Shade(reflect_color, refract_color, obj);
  end;
```

Refraction

```
c_1 \sin \theta_1 = c_2 \sin \theta_2
```

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 of area light producing soft shadows](image)

- Occluding surface
- Receiving surface
- Umbra (core shadow)
- Penumbra (partial shadow)

### 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)

### Integrating over Light Source

**Rewrite the integration**

- Instead of integrating over directions
  
  \[ I_{\text{reflected}} = \int_{\text{light}} \rho(a) \cdot V(a) \cdot I_{\text{light}}(a) \cdot \, da \]

- We can integrate over points on the light source
  
  \[ I_{\text{reflected}}(q) = \int \frac{\rho(p-q) \cdot V(p-q)}{1 \cdot q \cdot p} \cdot I_{\text{light}}(p) \cdot ds \cdot dt \]

- \( q \): 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 unsolvable 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

Formally:
- Approximate integral with finite sum
  \[ I_{\text{light}}(q) = \int \frac{\rho(p-q) \cdot V(p-q)}{1|p-q|^2} I_{\text{light}}(p) \cdot dp \cdot dt \]
  \[ \approx \frac{A}{N} \sum_{i=1}^{N} \frac{\rho(p_i-q) \cdot V(p_i-q)}{1|p_i-q|^2} I_{\text{light}}(p_i) \]

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

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

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

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 Wenn-Jensen

Global Illumination

Image by Henrik Wenn-Jensen

Rendering Equation

**Equation guiding global illumination:**

\[
L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{\Omega} p(x, \omega, \omega_o) L_i(\omega) \, d\omega
\]

Where

- \( p \) is the reflectance from \( \omega_i \) to \( \omega_o \) at point \( x \)
- \( L_o \) is the outgoing (i.e. reflected) **radiance** at point \( x \) in direction \( \omega_o \)
- **Radiance** is a specific physical quantity describing the amount of light along a ray
- **Radiance** is constant along a ray
- \( L_e \) 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 \( \omega_i \)

\[
L_o(\mathbf{x}, \Omega) = L_e(\mathbf{x}, \omega_o) + \int_{\Omega} p(\mathbf{x}, \omega, \omega_o) L_i(\omega) \, d\omega
\]

**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.

---

**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*

---

**More on Global Illumination**

- *This was a (very) quick overview*
  - Not offered this year, but in 2008/9.
Coming Up...

Next Week:
- Global illumination