TIEA311
Tietokonegrafiikan perusteet
kevät 2018

("Principles of Computer Graphics" – Spring 2018)

TIEA311 Tietokonegrafiikan perusteet – kevät 2018 (“Principles of Computer Graphics” – Spring 2018)

Adapted from: *Wojciech Matusik*, and *Frédo Durand*: 6.837 Computer Graphics. Fall 2012. Massachusetts Institute of Technology: MIT OpenCourseWare, https://ocw.mit.edu/.

Frontpage of the local course version, held during Spring 2018 at the Faculty of Information technology, University of Jyväskylä:
`http://users.jyu.fi/~nieminen/tgp18/`

TIEA311 - Today in Jyväskylä

Part 1:

- What is “dot product”? (Go google..)! Used a lot in the equations!
- Continue the theory of ray casting from last time, without much recap. Remember to have a break!

Part 2, After a break:

- Continue the theory as far as we can today.
- Goal: Assignment 4 gets to full speed. One week left, so we need things to think about!

Later today (March 1, 2018):

- Guidance available in computer class 16:15–17:45
- 18:00 Instanssi 2018 begins! Guidance available on-site.

Google time!

On the lecture, we spent time in the Internet, looking at some definitions and properties of the **Dot product** and **Inner products**.

Point of the exercise:

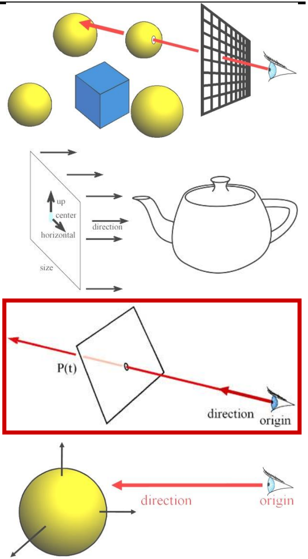
When you forget how some detail works, what do you do? You connect to the WWW and re-learn the detail.

When you need to learn a new detail, what do you do? You connect to the WWW and learn the detail. You only need a keyword to find it.

That much is simple. Things get interesting in putting the details to action.

Ray Casting

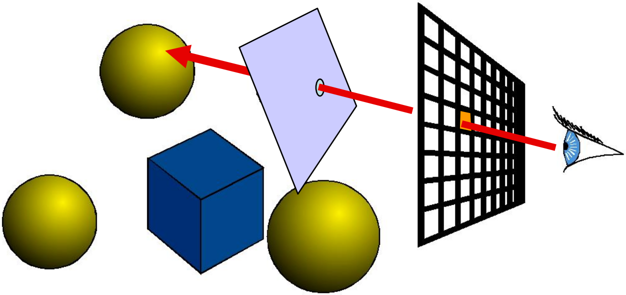
- Ray Casting Basics
- Camera and Ray Generation
- Ray-Plane Intersection
- Ray-Sphere Intersection



Ray Casting

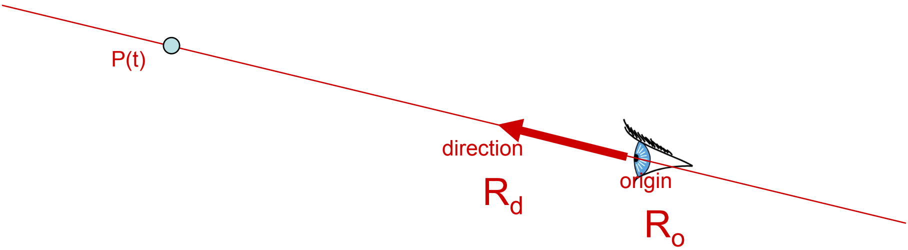
```
For every pixel
   Construct a ray from the eye
   For every object in the scene
      Find intersection with the ray
      Keep if closest
```

First we will study ray-plane intersection



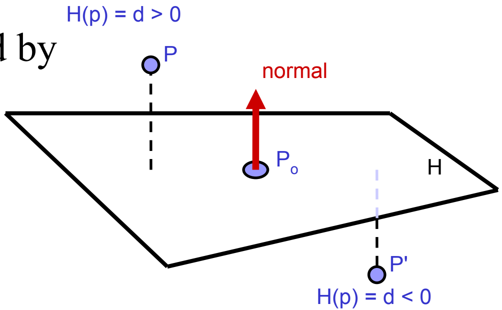
Recall: Ray Representation

- Parametric line
- $P(t) = R_o + t * R_d$
- Explicit representation



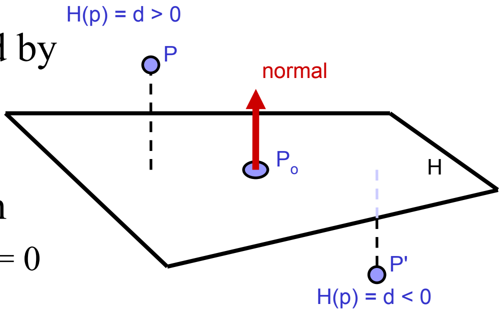
3D Plane Representation?

- (Infinite) plane defined by
  - $P_o = (x_0, y_0, z_0)$
  - $n = (A,B,C)$



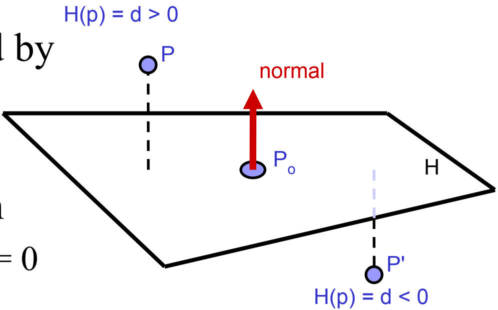
3D Plane Representation?

- (Infinite) plane defined by
  - $P_o = (x_0, y_0, z_0)$
  - $n = (A,B,C)$
- Implicit plane equation
  - $H(P) = Ax+By+Cz+D = 0$
    $= n \cdot P + D = 0$



3D Plane Representation?

- (Infinite) plane defined by
  - $P_o = (x_0, y_0, z_0)$
  - $n = (A,B,C)$
- Implicit plane equation
  - $H(P) = Ax+By+Cz+D = 0$
    $= n \cdot P + D = 0$
  - What is D?



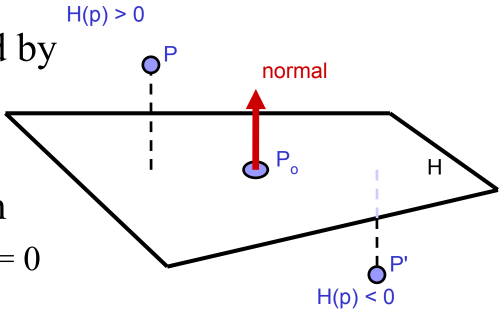
$$Ax_0 + By_0 + Cz_0 + D = 0$$

(Point P_0 must lie on plane)

$$\Rightarrow D = -Ax_0 - By_0 - Cz_0$$

3D Plane Representation?

- (Infinite) plane defined by
  - $P_o = (x_0, y_0, z_0)$
  - $n = (A,B,C)$
- Implicit plane equation
  - $H(P) = Ax+By+Cz+D = 0$
    $= n \cdot P + D = 0$
- Point-Plane distance?
  - If n is normalized, distance to plane is H(P)
  - it is a ***signed*** *distance*!



Explicit vs. Implicit?

- Ray equation is explicit $P(t) = R_o + t * R_d$
  - Parametric
  - Generates points
  - Hard to verify that a point is on the ray
- Plane equation is implicit $H(P) = n \cdot P + D = 0$
  - Solution of an equation
  - Does not generate points
  - Verifies that a point is on the plane

- Exercise: Explicit plane and implicit ray?

Ray-Plane Intersection

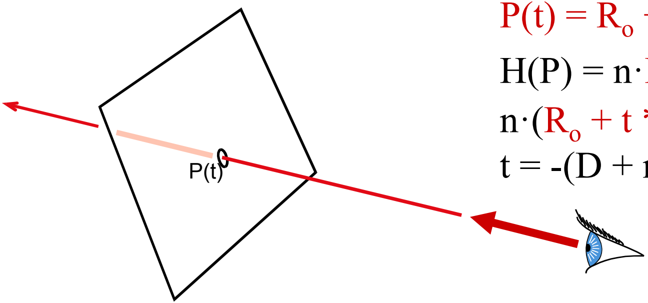
- Intersection means both are satisfied
- So, insert explicit equation of ray into implicit equation of plane & solve for t

$$P(t) = R_o + t * R_d$$

$$H(P) = n \cdot P + D = 0$$

$$n \cdot (R_o + t * R_d) + D = 0$$

$$t = -(D + n \cdot R_o) / n \cdot R_d$$



Done!

Done!? What the.. How?

Puzzled by **how** the final equation “suddenly appears”?

You **should be**, at least for a second. And then **as long as it takes**, until you are happy that you understand and agree.

This was talked through and sketched on lecture. What you **should always do** when attempting to fully understand “anything math” is to fill all the gaps either in your brain (**impossible at first**, becoming **possible** and then **faster** only with experience) or with pen and paper. **Suspect everything** until you agree, every step of the way! With your own hands, you can also use cleaner notation than in some slide set, for example to mark up vectors apart from scalars using “arrow hats”.

The next slide leaves not many gaps. Once you understand the “legal moves”, you can start combining them in your head, no more writing out those dull intermediate steps. Math articles and textbooks (even introductory ones!) leave out many “obvious”, “minor” details, because **they expect the reader to fill them in**, one way (brain) or the other (brain & paper)!

Done!? What the.. How?

$\vec{n} \cdot (\vec{R}_o + t\vec{R}_d) + D = 0$	Start with equation. Do stuff that keeps both sides equal, towards leaving only t on the left side.
$\vec{n} \cdot (\vec{R}_o + t\vec{R}_d) + D - D = 0 - D$	Added $-D$ to both sides. Different but equal.
$\vec{n} \cdot (\vec{R}_o + t\vec{R}_d) + (D - D) = 0 - D$	Regroup (real sums are associative)
$\vec{n} \cdot (\vec{R}_o + t\vec{R}_d) + 0 = 0 - D$	Sum of additive inverses yields zero (definition of "minus": $D - D = D + (-D) = 0$)
$\vec{n} \cdot (\vec{R}_o + t\vec{R}_d) = -D$	Rid of zeros (neutral element for addition, i.e., additive identity). Performing the steps up to here, all at once, should have become "obvious" in high school; underlying axiomatic algebra likely not.
$\vec{n} \cdot \vec{R}_o + \vec{n} \cdot (t\vec{R}_d) = -D$	Dot product is distributive over vector addition
$\vec{n} \cdot \vec{R}_o - \vec{n} \cdot \vec{R}_o + \vec{n} \cdot (t\vec{R}_d) = -D - \vec{n} \cdot \vec{R}_o$	Add $-\vec{n} \cdot \vec{R}_O$ (additive inverse, like $-D$ above) to both sides. Middle OK since sum is commutative.
$(\vec{n} \cdot \vec{R}_o - \vec{n} \cdot \vec{R}_o) + \vec{n} \cdot (t\vec{R}_d) = -D - \vec{n} \cdot \vec{R}_o$	Regroup (associativity again)
$0 + \vec{n} \cdot (t\vec{R}_d) = -D - \vec{n} \cdot \vec{R}_o$	Sum of additive inverses (again)
$\vec{n} \cdot (t\vec{R}_d) = -D - \vec{n} \cdot \vec{R}_o$	Rid of zero (additive identity)
$t(\vec{n} \cdot \vec{R}_d) = -D - \vec{n} \cdot \vec{R}_o$	Scalar multiplication property of dot product
$t(\vec{n} \cdot \vec{R}_d)(\vec{n} \cdot \vec{R}_d)^{-1} = (-D - \vec{n} \cdot \vec{R}_o)(\vec{n} \cdot \vec{R}_d)^{-1}$	Multiply both sides by multiplicative inverse ("divide"). **Such inverse is not defined for** 0 though!
$t * 1 = (-D - \vec{n} \cdot \vec{R}_o)(\vec{n} \cdot \vec{R}_d)^{-1}$	multiplication by inverse yields multiplicative identity 1; multiplication denoted $*$ for clarity
$t = -(D + \vec{n} \cdot \vec{R}_o)(\vec{n} \cdot \vec{R}_d)^{-1}$	Rid of 1 (multiplicative identity). Distributive and associative properties used on right to fit slide.
$t = -\dfrac{D + \vec{n} \cdot \vec{R}_o}{\vec{n} \cdot \vec{R}_d}$	Use fractional "divide-by" notation for multiplication by the multiplicative inverse

Done!? What the.. Oh, yes, done indeed!

And that was why

$$\vec{n} \cdot (\vec{R}_o + t\vec{R}_d) + D = 0$$

gives us

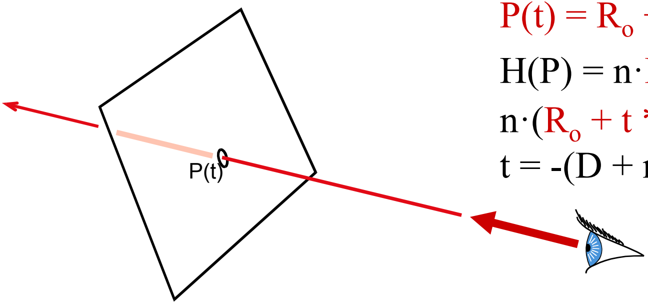
$$t = -\frac{D + \vec{n} \cdot \vec{R}_o}{\vec{n} \cdot \vec{R}_d}$$

“as the reader should verify” :).

Meanwhile, the reader will have noticed the possible case of division by zero! The reader will have attempted to figure out if and when it could happen, possibly by sketching figures, re-checking what the equations mean, and using real-world artefacts in front of real-world eye-rays (see the lecture video for example). If the reader hasn’t done this, he or she may have wasted time just looking at random equations and not learning too much.

Ray-Plane Intersection

- Intersection means both are satisfied
- So, insert explicit equation of ray into implicit equation of plane & solve for t



$P(t) = R_o + t * R_d$

$H(P) = n \cdot P + D = 0$

$n \cdot (R_o + t * R_d) + D = 0$

$t = -(D + n \cdot R_o) / n \cdot R_d$

Done!

Ray-Plane Intersection

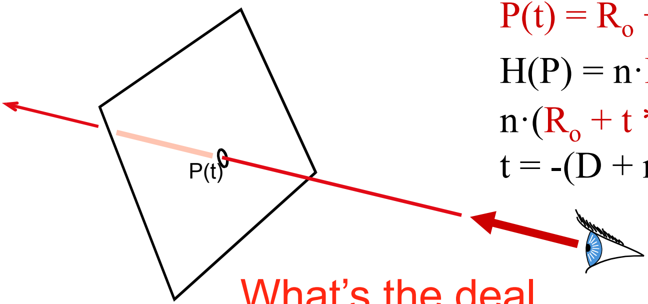
- Intersection means both are satisfied
- So, insert explicit equation of ray into implicit equation of plane & solve for t

$P(t) = R_o + t * R_d$

$H(P) = n \cdot P + D = 0$

$n \cdot (R_o + t * R_d) + D = 0$

$t = -(D + n \cdot R_o) / n \cdot R_d$



Done!

What's the deal when $n \cdot R_d = 0$?

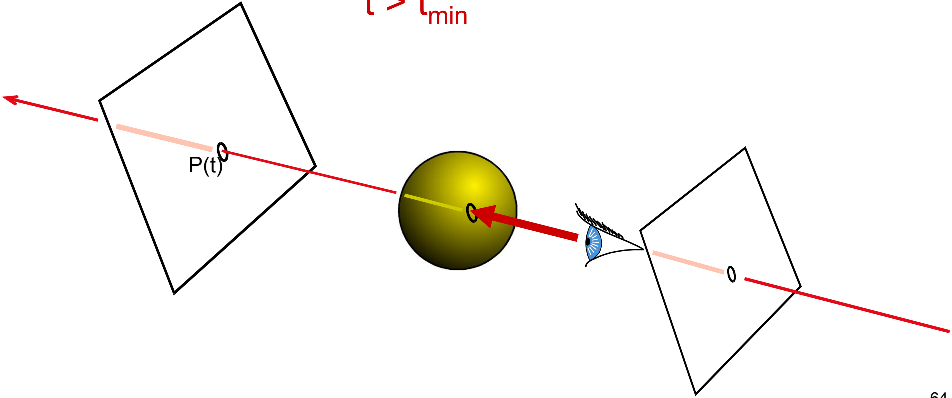
Additional Bookkeeping

- Verify that intersection is closer than previous

  $t < t_{current}$

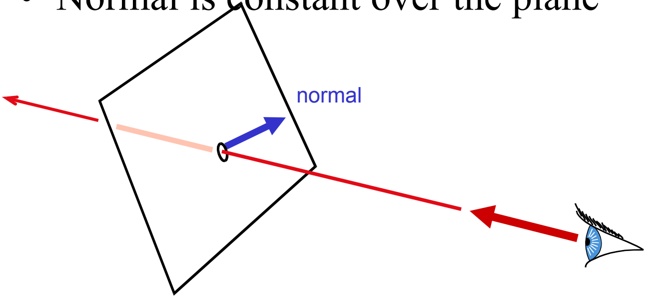
- Verify that it is not out of range (behind eye)

  $t > t_{min}$



Normal

- Also need surface normal for shading
  - (Diffuse: dot product between light direction and normal, clamp to zero)
- Normal is constant over the plane



Questions?

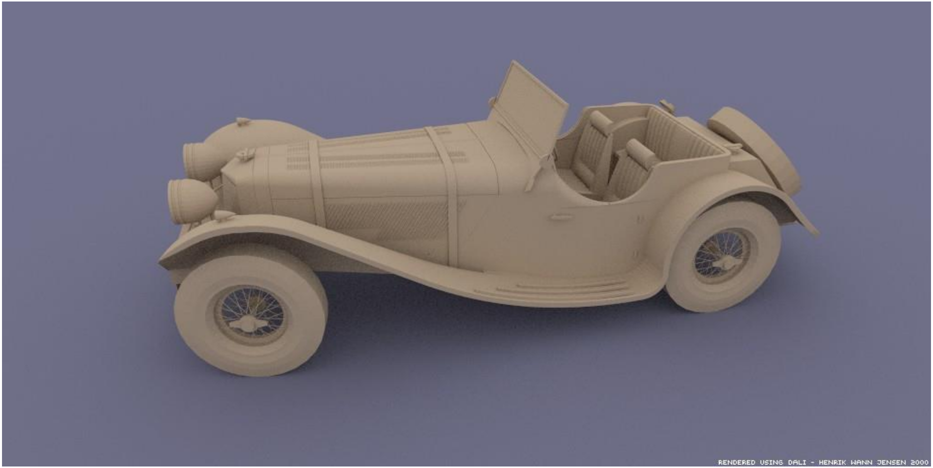
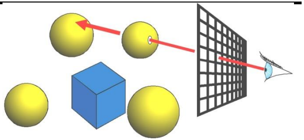
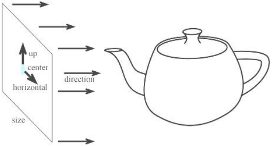


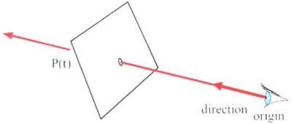
Image by Henrik Wann Jensen

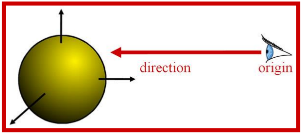
Ray Casting

- Ray Casting Basics
- Camera and Ray Generation
- Ray-Plane Intersection
- Ray-Sphere Intersection



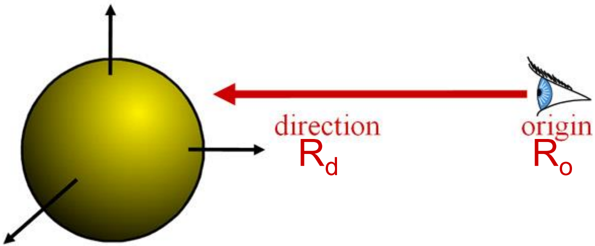






Sphere Representation?

- Implicit sphere equation
  - Assume centered at origin (easy to translate)
  - $H(P) = \|P\|^2 - r^2 = P \cdot P - r^2 = 0$



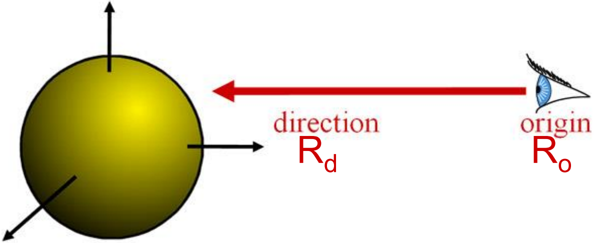
Ray-Sphere Intersection

- Insert explicit equation of ray into implicit equation of sphere & solve for t

$P(t) = R_o + t^*R_d \qquad ; \qquad H(P) = P \cdot P - r^2 = 0$

$$(R_o + tR_d) \cdot (R_o + tR_d) - r^2 = 0$$

$$R_d \cdot R_d t^2 + 2R_d \cdot R_o t + R_o \cdot R_o - r^2 = 0$$

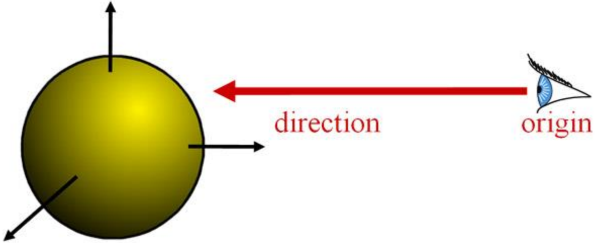


Ray-Sphere Intersection

- Quadratic: $at^2 + bt + c = 0$
  - $a = 1$ (remember, $||R_d|| = 1$)
  - $b = 2R_d \cdot R_o$
  - $c = R_o \cdot R_o - r^2$
- with discriminant $$d = \sqrt{b^2 - 4ac}$$
- and solutions $$t_{\pm} = \frac{-b \pm d}{2a}$$

Ray-Sphere Intersection

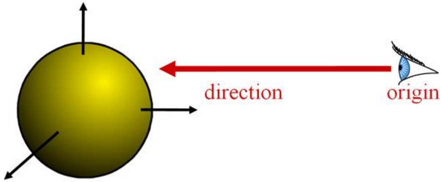
- 3 cases, depending on the sign of $b^2 - 4ac$
- What do these cases correspond to?
- Which root (t+ or t-) should you choose?
  - Closest positive!

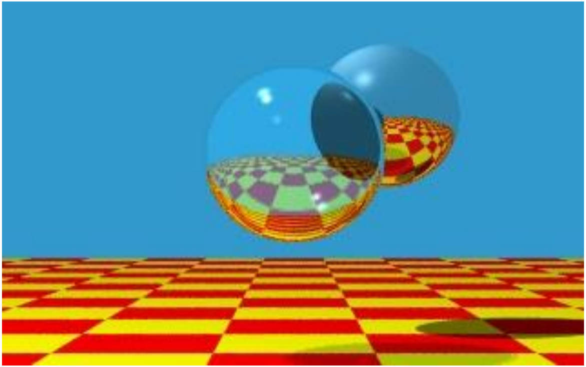


Ray-Sphere Intersection

- It's so easy that all ray-tracing images have spheres!

:-)

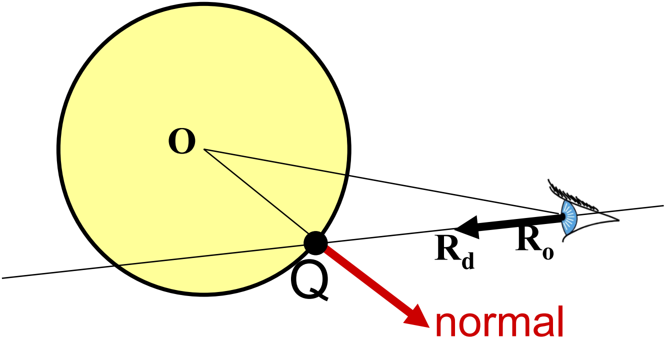




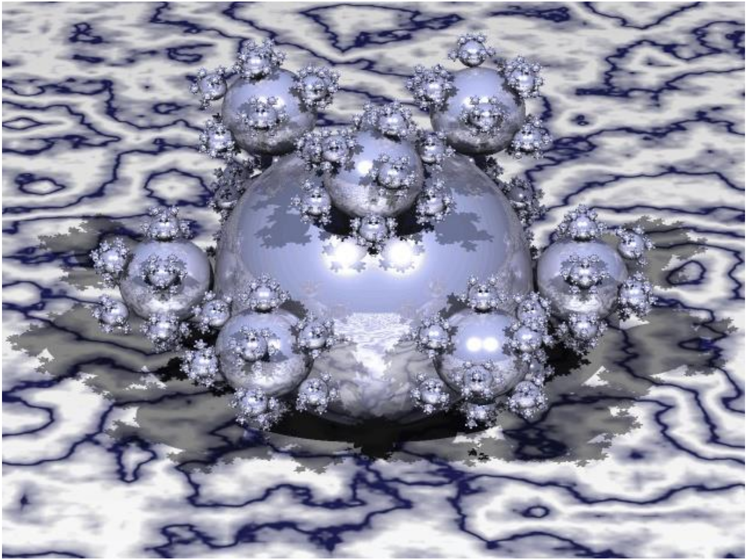
Turner Whitted

Sphere Normal

- Simply Q/||Q||
  - Q = P(t), intersection point
  - (for spheres centered at origin)



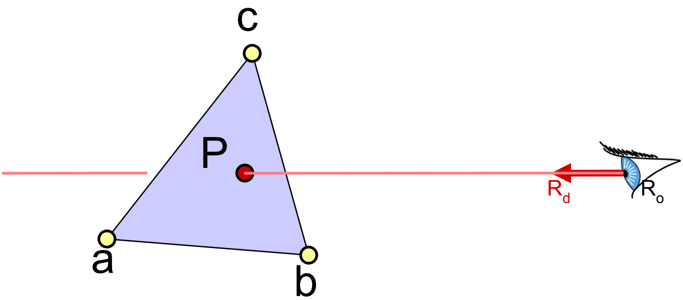
Questions?



Courtesy of Henrik Wann Jensen. Used with permission.

Ray-Triangle Intersection

- Use ray-plane intersection followed by in-triangle test
- Or try to be smarter
  - Use barycentric coordinates

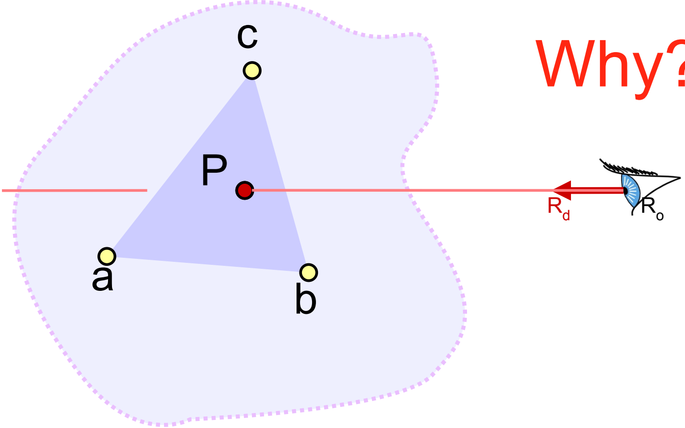


Barycentric Definition of a Plane

- A (non-degenerate) triangle (**a**,**b**,**c**) defines a plane
- Any point **P** on this plane can be written as

  $\mathbf{P}(\alpha,\beta,\gamma) = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$,

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



Why? How?

[Möbius, 1827]

Barycentric Coordinates

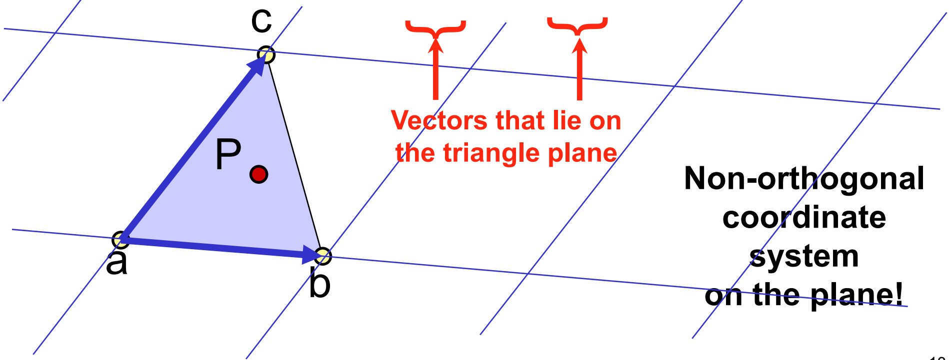
- Since $\alpha+\beta+\gamma=1$, we can write $\alpha = 1-\beta-\gamma$

$$\mathbf{P}(\alpha,\beta,\gamma) = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$$

rewrite

$$\mathbf{P}(\beta,\gamma) = (1-\beta-\gamma)\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$$

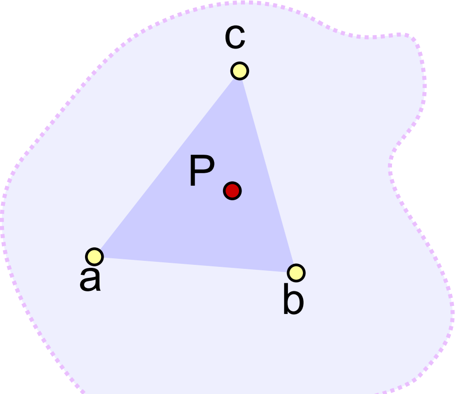
$$= \mathbf{a} + \beta\underbrace{(\mathbf{b}-\mathbf{a})} + \gamma\underbrace{(\mathbf{c}-\mathbf{a})}$$



Barycentric Definition of a Plane

[Möbius, 1827]

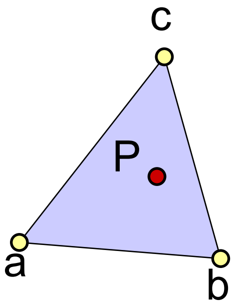
- $\mathbf{P}(\alpha,\beta,\gamma) = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$
  with $\alpha+\beta+\gamma = 1$
- Is it explicit or implicit?



Fun to know:
P is the **barycenter**, the single point upon which the triangle would balance if weights of size α, β, & γ are placed on points **a**, **b** & **c**.

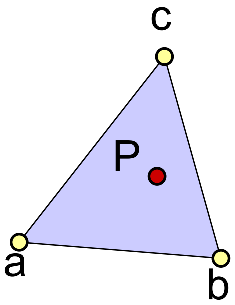
Barycentric Definition of a Triangle

- $\mathbf{P}(\alpha,\beta,\gamma) = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$
  with $\alpha+\beta+\gamma = 1$ parameterizes the entire plane



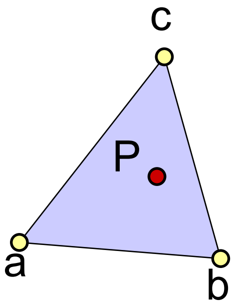
Barycentric Definition of a Triangle

- $\mathbf{P}(\alpha,\beta,\gamma) = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$
  with $\alpha+\beta+\gamma = 1$ parameterizes the entire plane
- If we require in addition that
  $\alpha, \beta, \gamma >= 0$, we get just the triangle!
  - Note that with $\alpha+\beta+\gamma = 1$ this implies
    $0 \leq \alpha \leq 1$ & $0 \leq \beta \leq 1$ & $0 \leq \gamma \leq 1$
  - Verify:
    - $\alpha = 0$ => **P** lies on line **b-c**
    - $\alpha, \beta = 0$ => **P** = **c**
    - etc.



Barycentric Definition of a Triangle

- $\mathbf{P}(\alpha,\beta,\gamma) = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$
- Condition to be barycentric coordinates: $\alpha+\beta+\gamma = 1$
- Condition to be inside the triangle: $\alpha, \beta, \gamma \geq 0$



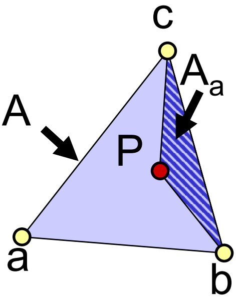
TIEA311 - Fast forward a bit

Let us briefly skim through a couple of slides about determining the barycentric coordinates of a point (already known to be within the plane).

(Example of some “math kinda stuff”)

How Do We Compute α, β, γ ?

- Ratio of opposite sub-triangle area to total area
  - $\alpha = A_a/A \qquad \beta = A_b/A \qquad \gamma = A_c/A$
- Use signed areas for points outside the triangle



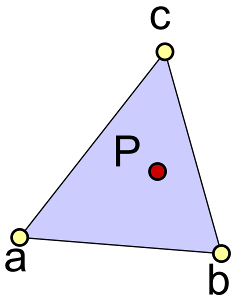
How Do We Compute α, β, γ ?

- Or write it as a 2×2 linear system
- $\mathbf{P}(\beta, \gamma) = \mathbf{a} + \beta\mathbf{e}_1 + \gamma\mathbf{e}_2$

  $\mathbf{e}_1 = (\mathbf{b}-\mathbf{a})$, $\mathbf{e}_2 = (\mathbf{c}-\mathbf{a})$

$$\boldsymbol{a} + \beta\boldsymbol{e}_1 + \gamma\boldsymbol{e}_2 - \boldsymbol{P} = 0$$

This should be zero



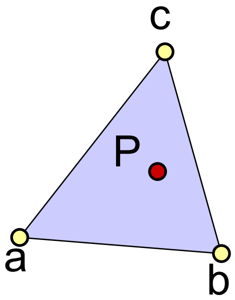
How Do We Compute α, β, γ ?

- Or write it as a 2×2 linear system
- $\mathbf{P}(\beta, \gamma) = \mathbf{a} + \beta\mathbf{e}_1 + \gamma\mathbf{e}_2$

  $\mathbf{e}_1 = (\mathbf{b}-\mathbf{a})$, $\mathbf{e}_2 = (\mathbf{c}-\mathbf{a})$

$$\boldsymbol{a} + \beta\boldsymbol{e}_1 + \gamma\boldsymbol{e}_2 - \boldsymbol{P} = 0$$

This should be zero



Something's wrong... This is a linear system of 3 equations and 2 unknowns!

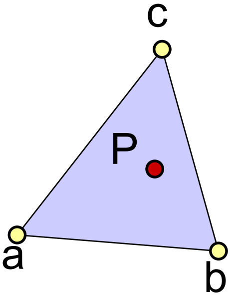
How Do We Compute α, β, γ ?

- Or write it as a 2×2 linear system
- $\mathbf{P}(\beta, \gamma) = \mathbf{a} + \beta\mathbf{e}_1 + \gamma\mathbf{e}_2$

  $\mathbf{e}_1 = (\mathbf{b}-\mathbf{a})$, $\mathbf{e}_2 = (\mathbf{c}-\mathbf{a})$

$$\langle \boldsymbol{e}_1,\ \boldsymbol{a} + \beta\boldsymbol{e}_1 + \gamma\boldsymbol{e}_2 - \boldsymbol{P} \rangle = 0$$
$$\langle \boldsymbol{e}_2,\ \boldsymbol{a} + \beta\boldsymbol{e}_1 + \gamma\boldsymbol{e}_2 - \boldsymbol{P} \rangle = 0$$

These should be zero



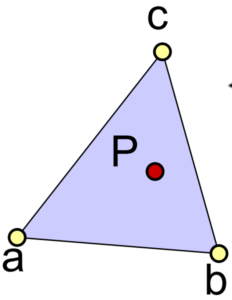
Ha! We'll take inner products of this equation with $\mathbf{e}_1$ & $\mathbf{e}_2$

How Do We Compute α, β, γ ?

- Or write it as a 2×2 linear system
- $\mathbf{P}(\beta, \gamma) = \mathbf{a} + \beta\mathbf{e}_1 + \gamma\mathbf{e}_2$

  $\mathbf{e}_1 = (\mathbf{b}\text{-}\mathbf{a}),\ \mathbf{e}_2 = (\mathbf{c}\text{-}\mathbf{a})$

$$\langle \boldsymbol{e}_1,\ \boldsymbol{a} + \beta\boldsymbol{e}_1 + \gamma\boldsymbol{e}_2 - \boldsymbol{P}\rangle = 0$$
$$\langle \boldsymbol{e}_2,\ \boldsymbol{a} + \beta\boldsymbol{e}_1 + \gamma\boldsymbol{e}_2 - \boldsymbol{P}\rangle = 0$$



$$\Leftrightarrow \boxed{\begin{pmatrix} \langle \boldsymbol{e}_1, \boldsymbol{e}_1\rangle & \langle \boldsymbol{e}_1, \boldsymbol{e}_2\rangle \\ \langle \boldsymbol{e}_2, \boldsymbol{e}_1\rangle & \langle \boldsymbol{e}_2, \boldsymbol{e}_2\rangle \end{pmatrix} \begin{pmatrix} \beta \\ \gamma \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}}$$

where $\begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} \langle(\boldsymbol{P} - \boldsymbol{a}), \boldsymbol{e}_1\rangle \\ \langle(\boldsymbol{P} - \boldsymbol{a}), \boldsymbol{e}_2\rangle \end{pmatrix}$

and <a,b> is the dot product.

Back to basics. . .

. . . which means: Just grab an equation from a “math person” and reproduce it in C++ (or any other language) for fun and/or profit.

Intersection with Barycentric Triangle

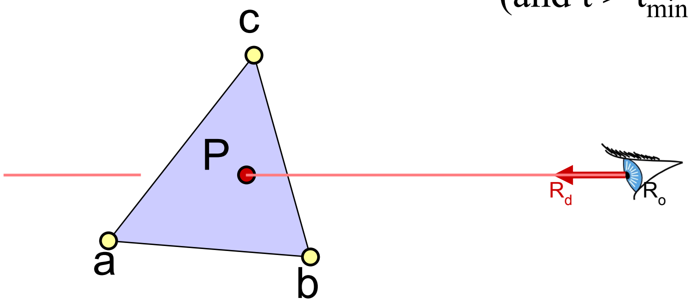
- Again, set ray equation equal to barycentric equation

$$\mathbf{P}(t) = \mathbf{P}(\beta, \gamma)$$

$$\mathbf{R}_o + t * \mathbf{R}_d = \mathbf{a} + \beta(\mathbf{b}\text{-}\mathbf{a}) + \gamma(\mathbf{c}\text{-}\mathbf{a})$$

- Intersection if $\beta + \gamma \leq 1$ & $\beta \geq 0$ & $\gamma \geq 0$

(and $t > t_{min}$ …)



Intersection with Barycentric Triangle

- $\mathbf{R_o} + t * \mathbf{R_d} = \mathbf{a} + \beta(\mathbf{b}-\mathbf{a}) + \gamma(\mathbf{c}-\mathbf{a})$

$$\left.\begin{aligned} R_{ox} + tR_{dx} &= a_x + \beta(b_x-a_x) + \gamma(c_x-a_x) \\ R_{oy} + tR_{dy} &= a_y + \beta(b_y-a_y) + \gamma(c_y-a_y) \\ R_{oz} + tR_{dz} &= a_z + \beta(b_z-a_z) + \gamma(c_z-a_z) \end{aligned}\right\} \text{3 equations, 3 unknowns}$$

- Regroup & write in matrix form $\mathbf{Ax}=\mathbf{b}$

$$\begin{bmatrix} a_x - b_x & a_x - c_x & R_{dx} \\ a_y - b_y & a_y - c_y & R_{dy} \\ a_z - b_z & a_z - c_z & R_{dz} \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} a_x - R_{ox} \\ a_y - R_{oy} \\ a_z - R_{oz} \end{bmatrix}$$

Cramer's Rule

- Used to solve for one variable at a time in system of equations

$$\beta = \frac{\begin{vmatrix} a_x - R_{ox} & a_x - c_x & R_{dx} \\ a_y - R_{oy} & a_y - c_y & R_{dy} \\ a_z - R_{oz} & a_z - c_z & R_{dz} \end{vmatrix}}{|A|} \qquad \gamma = \frac{\begin{vmatrix} a_x - b_x & a_x - R_{ox} & R_{dx} \\ a_y - b_y & a_y - R_{oy} & R_{dy} \\ a_z - b_z & a_z - R_{oz} & R_{dz} \end{vmatrix}}{|A|}$$

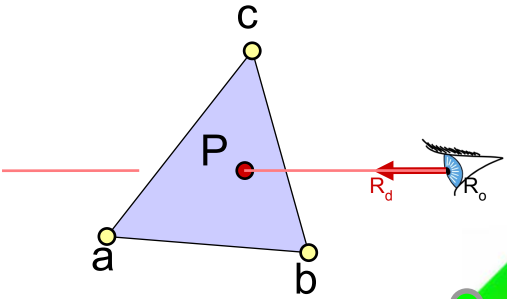
$$t = \frac{\begin{vmatrix} a_x - b_x & a_x - c_x & a_x - R_{ox} \\ a_y - b_y & a_y - c_y & a_y - R_{oy} \\ a_z - b_z & a_z - c_z & a_z - R_{oz} \end{vmatrix}}{|A|}$$

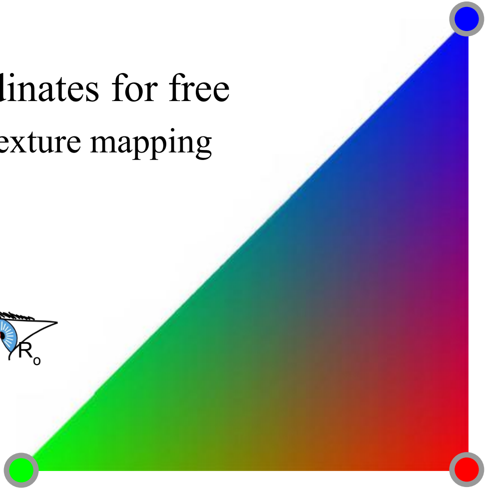
| | denotes the determinant

Can be copied mechanically into code

Barycentric Intersection Pros

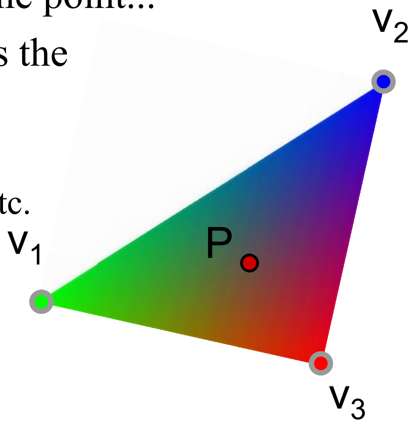
- Efficient
- Stores no plane equation
- Get the barycentric coordinates for free
  - Useful for interpolation, texture mapping





Barycentric Interpolation

- Values v_1, v_2, v_3 defined at **a**, **b**, **c**
  - Colors, normal, texture coordinates, etc.
- $\mathbf{P}(\alpha,\beta,\gamma) = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$ is the point...
- $v(\alpha,\beta,\gamma) = \alpha v_1 + \beta v_2 + \gamma v_3$ is the barycentric interpolation of v_1,v_2,v_3 at point **P**
  - Sanity check: $v(1,0,0) = v_1$, etc.
- I.e, once you know α,β,γ you can interpolate values using the same weights.
  - Convenient!

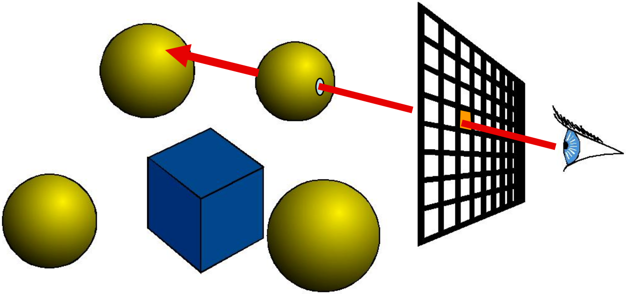


Questions?

- Image computed using the RADIANCE system by Greg Ward

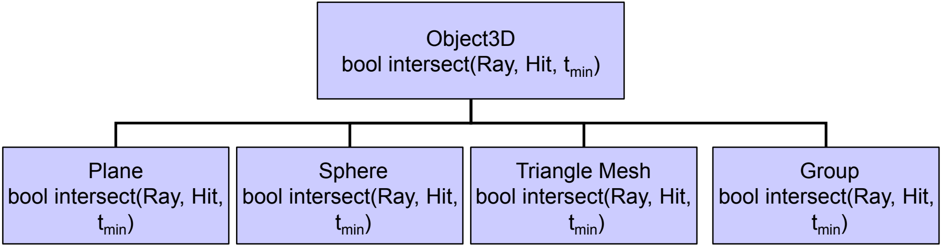
Ray Casting: Object Oriented Design

```
For every pixel
   Construct a ray from the eye
   For every object in the scene
       Find intersection with the ray
       Keep if closest
```



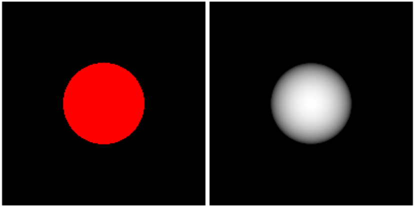
Object-Oriented Design

- We want to be able to add primitives easily
  - Inheritance and virtual methods
- Even the scene is derived from Object3D!

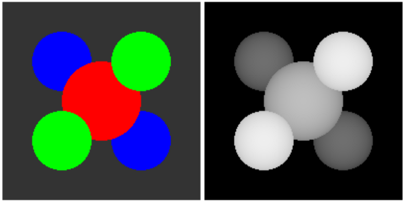


- Also cameras are abstracted (perspective/ortho)
  - Methods for generating rays for given image coordinates

Assignment 4 & 5: Ray Casting/Tracing



- Write a basic ray caster
  - Orthographic and perspective cameras
  - Spheres and triangles
  - 2 Display modes: color and distance
- We provide classes for
  - Ray: origin, direction
  - Hit: t, Material, (*normal*)
  - Scene Parsing
- You write ray generation, hit testing, simple shading



Books

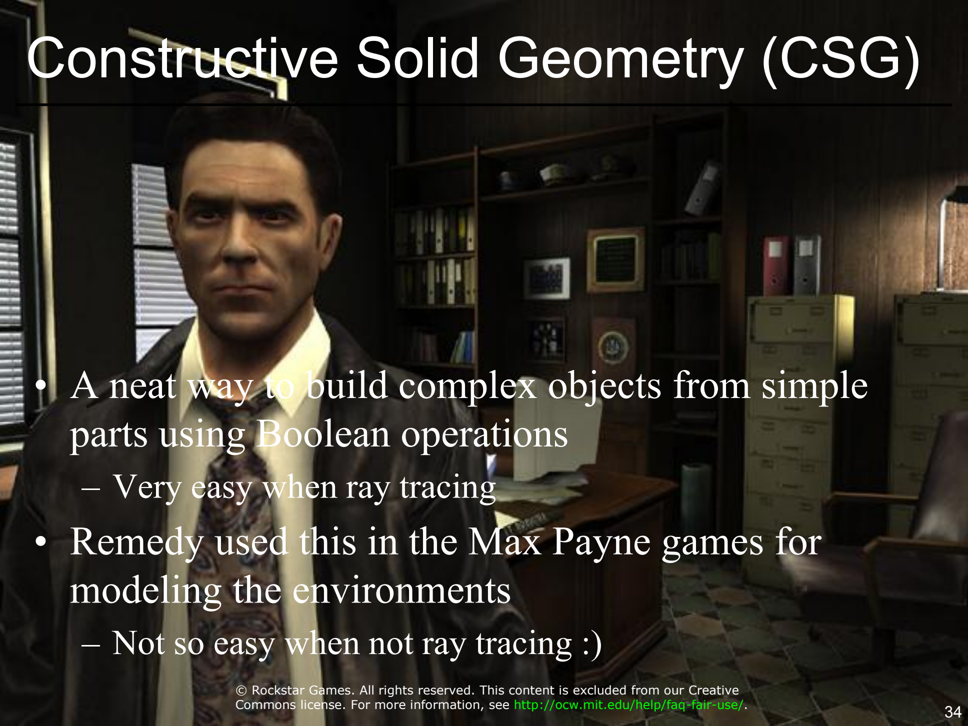
- Peter Shirley et al.:
  Fundamentals of Computer Graphics
  AK Peters

Remember the ones at books24x7 mentioned in the beginning!

- Ray Tracing
  - Jensen
  - Shirley
  - Glassner

Images of three book covers have been removed due to copyright restrictions. Please see the following books for more details:
-Shirley P., M. Ashikhmin and S. Marschner, *Fundamentals of Computer Graphics*
-Shirley P. and R.K. Morley, *Realistic Ray Tracing*
-Jensen H.W., *Realistic Image Synthesis Using Photon Mapping*

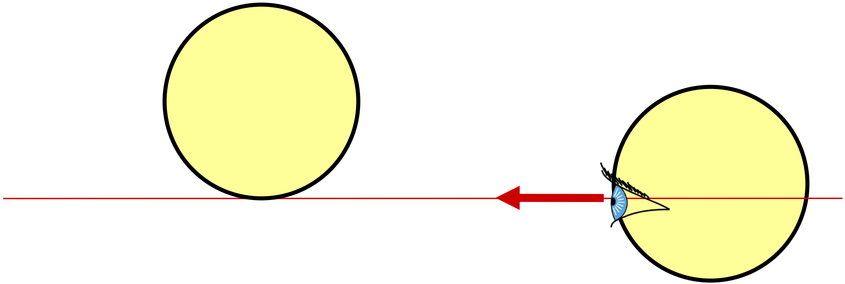
Constructive Solid Geometry (CSG)



- A neat way to build complex objects from simple parts using Boolean operations
  - Very easy when ray tracing
- Remedy used this in the Max Payne games for modeling the environments
  - Not so easy when not ray tracing :)

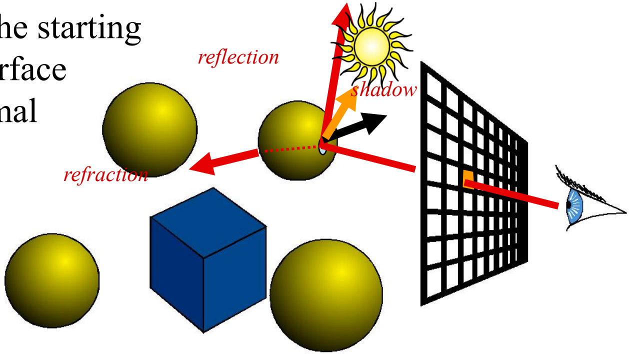
Precision

- What happens when
  - Ray Origin lies on an object?
  - Grazing rays?
- Problem with floating-point approximation



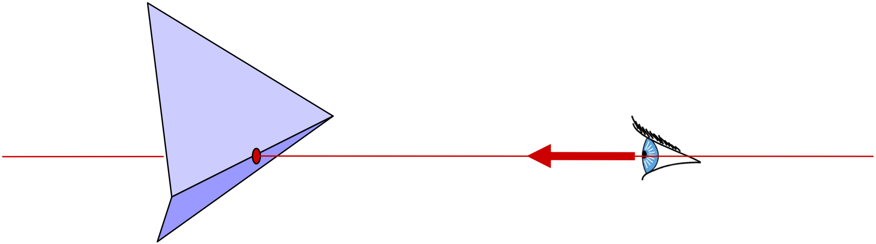
The Evil ε

- In ray tracing, do NOT report intersection for rays starting on surfaces
  - Secondary rays start on surfaces
  - Requires epsilons
  - Best to nudge the starting point off the surface e.g., along normal



The Evil ε

- Edges in triangle meshes
  - Must report intersection (otherwise not watertight)
  - Hard to get right



Questions?

Image by Henrik Wann Jensen

Transformations and Ray Casting

- We have seen that transformations such as affine transforms are useful for modeling & animation
- How do we incorporate them into ray casting?

Incorporating Transforms

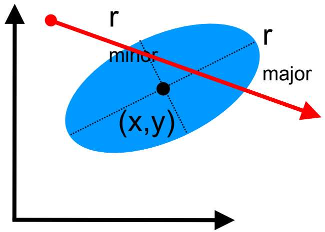
1. Make each primitive handle any applied transformations and produce a camera space description of its geometry

```
Transform {
    Translate { 1 0.5 0 }
    Scale { 2 2 2 }
    Sphere {
        center 0 0 0
        radius 1
    }
}
```

2. ...Or Transform the Rays

Primitives Handle Transforms

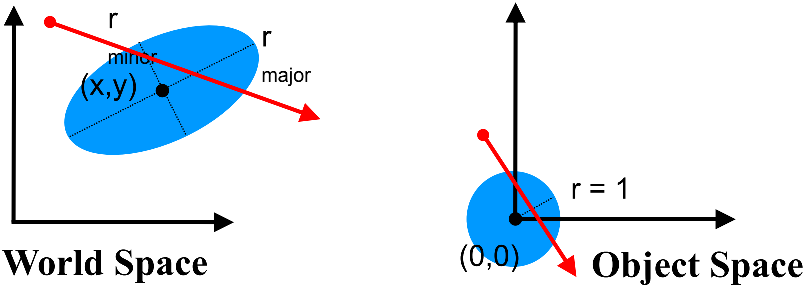
```
Sphere {
    center 3 2 0
    z_rotation 30
    r_major 2
    r_minor 1
}
```



- Complicated for many primitives

Transform Ray

- Move the ray from *World Space* to *Object Space*



$$p_{WS} = \mathbf{M} \; p_{OS}$$

$$p_{OS} = \mathbf{M}^{-1} \; p_{WS}$$

Transform Ray

- New origin:

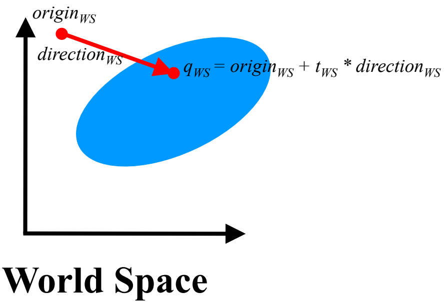
  $origin_{OS} = \mathbf{M}^{-1}\ origin_{WS}$

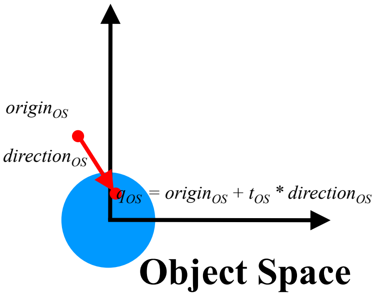
- New direction:

  $direction_{OS} = \mathbf{M}^{-1}\ (origin_{WS} + 1 * direction_{WS}) - \mathbf{M}^{-1}\ origin_{WS}$

  $direction_{OS} = \mathbf{M}^{-1}\ direction_{WS}$

Note that the w component of direction is 0





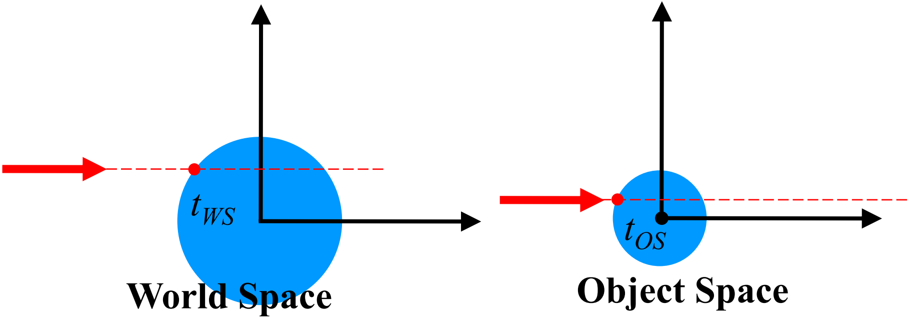
What About *t* ?

- If **M** includes scaling, $direction_{OS}$ ends up NOT be normalized after transformation

- Two solutions
  - Normalize the direction
  - Do not normalize the direction

1. Normalize Direction

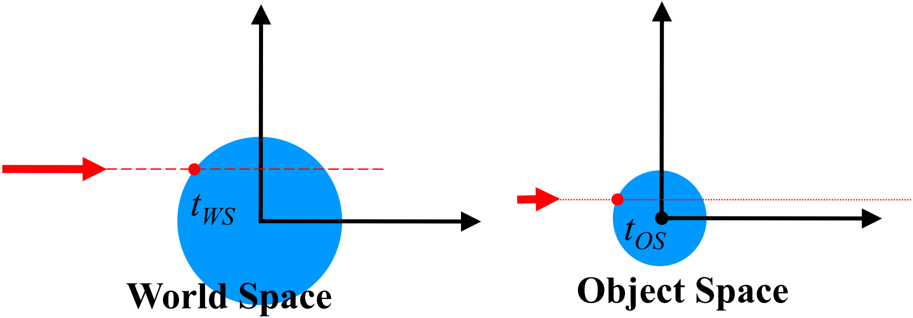
- $t_{OS} \neq t_{WS}$
  and must be rescaled after intersection
  ==> One more possible failure case...



2. Do Not Normalize Direction

Highly recommended

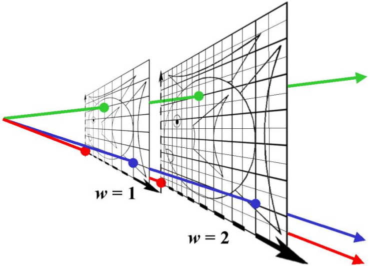
- $t_{OS} = t_{WS}$ → convenient!
- But you should not rely on t_{OS} being true distance in intersection routines (e.g. a≠1 in ray-sphere test)



Transforming Points & Directions

- Transform point

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} ax+by+cz+d \\ ex+fy+gz+h \\ ix+jy+kz+l \\ 1 \end{pmatrix}$$



- Transform direction

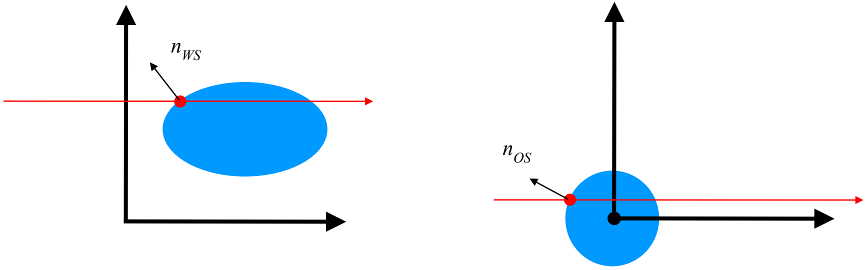
$$\begin{pmatrix} x' \\ y' \\ z' \\ 0 \end{pmatrix} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 0 \end{pmatrix} = \begin{pmatrix} ax+by+cz \\ ex+fy+gz \\ ix+jy+kz \\ 0 \end{pmatrix}$$

Homogeneous Coordinates: (x,y,z,w)

$w = 0$ is a point at infinity (direction)

- If you do not store w you need different routines to apply **M** to a point and to a direction ==> Store everything in 4D!

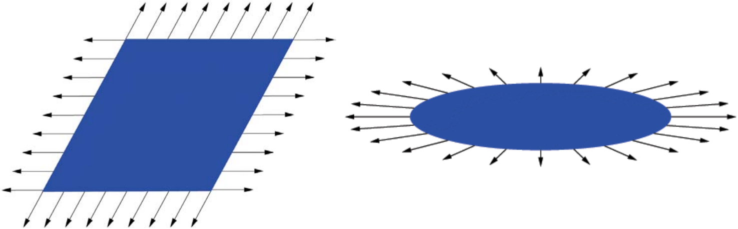
Recap: How to Transform Normals?



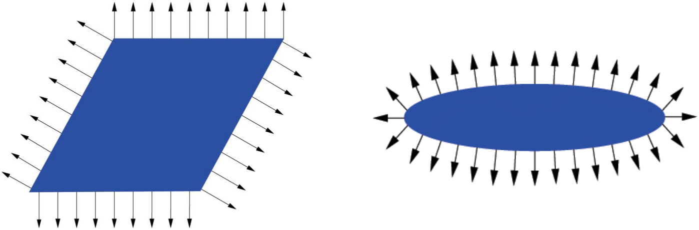
World Space **Object Space**

Transformation for Shear and Scale

Incorrect
Normal
Transformation

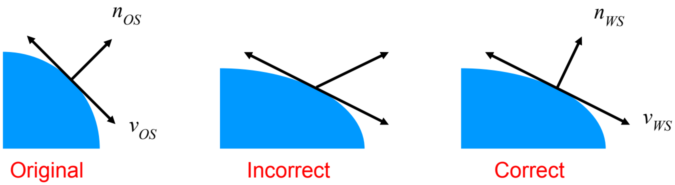


Correct
Normal
Transformation



So How Do We Do It Right?

- Think about transforming the *tangent plane* to the normal, not the normal *vector*



Pick any vector v_{OS} in the tangent plane, how is it transformed by matrix $\mathbf{M}$?

$$v_{WS} = \mathbf{M} \; v_{OS}$$

Transform Tangent Vector v

v is perpendicular to normal n:

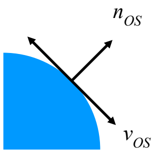
Dot product

$$n_{OS}^{\mathbf{T}}\ v_{OS} = 0$$

$$n_{OS}^{\mathbf{T}}\ (\mathbf{M}^{-1}\ \mathbf{M})\ v_{OS} = 0$$

$$(n_{OS}^{\mathbf{T}}\ \mathbf{M}^{-1})\ (\mathbf{M}\ v_{OS}) = 0$$

$$(n_{OS}^{\mathbf{T}}\ \mathbf{M}^{-1})\ v_{WS} = 0$$

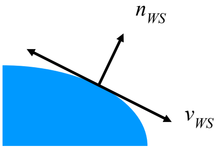


v_{WS} is perpendicular to normal n_{WS}:

$$n_{WS}^{\mathbf{T}}\ v_{WS} = 0$$

$$n_{WS}^{\mathbf{T}} = n_{OS}^{\mathbf{T}}\,(\mathbf{M}^{-1})$$

$$\boxed{n_{WS} = (\mathbf{M}^{-1})^{\mathbf{T}}\ n_{OS}}$$



Position, Direction, Normal

- Position
  - transformed by the full homogeneous matrix **M**
- Direction
  - transformed by **M** except the translation component
- Normal
  - transformed by $\mathbf{M}^{-T}$, no translation component

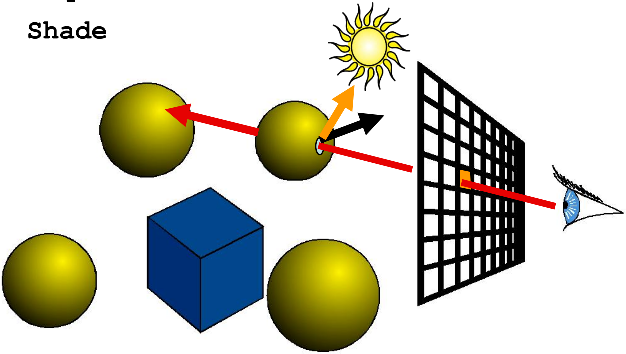
MIT EECS 6.837 Computer Graphics

Ray Tracing

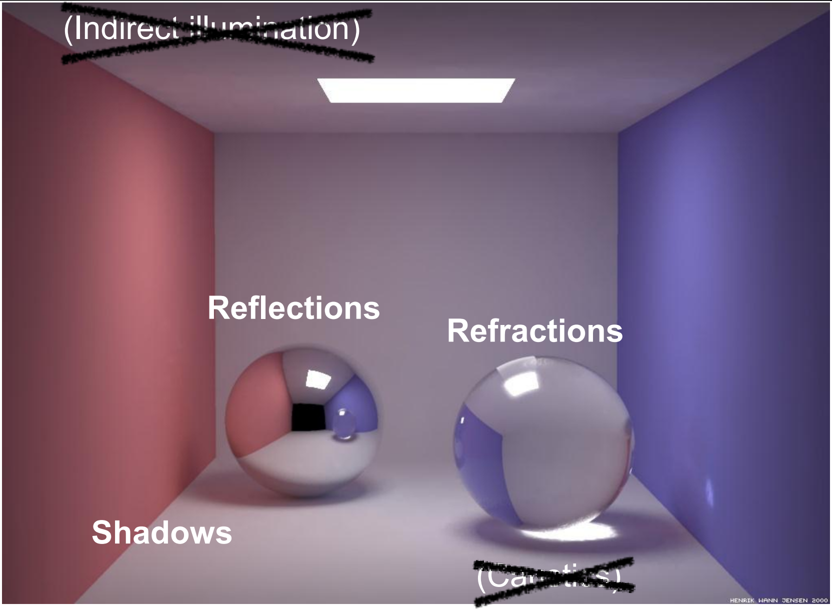
Wojciech Matusik, MIT EECS
Many slides from Jaakko Lehtinen and Fredo Durand

Ray Casting

```
For every pixel
   Construct a ray from the eye
   For every object in the scene
        Find intersection with the ray
        Keep if closest
         Shade
```



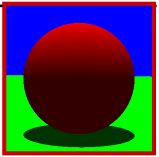
Today – Ray Tracing



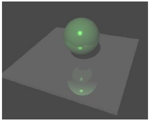
Henrik Wann Jensen

Overview of Today

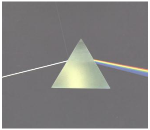
- Shadows



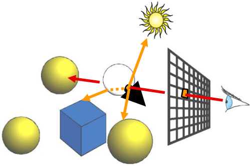
- Reflection



- Refraction

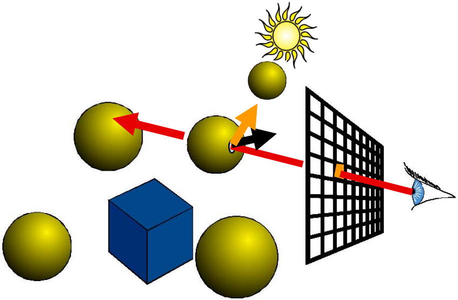


- Recursive Ray Tracing
  - "Hall of mirrors"



How Can We Add Shadows?

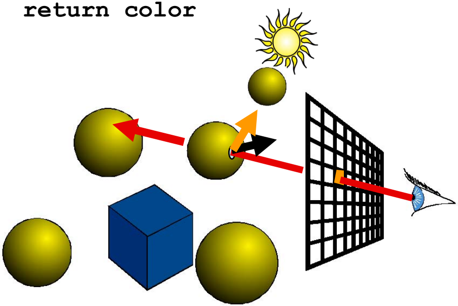
```
For every pixel
   Construct a ray from the eye
   For every object in the scene
       Find intersection with the ray
       Keep if closest
        Shade
```

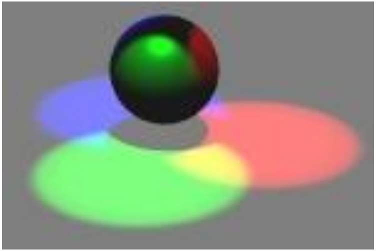


How Can We Add Shadows?

```
color = ambient*hit->getMaterial()->getDiffuseColor()
for every light
   Ray ray2(hitPoint, directionToLight)
   Hit hit2(distanceToLight, NULL, NULL)
   For every object
      object->intersect(ray2, hit2, 0)
   if (hit2->getT() = distanceToLight)
      color += hit->getMaterial()->Shade
                  (ray, hit, directionToLight, lightColor)
return color
```

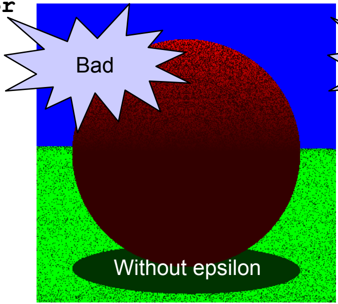
ambient = k_a
diffuseColor = k_d

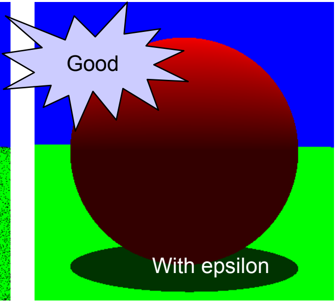




Problem: Self-Shadowing

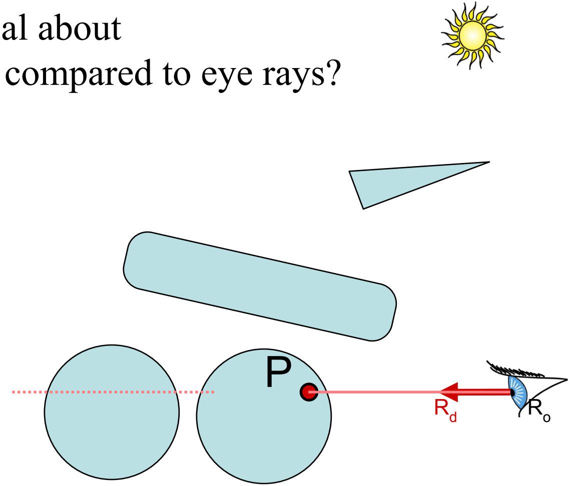
```
color = ambient*hit->getMaterial()->getDiffuseColor()
for every light
   Ray ray2(hitPoint, directionToLight)
   Hit hit2(distanceToLight, NULL, NULL)
   For every object
      object->intersect(ray2, hit2, epsilon)
   if (hit2->getT() = distanceToLight)
      color += hit->getMaterial()->Shade
                  (ray, hit, directionToLight, lightColor)
return color
```





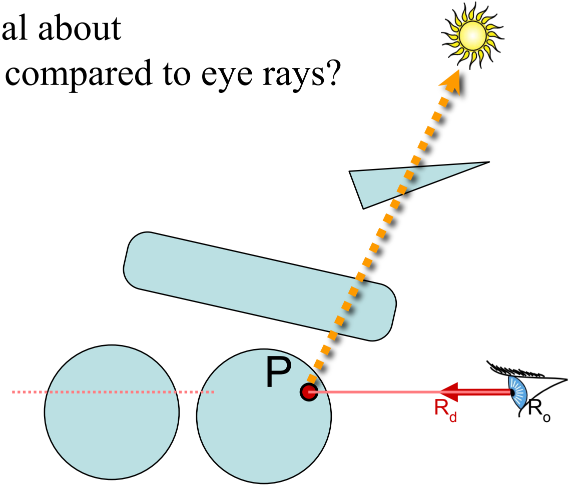
Let's Think About Shadow Rays

- What's special about shadow rays compared to eye rays?



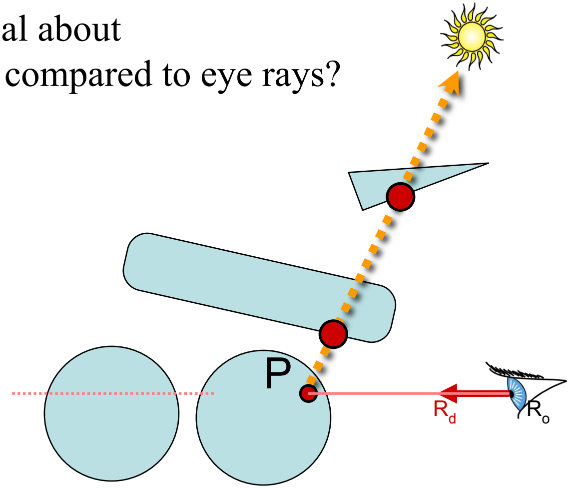
Let's Think About Shadow Rays

- What's special about shadow rays compared to eye rays?



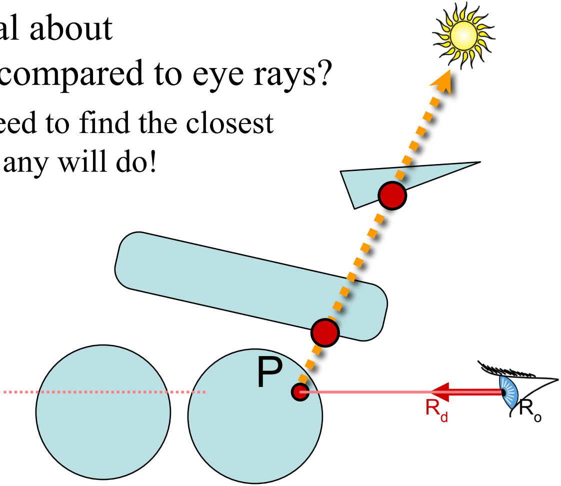
Let's Think About Shadow Rays

- What's special about shadow rays compared to eye rays?



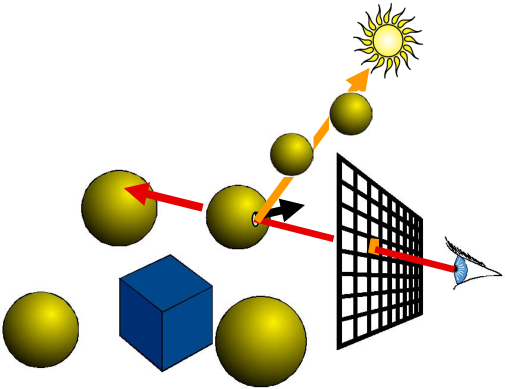
Let's Think About Shadow Rays

- What's special about shadow rays compared to eye rays?
  - We do not need to find the closest intersection, any will do!

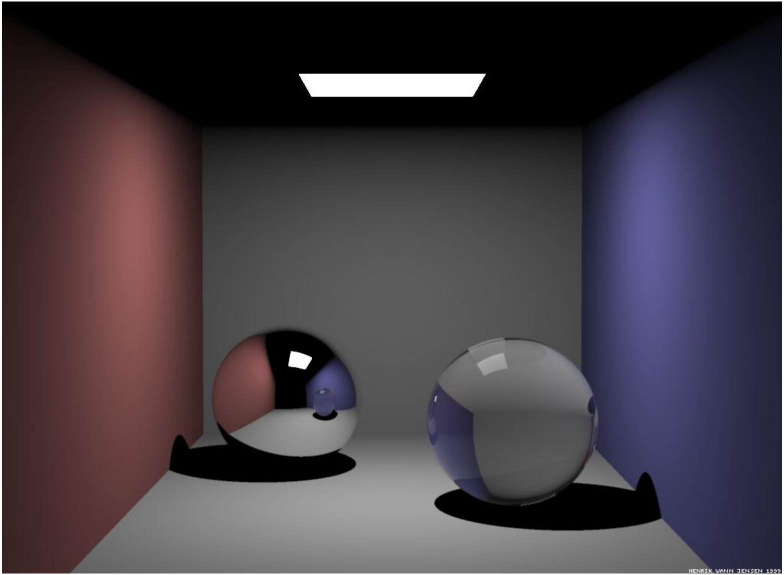


Shadow Optimization

- We only want to know whether there is an intersection, *not* which one is closest
- Special routine `Object3D::intersectShadowRay()`
  - Stops at first intersection



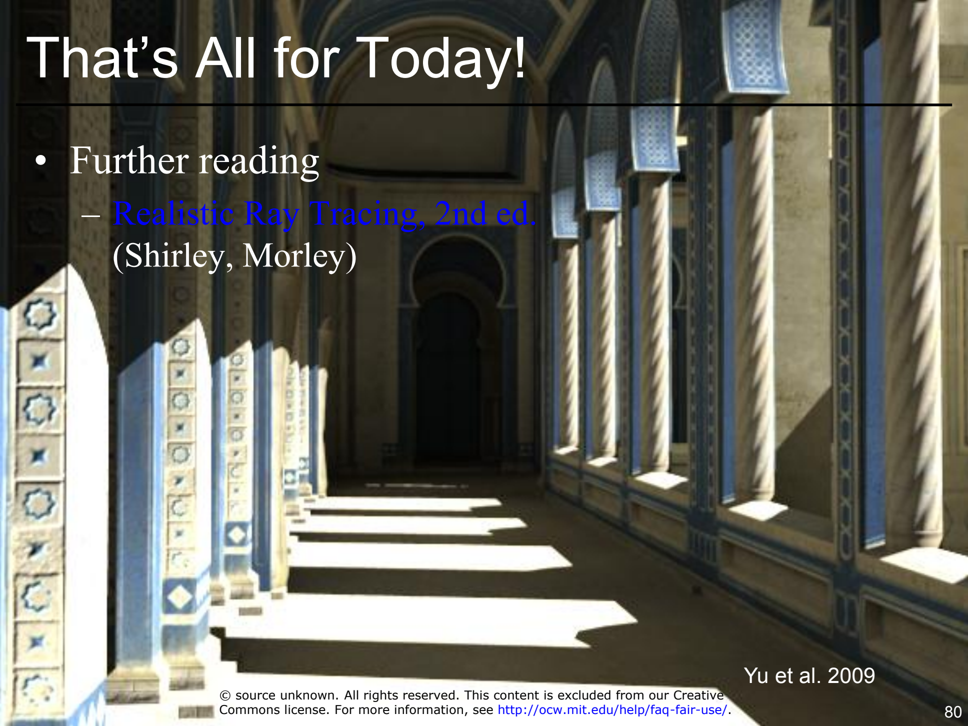
Questions?



Henrik Wann Jensen

That’s All for Today!

- Further reading
  - Realistic Ray Tracing, 2nd ed. (Shirley, Morley)



Yu et al. 2009