Computer Graphics
-Ray Tracing

Junjie Cao @ DLUT

Spring 2016

http://jjcao.github.io/ComputerGraphics/

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)
- …

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
- *Camera Ray Casting (choose ray directions)*
- Shadows / Reflections (core algorithm)
- Ray-Surface Intersection
  - Ray-object intersections
  - Ray-tracing transformed objects
- Optimizations
- Lighting calculations

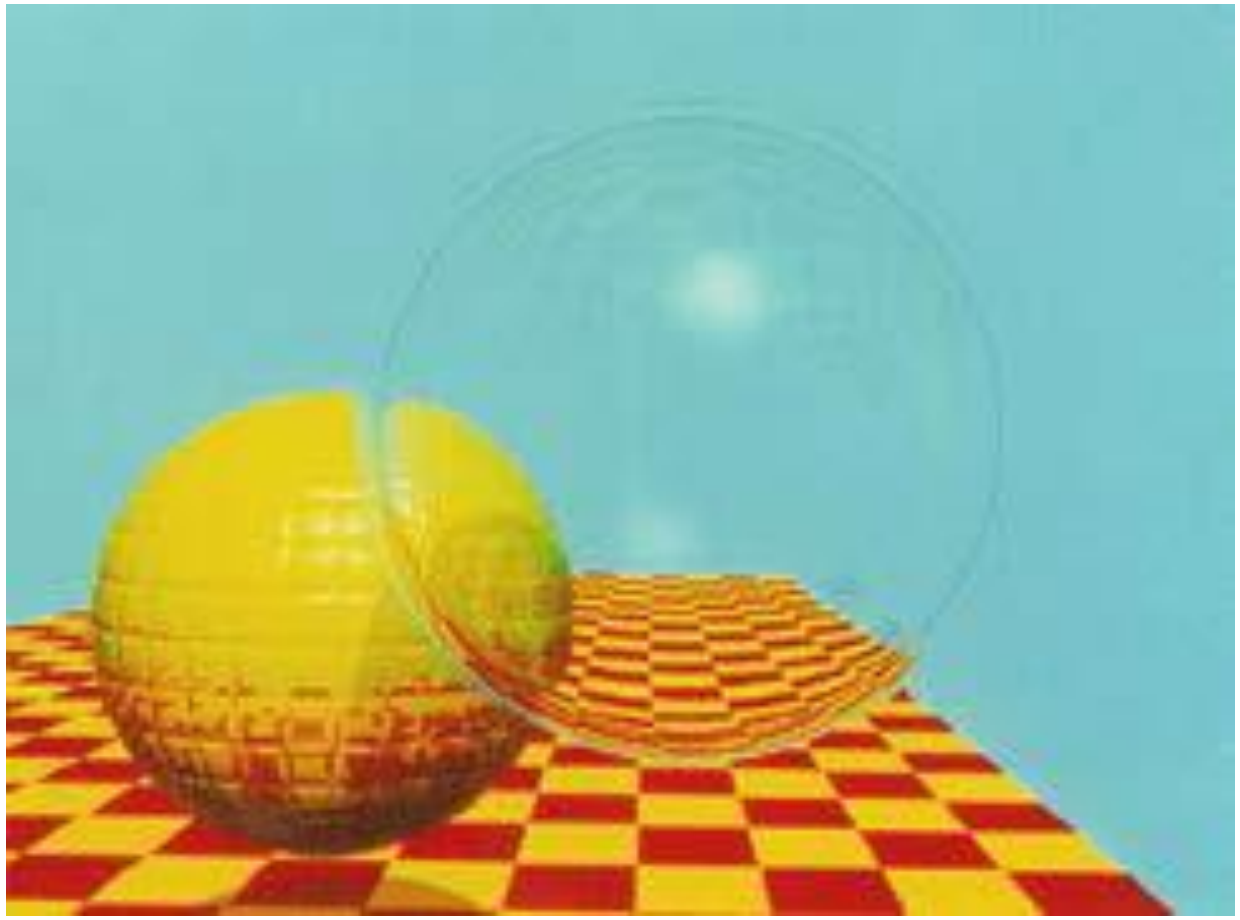
Ray Tracing: History

- 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

one of the most significant developments in the history of CG

Raytracing History

- “An improved illumination model for shaded display” by T. Whitted, CACM 1980
- 512*512, VAX 11/780
- 74 min, today real-time



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 ;
}
```

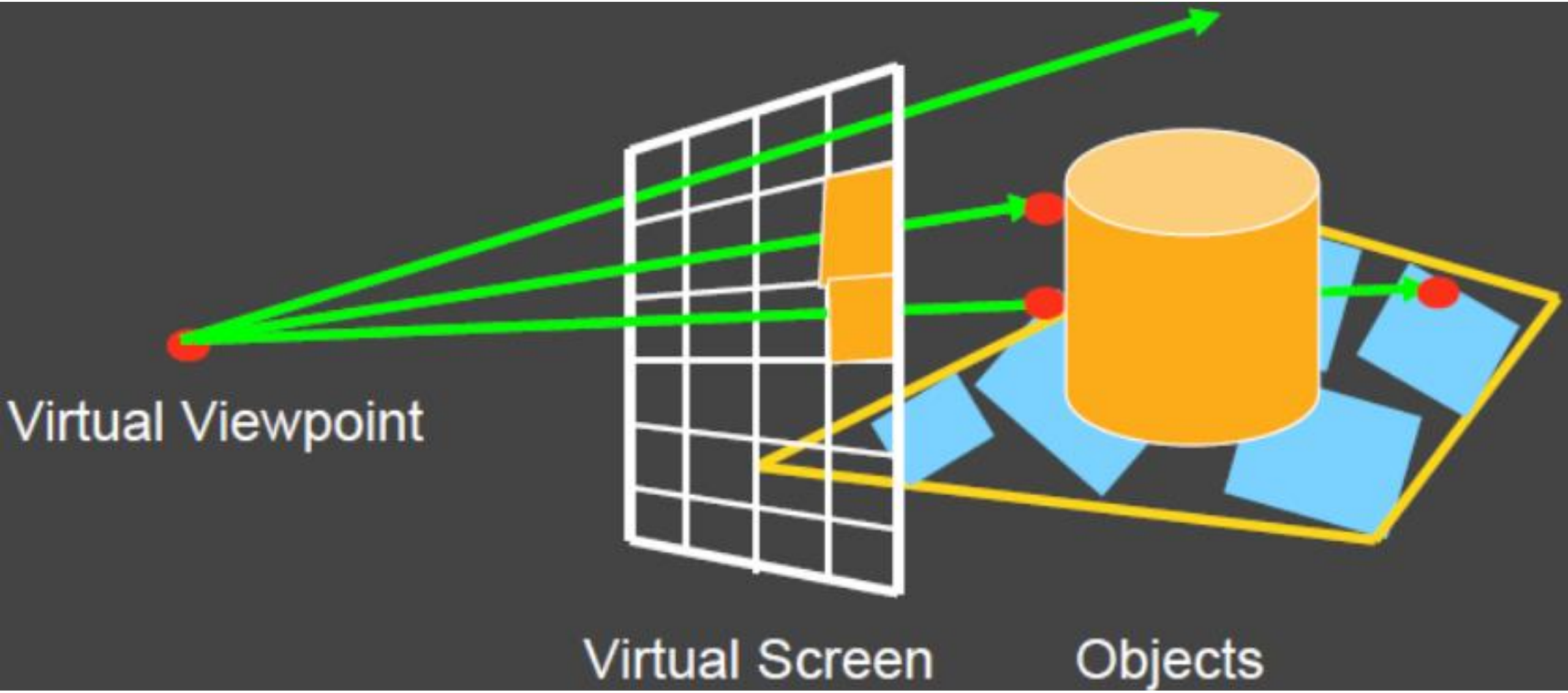
Outline

- History
- Basic Ray Casting (instead of rasterization)
  - Comparison to hardware scan conversion
- *Camera Ray Casting (choose ray directions)*
- Shadows / Reflections (core algorithm)
- Ray-Surface Intersection
  - Ray-object intersections
  - Ray-tracing transformed objects
- Optimizations
- Lighting calculations

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



Ray intersects object: shade using color
Ray misses all objects: Pixel colored black
Multiple intersections: Use closest one (as does OpenGL), lights, materials

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

Summarization

- Per-object vs per-pixel evaluation, costly

- But good for walkthroughs of extremely large models (amortize preprocessing, low complexity)

- More complex shading, lighting effects possible

Outline

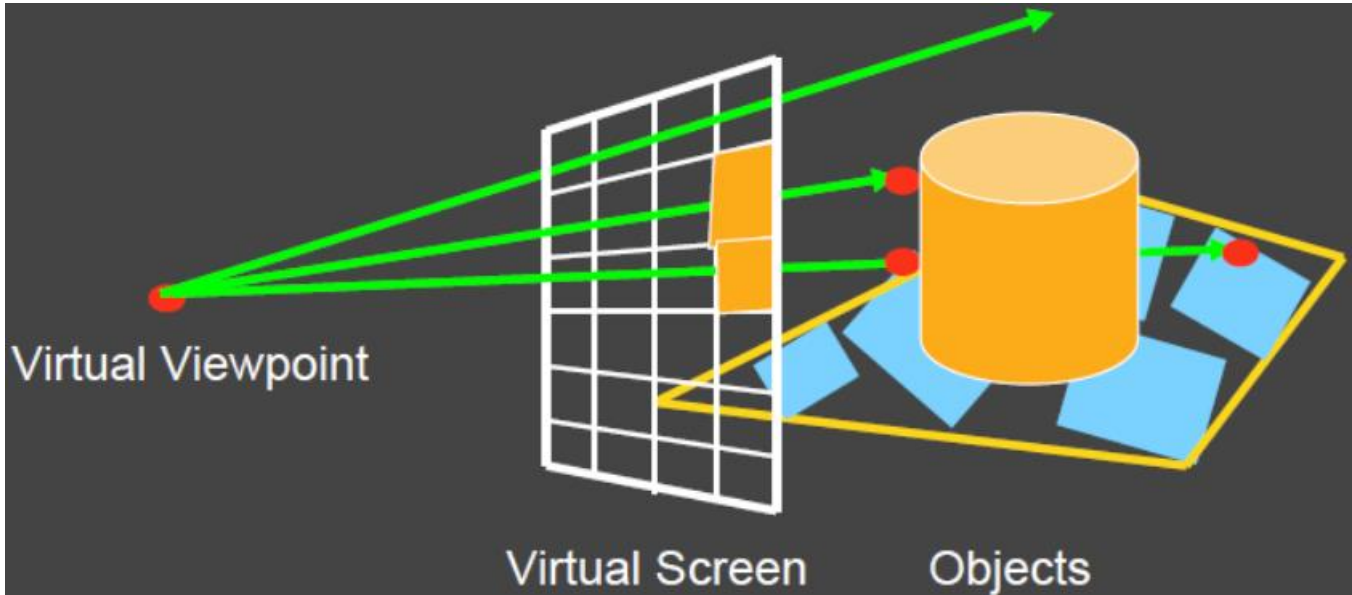
- History
- Basic Ray Casting (instead of rasterization)
  - Comparison to hardware scan conversion
- Camera Ray Casting *(choose ray directions)*
- Shadows / Reflections (core algorithm)
- Ray-Surface Intersection
  - Ray-object intersections
  - Ray-tracing transformed objects
- Optimizations
- Lighting calculations

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 ;
}
```

Finding Ray Direction

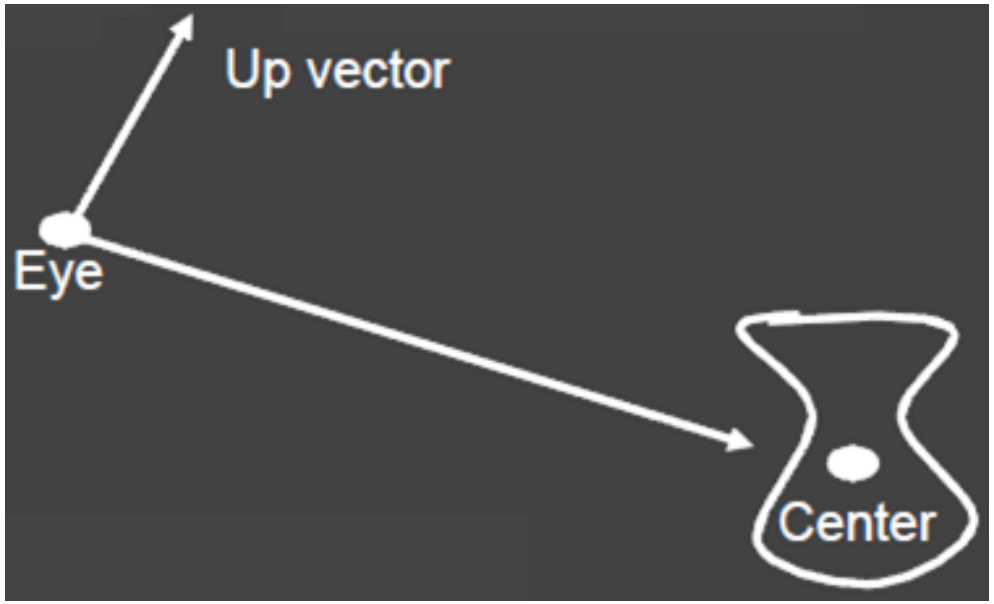
- Goal is to find ray direction for given pixel i and j
- Many ways to approach problem
  - Objects in world coord, find dirn of each ray (we do this)
  - Camera in canonical frame, transform objects (OpenGL)
- Basic idea
  - Ray has origin (camera center) and direction
  - Find direction given camera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



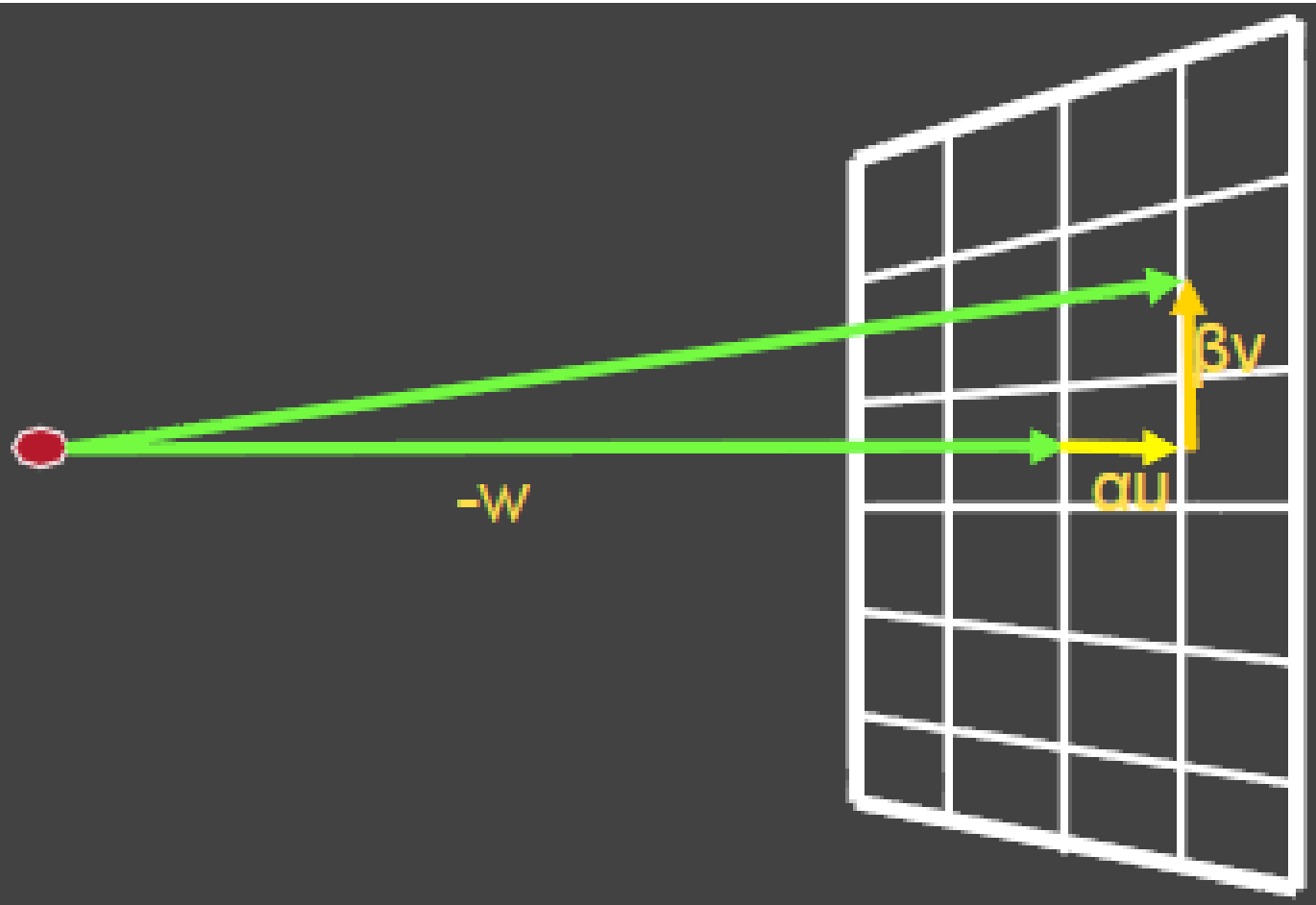
Constructing a coordinate frame?

- We want to position camera at origin, looking down –Z dirn
- Hence, vector **a** is given by **eye** – **center**
- The vector **b** is simply the **up** vector

$$w = \frac{a}{\|a\|} \qquad u = \frac{b \times w}{\|b \times w\|} \qquad v = w \times u$$



Canonical viewing geometry



$$ray = eye + \frac{\alpha u + \beta v - w}{|\alpha u + \beta v - w|}$$

$$\alpha = \tan\left(\frac{fovx}{2}\right) \times \left(\frac{j - (width\,/\,2)}{width\,/\,2}\right)$$

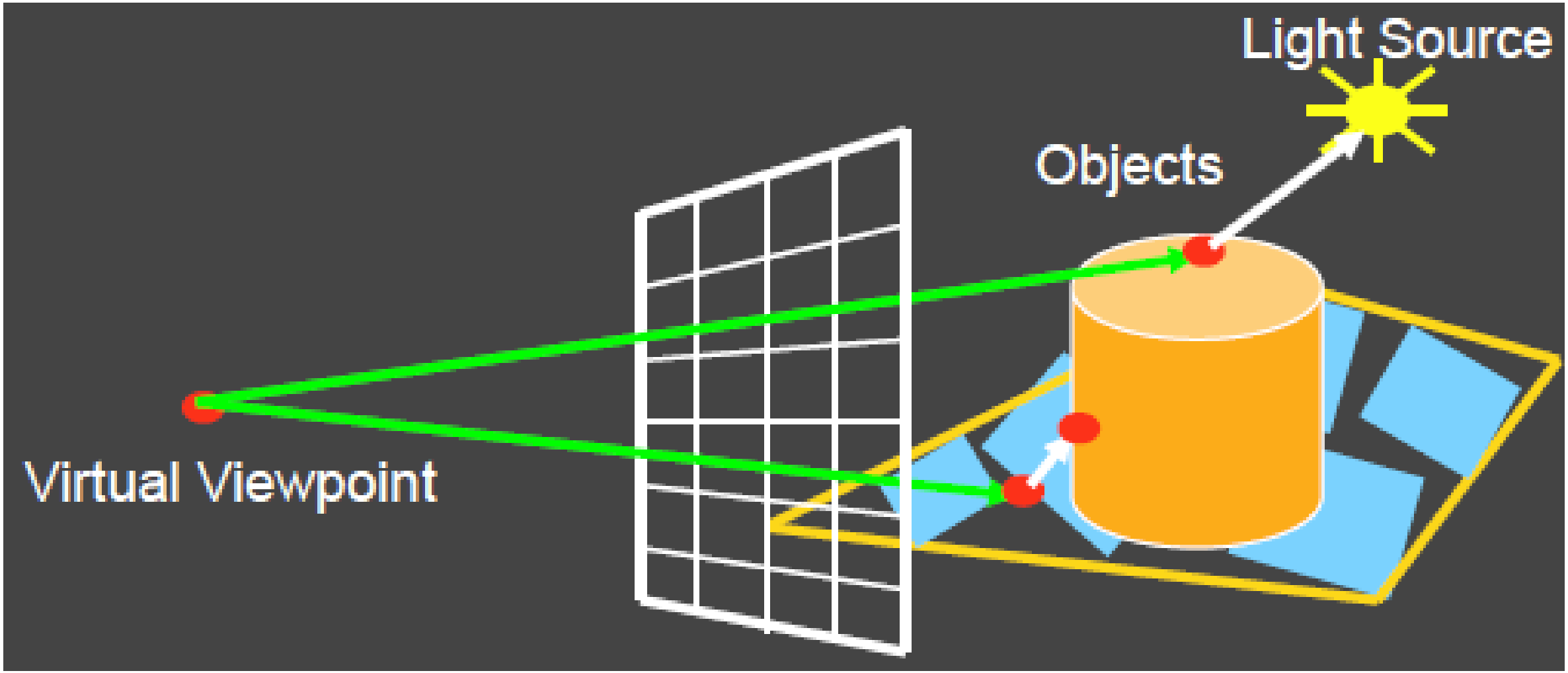
$$\beta = \tan\left(\frac{fovy}{2}\right) \times \left(\frac{(height\,/\,2) - i}{height\,/\,2}\right)$$

Outline

- History
- Basic Ray Casting (instead of rasterization)
  - Comparison to hardware scan conversion
- Camera Ray Casting *(choose ray directions)*
- Shadows / Reflections (core algorithm)
- Ray-Surface Intersection
  - Ray-object intersections
  - Ray-tracing transformed objects
- Optimizations
- Lighting calculations

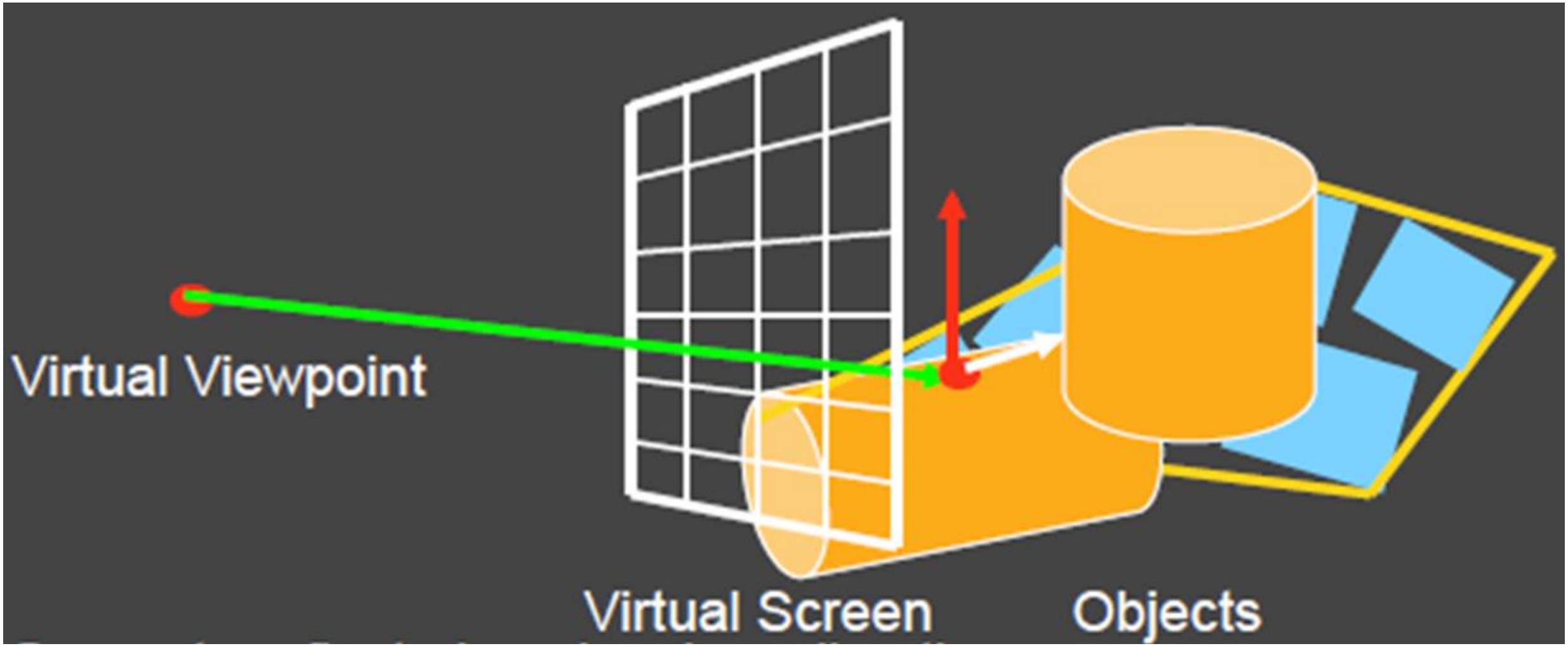
Shadows

- Shadow ray to light is unblocked: object visible
- Shadow ray to light is blocked: object in shadow



Mirror Reflections/Refractions

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



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)

- Discussed in this lecture
- Not discussed but possible with distribution ray tracing
- Hard (but not impossible) with ray tracing; radiosity methods

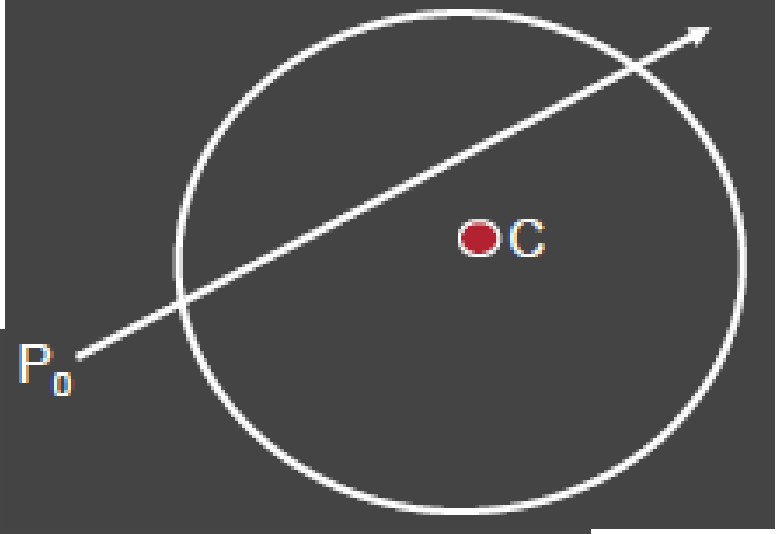
Outline

- History
- Basic Ray Casting (instead of rasterization)
  - Comparison to hardware scan conversion
- Camera Ray Casting *(choose ray directions)*
- Shadows / Reflections (core algorithm)
- Ray-Surface Intersection
  - Ray-object intersections
  - Ray-tracing transformed objects
- Optimizations
- Lighting calculations

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



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

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

Substitute

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

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

Simplify

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

Ray-Sphere Intersection

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

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-Triangle Intersection

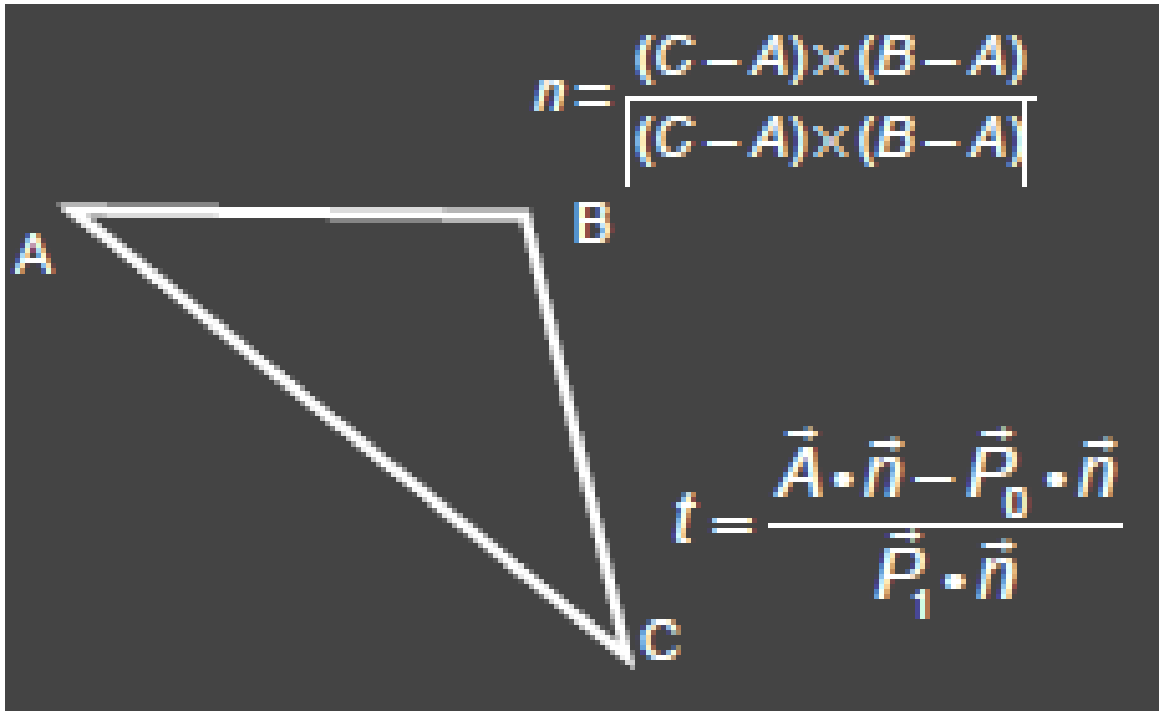
- One approach: Ray-Plane intersection, then check if inside triangle
- Plane equation:

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

- Combine with ray equation

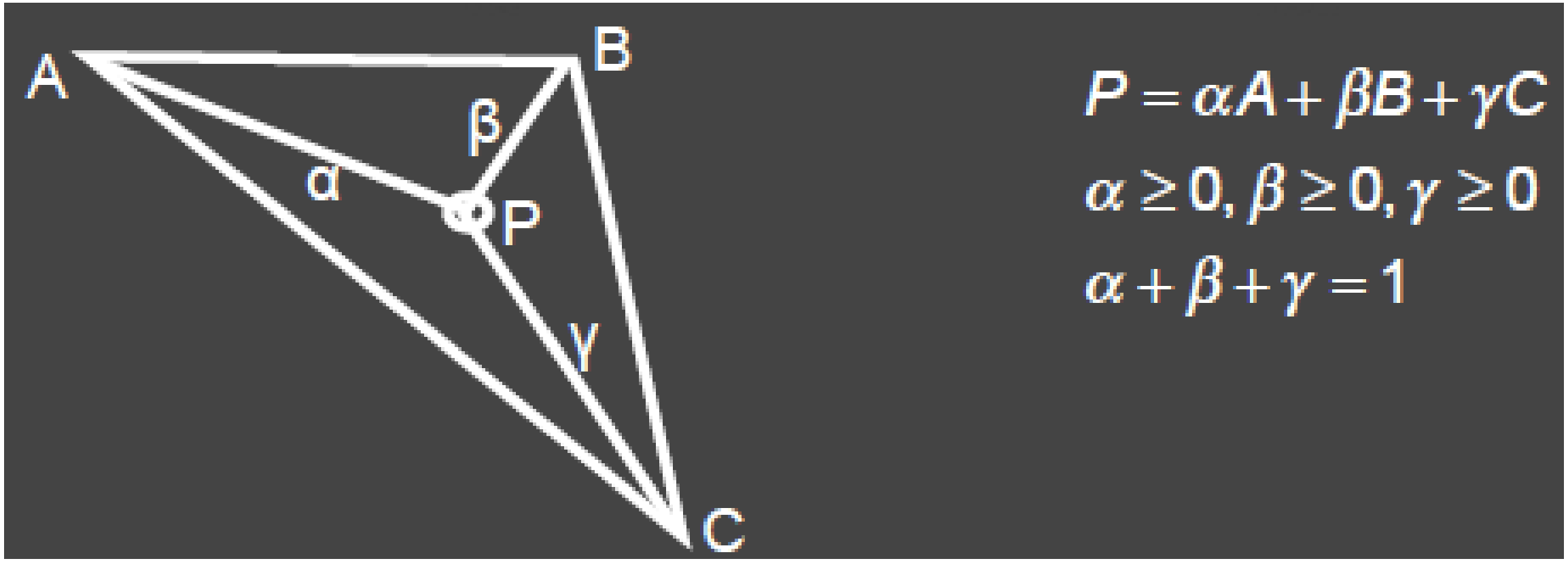
$$ray \equiv \vec{P} = \vec{P}_0 + \vec{P}_1 t$$

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



Ray inside Triangle

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



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-Tracing Transformed Objects

- Consider a general 4x4 transform M (matrix stacks)
- 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
  - Normals n transform as M-tn. Do all this before lighting

Outline

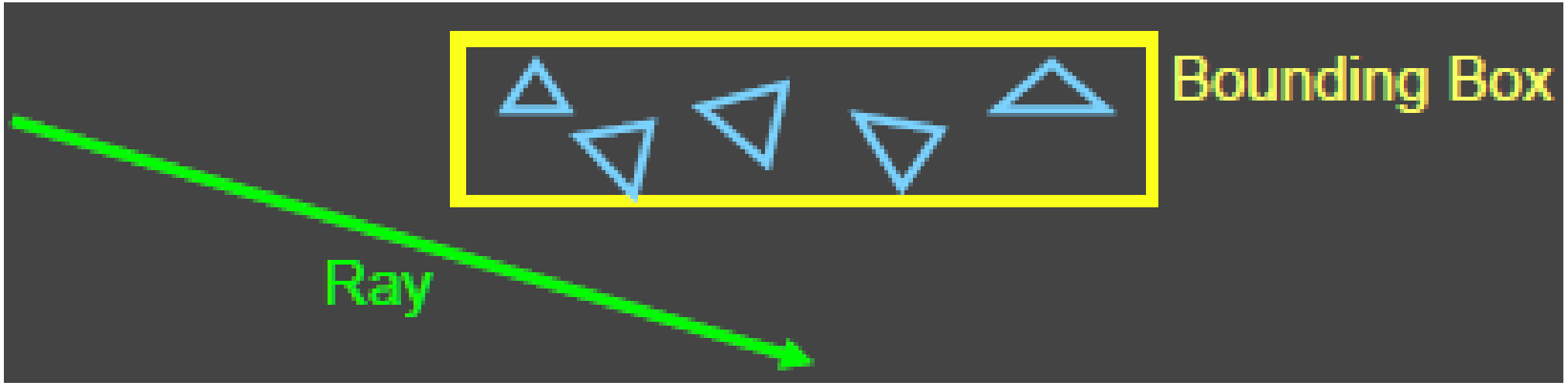
- History
- Basic Ray Casting (instead of rasterization)
  - Comparison to hardware scan conversion
- Camera Ray Casting *(choose ray directions)*
- Shadows / Reflections (core algorithm)
- Ray-Surface Intersection
  - Ray-object intersections
  - Ray-tracing transformed objects
- Optimizations
- Lighting 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 (more on this later)
    - Optimized Ray-Object Intersections
    - ***Fewer Intersections***

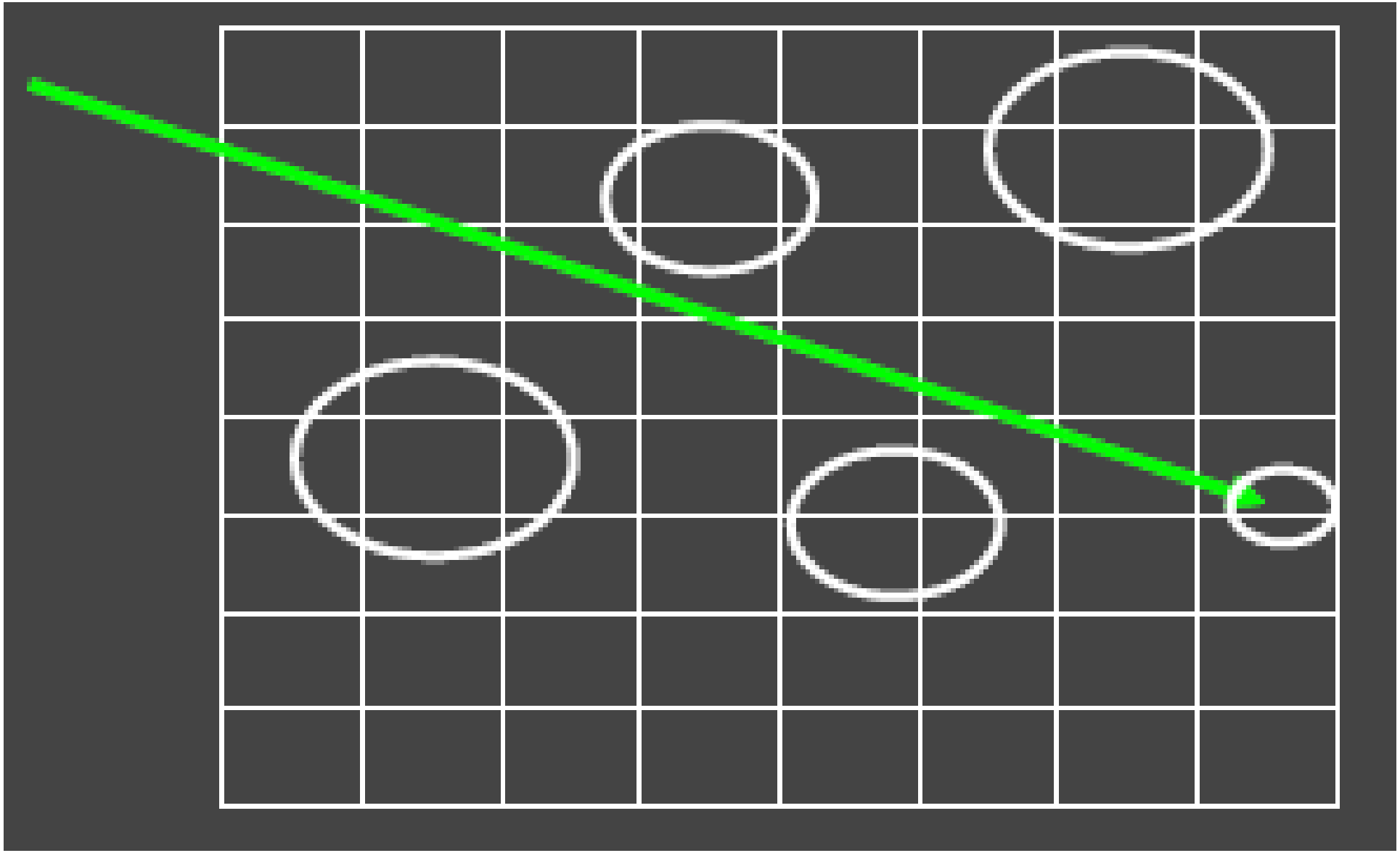
Acceleration Structures

- 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

Outline

- History
- Basic Ray Casting (instead of rasterization)
  - Comparison to hardware scan conversion
- Camera Ray Casting *(choose ray directions)*
- Shadows / Reflections (core algorithm)
- Ray-Surface Intersection
  - Ray-object intersections
  - Ray-tracing transformed objects
- Optimizations
- Lighting calculations

Lighting Model

- Similar to OpenGL
- Lighting model parameters (global)
  - Ambient r g b
  - Attenuation const linear quadratic
- Per light model parameters
  - Directional light (direction, RGB parameters)
  - Point light (location, RGB parameters)
  - Some differences from HW 2 syntax

$$L = \frac{L_0}{const + lin * d + quad * d^2}$$

Material Model

- Diffuse reflectance (r g b)
- Specular reflectance (r g b)
- Shininess s
- Emission (r g b)
- All as in OpenGL

Shading Model

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

- 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)

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

$$I = K_a + K_e + \sum_{i=1}^{n} V_i L_i (K_d \max(l_i \bullet n, 0) + K_s (\max(h_i \bullet n, 0))^s) + K_s I_R + K_T I_T$$

- Highlighted terms are recursive specularities [mirror reflections] and transmission (latter is extra)
- 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)

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