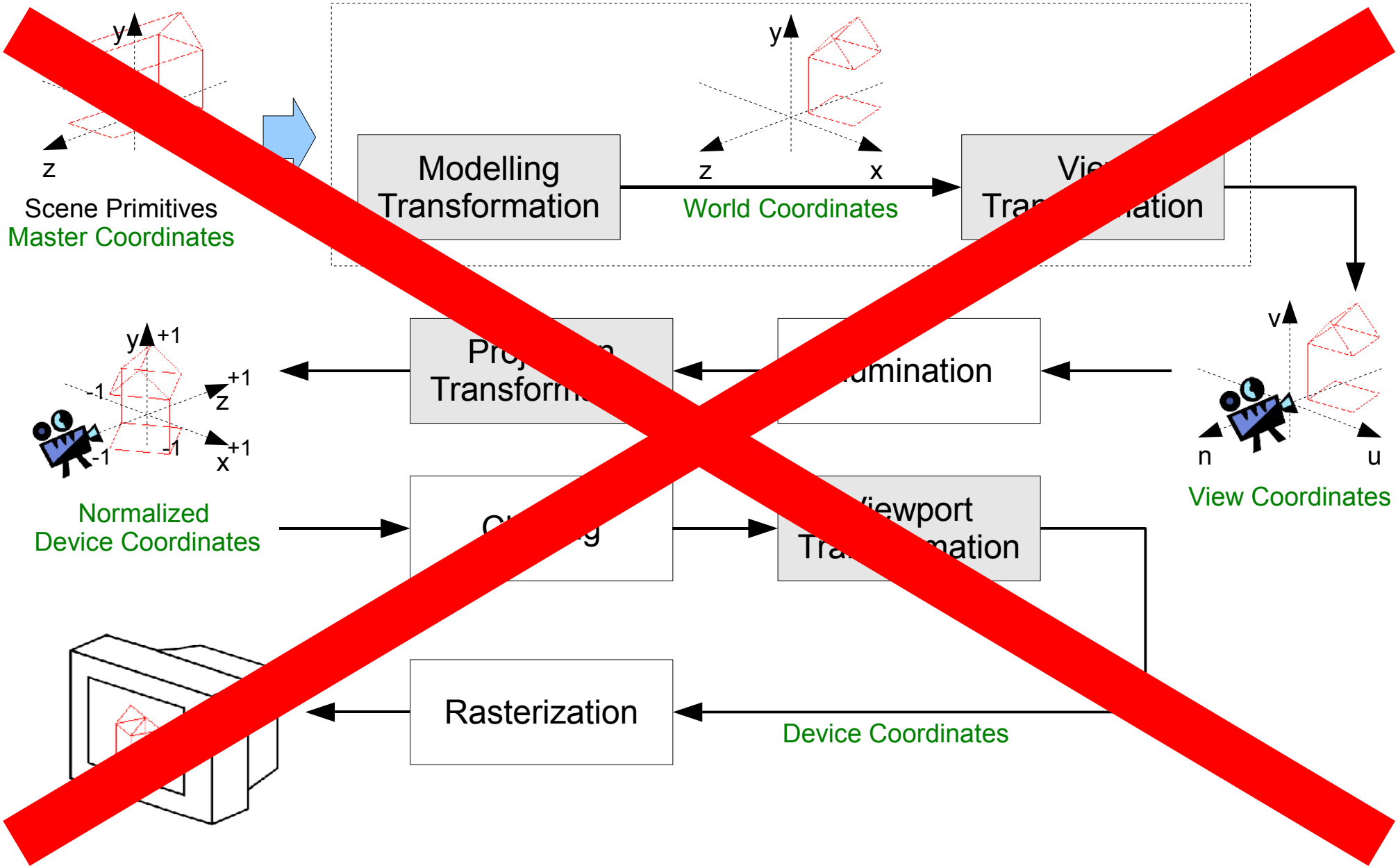
CompSci 372 – Tutorial

Part 9

Ray Tracing

OpenGL Render Pipeline

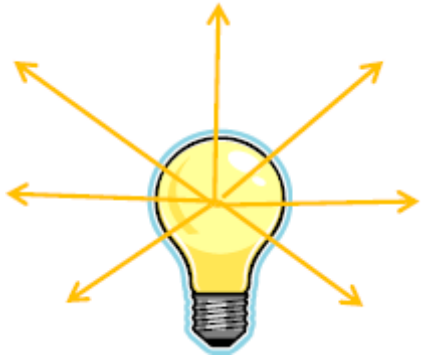


Ray Tracing

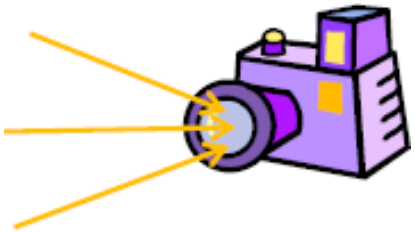
- Disadvantage Polygon Rendering:
  - No reflection, refraction
  - No proper shadows
- Ray Tracing
  - Calculate the path of light rays
  - Trace a light ray through **several** reflections, refractions, …
  - Proper shadows
  - But: Much slower

Ray Tracing

- Physically correct:
  - From Light to Camera
  - Computationally very expensive (but possible)
  - Only small proportion of rays hit the camera



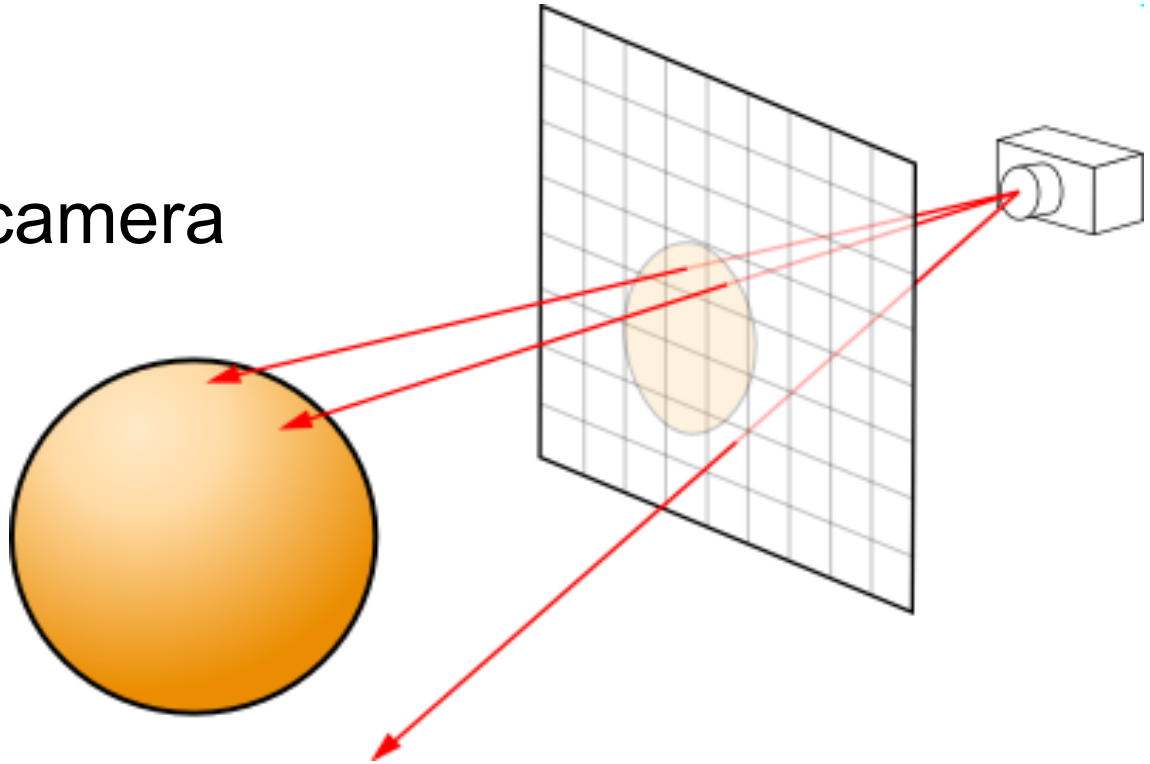
- More efficient:
  - From camera to Scene (to objects, to lights, etc.)
  - Not as accurate (e.g. focal points)



Ray Tracing

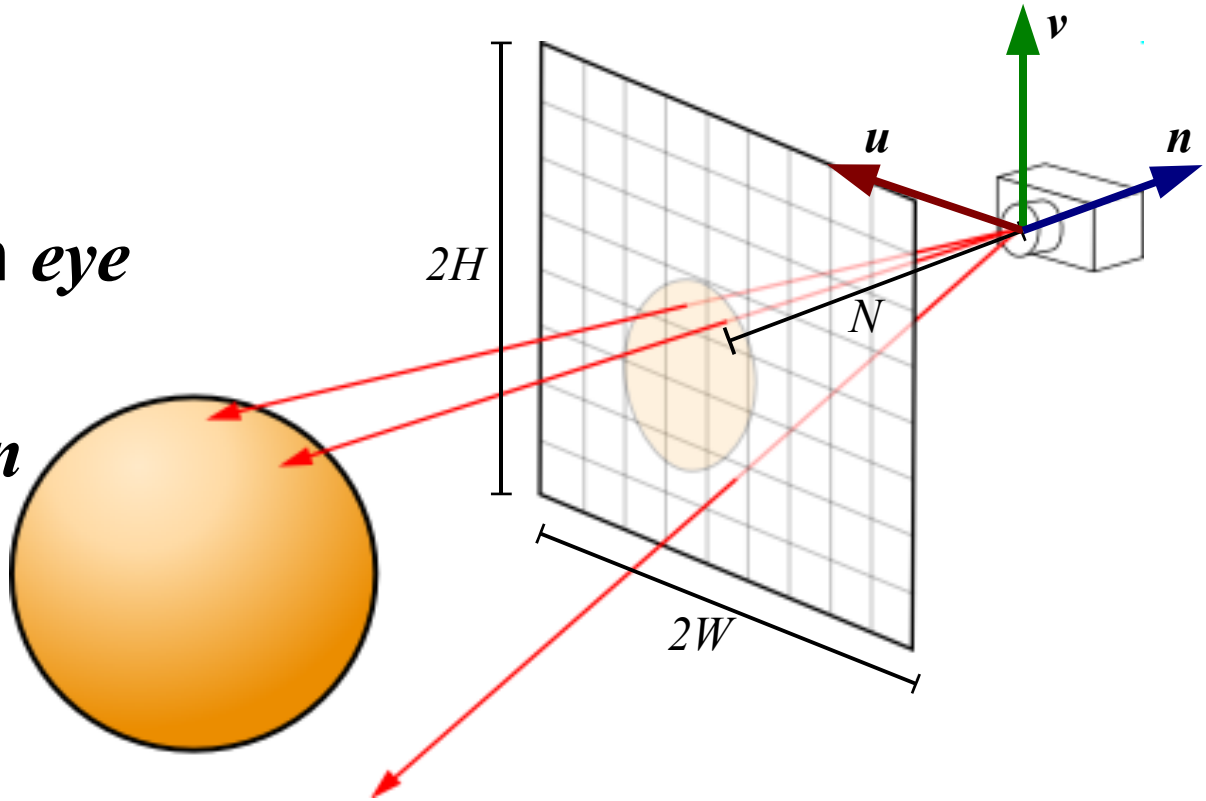
Principle

- For each pixel:
  - Shoot ray from camera through pixel
  - Which object does it intersect first
  - Calculate pixel colour from that information



Principle

- Given:
  - Camera position ***eye***
  - Camera orientation ***u, v, n***
  - View plane
    - Size *2W, 2H*
    - Distance *N*
    - Resolution *nCols, nRows*
- Wanted:
  - Ray equation $\boldsymbol{p}(t) = \boldsymbol{p}_0 + \boldsymbol{d}t$

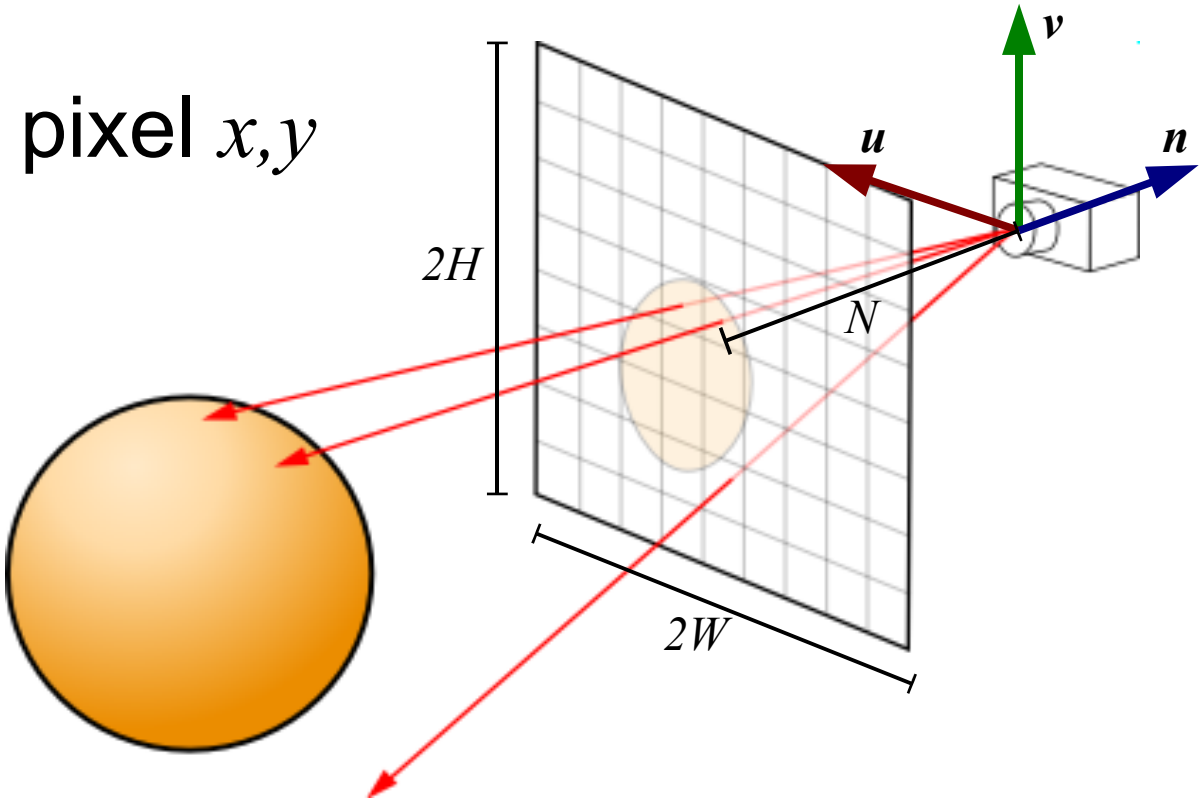


Principle

- Ray equation for pixel x,y
  - Starting point:

    $\boldsymbol{p}_0 = \boldsymbol{eye}$
  - To centre of view plane:

    $\boldsymbol{d}_c = -N\,\boldsymbol{n}$
  - To pixel x,y:



$$\boldsymbol{d} = -N\,\boldsymbol{n} + W\left(\frac{2x}{nCols} - 1\right)\boldsymbol{u} + H\left(\frac{2y}{nRows} - 1\right)\boldsymbol{v}$$

Principle

Define scene (objects, lights, camera)

```
for (int y = 0; y < nRows; y++) {
  for (int x = 0; x < nCols; x++) {
    Construct ray going through (x, y)
    Find closest (=smallest t) intersection of ray with an object
    Calculate colour of intersection point P

    Paint the pixel at (x, y)
  }
}
```

Principle

Define scene (objects, lights, camera)

```
for (int y = 0; y < nRows; y++) {
  for (int x = 0; x < nCols; x++) {
    Vector d = Vector( ? , ? , ? );
    Hit hit = intersect(eye, d);
    Color color = shade(hit);
    glColor3f(color.r, color.g, color.b);
    glVertex2f((GLfloat) x, (GLfloat) y);
  }
}
```

Ray/Object Intersection

- Principle
  - Define object as implicit function $f(\mathbf{p})=0$
  - Insert ray equation $\mathbf{p}(t)=\mathbf{eye}+\mathbf{d}t$ and solve for t
    - Smallest t is closest to eye
    - $t<0$: behind eye point, not visible
      (Think of the definition of a ray!)

Ray/Plane Intersection

- Given
  - Normal $\boldsymbol{n}$
  - Distance a

$$\boldsymbol{p}\cdot\boldsymbol{n}-a=0$$

$$(\boldsymbol{eye}+\boldsymbol{d}\,t)\cdot\boldsymbol{n}-a=0$$

$$\boldsymbol{eye}\cdot\boldsymbol{n}+\boldsymbol{d}\,t\cdot\boldsymbol{n}=a$$

$$t=\frac{a-\boldsymbol{eye}\cdot\boldsymbol{n}}{\boldsymbol{d}\cdot\boldsymbol{n}}$$

Ray/Sphere Intersection

- Given
  - Radius r

$$|\boldsymbol{p}| - r = 0 \Rightarrow \boldsymbol{p}\cdot\boldsymbol{p} - r^2 = 0$$

$$(\boldsymbol{eye} + \boldsymbol{d}\,t)\cdot(\boldsymbol{eye} + \boldsymbol{d}\,t) - r^2 = 0$$

$$\boldsymbol{d}\cdot\boldsymbol{d}\,t^2 + 2\,\boldsymbol{eye}\cdot\boldsymbol{d}\,t + \boldsymbol{eye}\cdot\boldsymbol{eye} - r^2 = 0$$

$$t_{1,2} = \frac{-B \pm \sqrt{B^2 - 4\,A\,C}}{2\,A}$$

$$A = \boldsymbol{d}\cdot\boldsymbol{d}\,, \quad B = 2\,\boldsymbol{eye}\cdot\boldsymbol{d}\,, \quad C = \boldsymbol{eye}\cdot\boldsymbol{eye} - r^2$$

Ray/Sphere Intersection

- $B^2\text{-}4AC$:

  <0: Ray misses sphere

  =0: Ray grazes sphere (=miss)

  >0: Solve for t_1 and t_2 (entry/exit point)

$$t_{1,2} = \frac{-B \pm \sqrt{B^2 - 4\,A\,C}}{2\,A}$$

Ray/Triangle Intersection

- Given
  - Points A,B,C
  - Normal n
    $(B-A)\times(C-A)$

$$(p-A)\cdot n=0$$

$$(eye+d\,t-A)\cdot n=0$$

$$eye\cdot n+d\,t\cdot n-A\cdot n=0$$

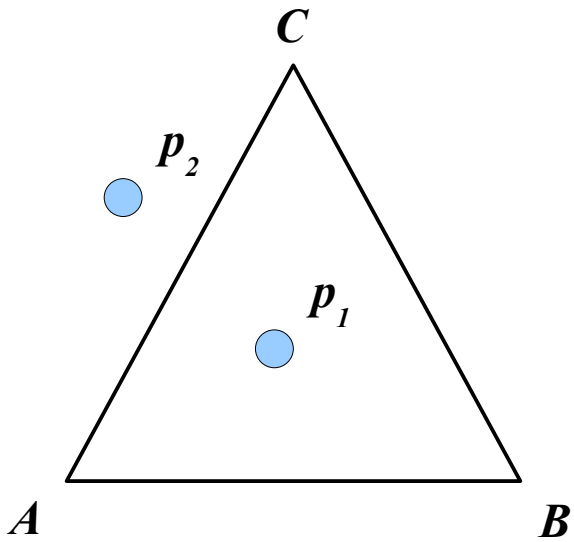
$$eye\cdot n+d\cdot n\,t=A\cdot n$$

$$t=\frac{(A-eye)\cdot n}{d\cdot n}$$

Ray/Triangle Intersection

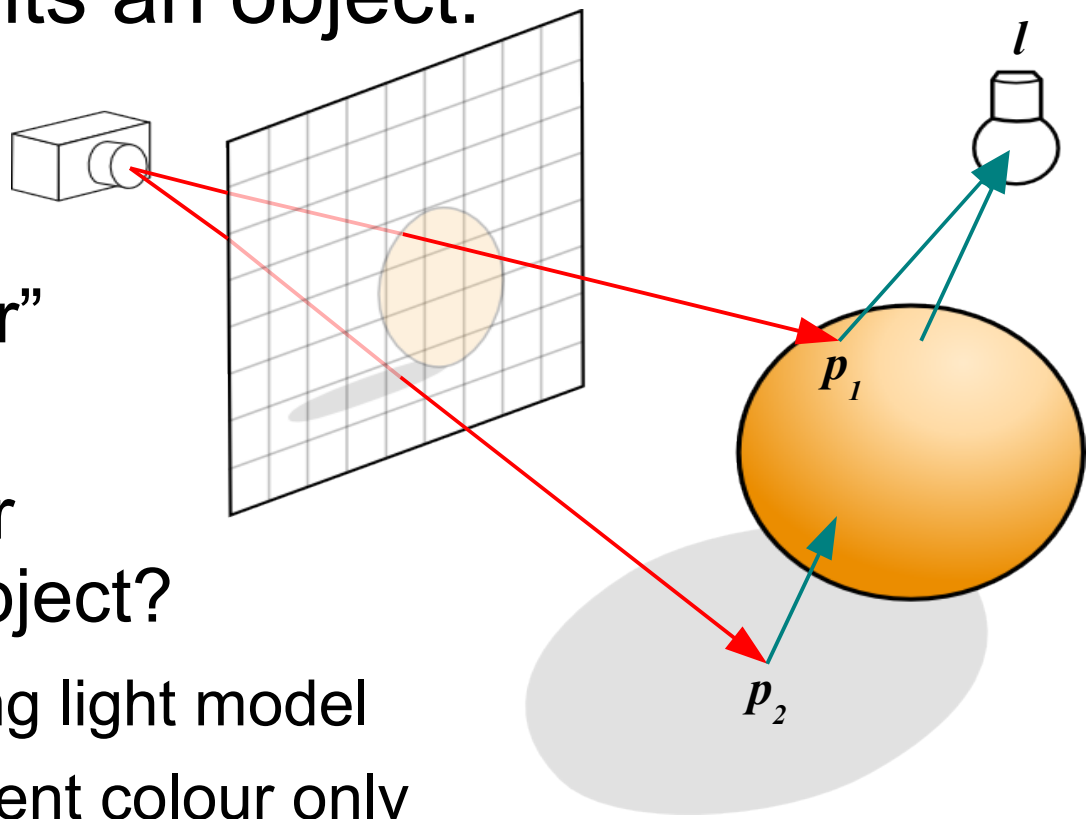
- Check if $\boldsymbol{p}(t)$ is inside the triangle
  - a) $((\boldsymbol{B}-\boldsymbol{A}) \times (\boldsymbol{p}-\boldsymbol{A}))\cdot\boldsymbol{n}$
  - b) $((\boldsymbol{C}-\boldsymbol{B}) \times (\boldsymbol{p}-\boldsymbol{B}))\cdot\boldsymbol{n}$
  - c) $((\boldsymbol{A}-\boldsymbol{C}) \times (\boldsymbol{p}-\boldsymbol{C}))\cdot\boldsymbol{n}$
- Must all be the same sign

$$t = \frac{(A - \boldsymbol{eye})\cdot\boldsymbol{n}}{\boldsymbol{d}\cdot\boldsymbol{n}}$$



Light and Shadow

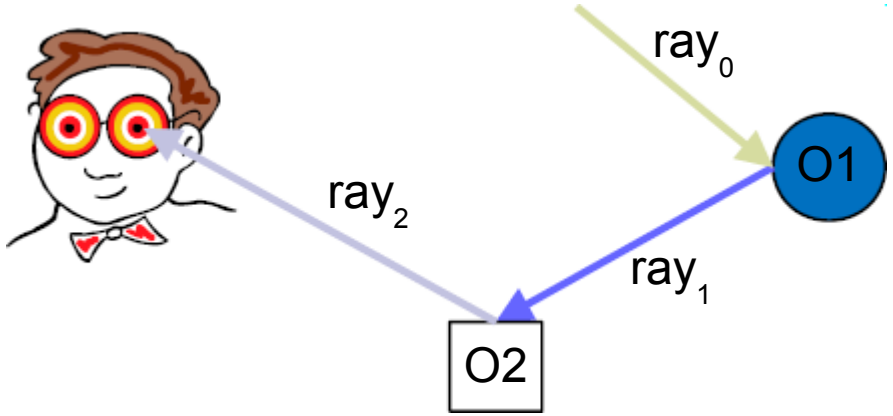
- For every ray that hits an object:
  - Determine hit point $\boldsymbol{p}$
  - Cast "shadow feeler" from $\boldsymbol{p}$ to $\boldsymbol{l}$
  - Does shadow feeler intersect another object?
    - No: Light: Full Phong light model
    - Yes: Shadow: Ambient colour only



$$\boldsymbol{sf}(t) = \boldsymbol{p} + (\boldsymbol{l} - \boldsymbol{p})t, \quad t \in [0 \ldots 1]$$

Reflections

- Reflected ray carries the colour of the previous reflection

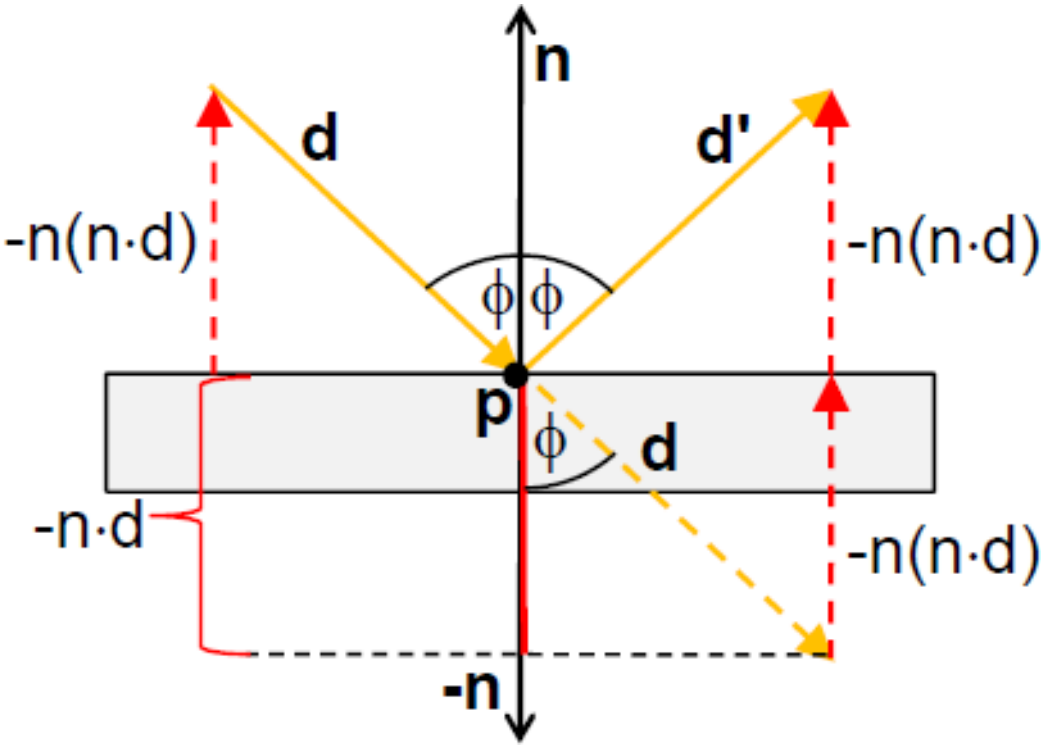


$$R_n = R_{amb,n} + R_{diff,n} + R_{spec,n} + reflectivity \cdot R_{n-1}$$

Reflections

- Reflected ray

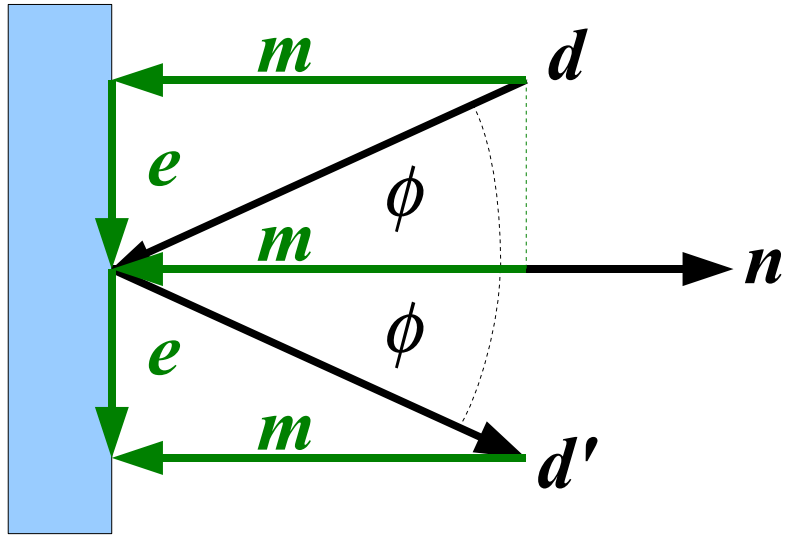
$$d' = d - 2n(n \cdot d)$$



Reflections

$$\boldsymbol{m} = \frac{\boldsymbol{d}\cdot\boldsymbol{n}}{\boldsymbol{n}\cdot\boldsymbol{n}}\boldsymbol{n}$$

$$= (\boldsymbol{d}\cdot\hat{\boldsymbol{n}})\hat{\boldsymbol{n}}$$



$$\boldsymbol{d} = \boldsymbol{m} + \boldsymbol{e}$$

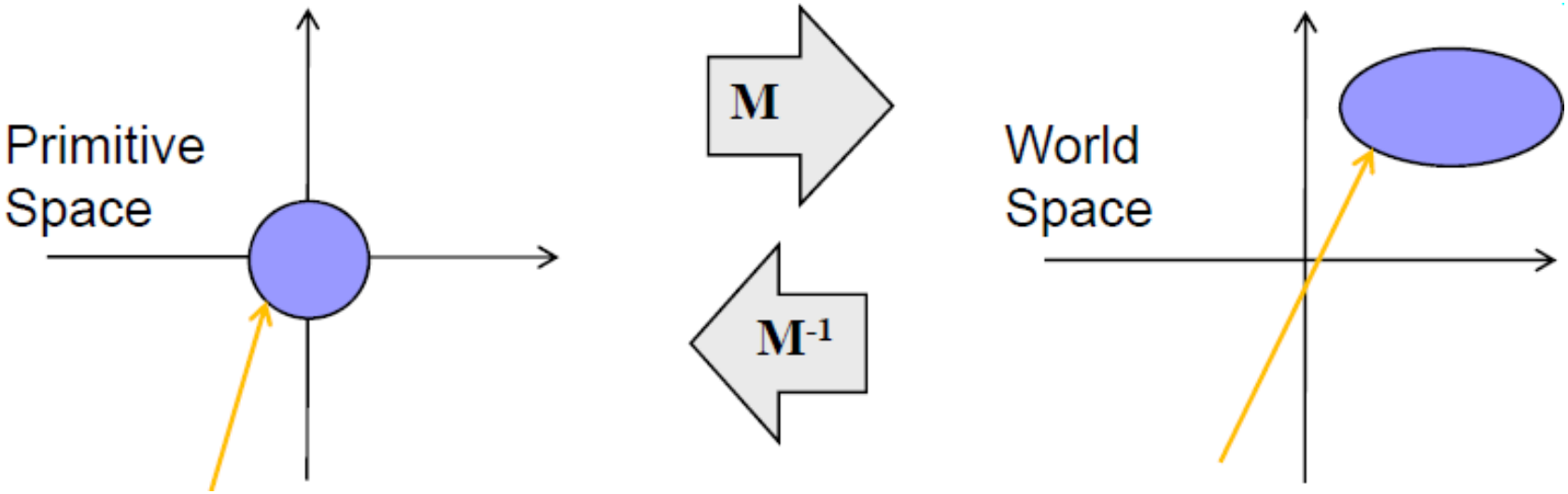
$$\boldsymbol{d}' = \boldsymbol{e} - \boldsymbol{m} = (\boldsymbol{d} - \boldsymbol{m}) - \boldsymbol{m} = \boldsymbol{d} - 2\boldsymbol{m}$$

$$= \boldsymbol{d} - 2(\boldsymbol{d}\cdot\hat{\boldsymbol{n}})\hat{\boldsymbol{n}}$$

Reflections

- Limit amount of reflections
  - Number
  - Strength

Transformations



$$ray'(t) = M^{-1} ray(t)$$

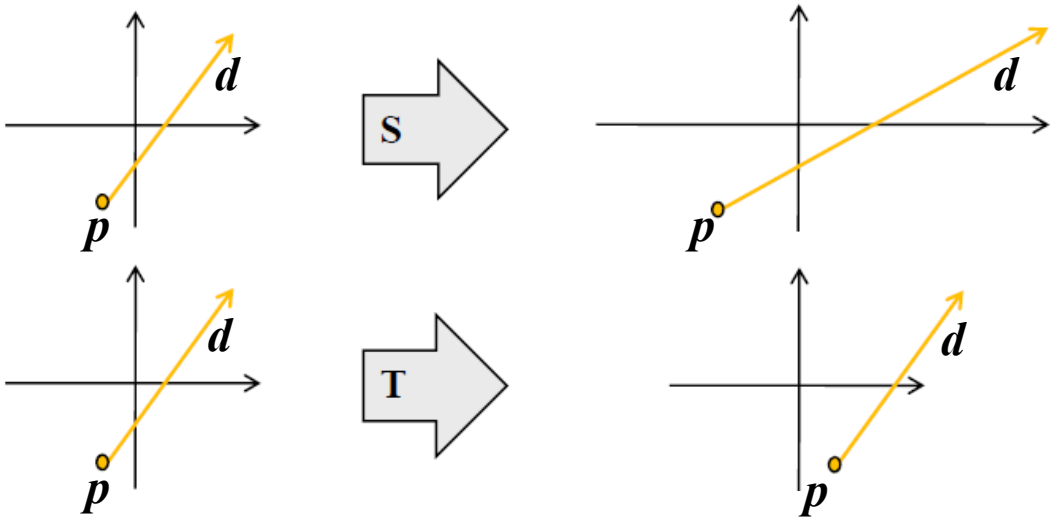
$$= M^{-1} p + M^{-1} d\, t$$

$$ray(t) = p + d\, t$$

!! t stays the same !!

Transformations

- Rotation and Scaling applies to $\boldsymbol{p}$ and $\boldsymbol{d}$
- Translation applies only to $\boldsymbol{p}$
  - Can be achieved by setting $w=0$



$$\boldsymbol{p}=\begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$ Point

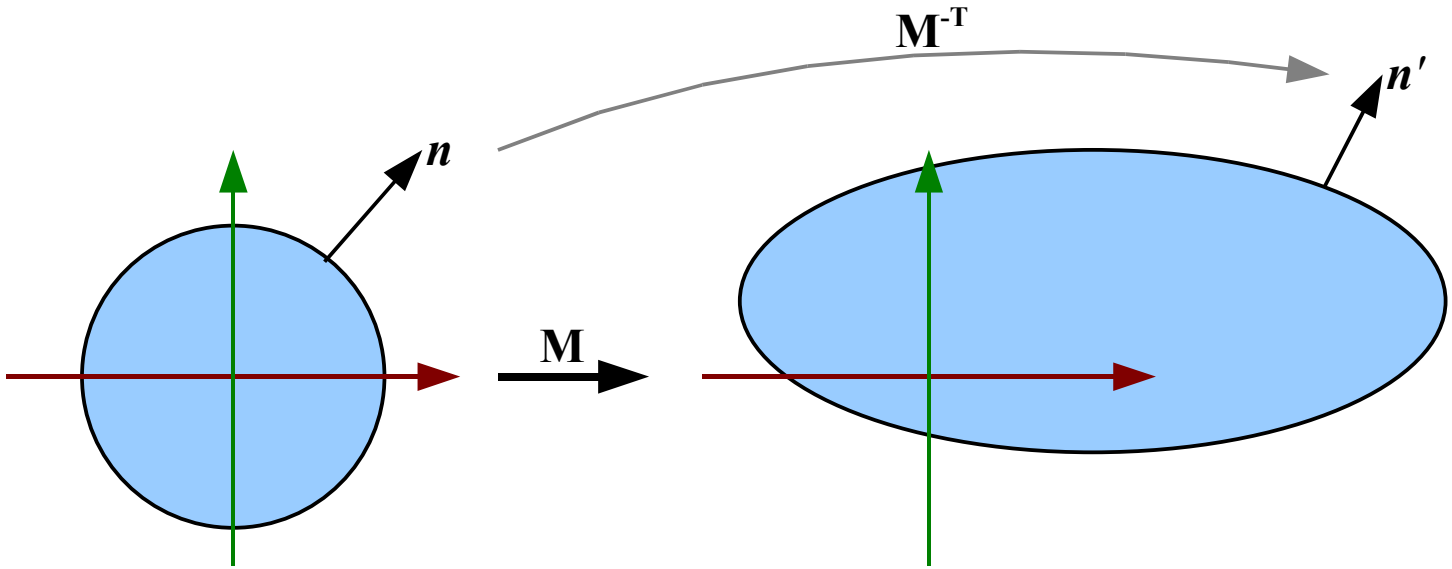
$$\boldsymbol{d}=\begin{pmatrix} d_x \\ d_y \\ d_z \\ 0 \end{pmatrix}$$ Direction

Transformations

- For normals: $\mathrm{M}^{-\mathrm{T}}$ without translation

Normalise afterwards!

$$\boldsymbol{M} = \begin{pmatrix} 3 & 0 & 0 & \cancel{1} \\ 0 & 1 & 0 & \cancel{1} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \boldsymbol{n} = \begin{pmatrix} n_x \\ n_y \\ n_z \\ 0 \end{pmatrix} \quad \boldsymbol{M}^{-T} = \begin{pmatrix} 1/3 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \boldsymbol{n}' = \begin{pmatrix} 1/3\, n_x \\ 1\, n_y \\ 1\, n_z \\ 0 \end{pmatrix}$$



Speedup

- Multithreading/Parallel computing
- Object extents
  - Spheres (only one half of the formula needed)
  - AABB's
- Subdivision (BSP, Quadtree, Octree)
- Item buffer
  (Ray tracing meets Polygon rendering)