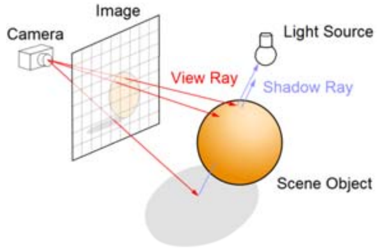
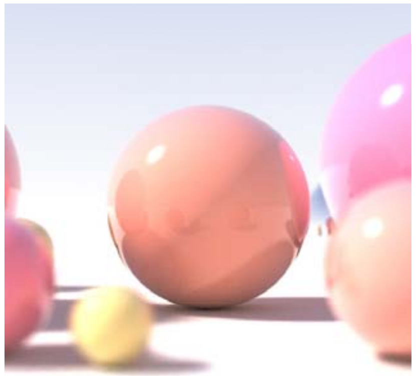
Computer Graphics: Ray Tracing I

Part 2 – Lecture 7

Today's Outline

- Introduction to Ray Tracing
- Ray Casting
- Intersecting Rays with Primitives
- Intersecting Rays with Transformed Primitives

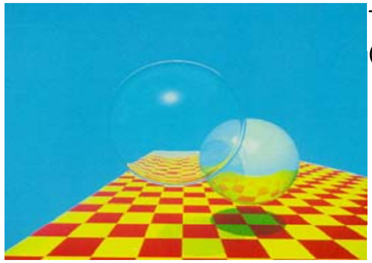




INTRODUCTION TO RAY TRACING

Images thanks to Henrik and Tim Babb

Ray Tracing Image Gallery



T. Whitted,1979
(44 mins on VAX)

"A Dirty Lab"
M. Borbely, 2000
Internet Ray Tracing Competition Winner

Terragen, thanks to T1g4h

Introduction to Ray Tracing

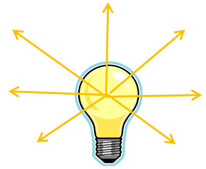
Recap: **Polygon Rendering**

- Visible polygons are projected onto a view plane
- Polygons made more realistic with texture mapping and shading
- Considers only a single light reflection per pixel
  (no reflections of scene on water, …)
- No refraction, no proper shadows

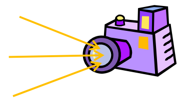
Ray Tracing

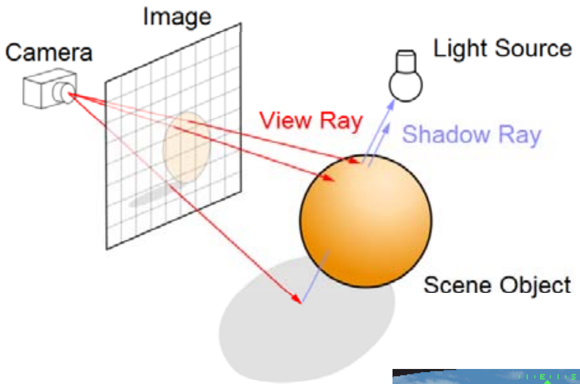
- Calculate the path of light rays
- Trace a light ray through several reflections, refractions, …
- Proper shadows where light rays do not hit a surface directly
- Much slower than polygon rendering, but can be photorealistic!

How to Trace Rays?



1. Trace light rays starting from a light source
   - Physically accurate
   - Essentially unlimited number of rays
   - Only a small fraction actually reaches the eye
   - Very, very slow for CGI, but possible (→ Monte-Carlo)
2. Trace only the light rays that hit the eye backwards
   - Not as accurate, but much faster!!!
   - Can follow the ray backwards through as many reflections and refractions as we like
   - For each object-ray intersection, we can calculate reflected light using an illumination model





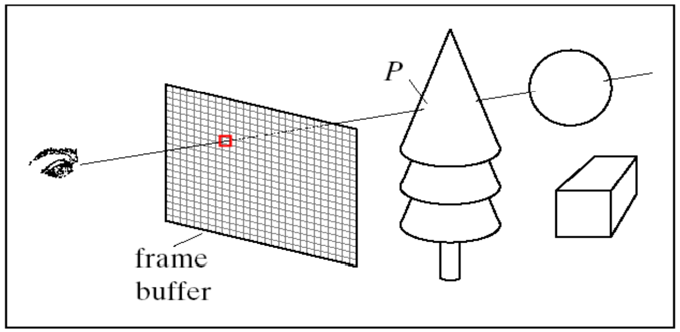
RAY CASTING

Comanche, 1992

Looking through the View Plane

What do we see through pixel (*c*,*r*) for all *c* and *r* ?

- Trace rays from eye into scene
- Describe each point **p** on a ray going through **a** and **b** by:
  p = **a** + t **b** (one point for each t)

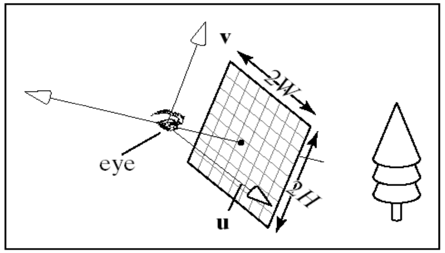


Hill textbook, Fig. 14.1

Setting Up the View Plane

Given:

- Camera position ***eye***
- View coord. system axes ***(u, v, n)***
- View plane width 2*W*
- View plane height 2*H*
- Number of pixel rows *nRows* and pixel columns *nCols* in view plane



Wanted: ray (***startPoint***, ***direction***) from eye through every pixel

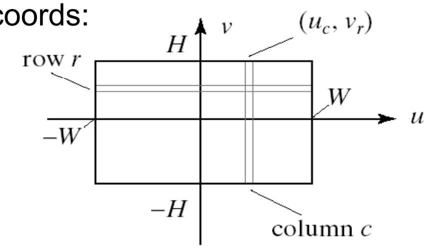
- **startPoint** is always **eye**
- Center of view plane: ***center*** = ***eye*** – *N* ***n***
  (N is distance from eye, usually choose N = 1)

Constructing Rays

Wanted: ray (***startPoint***, ***direction***) from eye through every pixel

- Corners of the view plane in world coords:

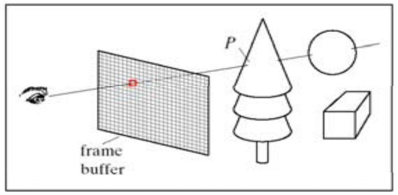
  *bottomLeft = centre + (-W**u**, -H**v**)*
  *bottomRight = centre + (W**u**, -H**v**)*
  *topLeft = centre + (-W**u**, H**v**)*
  *topRight = centre + (W**u**, H**v**)*



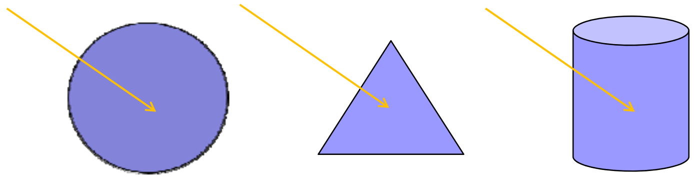
- Go through all pixels, with column 0 and row 0 at *bottomLeft*
- Ray direction ***d*** = ***pixelPos*** - ***eye***

$$\mathbf{d} = -N\mathbf{n} + W\left(\frac{2c}{nCols} - 1\right)\mathbf{u} + H\left(\frac{2r}{nRows} - 1\right)\mathbf{v}$$

Ray Casting Algorithm



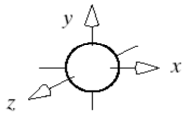

```
define scene = ({ objects }, { lights })
define camera (eye, u, v, n)
for (int r = 0; r < nRows; r++) {
    for (int c = 0; c < nCols; c++) {
        construct ray going through (c, r)
        find closest intersection of ray with an object (smallest t)
        find intersection point P
        get the surface normal at P
        pixel(c, r) =  the color at P as seen by the eye
    }
}
```

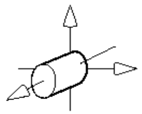



INTERSECTING RAYS WITH PRIMITIVES

Ray-Object Intersection

- Define each object as an implicit function f:
  f(**p**) = 0 for every point **p** on the surface of the object
  (if **p** is not on surface, then f(**p**) $\neq$ 0)
- Examples for simple objects ("primitives"):
  - Sphere (center at origin, radius 1)
    $f(\mathbf{p}) = x^2 + y^2 + z^2 - 1 = |\mathbf{p}|^2 - 1$
  - Cylinder (around z-axis, radius 1)
    $f(\mathbf{p}) = x^2 + y^2 - 1$
- Where a ray (**eye** + **d** *t*) meets the object:
  f(**eye** + **d** *t*) = 0
  $\rightarrow$ solve for *t* and get intersection point **eye** + **d** *t*



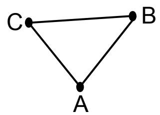


Ray-Plane Intersection

- Implicit function for plane (normal **n**, distance *a* from origin):
  p·n – *a* = 0
- Intersection when (**eye** + *t* **d**)·**n** – *a* = 0
  i.e. $$t = \frac{a - \mathbf{eye} \cdot \mathbf{n}}{\mathbf{d} \cdot \mathbf{n}}$$
- If **d·n**=0 then ray parallel to plane (ignore plane)
- If *t* negative then ray directed away from plane (no intersection)

Ray-Triangle Intersection

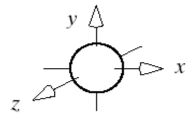
- Normal **n** = (B-A) $\times$ (C-A)
- Implicit function for corresponding plane: (**p**-A)·**n** = 0
- Intersection with plane when (**eye** + *t* **d** - A)·**n** = 0
  i.e. $$t = \frac{-(\mathbf{eye} - \mathbf{A}) \cdot \mathbf{n}}{\mathbf{d} \cdot \mathbf{n}}$$



- Is the ray-plane intersection at **p**=(**eye** + *t* **d**) inside the triangle?
  - Calculate scalars for three cross products (normals for the plane):
    ((B-A) $\times$ (**p**-A))·**n** and ((C-B) $\times$ (**p**-B))·**n** and ((A-C) $\times$ (**p**-C))·**n**
  - If they all have the same sign (e.g. all negative) then **p** is inside the triangle
  - Each tests if **p** is on the inside or outside of one of the edges
  - If p on the inside of an edge, then normal is directed towards us

Ray-Sphere Intersection

- Implicit function for sphere (center at origin, radius 1):
  p·p – 1 = 0
- Intersection when (**eye** + *t* **d**)·(**eye** + *t* **d**) – 1 = 0
  i.e. $$\mathbf{d} \cdot \mathbf{d}\ t^2 + 2\ \mathbf{eye}\ \mathbf{d}\ t + \mathbf{eye} \cdot \mathbf{eye} - 1 = 0$$



- Solve quadratic equation for *t*

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

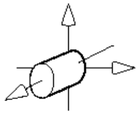
with *A*=**d·d**
B=2 **eye·d**
C= **eye·eye-1**

- If (B^2-4AC)<0 then ray misses sphere
- If (B^2-4AC)=0 then ray grazes sphere (treat as miss)
- If (B^2-4AC)>0 then one t for entry and one t for exit point (use smaller t for intersection, $\rightarrow$ closer to eye)

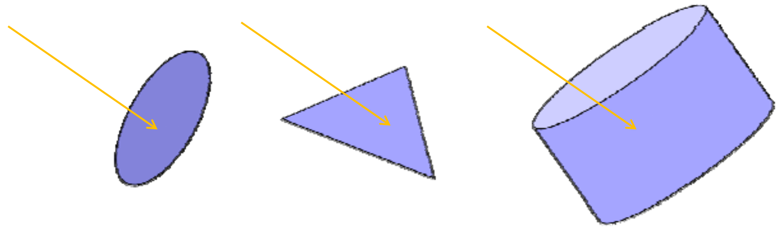
Ray-Cylinder Intersection

- Implicit function for sphere (one end at origin, radius 1, length 1):
  $x^2 + y^2 - 1 = 0$ and $0 \le z \le 1$
- Intersection when $0 \le \mathbf{eye}_z + t\,\mathbf{d}_z \le 1$
  and
  $(\mathbf{eye}_x + t\,\mathbf{d}_x)\cdot(\mathbf{eye}_x + t\,\mathbf{d}_x) + (\mathbf{eye}_y + t\,\mathbf{d}_y)\cdot(\mathbf{eye}_y + t\,\mathbf{d}_y) - 1 = 0$

  i.e. $(\mathbf{d}_x^2 + \mathbf{d}_y^2)\ t^2 + 2(\mathbf{eye}_x\mathbf{d}_x + \mathbf{eye}_y\mathbf{d}_y)t + \mathbf{eye}_x^2 + \mathbf{eye}_y^2 - 1 = 0$



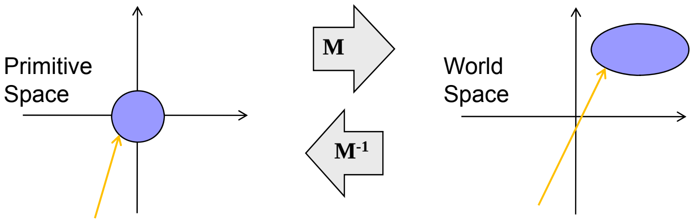
- Solve quadratic equation for t (same as for sphere)
- Check if $0 \le \mathbf{eye}_z + t\,\mathbf{d}_z \le 1$ (otherwise miss)
- **Note**: cylinder is open at the ends and hollow on the inside



INTERSECTING RAYS WITH TRANSFORMED PRIMITIVES

Transformed Primitives

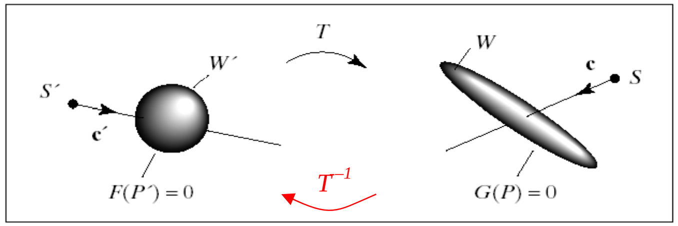
Problem: How to intersect with transformed primitives?
(e.g. scaled and translated unit sphere)



Solution: intersection of ray with transformed primitive is the same as intersection with inversely transformed ray and primitive

- Intersect with transformed ray ($\mathbf{eye}_t + \mathbf{d}_t\, t$)
  i.e. $\mathbf{eye}_t = \mathbf{M}^{-1}\,\mathbf{eye}$ and $\mathbf{d}_t = \mathbf{M}^{-1}\,\mathbf{d}$
- t for the intersection is the same in world and primitive space

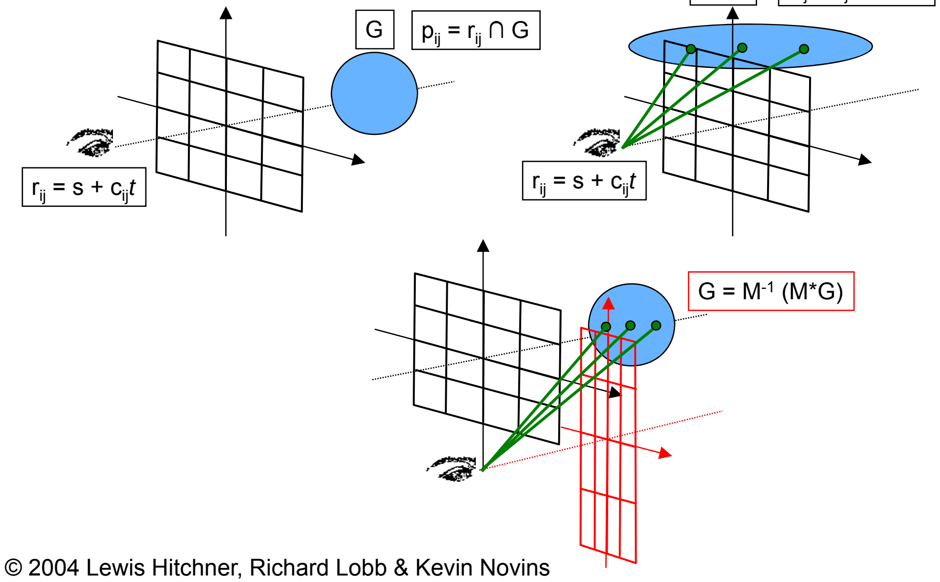
Transformed Primitives



$$\tilde{r}(t) = \mathbf{M}^{-1}\begin{pmatrix} S_x \\ S_y \\ S_z \\ 1 \end{pmatrix} + \mathbf{M}^{-1}\begin{pmatrix} c_x \\ c_y \\ c_z \\ 0 \end{pmatrix} t = \tilde{S}' + \tilde{\mathbf{c}}' t$$

Note this

Transformed Primitives

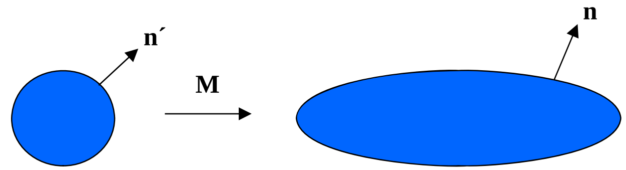


Normals for Transformed Primitives

- We need the intersection point and the surface normal
- **Recap**: given a normal **n**, after a transformation **M** the new normal is **n'** with **n'** = $\mathbf{M}^{-T}\,\mathbf{n}$ (but **n'** is not always normalized)

Example (in 2D):

$$\mathbf{M} = \begin{pmatrix} 3 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \mathbf{n} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \quad \mathbf{M}^{-T} = \begin{pmatrix} 1/3 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \mathbf{n}' = \begin{pmatrix} 1/3 \\ 1 \\ 0 \end{pmatrix}$$



SUMMARY

Summary

1. Ray tracing is slow, but can give us photorealistic images
   - Tracing the light rays that hit the eye backwards
   - Trace each light ray through several reflections, refractions, …
2. Ray casting: simple form of ray tracing
   - Trace one ray per screen pixel on the frame buffer
   - Trace only until it hits an object (i.e. only one reflection)
3. Calculate ray-object intersections by putting ray equation into implicit object function and solving for t

References:

- Introduction to Ray Tracing: Hill, Chapters 12.1 - 12.3
- Ray-Object Intersection: Hill, Chapter 12.4

Quiz

1. What is ray tracing? Give a brief general description.
2. Give pseudo-code for the ray casting algorithm.
3. What is the general approach when looking for a ray-object intersection?
4. What is the common approach when looking for intersections with transformed primitives?