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
- Shadows / Reflections (core algorithm)
- Ray-Surface Intersection
- Optimizations
- Current Research

Chapter 4 in text
Ray Tracing History

Outlook 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 ;
} // Corresponds to ray generation, intersection, shading in 4.1

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

Outline

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

---

**Outline**

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

---

**Shadows: Numerical Issues**

- 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

---

**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
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 (13)
Hard (but not impossible) with ray tracing; radiosity methods

Outline

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

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

\[
\text{ray} = \hat{P} = \hat{P}_0 + \hat{P}_t t
\]
\[
\text{sphere} = (\hat{P} - \hat{C})(\hat{P} - \hat{C}) - r^2 = 0
\]
Ray-Sphere Intersection

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

Substitute

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

Simplify

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

Ray-Sphere Intersection

- **Intersection point**: \( \vec{P} = \vec{P}_0 + \vec{t} \)
- **Normal**: for sphere, this is same as coordinates in sphere frame of reference, useful for other tasks

\( \text{Normal} = \frac{\vec{P} - \vec{C}}{|\vec{P} - \vec{C}|} \)

Ray-Triangle Intersection

- **One approach**: Ray-Plane intersection, then check if inside triangle
- **Plane equation**: \( \vec{n} \cdot (\vec{P} - \vec{A}) = 0 \)

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)
Ray inside Triangle

\[ P = \alpha A + \beta B + \gamma C \]
\[ \alpha \geq 0, \beta \geq 0, \gamma \geq 0 \]
\[ \alpha + \beta + \gamma = 1 \]
\[ P - A = \theta(B - A) + \gamma(C - A) \]
\[ 0 \leq \beta \leq 1, \quad 0 \leq \gamma \leq 1 \]
\[ \beta + \gamma \leq 1 \]

Other primitives

- Much early work in ray tracing focused on ray-primitive intersection tests
- Cones, cylinders, ellipsoides
- Boxes (especially useful for bounding boxes)
- General planar polygons
- Many more
- Many references. For example, chapter in Glassner introduction to ray tracing (see me if interested)

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)

Mathematical details worked out in class

13.2 in text

Transformed Objects

- Consider a general 4x4 transform \( M \)
  - Will need to implement matrix stacks like in OpenGL
- Apply inverse transform \( M^{-1} \) to ray
  - Locations stored and transform in homogeneous coordinates
  - Vectors (ray directions) have homogeneous coordinate set to 0 [so there is no action because of translations]
- Do standard ray-surface intersection as modified
- Transform intersection back to actual coordinates
  - Intersection point \( p \) transforms as \( Mp \)
  - Distance to intersection if used may need recalculation
  - Normal \( n \) transform as \( M^n \). Do all this before lighting

Outline

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

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; chapter 13 briefly covers
Acceleration Structures
Bounding boxes (possibly hierarchical)
If no intersection bounding box, needn’t check objects

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

Outline
- History
- Basic Ray Casting (instead of rasterization)
  - Comparison to hardware scan conversion
- Shadows / Reflections (core algorithm)
- Ray-Surface Intersection
- 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
- OpenRT project real-time ray tracing (http://www.openrt.de)

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