

Computer Graphics

CSE 167 [Win 17], Lecture 15: Ray Tracing

Ravi Ramamoorthi

<http://viscomp.ucsd.edu/classes/cse167/wi17>

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

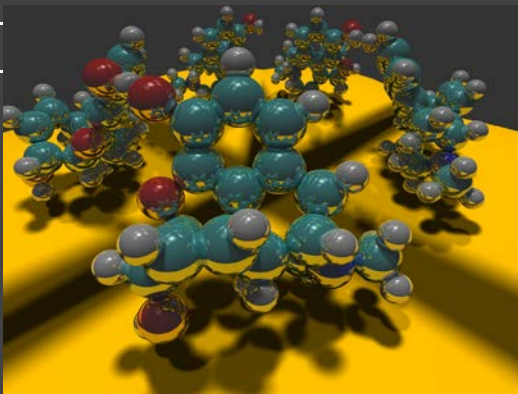


Image courtesy Paul Heckbert 1983

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

Ray Tracing: History

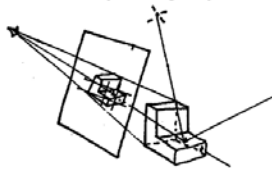
- Appel 68
- Whitted 80 [recursive ray tracing]
 - Landmark in computer graphics
- Lots of work on various geometric primitives
- Lots of work on accelerations
- Current Research
 - Real-Time raytracing (historically, slow technique)
 - Ray tracing architecture

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



CS348B Lecture 2

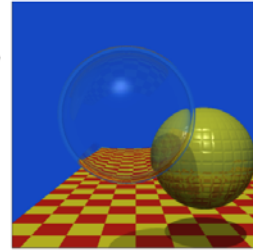
Pat Hanrahan, Spring 2009

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.
PC (2006)
6 sec.



Spheres and Checkerboard, T. Whitted, 1979

CS348B Lecture 2

Pat Hanrahan, Spring 2009

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 ;
}
```

Outline

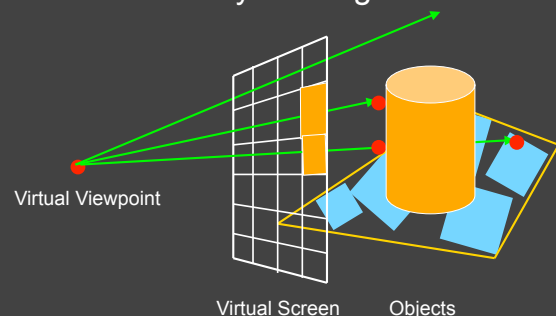
- History
- Basic Ray Casting** (instead of rasterization)
 - Comparison to hardware scan conversion
- Shadows / Reflections (core algorithm)
- Ray-Surface Intersection
- Optimizations
- Current Research

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



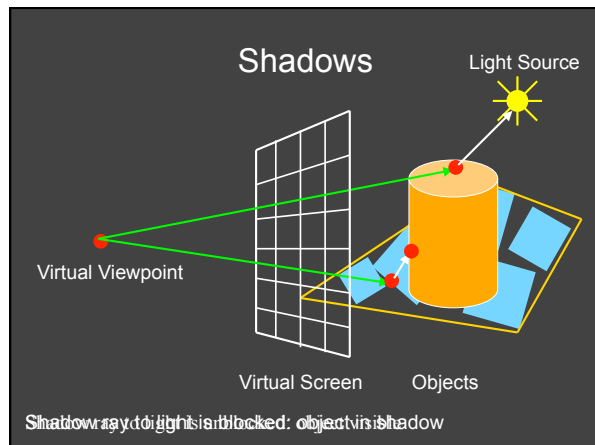
Multiple intersections! Use nearest hit (as does OpenGL)

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

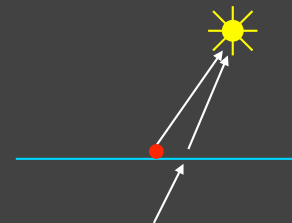
Outline

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

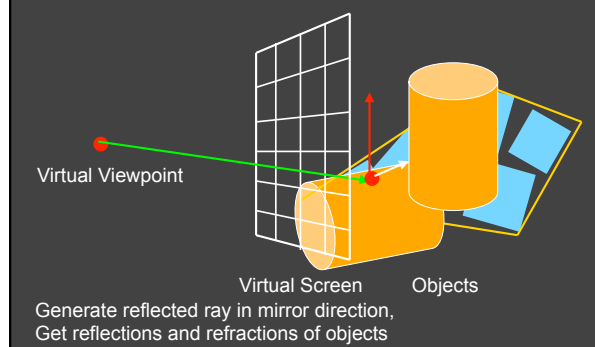


Shadows: Numerical Issues

- 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



Mirror Reflections/Refractions



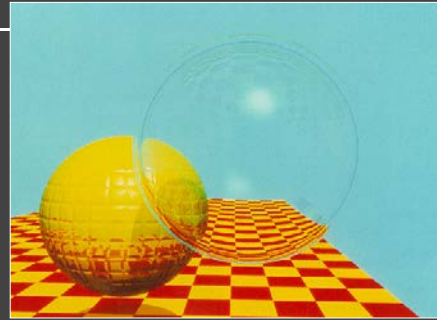
Recursive Ray Tracing

For each pixel

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

Problems with Recursion

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



Turner Whitted 1980

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; 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

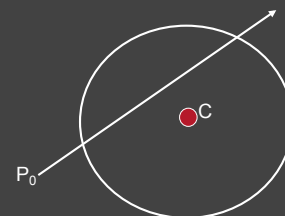
Ray/Object Intersections

- 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

$$\text{ray} \equiv \vec{P} = \vec{P}_0 + \vec{P}_t t$$

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



Ray-Sphere Intersection

$$\text{ray} \equiv \vec{P} = \vec{P}_0 + \vec{P}_1 t$$

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

Substitute

$$\text{ray} \equiv \vec{P} = \vec{P}_0 + \vec{P}_1 t$$

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

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

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

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-Sphere Intersection

- Intersection point: $\text{ray} \equiv \vec{P} = \vec{P}_0 + \vec{P}_1 t$
- Normal (for sphere, this is same as coordinates in sphere frame of reference, useful other tasks)

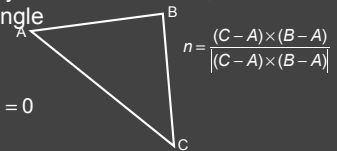
$$\text{normal} = \frac{\vec{P} - \vec{C}}{|\vec{P} - \vec{C}|}$$

Ray-Triangle Intersection

- One approach: Ray-Plane intersection, then check if inside triangle

- Plane equation:

$$\text{plane} \equiv \vec{P} \cdot \vec{n} - \vec{A} \cdot \vec{n} = 0$$



Ray-Triangle Intersection

- One approach: Ray-Plane intersection, then check if inside triangle

- Plane equation:

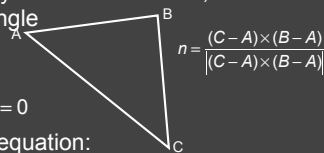
$$\text{plane} \equiv \vec{P} \cdot \vec{n} - \vec{A} \cdot \vec{n} = 0$$

- Combine with ray equation:

$$\text{ray} \equiv \vec{P} = \vec{P}_0 + \vec{P}_1 t$$

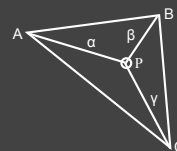
$$(\vec{P}_0 + \vec{P}_1 t) \cdot \vec{n} = \vec{A} \cdot \vec{n}$$

$$t = \frac{\vec{A} \cdot \vec{n} - \vec{P}_0 \cdot \vec{n}}{\vec{P}_1 \cdot \vec{n}}$$



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)

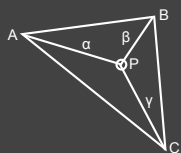


$$P = \alpha A + \beta B + \gamma C$$

$$\alpha \geq 0, \beta \geq 0, \gamma \geq 0$$

$$\alpha + \beta + \gamma = 1$$

Ray inside Triangle



$$P = \alpha A + \beta B + \gamma C$$
$$\alpha \geq 0, \beta \geq 0, \gamma \geq 0$$
$$\alpha + \beta + \gamma = 1$$

$$P - A = \beta(B - A) + \gamma(C - A)$$

$$0 \leq \beta \leq 1, 0 \leq \gamma \leq 1$$

$$\beta + \gamma \leq 1$$

Other primitives

- Much early work in ray tracing focused on ray-primitive intersection tests
- Cones, cylinders, ellipsoids
- 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

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
 - Normals n transform as $M^{-T}n$. 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

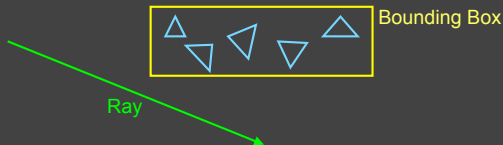
Testing each object for each ray is slow

- 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

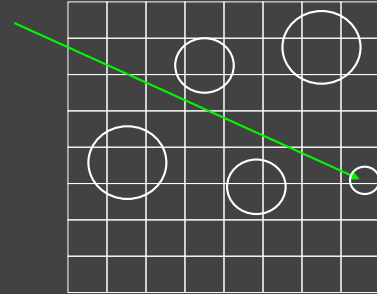
Acceleration Structures

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



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

Acceleration Structures: Grids



Acceleration and Regular Grids

- Simplest acceleration, for example 5x5x5 grid
- For each grid cell, store overlapping triangles
- 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
- Try to implement some acceleration in HW 4

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
- NVIDIA OptiX ray-tracing API like OpenGL



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>

