Motivation

- Ray Tracing is a core aspect of both offline and real-time rendering today
- Basic topic which I cover in CSE 167
- But not everyone does, this class also covers 168 (ray tracing is a prelude to key path tracing algorithm)
- Background for some (most?) of you, but critical topic for those who haven’t seen it before, go over it fast
- Important if you want to do the optional path tracer assignment (includes a ray tracing assignment)

http://www.edx.org/course/computer-graphics

Effects needed for Realism

- (Soft) Shadows
- Reflections (Mirrors and Glossy)
- Transparency (Water, Glass)
- Interreflections (Color Bleeding)
- Complex Illumination (Natural, Area Light)
- Realistic Materials (Velvet, Paints, Glass)
- And many more

Ray Tracing

- Different Approach to Image Synthesis as compared to Hardware pipeline (OpenGL)
- Pixel by Pixel instead of Object by Object
- Easy to compute shadows/transparency/etc

Outline

- **History**
  - Basic Ray Casting (instead of rasterization)
    - Comparison to hardware scan conversion
  - Ray-Surface Intersection
  - Shadows / Reflections (core algorithm)
  - Optimizations
  - Current Research
Ray Tracing: History

- Appel 68
- Whitted 80 [recursive ray tracing]
  - Landmark in computer graphics
- Lots of work on various geometric primitives
- Lots of work on accelerations
- Current Research
  - Real-Time ray tracing (historically, slow technique)
  - Ray tracing architecture

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

Outline

- History
- Basic Ray Casting (instead of rasterization)
  - Comparison to hardware scan conversion
- Ray-Surface Intersection
- Shadows / Reflections (core algorithm)
- Optimizations
- Current Research

Outline in Code

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;

Ray Casting

Produce same images as with OpenGL
- Visibility per pixel instead of Z-buffer
- Find nearest object by shooting rays into scene
- Shade it as in standard OpenGL
Ray Casting

Virtual Viewpoint

Virtual Screen

Objects

Comparison to hardware scan-line

- Per-pixel evaluation, per-pixel rays (not scan-convert each object). On face of it, costly
- But good for walkthroughs of extremely large models (amortize preprocessing, low complexity)
- More complex shading, lighting effects possible

Finding Ray Direction

- Goal is to find ray direction for given pixel i and j
- Many ways to approach problem
  - Objects in world coord, find dir of each ray (we do this)
  - Camera in canonical frame, transform objects (OpenGL)
- Basic idea
  - Ray has origin (camera center) and direction
  - Find direction given camera params and i and j
- Camera params as in gluLookAt
  - Lookfrom[3], LookAt[3], up[3], fov

Similar to gluLookAt derivation

- gluLookAt(eyex, eyey, eyez, centerx, centery, centerz, upx, upy, upz)
- Camera at eye, looking at center, with up direction being up

Constructing a coordinate frame?

We want to associate \( w \) with \( a \), and \( v \) with \( b \)
- But \( a \) and \( b \) are neither orthogonal nor unit norm
- And we also need to find \( u \)

\[
\begin{align*}
  w &= \frac{a}{|a|} \\
  u &= \frac{b \times w}{|b \times w|} \\
  v &= w \times u
\end{align*}
\]

Camera coordinate frame

\[
\begin{align*}
  w &= \frac{a}{|a|} \\
  u &= \frac{b \times w}{|b \times w|} \\
  v &= w \times u
\end{align*}
\]

- We want to position camera at origin, looking down \(-Z\) dirn
- Hence, vector \( a \) is given by \( \text{eye} - \text{center} \)
- The vector \( b \) is simply the \( \text{up} \) vector
Canonical viewing geometry

\[ \alpha = \tan \left( \frac{\text{fovx}}{2} \right) \times \left( \frac{j - \text{width} \times i}{\text{width} / 2} \right) \]

\[ \beta = \tan \left( \frac{\text{fovy}}{2} \right) \times \left( \frac{(\text{height} / 2) - i}{\text{height} / 2} \right) \]

Ray/Object Intersections

- Heart of Ray Tracer
  - One of the main initial research areas
  - Optimized routines for wide variety of primitives
- Various types of info
  - Shadow rays: Intersection/No Intersection
  - Primary rays: Point of intersection, material, normals
  - Texture coordinates
- Work out examples
  - Triangle, sphere, polygon, general implicit surface

Ray-Sphere Intersection

\[ \text{ray} = \vec{P} = \vec{P}_0 + \vec{P}_1 t \]

\[ \text{sphere} = (\vec{P} - \vec{C}) \cdot (\vec{P} - \vec{C}) - r^2 = 0 \]

Outline

- History
- Basic Ray Casting (instead of rasterization)
  - Comparison to hardware scan conversion
- Ray-Surface Intersection
- Shadows / Reflections (core algorithm)
- Optimizations
- Current Research

Outline in Code

```java
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 ;
}
```

Ray-Sphere Intersection

\[ \text{ray} = \vec{P} = \vec{P}_0 + \vec{P}_1 t \]

\[ \text{sphere} = (\vec{P} - \vec{C}) \cdot (\vec{P} - \vec{C}) - r^2 = 0 \]

Substitute

\[ \text{ray} = \vec{P} = \vec{P}_0 + \vec{P}_1 t \]

\[ \text{sphere} = (\vec{P}_0 + \vec{P}_1 t - \vec{C}) \cdot (\vec{P}_0 + \vec{P}_1 t - \vec{C}) - r^2 = 0 \]

Simplify

\[ t^2 (\vec{P}_1 \cdot \vec{P}_1) + 2t \vec{P}_1 \cdot (\vec{P}_0 - \vec{C}) + (\vec{P}_0 - \vec{C}) \cdot (\vec{P}_0 - \vec{C}) - r^2 = 0 \]
Ray-Sphere Intersection

\[ t^2(\vec{P}_1 \cdot \vec{P}_1) + 2t(\vec{P}_0 \cdot (\vec{P}_0 - \vec{C}) + (\vec{P}_1 \cdot (\vec{P}_1 - \vec{C}) - r^2 = 0 \]

Solve quadratic equations for \( t \)

- 2 real positive roots: pick smaller root
- Both roots same: tangent to sphere
- One positive, one negative root: ray origin inside sphere (pick + root)
- Complex roots: no intersection (check discriminant of equation first)

Ray inside Triangle

- Once intersect with plane, still need to find if in triangle
- Many possibilities for triangles, general polygons (point in polygon tests)
- We find parametrically [barycentric coordinates]. Also useful for other applications (texture mapping)

\[ P = \alpha \vec{A} + \beta \vec{B} + \gamma \vec{C} \]
\[ \alpha \geq 0, \beta \geq 0, \gamma \geq 0 \]
\[ \alpha + \beta + \gamma = 1 \]
Other primitives

- Much early work in ray tracing focused on ray-primitive intersection tests
- Cones, cylinders, ellipsoids
- Boxes (especially useful for bounding boxes)
- General planar polygons
- Many more

Ray Scene Intersection

```java
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)
}
```

Transformed Objects

- E.g. transform sphere into ellipsoid
- Could develop routine to trace ellipsoid (compute parameters after transformation)
- May be useful for triangles, since triangle after transformation is still a triangle in any case
- But can also use original optimized routines

Ray-Tracing Transformed Objects

We have an optimized ray-sphere test

- But we want to ray trace an ellipsoid...

Solution: Ellipsoid transforms sphere

- Apply inverse transform to ray, use ray-sphere
- Allows for instancing (traffic jam of cars)
- Same idea for other primitives

Outline

- History
- Basic Ray Casting (instead of rasterization)
  - Comparison to hardware scan conversion
- Ray-Surface Intersection
- Shadows / Reflections (core algorithm)
- Optimizations
- Current Research
Outline in Code

```java
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 ;
}
```

Shadows

- Numerical inaccuracy may cause intersection to be below surface (effect exaggerated in figure)
- Causing surface to incorrectly shadow itself
- Move a little towards light before shooting shadow ray

Shading Model

\[
I = K_a + K_e + \sum_{i=1}^{n} V_i (K_d \max (l_i \cdot n, 0) + K_s \max (h_i \cdot n, 0)^s)
\]

- Global ambient term, emission from material
- For each light, diffuse specular terms
- Note visibility/shadowing for each light (not in OpenGL)
- Evaluated per pixel per light (not per vertex)

Mirror Reflections/Refractions

- Generate reflected ray in mirror direction,
  Get reflections and refractions of objects

Turner Whitted 1980
Recursive Ray Tracing

For each pixel
- Trace Primary Eye Ray, find intersection
  - Trace Secondary Shadow Ray(s) to all light(s)
    - Color = Visible ? Illumination Model : 0 ;
  - Trace Reflected Ray
    - Color += reflectivity * Color of reflected ray

Recursive Shading Model

\[ I = K_a + K_s + \sum_{i} [L_i (K_r \max (I_i \cdot n, 0) + K_f \max (h_i \cdot n, 0))] + K_{ls} + K_{ft} \]

- Highlighted terms are recursive specularities [mirror reflections] and transmission (latter is extra credit)
- Trace secondary rays for mirror reflections and refractions, include contribution in lighting model
- GetColor calls RayTrace recursively (the I values in equation above of secondary rays are obtained by recursive calls)

Problems with Recursion

- Reflection rays may be traced forever
- Generally, set maximum recursion depth
- Same for transmitted rays (take refraction into account)

Effects needed for Realism

- (Soft) Shadows
- Reflections (Mirrors and Glossy)
- Transparency (Water, Glass)
- Interreflections (Color Bleeding)
- Complex Illumination (Natural, Area Light)
- Realistic Materials (Velvet, Paints, Glass)

Discussed in this lecture
Not discussed but possible with distribution ray tracing
Hard (but not impossible) with ray tracing; path tracing next time

Outline

- History
- Basic Ray Casting (instead of rasterization)
  - Comparison to hardware scan conversion
- Ray-Surface Intersection
- Shadows / Reflections (core algorithm)
- Optimizations
- Current Research

Some basic add ons

- Area light sources and soft shadows: break into grid of n x n point lights
  - Use jittering: Randomize direction of shadow ray within small box for given light source direction
  - Jittering also useful for antialiasing shadows when shooting primary rays
- More complex reflectance models
  - Simply update shading model
  - But at present, we can handle only mirror global illumination calculations
### Acceleration

Testing each object for each ray is slow
- Fewer Rays
- Adaptive sampling, depth control
- Generalized Rays
  - Beam tracing, cone tracing, pencil tracing etc.
- Faster Intersections
  - Optimized Ray-Object Intersections
  - Fewer Intersections

We just discuss some approaches at high level.

### Acceleration Structures

Bounding boxes (possibly hierarchical)
- If no intersection bounding box, needn’t check objects

Spatial Hierarchies (Oct-trees, kd trees, BSP trees)

### Acceleration Structures: Grids

```
     +-----+-----+-----+-----+-----+-----+-----+-----+
     |     |     |     |     |     |     |     |     |
     +-----+-----+-----+-----+-----+-----+-----+-----+
     |     |     |     |     |     |     |     |     |
     +-----+-----+-----+-----+-----+-----+-----+-----+
     |     |     |     |     |     |     |     |     |
     +-----+-----+-----+-----+-----+-----+-----+-----+
     |     |     |     |     |     |     |     |     |
     +-----+-----+-----+-----+-----+-----+-----+-----+
     |     |     |     |     |     |     |     |     |
     +-----+-----+-----+-----+-----+-----+-----+-----+
```

March ray along grid (need to be careful with this), test against each triangle in grid cell

More sophisticated: kd-tree, oct-tree bsp-tree
- Or use (hierarchical) bounding boxes
- Some acceleration is critical for path tracing

### Outline

- History
- Basic Ray Casting (instead of rasterization)
  - Comparison to hardware scan conversion
- Ray-Surface Intersection
- Shadows / Reflections (core algorithm)
- Optimizations
- **Current Research**

### Interactive Raytracing

- Ray tracing historically slow
- Now viable alternative for complex scenes
  - Key is sublinear complexity with acceleration: need not process all triangles in scene
- Allows many effects hard in hardware
- NVIDIA OptiX ray-tracing API like OpenGL
- Recent NVIDIA OptiX release major advance
  - Ray tracing now practical for games
  - Integration with Microsoft’s DirectX
  - Dedicated Hardware
  - Machine Learning for denoising (later in course)
Raytracing on Graphics Hardware

- Modern Programmable Hardware general streaming architecture
- Can map various elements of ray tracing
- Kernels like eye rays, intersect etc.
- In vertex or fragment programs
- Convergence between hardware, ray tracing

[Purcell et al. 2002, 2003]
http://graphics.stanford.edu/papers/photongfx