# CS174A Lecture 14

# Announcements & Reminders

- ***Midterm and project 2 grades should be posted today***
- ***Dec 1: projects due***
- ***Dec 3 and 5: project presentations, in class***
- ***Dec 12: final exam, 11:30 AM – 2:30 PM, Place TBD***

# TA Session This Friday

- *Midterms*
- *Project #4*
- *Collision Detection*

# Last Lecture Recap

- ***Non-Photorealistic Rendering***
- ***Global Illumination: ray tracing & radiosity***
- ***Mappings: Texture, Bump, Displacement, Environment***
- ***Shadows***
    - 2-pass z-buffer algorithm
    - Shadow volumes

# Next Up

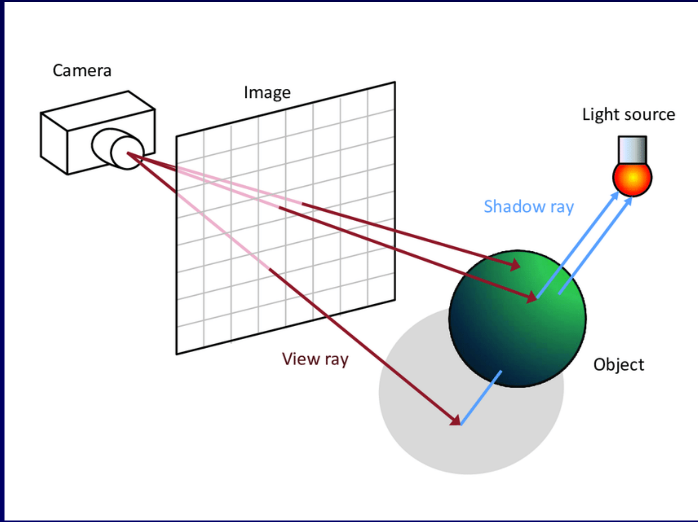
- ***Hidden Surface Removal***
  - Ray casting
- ***Ray Tracing***

# HSR: Ray Casting Algorithm

- ***Ray Casting is a WS or ES visible surface algorithm***
- ***Forward vs. backward ray casting***
- ***Window is divided into a rectangular array of pixels***
- ***Algorithm***

  For each pixel in viewport

  1. Generate ray emanating from Eye (O) through pixel
  2. Find intersection between ray and objects
  3. Pick intersection point closest to Eye (O)
  4. Plot pixel with color of closest object



# Ray Casting Algorithm

***Step 1: Generate ray emanating from Eye (O) through pixel***

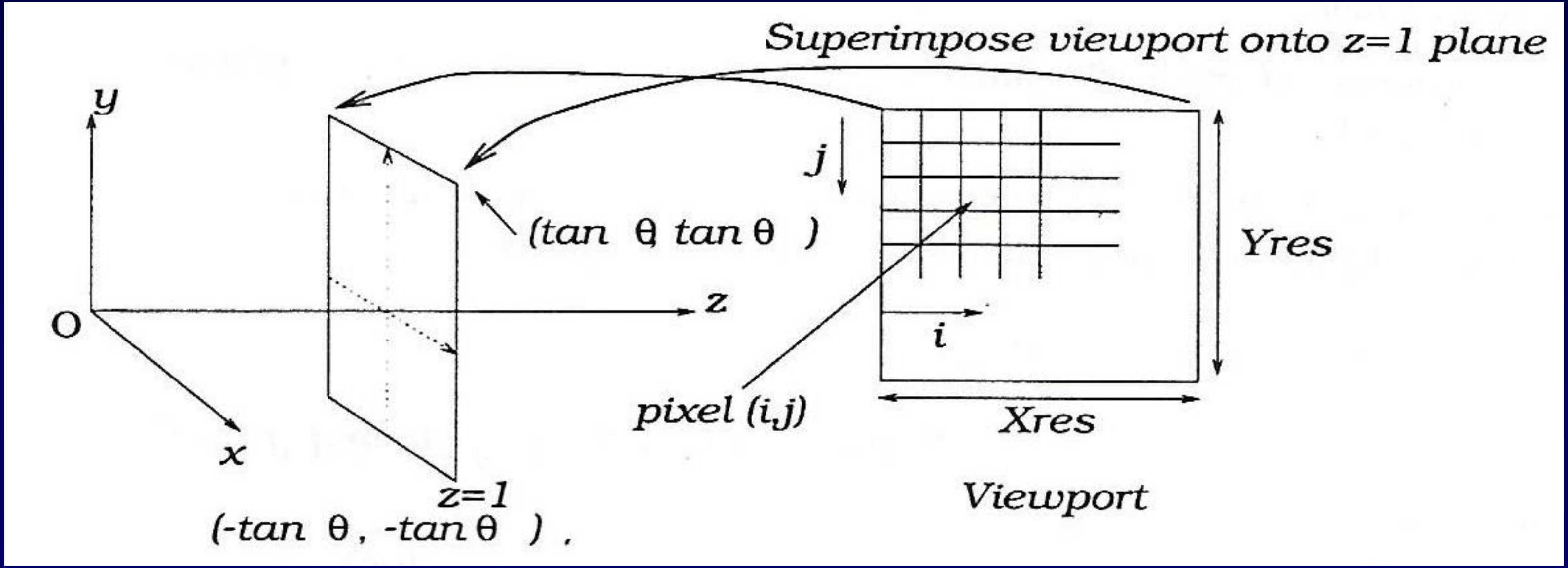
- ***Superimpose a grid (representing viewport) onto the area defined by ($-\tan\theta$, $-\tan\theta/A_r$), ($\tan\theta$, $\tan\theta/A_r$) on z = 1 plane***
- ***Width of a pixel on the z = 1 plane will be $\frac{2\tan\theta}{X_{res}}$***
- ***Point corresponding to center of pixel (i,j) is P = ($x_p$, $y_p$, 1)***

$$x_p = -\tan\theta + (i + 0.5)\frac{2\tan\theta}{X_{res}} \qquad y_p = \tan\theta/A_r - (j + 0.5)\frac{2\tan\theta}{Y_{res}}$$

- ***Parametric ray emanating from eye (origin) through pixel P:***
  - $x = x_e + t(x_p - x_e)$, $y = y_e + t(y_p - y_e)$, $z = z_e + t(z_p - z_e)$
  - For ES ($x_e = y_e = z_e = 0$): $x = tx_p$, $y = ty_p$, $z = t$

# Ray Casting Algorithm

**Step 1: Generate ray emanating from Eye (O) through pixel**



# Ray Casting Algorithm

***Step 2: Find intersection between ray and objects***

a. Intersection test: if no intersection, skip part b
b. Intersection calculation: for polygons and for spheres

# Ray Casting Polygons

- ***For polygons***
  a. Transform poly to ES
  b. Equation of the plane containing poly: $Ax + By + Cz + D = 0$
  c. Parametric equation of ray (ES): $x = tx_p$, $y = ty_p$, $z = t$
  d. Find intersection of ray with plane: plug in eqn of ray in eqn of plane
     i. *If denom = 0, ray is parallel to plane $\Rightarrow$ no intersection*
  e. Check if intersection point P is contained inside the poly
     i. *Project poly and point onto one of the primary planes (use max of N components)*
     ii. *Use one of the "point inside poly" tests*
  f. Find normal N at intersection point using trilinear interpolation
  g. Illuminate using P, N, E of closest +ve t

# Ray Casting Spheres

- ***For spheres***
  a. Transform center of sphere to ES (radius remains same)
  b. Equation of sphere: $(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 - R^2 = 0$
  c. Parametric equation of ray (ES): $x = tx_p$, $y = ty_p$, $z = t$
  d. Find intersection of ray with sphere: plug in eqn of ray in eqn of sphere
     $At^2 + Bt + C = 0$
     i. *No real solution ⇒ ray does not intersect sphere*
     ii. *1 real solution ⇒ ray grazes sphere*
     iii. *2 real solutions ⇒ entering & exiting points, pick lower +ve t, and calculate P*
  e. Find normal at intersection point: $N = P - P_c$
  f. Illuminate using P, N, E of closest +ve t

# Ray Casting in World Space

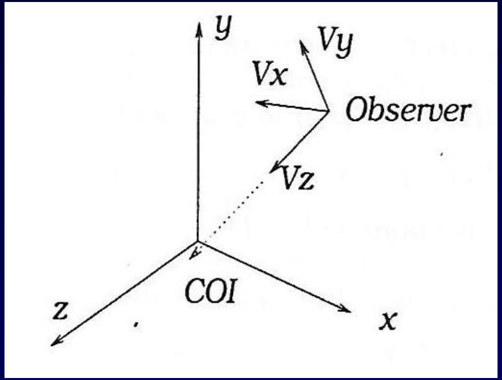
- ***Vectors***
  - $V_z$ = normalize(COI – OBS)
  - $V_x = V_z \times (0,1,0)$
  - $V_y = V_x \times V_z$
- ***Ray***
  - Parametric equation: OBS + t(P – OBS), where P is given by

$$x_p = -\tan\theta + (i + 0.5)\,\frac{2\tan\theta}{X_{res}}\,V_x$$

$$y_p = \tan\theta/A_r - (j + 0.5)\,\frac{2\tan\theta}{Y_{res}}\,V_y$$

$$z_p = V_z$$

  - Rest of calculations are similar to ray casting in ES



# Ray Tracing

- *Also referred to as Recursive Ray Tracing*
- *Handles reflections, refractions, shadows*
- *Primary ray: ray from eye into the world*
- *Secondary rays: all other reflected, refracted, or shadow rays*
- *Illumination for ray tree:*
  $I = ambient + diffuse + specular + k_s I_r + k_t I_t$
- *Attenuate $I_r$ and $I_t$ by distance the ray travels*

# Ray Tracing

- ***Reflected Ray Direction***

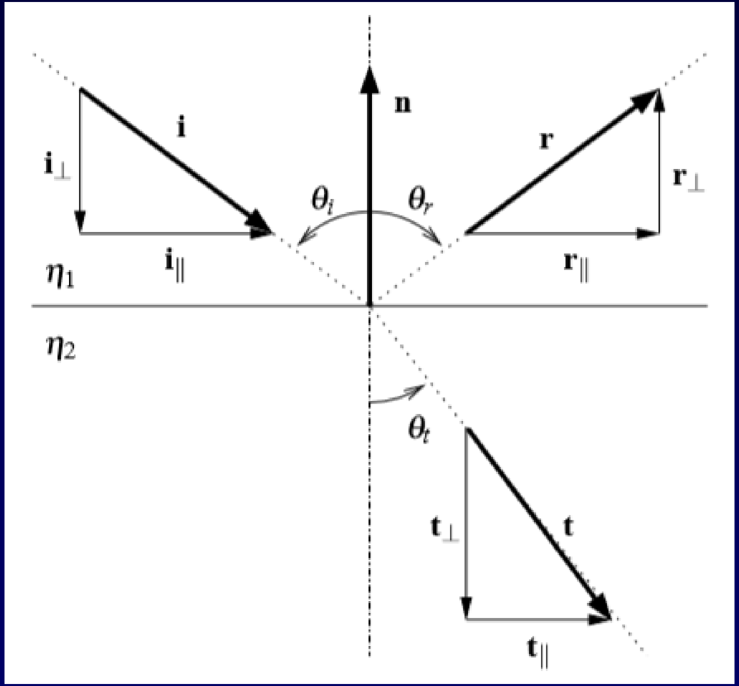
  $r = i - 2(i \cdot n)n$

- ***Refracted Ray Direction***

  Snell's Law: $\eta_1 \sin\theta_i = \eta_2 \sin\theta_t$

  $t = \frac{\eta_1}{\eta_2} i + \left(\frac{\eta_1}{\eta_2}\cos\theta_i - \sqrt{1 - \sin^2\theta_t}\right)n$

  $\sin^2\theta_t = \left(\frac{\eta_1}{\eta_2}\right)^2 \sin^2\theta_i = \left(\frac{\eta_1}{\eta_2}\right)^2 (1 - \cos^2\theta_i)$



# Ray Tracing: Illumination

- ***Ray Tree***
  - Formed of primary and secondary rays
  - Evaluated bottom up
- ***Tree terminates if***
  - No intersection for a ray
  - Tree depth has reached a specified level
  - Intensity of $I_r$ and $I_t$ becomes very low

# Ray Tracing: Speed

- ***Efficiency Considerations***
  - Total # of shadow rays spawned = $m(2^n - 1)$
    m = # light sources; n = depth of ray-tree
  - Total # of rays = $(m + 1)(2^n - 1)$
  - Back faces cannot be culled
  - Clipping cannot be done for view volume or behind eye
  - 75%-95% of time is spent in intersection calculations
  - Use bounding box testing or hierarchies (octrees)