Ray Tracing Introduction

- A different algorithm for rendering 3D scenes

<table>
<thead>
<tr>
<th>Rasterization</th>
<th>Ray Tracing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Renders an image:</td>
<td></td>
</tr>
<tr>
<td>Based on:</td>
<td>Complex effects (shadows,</td>
</tr>
<tr>
<td></td>
<td>reflections) are:</td>
</tr>
<tr>
<td>Object-by-object</td>
<td>Transforming Geometry</td>
</tr>
<tr>
<td>Transforming Geometry</td>
<td>Difficult to incorporate</td>
</tr>
<tr>
<td>Pixel-by-pixel</td>
<td>Geometric reasoning about</td>
</tr>
<tr>
<td></td>
<td>light rays</td>
</tr>
<tr>
<td></td>
<td>Easy to compute in a physically-accurate manner</td>
</tr>
</tbody>
</table>
Algorithm Overview

Create a ‘virtual window’ into the scene
Algorithm Overview

Shoot ray from camera through pixel, see what it hits
Algorithm Overview

Shoot ray toward light to see if point is in shadow
Algorithm Overview

Compute shading from light source

\[ k_d l \max(\hat{\mathbf{l}} \cdot \hat{\mathbf{n}}, 0) + k_s l \max(\hat{\mathbf{r}} \cdot \hat{\mathbf{v}}, 0)^p \]
Algorithm Overview

Recursively trace ray to calculate mirror reflections
Algorithm Overview

Record pixel color
Image Raytrace (Camera cam, Scene scene, int width, int height) 
{
    Image image = new Image (width, height) ;
    for (int i = 0 ; i < height ; i++)
        for (int j = 0 ; j < width ; j++)
            {
                Ray ray = RayThruPixel (cam, i, j) ;
                Intersection hit = Intersect (ray, scene) ;
                image[i][j] = FindColor (hit) ;
            }
    return image ;
}
Generating Viewing Rays

Rectilinear image plane build from four points

\[ P = (1 - u)((1 - v)LL + (v)UL) + (u)((1 - v)LR + (v)UR) \]
Generating Viewing Rays

• Ray equation

\[ R(t) = E + t(P - E) \]

\[ t \in [1 \ldots + \infty] \]

• Through eye at \( t = 0 \)
• At pixel center at \( t = 1 \)
Defining Camera & Image Plane

- Camera at eye, looking at lookAt, with up direction being up

\[ a = \text{lookAt} - \text{eye} \]
\[ b = \text{up} \]
Defining Camera & Image Plane

- \( \mathbf{a}, \mathbf{b} \) not guaranteed to be orthogonal or unit length
- Form an orthonormal basis:

\[
\begin{align*}
\mathbf{w} &= \frac{\mathbf{a}}{\|\mathbf{a}\|} \\
\mathbf{u} &= \frac{\mathbf{b} \times \mathbf{w}}{\|\mathbf{b} \times \mathbf{w}\|} \\
\mathbf{v} &= \mathbf{w} \times \mathbf{u}
\end{align*}
\]
Defining Camera & Image Plane

\[ C = \text{eye} + w \]
\[ y = \tan(\text{fovy}/2) \]
\[ x = \tan(\text{aspect} \cdot \text{fovy} / 2) \]

\[ LL = C + (x \cdot u) - (y \cdot v) \]
\[ UL = C + (x \cdot u) + (y \cdot v) \]
\[ LR = C - (x \cdot u) - (y \cdot v) \]
\[ UR = C - (x \cdot u) + (y \cdot v) \]
Ray-Object Intersection: Spheres

• Ray equation: \( \mathbf{R}(t) = \mathbf{A} + t \mathbf{D} \)

• Implicit equation for sphere: \( |\mathbf{X} - \mathbf{C}|^2 - r^2 = 0 \)

• Combine:
  \[
  |\mathbf{R}(t) - \mathbf{C}|^2 - r^2 = 0 \\
  |\mathbf{A} + t \mathbf{D} - \mathbf{C}|^2 - r^2 = 0 \\
  \]

• Quadratic equation in \( t \)
Ray-Object Intersection: Spheres

Two solutions

One solution

Imaginary
Ray-Object Intersection: Spheres

• Intersection Point:

\[ P = A + tD \]

• Intersection Normal:

\[ \frac{P - C}{|P - C|} \]
Ray-Object Intersection: Triangles

• See Shirley Ch. 4
Ray-Scene Intersection

Intersection FindIntersection(Ray ray, Scene scene)
{
    min_t = infinity
    min_primitive = NULL
    For each primitive in scene {
        t = Intersect(ray, primitive);
        if (t > 0 && t < min_t) then
            min_primitive = primitive
            min_t = t
    }
}
return Intersection(min_t, min_primitive)
Local Shading

\[ C = k_a \sum_{l \in E_a} I + \sum_{l \in E} k_d I \max(0, L \cdot N) + k_s I \max(0, R \cdot V)^p \]

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(k_a)</td>
<td>Ambient reflectance</td>
</tr>
<tr>
<td>(k_d)</td>
<td>Diffuse reflectance</td>
</tr>
<tr>
<td>(k_s)</td>
<td>Specular reflectance</td>
</tr>
<tr>
<td>(E_a)</td>
<td>Ambient lights in scene</td>
</tr>
<tr>
<td>(E)</td>
<td>Other lights in scene</td>
</tr>
<tr>
<td>(I)</td>
<td>Intensity of light (l)</td>
</tr>
<tr>
<td>(p)</td>
<td>‘Shininess’ exponent</td>
</tr>
</tbody>
</table>
Light Types

• Point: \[ L = p_l - p_s \]
  (Light comes from a single point)

• Directional: \[ L = -d_l \]
  (Light comes from infinitely far away)
Shadows

- Detect shadow by rays to light source

\[ R(t) = S + t(L - S) \]
\[ t \in [\varepsilon \ldots 1) \]
Shadows

• Test for occluder
  • No occluder, shade normally (e.g. Phong model)
  • Yes occluder, skip light (don’t skip ambient)

• Self shadowing
  • Add shadow bias ($\epsilon$)
Mirror Reflections

- Recursive shading
  - Ray bounces off object
  - Treat bounce rays (mostly) like eye rays
  - Shade bounce ray and return color
    - Shadow rays
    - Recursive reflections
  - Add color to shading at original point
    - Specular or separate reflection coefficient

\[
R(t) = S + t B \\
t \in [\varepsilon \ldots + \infty)
\]
Shading with Reflections

\[ C = k_a \sum_{l \in E_a} I + \sum_{l \in E} k_d I \max(0, L \cdot N) + k_s I \max(0, R \cdot V)^p + k_r C_r \]

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( k_a )</td>
<td>Ambient reflectance</td>
</tr>
<tr>
<td>( k_d )</td>
<td>Diffuse reflectance</td>
</tr>
<tr>
<td>( k_s )</td>
<td>Specular reflectance</td>
</tr>
<tr>
<td>( k_r )</td>
<td>Mirror reflectance</td>
</tr>
<tr>
<td>( E_a )</td>
<td>Ambient lights in scene</td>
</tr>
<tr>
<td>( E )</td>
<td>Other lights in scene</td>
</tr>
<tr>
<td>( I )</td>
<td>Intensity of light ( l )</td>
</tr>
<tr>
<td>( p )</td>
<td>‘Shininess’ exponent</td>
</tr>
</tbody>
</table>
Mirror Reflections

- Recursion Depth
  - Truncate at fixed number of bounces

More bounces

Fewer bounces
Ray Tracing Transformed Objects

- Triangle: Still a triangle after transformation
- Sphere: Becomes ellipsoid
  - Write another intersection routine?
  - ...or reuse ray-sphere intersection code?
Ray Tracing Transformed Objects

- Idea: Intersect untransformed object with inverse-transformed ray

\[ M \cdot S = M^{-1} \cdot R \]
Ray Tracing Transformed Objects

• Transform intersection back to world coords

  – Intersection point: \( \bar{\mathbf{p}} = \mathbf{M} \cdot \mathbf{p} \)

  – Intersection normal: \( \bar{\mathbf{n}} = (\mathbf{M}^{-1})^T \cdot \mathbf{n} \)
Ray Tracing History

Ray Tracing in Computer Graphics

Appel 1968 - Ray casting

1. Generate an image by sending one ray per pixel
2. Check for shadows by sending a ray to the light
Ray Tracing History

Ray Tracing in Computer Graphics

“An improved Illumination model for shaded display,”
T. Whitted, CACM 1980

Resolution: 512 x 512
Time:
VAX 11/780 (1979) 74 min.

Spheres and Checkerboard, T. Whitted, 1979
Ray Tracing History

Image courtesy Paul Heckbert 1983
Ray Tracing History

Figure 6. A sample image. All objects are neutral grey. Color on the objects is due to caustics from the green glass balls and color bleeding from the base polygon.

Kajiya
1986
Real-Time Ray Tracing?

• Asymptotic Time Complexity

  – Rasterization: \( O(b \cdot m) \)

  – Ray Tracing: \( O(b \cdot p) \)

  – Accelerated Ray Tracing: \( O(\log b \cdot p) \)

<table>
<thead>
<tr>
<th>( p )</th>
<th># of pixels on screen</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b )</td>
<td># of objects in scene</td>
</tr>
<tr>
<td>( m )</td>
<td>Avg. # of pixels per object</td>
</tr>
</tbody>
</table>
Real-Time Ray Tracing?

"Ray tracing is embarrassingly parallelizable"  [NVIDIA, SIGGRAPH 2008]
Extras
Distribution Ray Tracing

• Anti-aliasing
  
  Send multiple rays through each pixel

One Sample  5x5 Grid  5x5 Jittered Grid
Average results together

Jittering trades aliasing for noise
Distribution Ray Tracing

• Anti-aliasing

One sample per pixel

Multiple samples per pixel
Distribution Ray Tracing

- Soft Shadows

Distribute shadow rays over light surface

- All shadow rays go through
- No shadow rays go through
- Some shadow rays go through

Figure from S. Chenney
Distribution Ray Tracing

• Soft Shadows
Distribution Ray Tracing

- Motion Blur

Distribute rays over *time*
Distribution Ray Tracing

- **Depth of Field**
  - Distribute rays across a finite camera aperture

A disk, instead of a single point
Distribution Ray Tracing

• Depth of Field

No DoF

More rays

Jittered rays for DoF

Multiple images for DoF

Even more rays
Acceleration Structures: BVH

- Build hierarchy of bounding volumes
  - Bounding volume of interior node contains all children
Acceleration Structures: BVH

- Use hierarchy to accelerate ray intersections
  - Intersect node contents only if hit bounding volume
Acceleration Structures: BVH

- Sort hits & detect early termination

```c
FindIntersection(Ray ray, Node node)
{
    // Find intersections with child node bounding volumes
    ...
    // Sort intersections front to back
    ...
    // Process intersections (checking for early termination)
    min_t = infinity;
    for each intersected child i {
        if (min_t < bv_t[i]) break;
        shape_t = FindIntersection(ray, child);
        if (shape_t < min_t) { min_t = shape_t; }
    }
    return min_t;
}
```
Acknowledgments

• Many slides shamelessly borrowed from James O’Brien, Ravi Ramamoorthi, and Pat Hanrahan