

TIEA311

Tietokonegrafiikan perusteet

kevät 2019

(“Principles of Computer Graphics” – Spring 2019)

Copyright and Fair Use Notice:

The lecture videos of this course are made available for registered students only. Please, do not redistribute them for other purposes. Use of auxiliary copyrighted material (academic papers, industrial standards, web pages, videos, and other materials) as a part of this lecture is intended to happen under academic “fair use” to illustrate key points of the subject matter. The lecturer may be contacted for take-down requests or other copyright concerns (email: paavo.j.nieminen@jyu.fi).

TIEA311 Tietokonegrafiikan perusteet – kevät 2019 ("Principles of Computer Graphics" – Spring 2019)

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

License: Creative Commons BY-NC-SA

Original license terms apply. Re-arrangement and new content copyright 2017-2019 by *Paavo Nieminen* and *Jarno Kansanaho*

Frontpage of the local course version, held during Spring 2019 at the Faculty of Information technology, University of Jyväskylä:

<http://users.jyu.fi/~nieminen/tgp19/>

TIEA311 - Today in Jyväskylä

Plan for today:

- ▶ Usual warm-up.
- ▶ Continue from yesterday
- ▶ Go through theory
- ▶ Remember to have a break!
- ▶ The teacher will try to remember and make use of the fact that we have groups of 2-3 students with pen and paper.

TIEA311 - Local plan for today

- ▶ Maybe some things I forgot to mention yesterday?
- ▶ Very brief recap of what went on previously.
- ▶ Then forward, with full speed!

MIT EECS 6.837 Computer Graphics

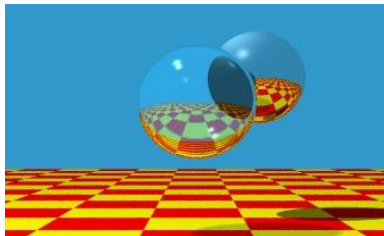
Part 2 – Rendering

Today: Intro to Rendering, Ray Casting



© NVIDIA Inc. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <http://www.mit.edu/help/faq-fair-use/>.

Ray Tracing



Reflections, refractions

© Turner Whitted, Bell Laboratories. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/help/faq-fair-use/>.

Reflections



Courtesy of Henrik Wann Jensen. Used with permission.



Caustics

HENRIK WANN JENSEN 2000

© source unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/help/faq-fair-use/>.

Ray Casting

For every pixel

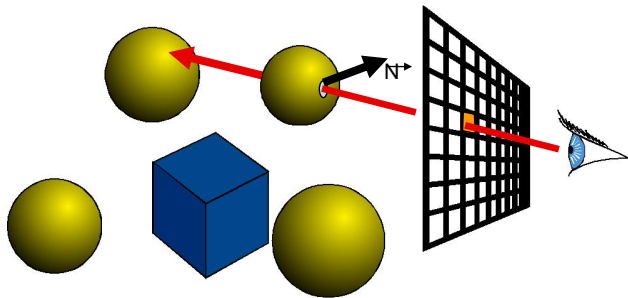
Construct a ray from the eye

For every object in the scene

Find intersection with the ray

Keep if closest

Shade depending on light and **normal** vector

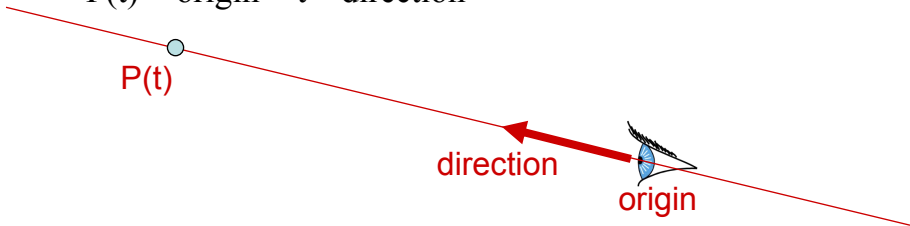


Finding the **intersection point** and **normal** is the central part of ray casting

Ray Representation

- Origin – Point
- Direction – Vector
 - normalized is better
- Parametric line
 - $P(t) = \text{origin} + t * \text{direction}$

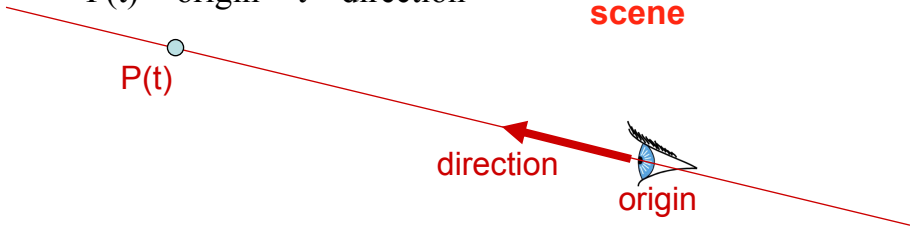
How would you represent a ray?



Ray Representation

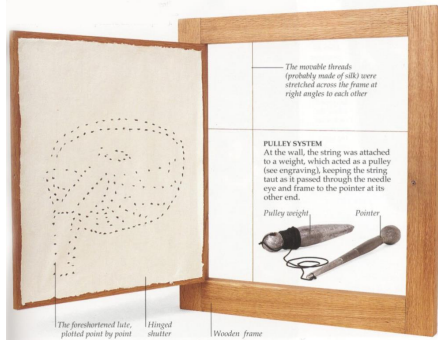
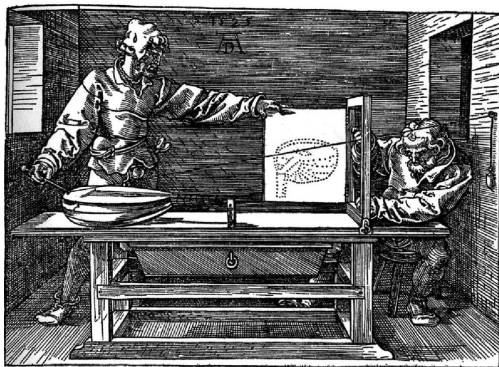
- Origin – Point
- Direction – Vector
 - normalized is better
- Parametric line
 - $P(t) = \text{origin} + t * \text{direction}$

Another way to put the ray casting problem statement:
Find smallest $t > 0$ such that $P(t)$ lies on a surface in the scene



Dürer's Ray Casting Machine

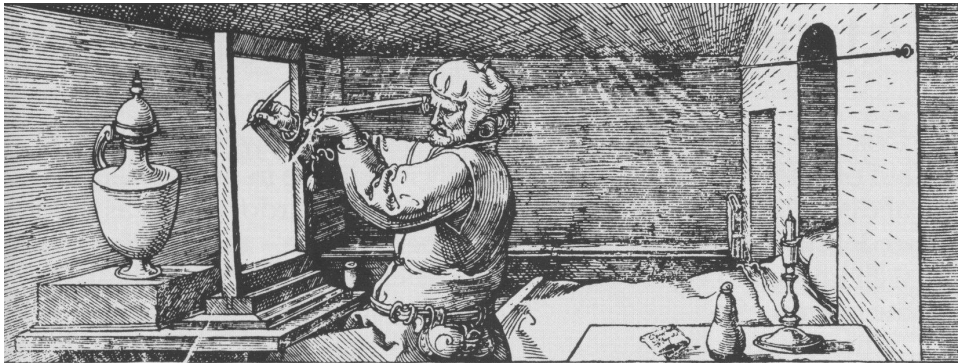
- **Albrecht Dürer**, 16th century



© source unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/help/faq-fair-use/>.

Dürer's Ray Casting Machine

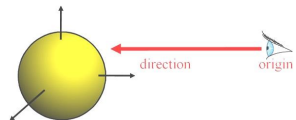
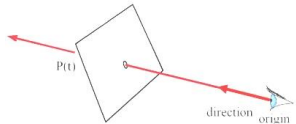
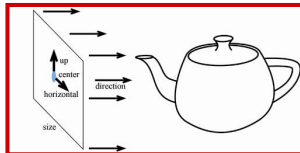
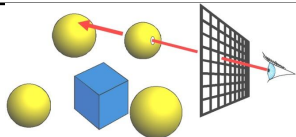
- Albrecht Dürer, 16th century



© source unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/help/faq-fair-use/>.

Ray Casting

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



Cameras

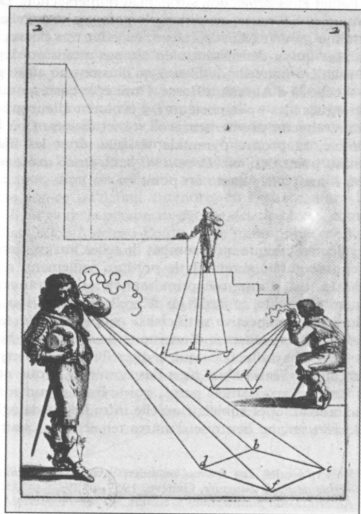
For every pixel

Construct a ray from the eye

For every object in the scene

Find intersection with ray

Keep if closest

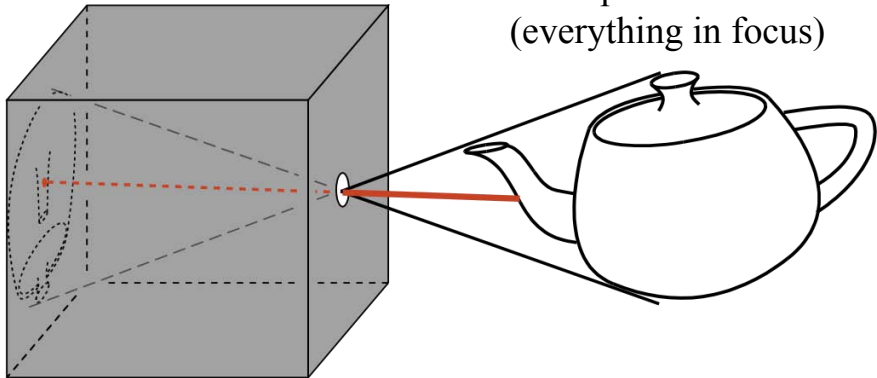


Abraham Bosse, *Les Perspectiveurs*. Gravure extraite de la *Manière*

© source unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/help/faq-fair-use/>.

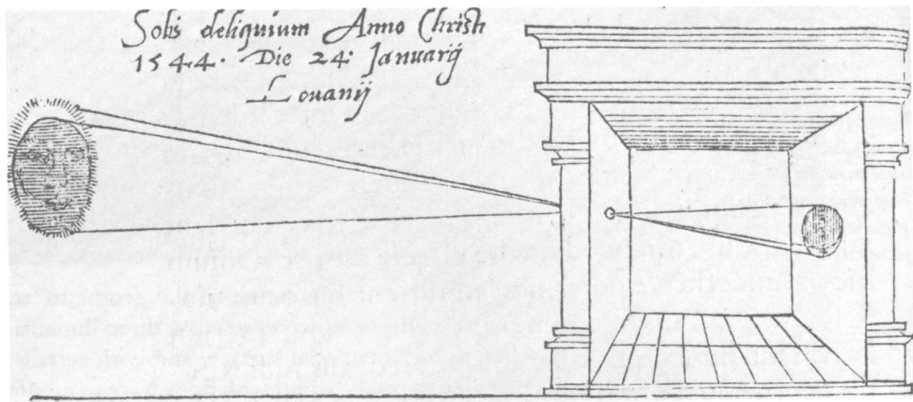
Pinhole Camera

- Box with a tiny hole
- Inverted image
- Similar triangles
- Perfect image if hole infinitely small
- Pure geometric optics
- No depth of field issue (everything in focus)



Oldest Illustration

- From Gemma Frisius, 1545



© source unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/help/faq-fair-use/>.

Also Called “Camera Obscura”

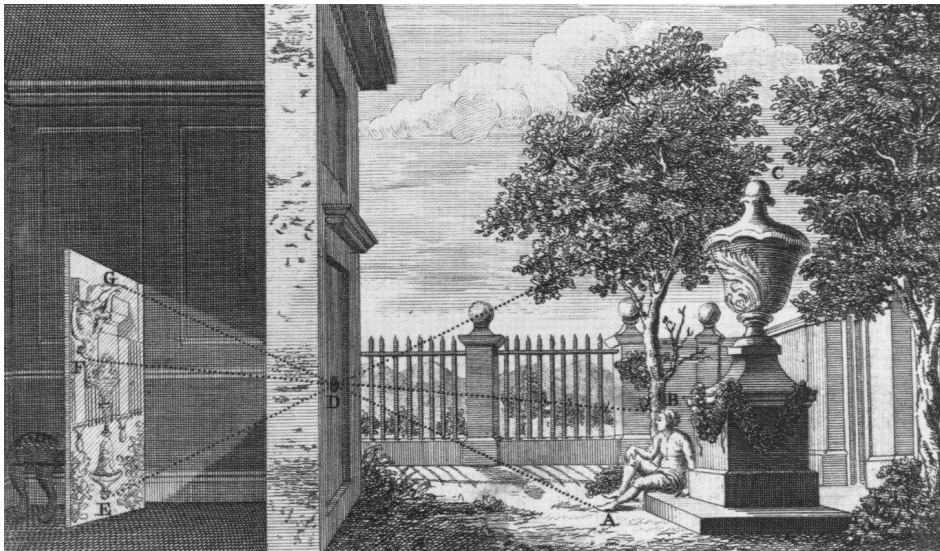


Image courtesy of Wellcome Library, London. License: CC-BY-NC. This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/help/faq-fair-use/>.

Camera Obscura Today

Images removed due to copyright restrictions -- please see
http://www.abelardomorell.net/photography/cameraobsc_01/cameraobsc_17.html
<http://www.abelardomorell.net/posts/camera-obscura/>
http://www.abelardomorell.net/photography/cameraobsc_49/cameraobsc_63.html
for further details.

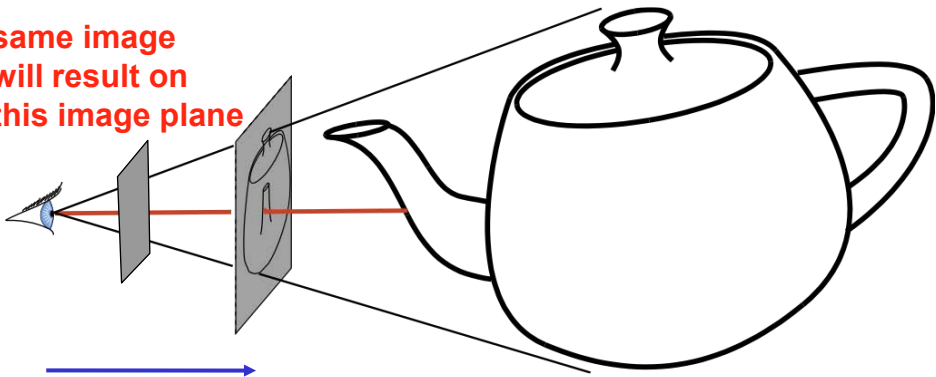
Abelardo Morell

`www.abelardomorell.net`

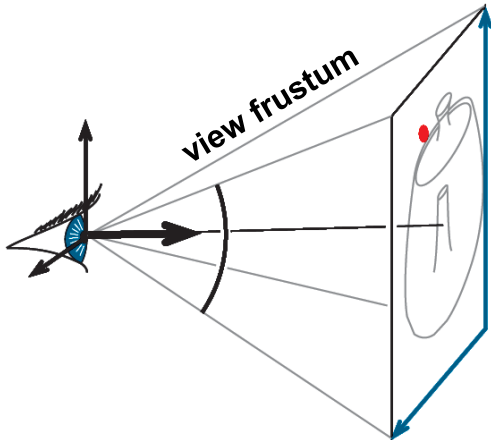
Simplified Pinhole Camera

- Eye-image pyramid (view frustum)
- Note that the distance/size of image are arbitrary

same image
will result on
this image plane



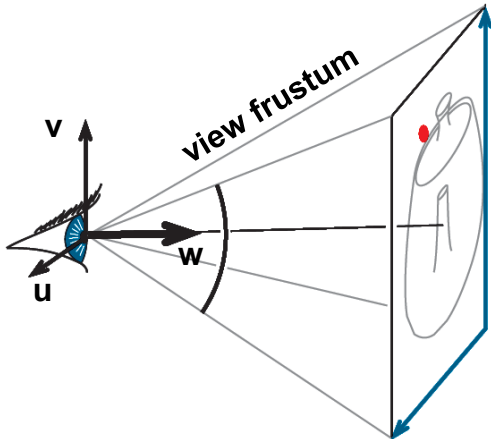
Camera Description?



Camera Description?

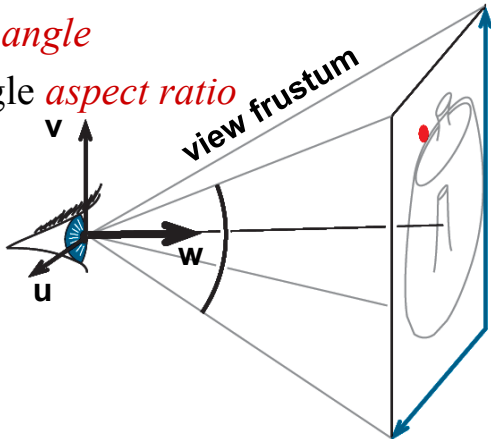
- Eye point e (*center*)
- Orthobasis u, v, w (*horizontal, up, direction*)

Object
coordinates
World
coordinates
**View
coordinates**
Image
coordinates



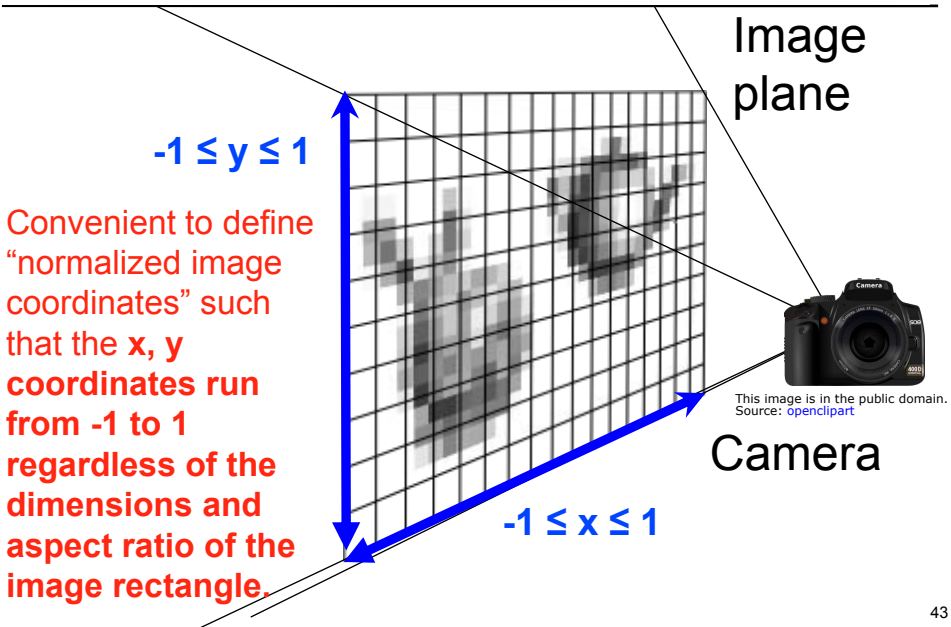
Camera Description?

- Eye point e (*center*)
- Orthobasis u, v, w (*horizontal, up, direction*)
- Field of view *angle*
- Image rectangle *aspect ratio*

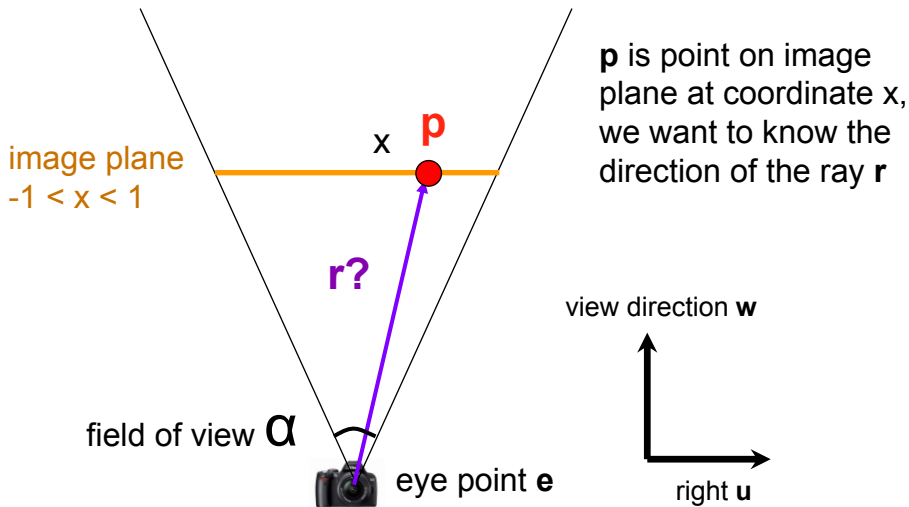


Object
coordinates
World
coordinates
View
coordinates
Image
coordinates

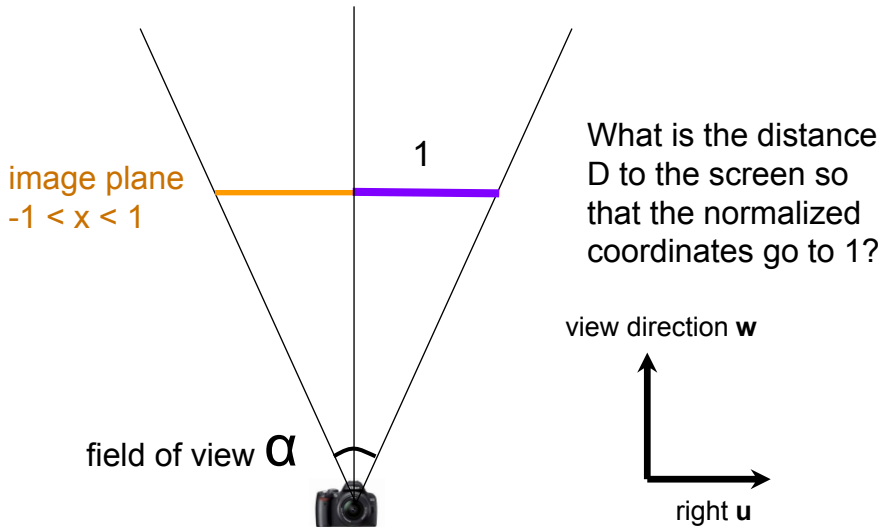
Image Coordinates



Ray Generation in 2D



Ray Generation in 2D



This image is in the public domain. Source: [openclipart](#)

Corrections

CORRECTION: In the following few slides, the ideas are brilliantly visualized, but some of the equations are rubbish. These are OK:

$$\tan(\alpha/2) = 1/D \implies D = 1/\tan(\alpha/2)$$

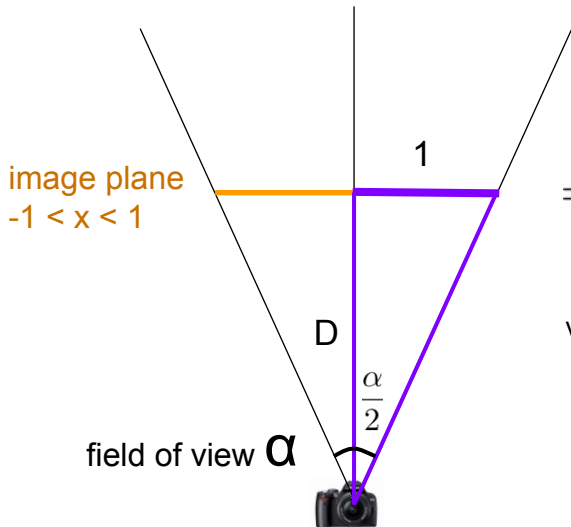
The others should read as:

$$\mathbf{r} = \mathbf{p} - \mathbf{e} = x\mathbf{u} + D\mathbf{w}$$

And (for the 3D case):

$$\mathbf{r} = x\mathbf{u} + \text{aspect} \cdot y\mathbf{v} + D\mathbf{w}$$

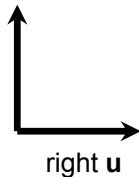
Ray Generation in 2D



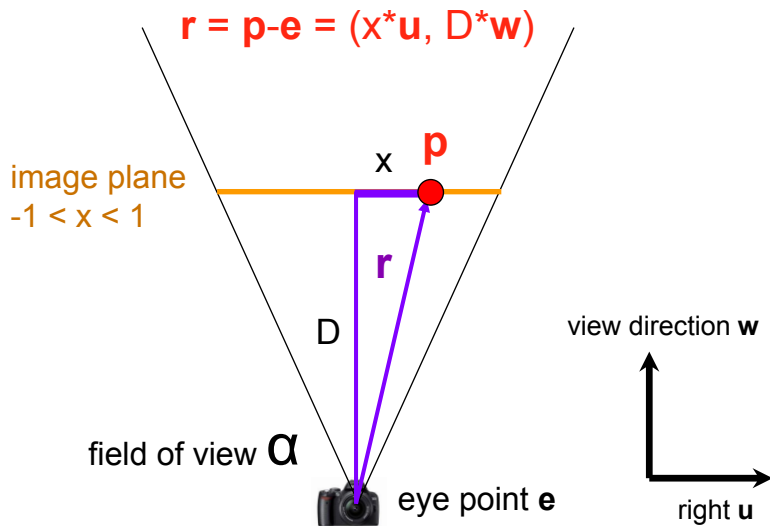
$$\tan \frac{\alpha}{2} = \frac{1}{D}$$

$$\Rightarrow D = \frac{1}{\tan(\alpha/2)}$$

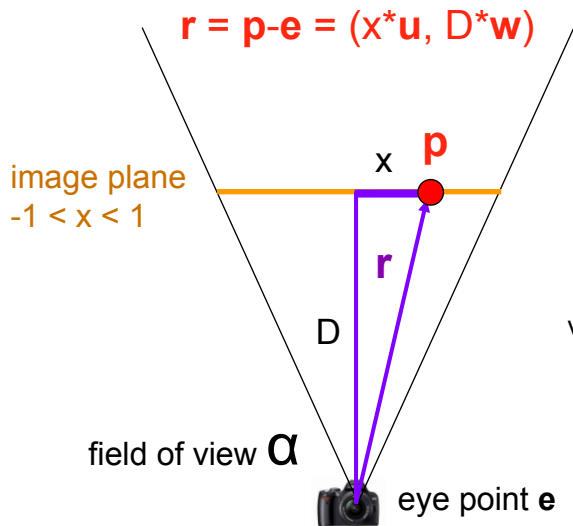
view direction \mathbf{w}



Ray Generation in 2D



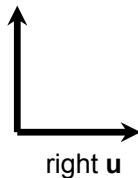
Ray Generation in 2D



then we just
normalize \mathbf{r} to get
the ray

$$\mathbf{P}(t) = \mathbf{e} + t\mathbf{d}$$
$$(\mathbf{d} = \mathbf{r} / \|\mathbf{r}\|)$$

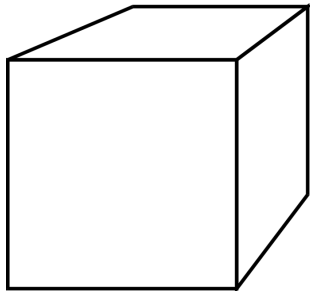
view direction \mathbf{w}



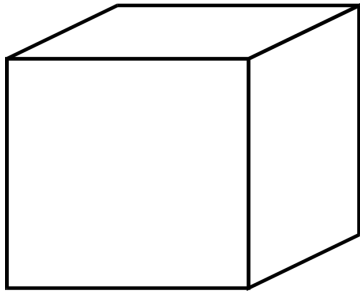
That was 2D, 3D is just as simple

- y coordinate is treated just like x , except accounting for aspect ratio
 - $\mathbf{r} = (x*\mathbf{u}, \text{aspect}*y*\mathbf{v}, D*\mathbf{w})$
 - Again, \mathbf{u} , \mathbf{v} , \mathbf{w} are the basis vectors of the view coordinate system
 - Aspect ratio handles non-square viewports
 - Think of your 16:9 widescreen TV
- The point of the exercise with computing D was to allow us to use the $[-1,1]$ image coordinate system regardless of field of view.

Perspective vs. Orthographic



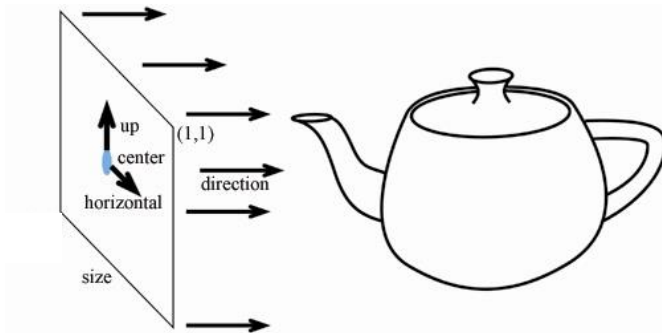
perspective



orthographic

- Parallel projection
- No foreshortening
- No vanishing point

Orthographic Camera



- Ray Generation?

- Origin = $\mathbf{e} + x \cdot \text{size} \cdot \mathbf{u} + y \cdot \text{size} \cdot \mathbf{v}$
- Direction is constant: \mathbf{w}

Other Weird Cameras

- E.g. fish eye, omnimax, parabolic



CAVE Columbia University

© CAVE Lab, Columbia University. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/help/faq-fair-use/>.

© source unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/help/faq-fair-use/>.

Questions?



Even Funkier
Multiperspective
Imaging

Courtesy of Paul Rademacher. Used with permission.

Plan

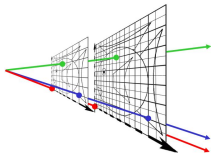
- Vectors



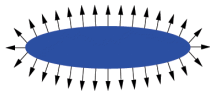
- Points



- **Homogenous coordinates**



- Normals



Homogeneous Coordinates

- Add an extra dimension (same as frames)
 - in 2D, we use 3-vectors and 3 x 3 matrices
 - In 3D, we use 4-vectors and 4 x 4 matrices
- The extra coordinate is now an **arbitrary** value, w
 - You can think of it as “scale,” or “weight”
 - For all transformations

except perspective, you can just set $w=1$ and not worry about it

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Projective Equivalence

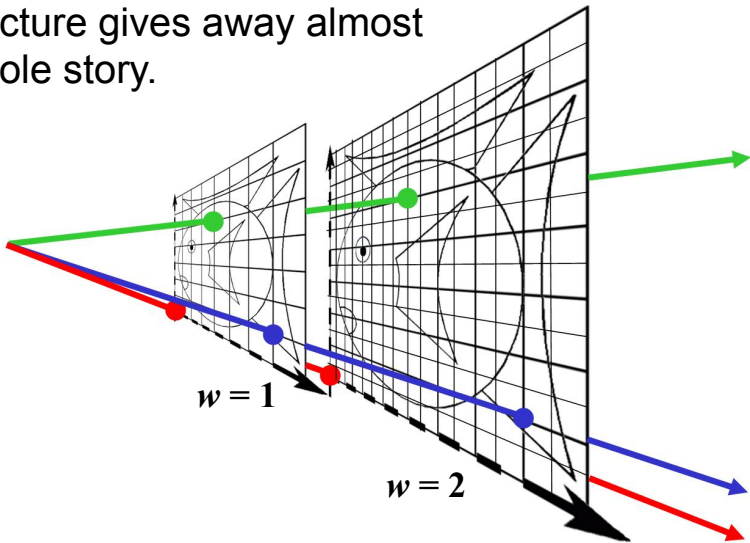
- All non-zero scalar multiples of a point are considered identical
- to get the equivalent Euclidean point, divide by w

$$\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} ax \\ ay \\ az \\ aw \end{pmatrix} \stackrel{w \neq 0}{=} \begin{pmatrix} x/w \\ y/w \\ z/w \\ 1 \end{pmatrix}$$

$$a \neq 0$$

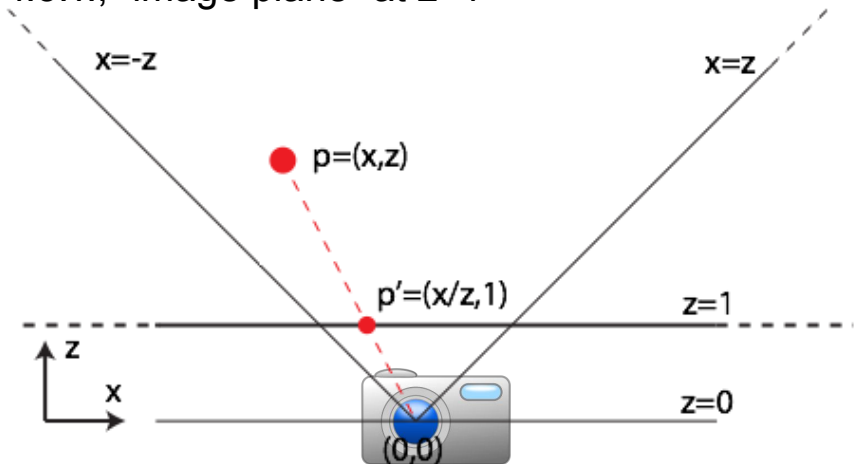
Why bother with extra coord?

- This picture gives away almost the whole story.



Perspective in 2D

- Camera at origin, looking along z, 90 degree f.o.v., “image plane” at $z=1$

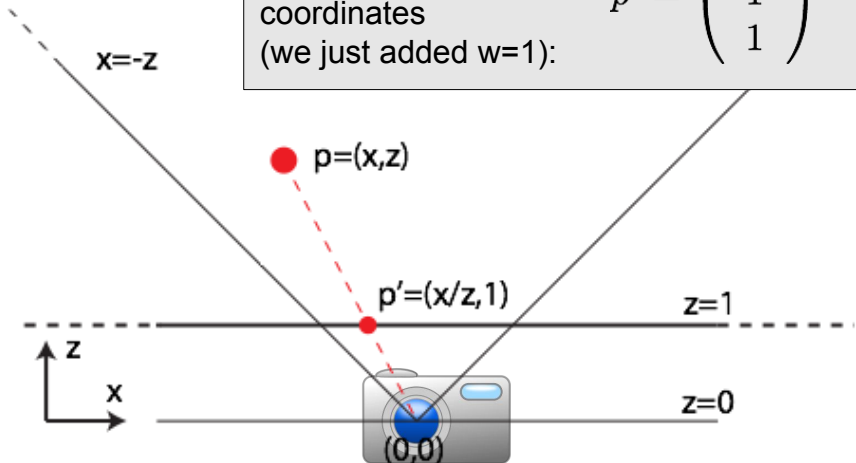


This image is in the public domain.
Source: <http://openclipart.org/detail/34051/digicam-by-thesaurus>.

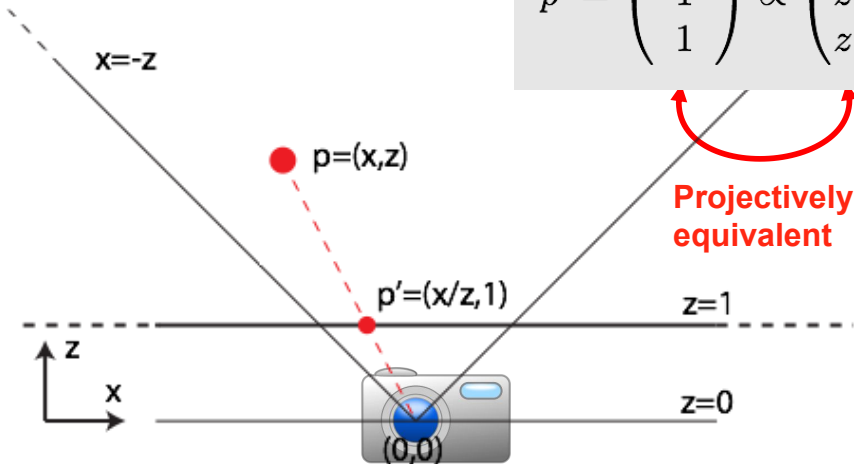
Perspective in 2D

The projected point in homogeneous coordinates (we just added $w=1$):

$$p' = \begin{pmatrix} x/z \\ 1 \\ 1 \end{pmatrix}$$



Perspective in 2D

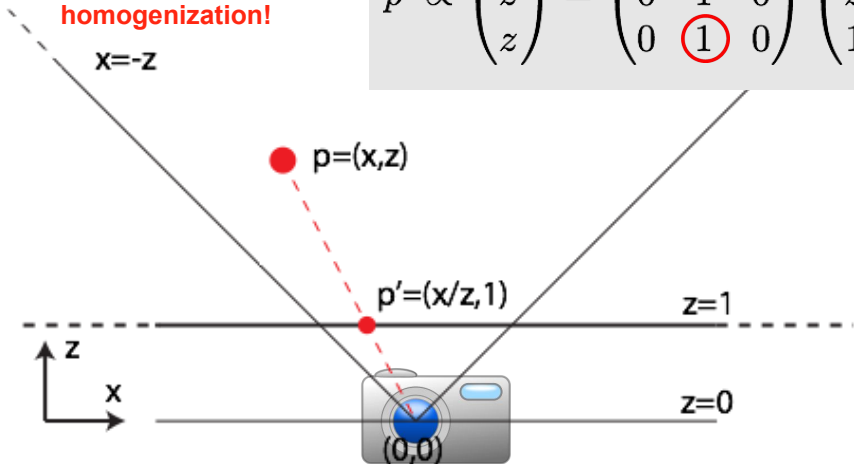


This image is in the public domain.
Source: <http://openclipart.org/detail/34051/digicam-by-thesaurus>.

Perspective in 2D

We'll just copy z to w ,
and get the projected
point after
homogenization!

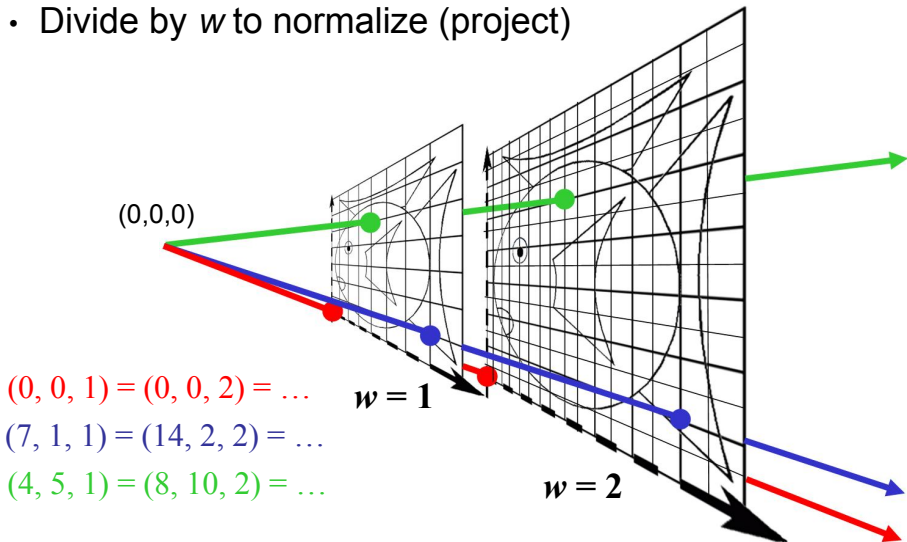
$$p' \propto \begin{pmatrix} x \\ z \\ z \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ z \\ 1 \end{pmatrix}$$



This image is in the public domain.
Source: <http://openciptart.org/detail/34051/digicam-by-thesaurus>.

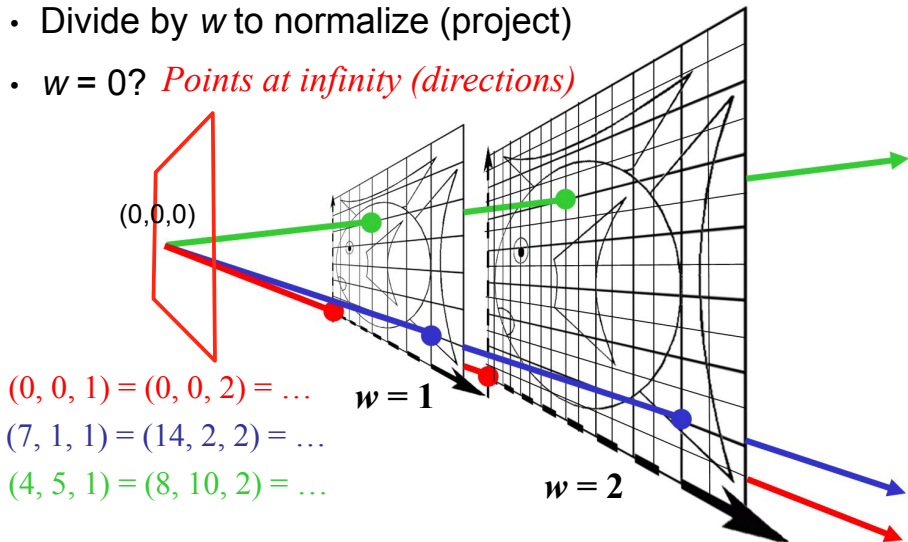
Homogeneous Visualization

- Divide by w to normalize (project)



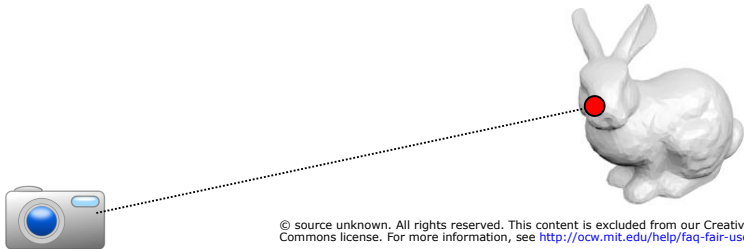
Homogeneous Visualization

- Divide by w to normalize (project)
- $w = 0$? *Points at infinity (directions)*



Projective Equivalence – Why?

- For affine transformations, adding $w=1$ in the end proved to be convenient.
- The real showpiece is **perspective**.



This image is in the public domain.
Source: <http://openclipart.org/detail/34051/digicam-by-thesaurus>.

© source unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/help/faq-fair-use/>.

Questions?

Eye candy: photo tourism

- Application of homogenous coordinates
- Goal: given N photos of a scene
 - find where they were taken
 - get 3D geometry for points in the scene

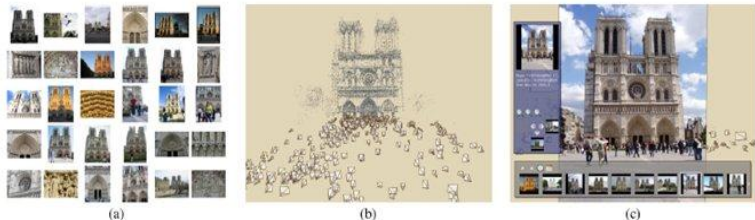


Figure 1: Our system takes unstructured collections of photographs such as those from online image searches (a) and reconstructs 3D points and viewpoints (b) to enable novel ways of browsing the photos (c).

©ACM. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/help/faq-fair-use/>.

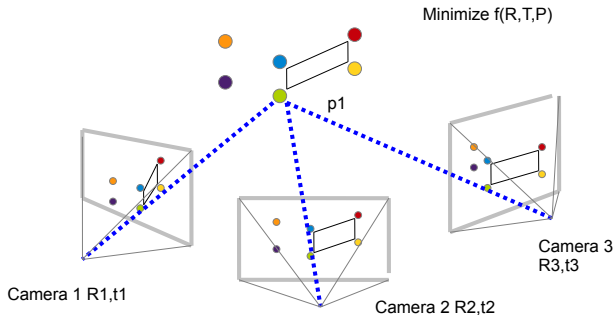
From *Photo Tourism: Exploring Photo Collections in 3D*, used with permission from ACM, Inc.

Step 1: point correspondences

- Extract salient points (corners) from images
- Find the same scene point in other images
- To learn how it's done, take 6.815

Structure from motion

- Given point correspondences
- Unknowns: 3D point location, camera poses
- For each point in each image, write perspective equations



Eye candy: photo tourism

Photo Tourism

Exploring photo collections in 3D

Noah Snavely Steven M. Seitz Richard Szeliski
University of Washington *Microsoft Research*

SIGGRAPH 2006

TIEA311

Tietokonegrafiikan perusteet

kevät 2019

(“Principles of Computer Graphics” – Spring 2019)

Copyright and Fair Use Notice:

The lecture videos of this course are made available for registered students only. Please, do not redistribute them for other purposes. Use of auxiliary copyrighted material (academic papers, industrial standards, web pages, videos, and other materials) as a part of this lecture is intended to happen under academic “fair use” to illustrate key points of the subject matter. The lecturer may be contacted for take-down requests or other copyright concerns (email: paavo.j.nieminen@jyu.fi).

TIEA311 Tietokonegrafiikan perusteet – kevät 2019 ("Principles of Computer Graphics" – Spring 2019)

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

License: Creative Commons BY-NC-SA

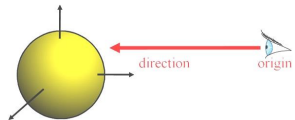
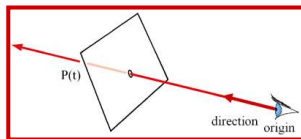
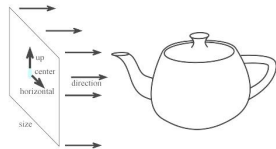
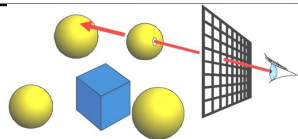
Original license terms apply. Re-arrangement and new content copyright 2017-2019 by *Paavo Nieminen* and *Jarno Kansanaho*

Frontpage of the local course version, held during Spring 2019 at the Faculty of Information technology, University of Jyväskylä:

<http://users.jyu.fi/~nieminen/tgp19/>

Ray Casting

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



Ray Casting

For every pixel

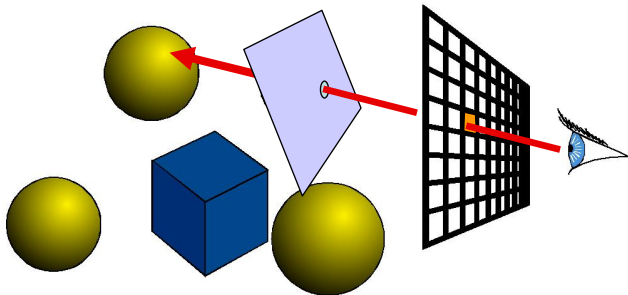
Construct a ray from the eye

For every object in the scene

Find intersection with the ray

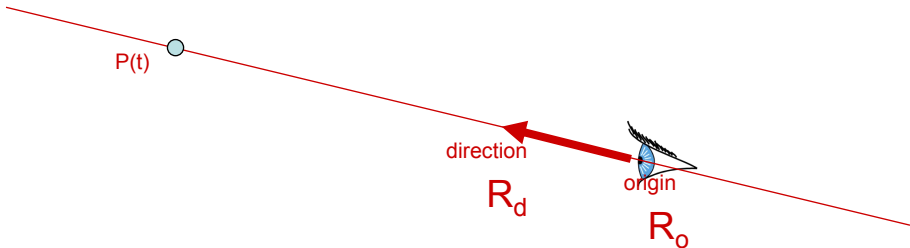
Keep if closest

First we will study ray-plane intersection



Recall: Ray Representation

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

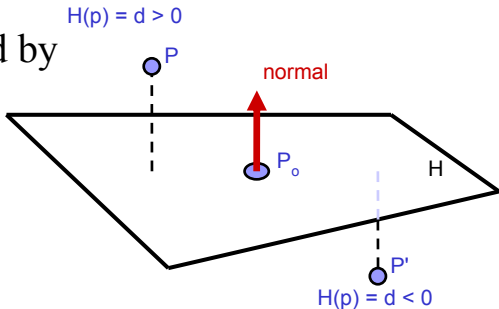


3D Plane Representation?

- (Infinite) plane defined by

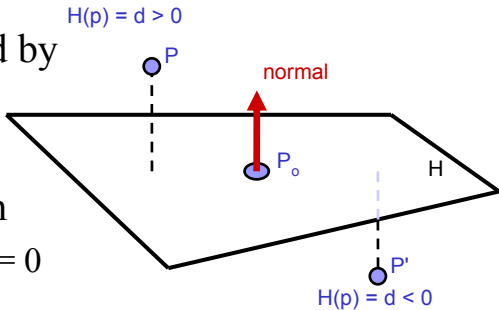
- $P_o = (x_0, y_0, z_0)$

- $n = (A, B, C)$



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

- $P_0 = (x_0, y_0, z_0)$

- $n = (A, B, C)$

- Implicit plane equation

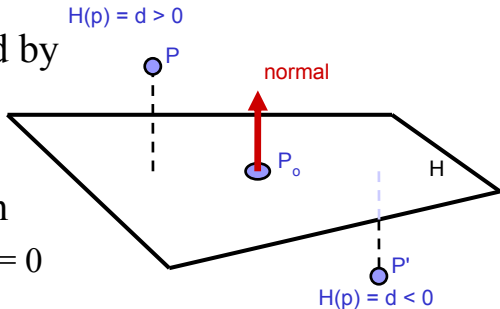
- $H(P) = Ax + By + Cz + D = 0$

- $= n \cdot P + D = 0$

- 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

- $P_o = (x_0, y_0, z_0)$

- $n = (A, B, C)$

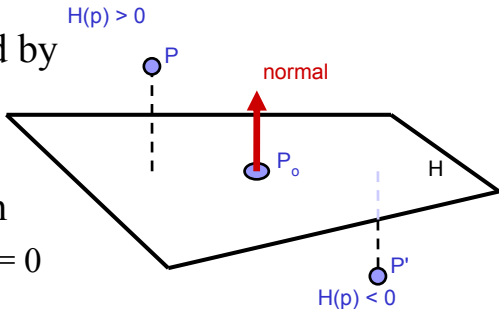
- Implicit plane equation

- $H(P) = Ax + By + Cz + D = 0$

- $= n \cdot P + D = 0$

- Point-Plane distance?

- If n is normalized,
distance to plane is $H(P)$
 - it is a *signed distance*!

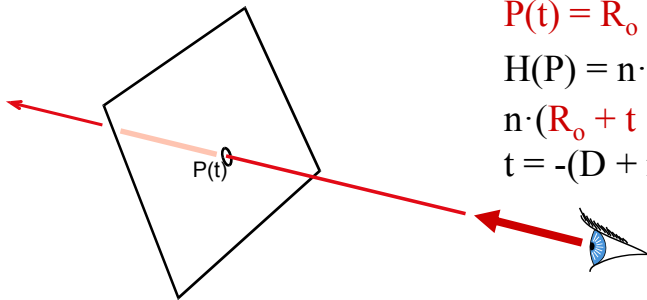


Explicit vs. Implicit?

- 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?

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

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

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

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

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

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

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

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

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

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

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

$$t(\vec{n} \cdot \vec{R}_d)(\vec{n} \cdot \vec{R}_d)^{-1} = (-D - \vec{n} \cdot \vec{R}_o)(\vec{n} \cdot \vec{R}_d)^{-1}$$

$$t * 1 = (-D - \vec{n} \cdot \vec{R}_o)(\vec{n} \cdot \vec{R}_d)^{-1}$$

$$t = -(D + \vec{n} \cdot \vec{R}_o)(\vec{n} \cdot \vec{R}_d)^{-1}$$

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

Start with equation. Do stuff that keeps both sides equal, towards leaving only t on the left side.

Added $-D$ to both sides. Different but equal.

Regroup (real sums are associative)

Sum of additive inverses yields zero (definition of "minus": $D - D = D + (-D) = 0$)

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.

Dot product is distributive over vector addition

Add $-\vec{n} \cdot \vec{R}_o$ (additive inverse, like $-D$ above) to both sides. Middle OK since sum is commutative.

Regroup (associativity again)

Sum of additive inverses (again)

Rid of zero (additive identity)

Scalar multiplication property of dot product

Multiply both sides by multiplicative inverse ("divide"). **Such inverse is not defined for 0** though!

multiplication by inverse yields multiplicative identity 1; multiplication denoted $*$ for clarity

Rid of 1 (multiplicative identity). Distributive and associative properties used on right to fit slide.

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

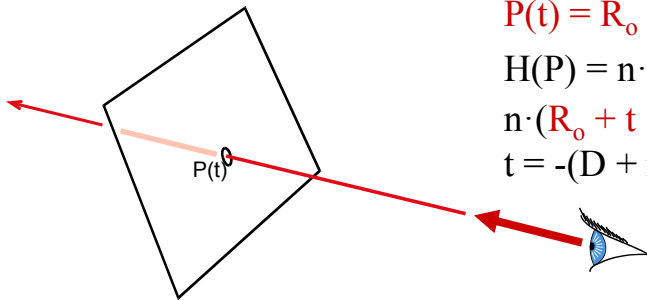
$$t = -\frac{D + \vec{n} \cdot \vec{R}_o}{\vec{n} \cdot \vec{R}_d}$$

“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



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

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

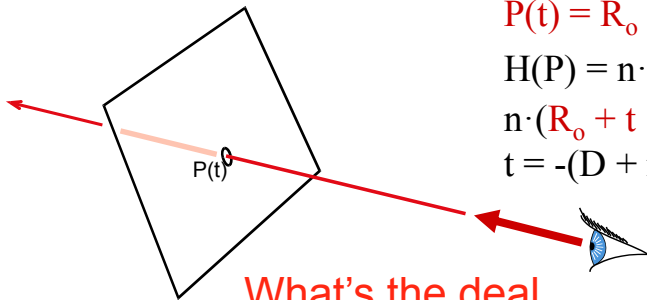
$$n \cdot (R_o + t * R_d) + D = 0$$

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

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 $n \cdot R_d = 0$?

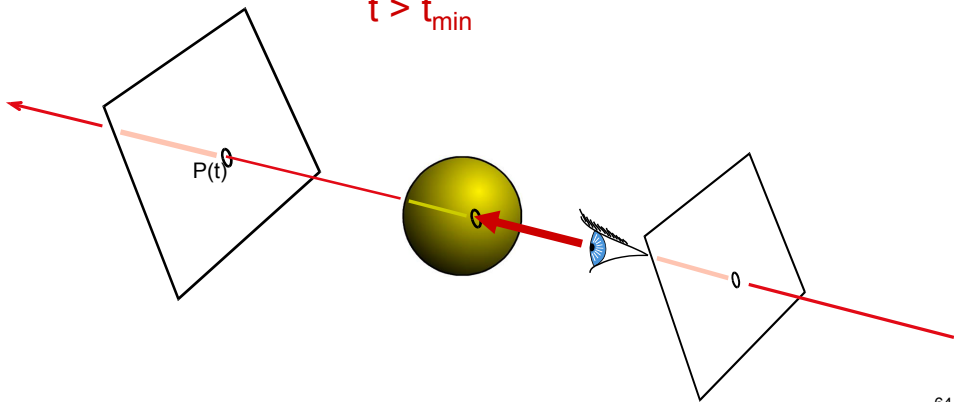
Additional Bookkeeping

- Verify that intersection is closer than previous

$$t < t_{\text{current}}$$

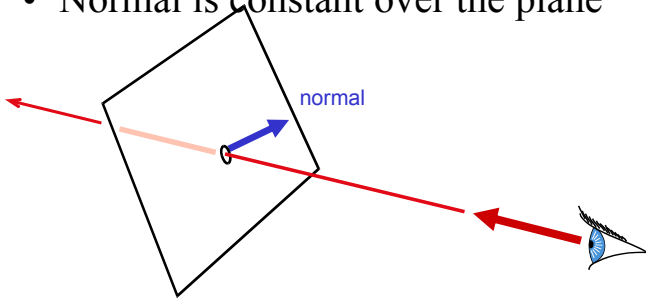
- Verify that it is not out of range (behind eye)

$$t > t_{\text{min}}$$



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?



RENDERED USING OPRT - HENRIK WANN JENSEN 2009

Courtesy of Henrik Wann Jensen. Used with permission.

Image by Henrik Wann Jensen

TIEA311

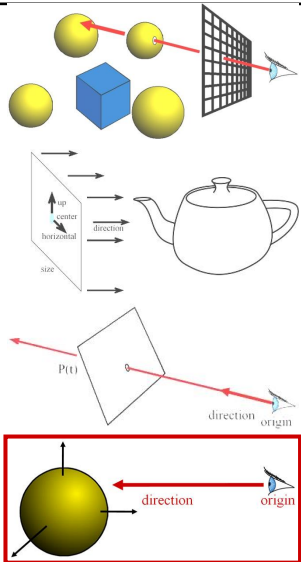
The following slides were not shown on the lecture (yet).

They are a **preview** of what we will talk about next, very soon.

Questions based on your preview will be **much appreciated** when we meet on the next lecture!

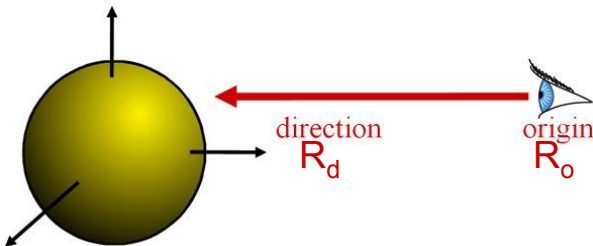
Ray Casting

- 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)
 - $H(P) = \|P\|^2 - r^2 = P \cdot P - r^2 = 0$



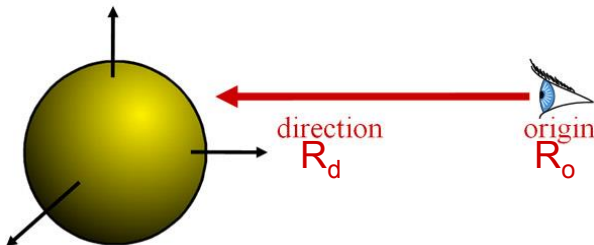
Ray-Sphere Intersection

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

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

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

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



Ray-Sphere Intersection

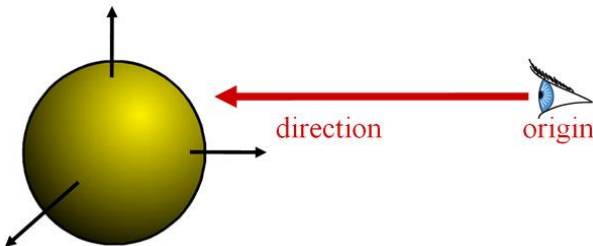
- Quadratic: $at^2 + bt + c = 0$
 - $a = 1$ (remember, $\|\mathbf{R}_d\| = 1$)
 - $b = 2\mathbf{R}_d \cdot \mathbf{R}_o$
 - $c = \mathbf{R}_o \cdot \mathbf{R}_o - r^2$

- with discriminant $d = \sqrt{b^2 - 4ac}$

- and solutions $t_{\pm} = \frac{-b \pm d}{2a}$

Ray-Sphere Intersection

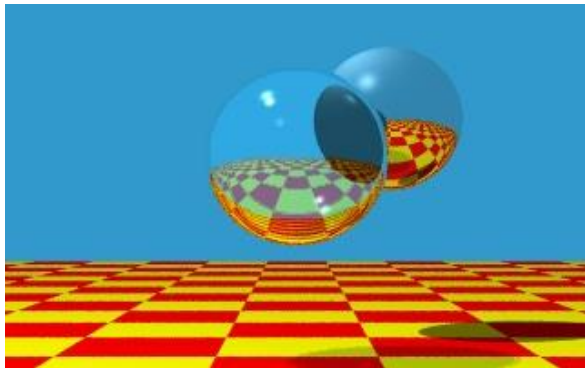
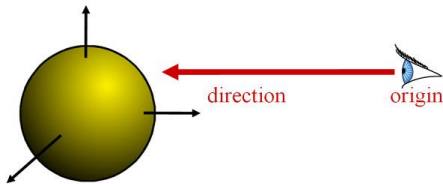
- 3 cases, depending on the sign of $b^2 - 4ac$
- What do these cases correspond to?
- Which root (t^+ or t^-) should you choose?
 - Closest positive!



Ray-Sphere Intersection

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

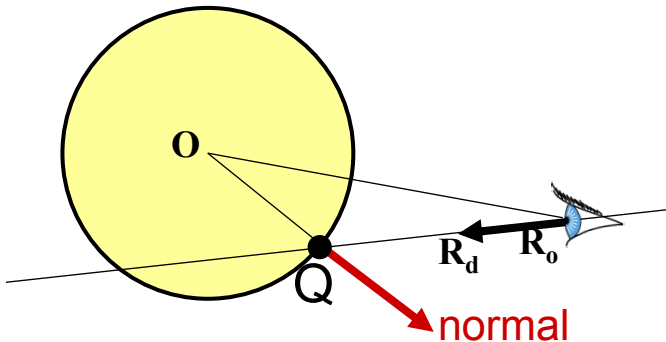
:-)



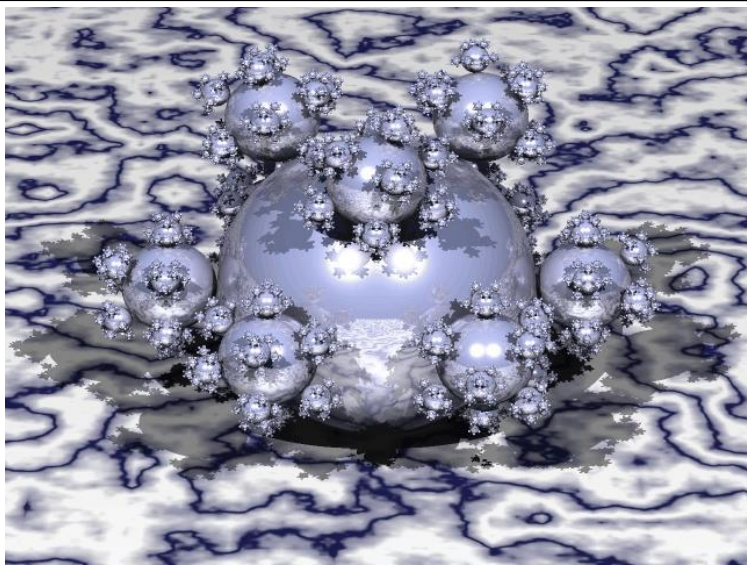
© source unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/help/faq-fair-use/>.

Sphere Normal

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



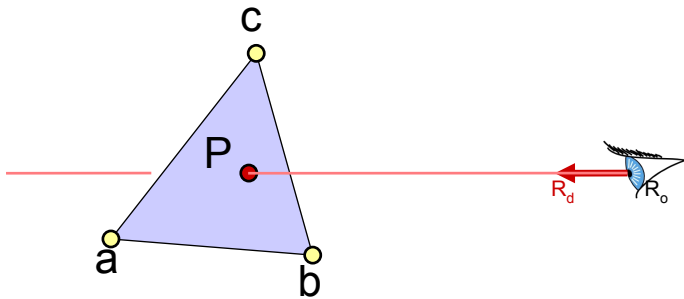
Questions?



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

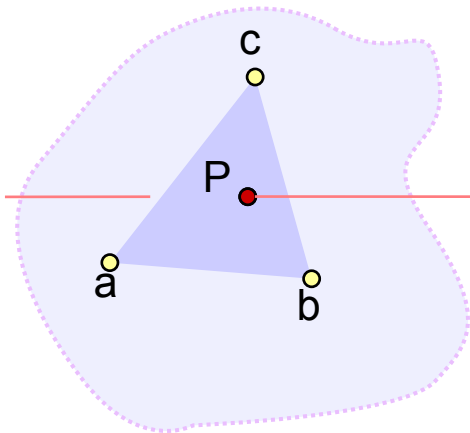


Barycentric Definition of a Plane

- A (non-degenerate) triangle $(\mathbf{a}, \mathbf{b}, \mathbf{c})$ defines a plane
- Any point \mathbf{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]

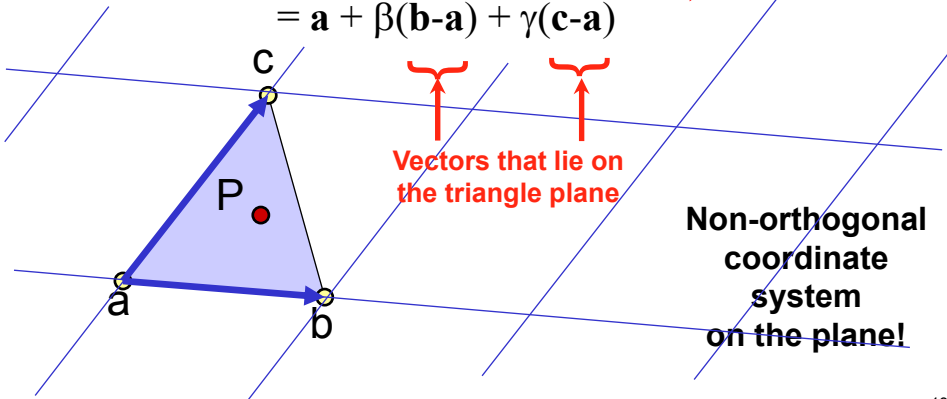
Barycentric Coordinates

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

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

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

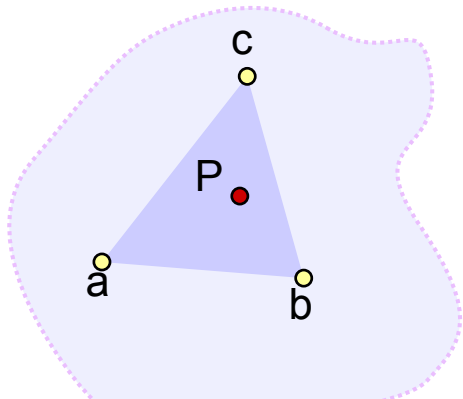
rewrite



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?

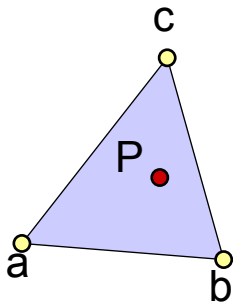


Fun to know:

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**.

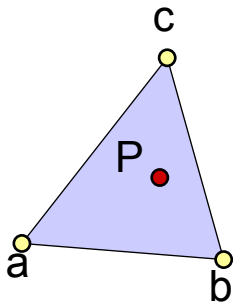
Barycentric Definition of a Triangle

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



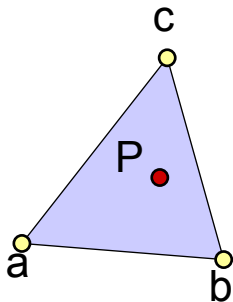
Barycentric Definition of a Triangle

- $\mathbf{P}(\alpha, \beta, \gamma) = \alpha \mathbf{a} + \beta \mathbf{b} + \gamma \mathbf{c}$
with $\alpha + \beta + \gamma = 1$ parameterizes the entire plane
- If we require in addition that $\alpha, \beta, \gamma \geq 0$, 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:
 - $\alpha = 0 \Rightarrow \mathbf{P}$ lies on line $\mathbf{b}-\mathbf{c}$
 - $\alpha, \beta = 0 \Rightarrow \mathbf{P} = \mathbf{c}$
 - 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:
 $\alpha, \beta, \gamma \geq 0$



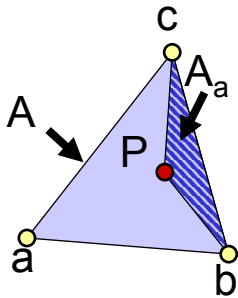
TIEA311 - Fast forward a bit

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
 - $\alpha = A_a/A$ $\beta = A_b/A$ $\gamma = A_c/A$
- Use signed areas for points outside the triangle

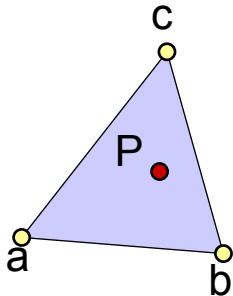


How Do We Compute α, β, γ ?

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

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

This should be zero

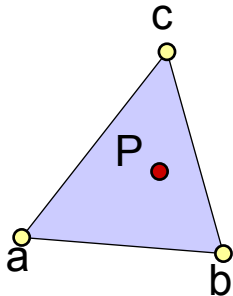


How Do We Compute α, β, γ ?

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

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

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 ?

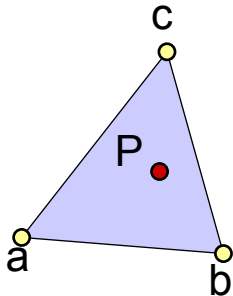
- Or write it as a 2×2 linear system
- $\mathbf{P}(\beta, \gamma) = \mathbf{a} + \beta \mathbf{e}_1 + \gamma \mathbf{e}_2$

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

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

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

These should be zero



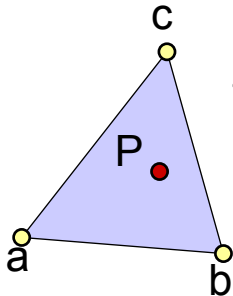
Ha! We'll take inner products of this equation with \mathbf{e}_1 & \mathbf{e}_2

How Do We Compute α, β, γ ?

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

$$\mathbf{e}_1 = (\mathbf{b}-\mathbf{a}), \mathbf{e}_2 = (\mathbf{c}-\mathbf{a}) \quad \langle \mathbf{e}_1, \mathbf{a} + \beta \mathbf{e}_1 + \gamma \mathbf{e}_2 - \mathbf{P} \rangle = 0$$

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



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

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

and $\langle \mathbf{a}, \mathbf{b} \rangle$ is the dot product.

TIEA311

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.

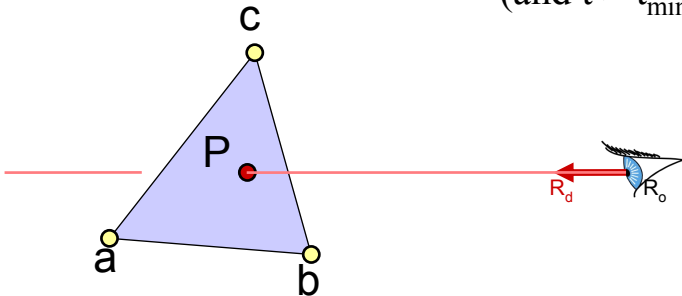
Intersection with Barycentric Triangle

- Again, set ray equation equal to barycentric equation

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

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

- Intersection if $\beta + \gamma \leq 1$ & $\beta \geq 0$ & $\gamma \geq 0$
(and $t > t_{\min} \dots$)



Intersection with Barycentric Triangle

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

$$R_{ox} + tR_{dx} = a_x + \beta(b_x - a_x) + \gamma(c_x - a_x)$$

$$R_{oy} + tR_{dy} = a_y + \beta(b_y - a_y) + \gamma(c_y - a_y)$$

$$R_{oz} + tR_{dz} = a_z + \beta(b_z - a_z) + \gamma(c_z - a_z)$$

} 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

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

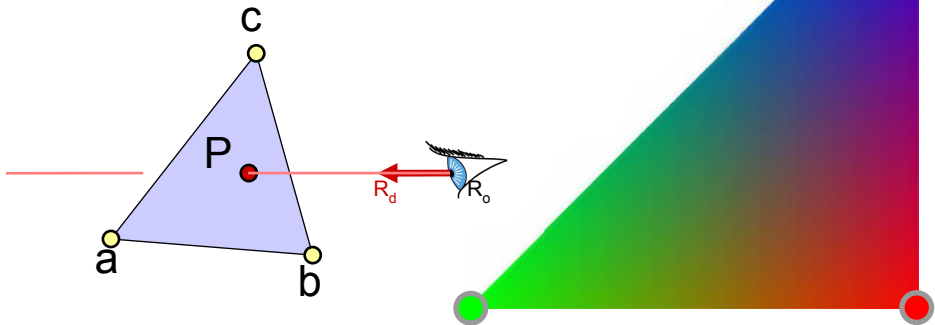
$$t = \frac{\begin{vmatrix} a_x - b_x & a_x - c_x & a_x - R_{ox} \\ a_y - b_y & a_y - c_y & a_y - R_{oy} \\ a_z - b_z & a_z - c_z & a_z - R_{oz} \end{vmatrix}}{|A|}$$

| | denotes the determinant

Can be copied mechanically into code

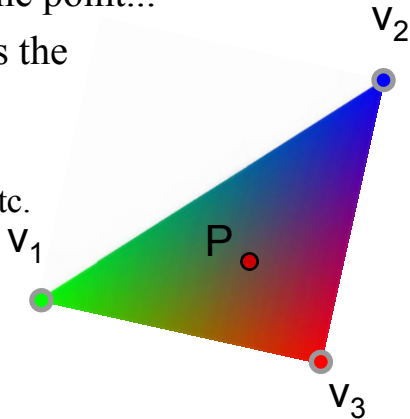
Barycentric Intersection Pros

- 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 $\mathbf{a}, \mathbf{b}, \mathbf{c}$
 - 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 \mathbf{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

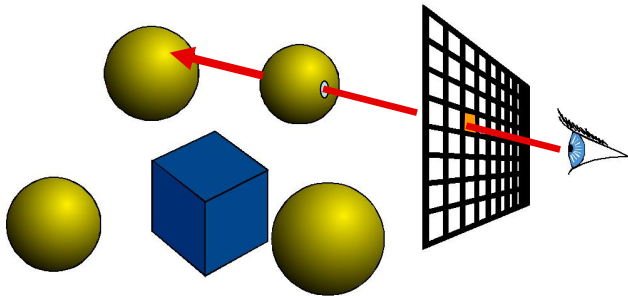
For every pixel

Construct a ray from the eye

For every object in the scene

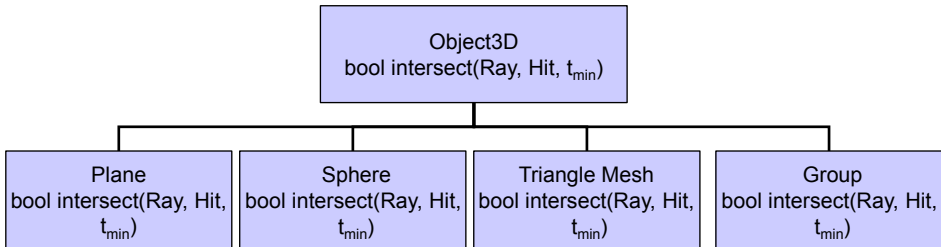
Find intersection with the ray

Keep if closest



Object-Oriented Design

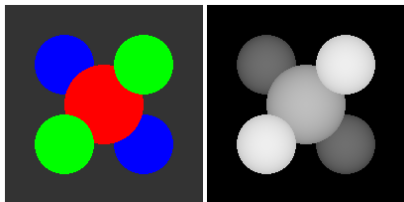
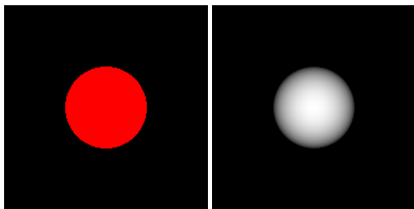
- We want to be able to add primitives easily
 - Inheritance and virtual methods
- Even the scene is derived from Object3D!



- 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:

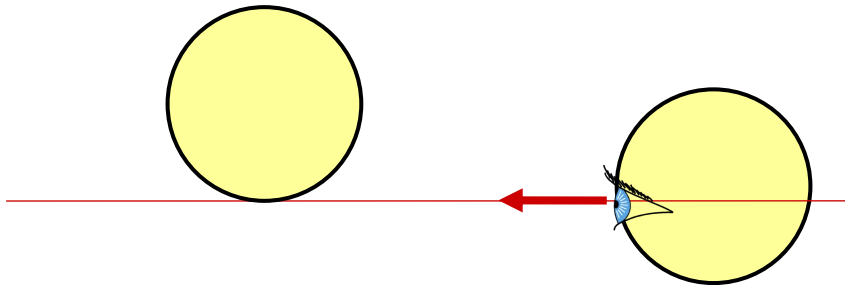
- Shirley P., M. Ashikhmin and S. Marschner, *Fundamentals of Computer Graphics*
- Shirley P. and R.K. Morley, *Realistic Ray Tracing*
- Jensen H.W., *Realistic Image Synthesis Using Photon Mapping*

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 :)

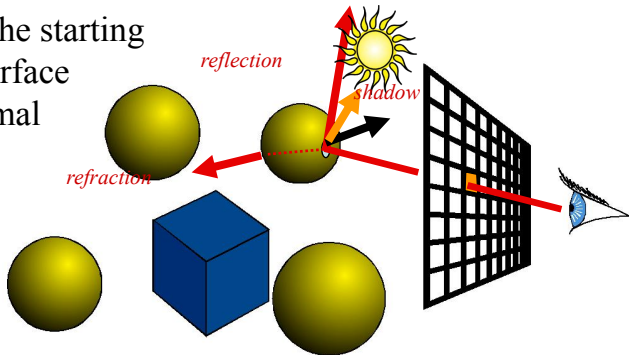
Precision

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



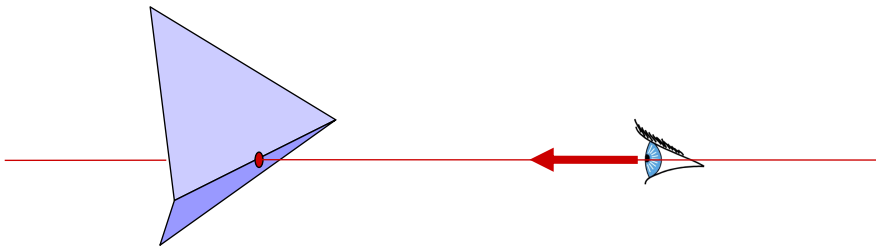
The Evil ϵ

- 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

Courtesy of Henrik Wann Jensen. Used with permission.

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

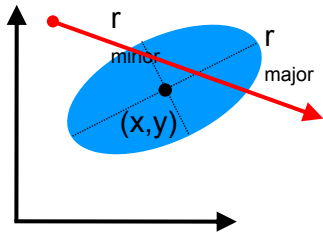
1. Make each primitive handle any applied transformations and produce a camera space description of its geometry

```
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

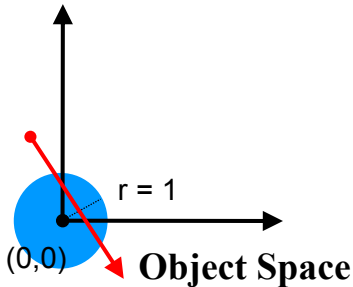
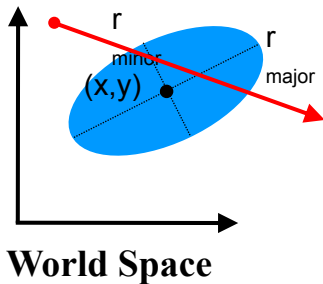
```
Sphere {  
  center 3 2 0  
  z_rotation 30  
  r_major 2  
  r_minor 1  
}
```



- 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:

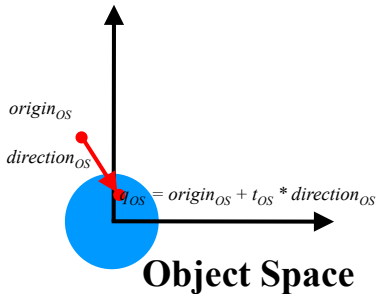
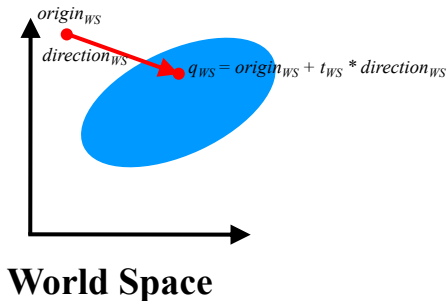
$$origin_{OS} = \mathbf{M}^{-1} origin_{WS}$$

- 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

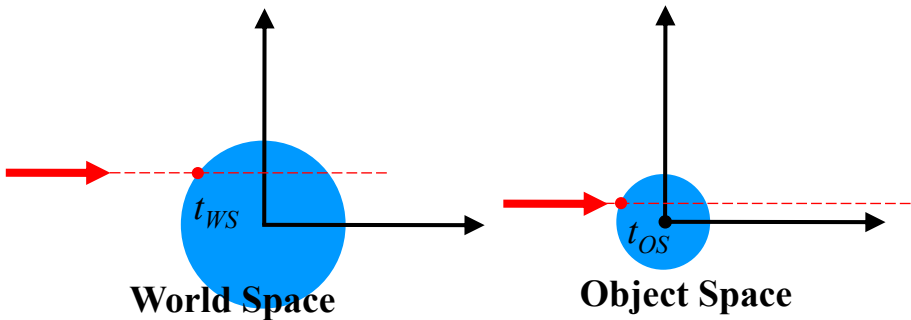


What About t ?

- If \mathbf{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

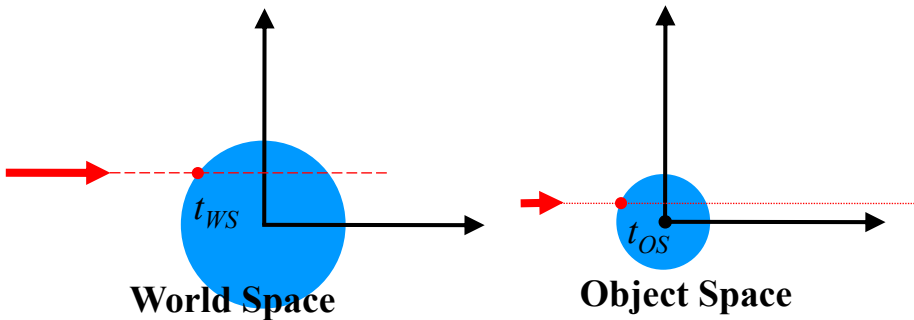
- $t_{OS} \neq t_{WS}$
and must be rescaled after intersection
 \implies One more possible failure case...



2. Do Not Normalize Direction

Highly
recommended

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



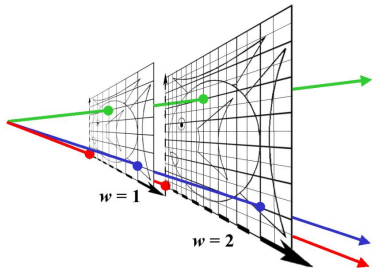
Transforming Points & Directions

- 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 \mathbf{M} to a point and to a direction ==> Store everything in 4D!

Different objects

- **Points**

- represent locations



- **Vectors**

- represent movement, force, displacement from A to B



- **Normals**

- represent orientation, unit length



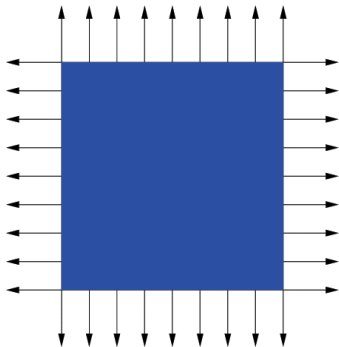
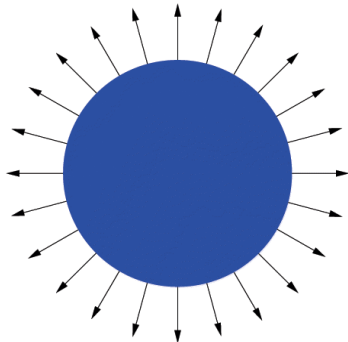
- **Coordinates**

- numerical representation of the above objects
in a given coordinate system

$$\begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

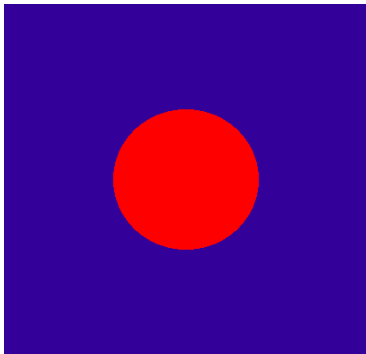
Normal

- Surface Normal: unit vector that is locally perpendicular to the surface

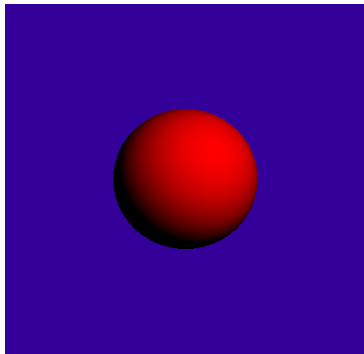


Why is the Normal important?

- It's used for shading — makes things look 3D!

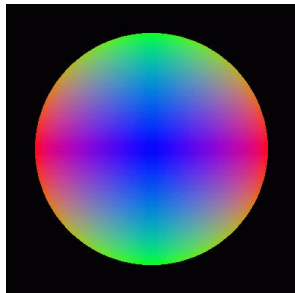
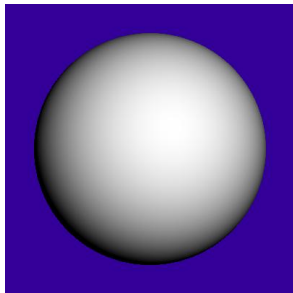
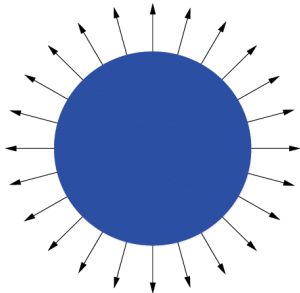


object color only



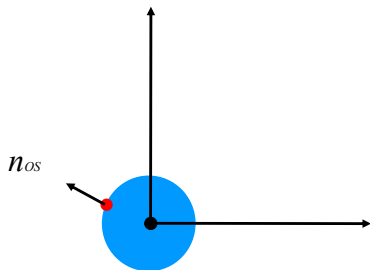
Diffuse Shading

Visualization of Surface Normal

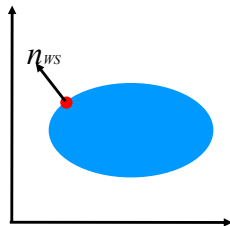


$\pm x = \text{Red}$
 $\pm y = \text{Green}$
 $\pm z = \text{Blue}$

How do we transform normals?



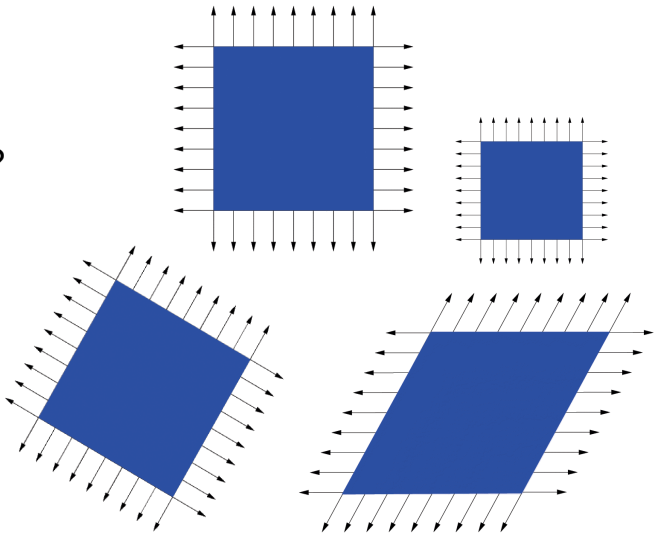
Object Space



World Space

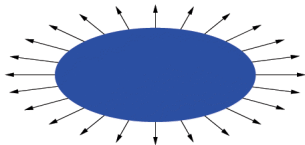
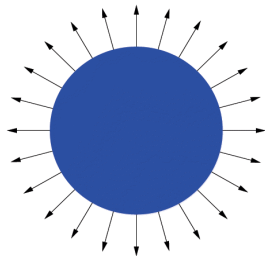
Transform Normal like Object?

- translation?
- rotation?
- isotropic scale?
- scale?
- reflection?
- shear?
- perspective?



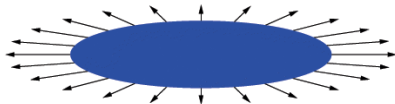
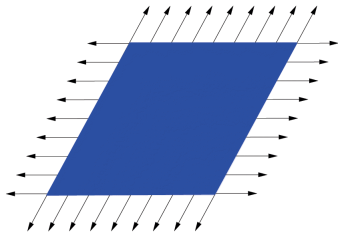
Transform Normal like Object?

- translation?
- rotation?
- isotropic scale?
- scale?
- reflection?
- shear?
- perspective?

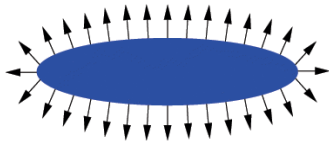
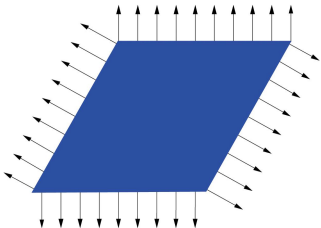


Transformation for shear and scale

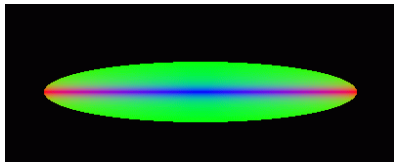
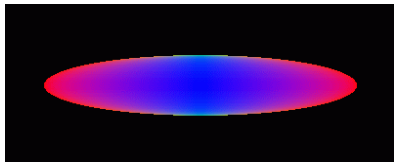
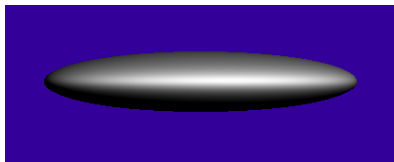
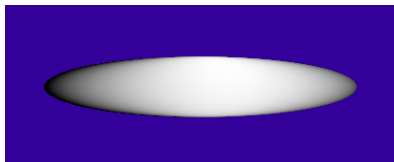
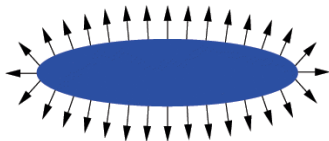
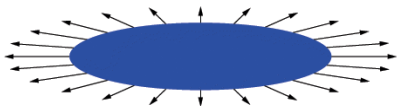
Incorrect
Normal
Transformation



Correct
Normal
Transformation



More Normal Visualizations

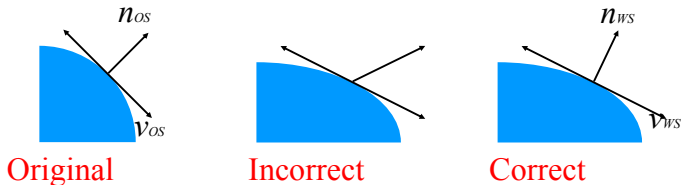


Incorrect Normal Transformation

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

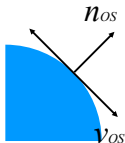
v is perpendicular to normal n :

Dot product $n_{os}^T v_{os} = 0$

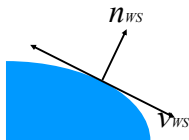
$$n_{os}^T (\mathbf{M}^{-1} \mathbf{M}) v_{os} = 0$$

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

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



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



$$n_{ws}^T = n_{os}^T (\mathbf{M}^{-1})$$

$$n_{ws} = (\mathbf{M}^{-1})^T n_{os}$$

$$n_{ws}^T v_{ws} = 0$$

Digression

$$n_{ws} = (\mathbf{M}^{-1})^T n_{os}$$

- The previous proof is not quite rigorous; first you'd need to prove that tangents indeed transform with \mathbf{M} .
 - Turns out they do, but we'll take it on faith here.
 - If you believe that, then the above formula follows.

Comment

- So the correct way to transform normals is:

$$n_{ws} = (\mathbf{M}^{-1})^{\top} n_{os}$$

Sometimes denoted $\mathbf{M}^{-\top}$

- But why did $n_{ws} = \mathbf{M} n_{os}$ work for similitudes?
- Because for similitude / similarity transforms,

$$(\mathbf{M}^{-1})^{\top} = \lambda \mathbf{M}$$

- e.g. for orthonormal basis:

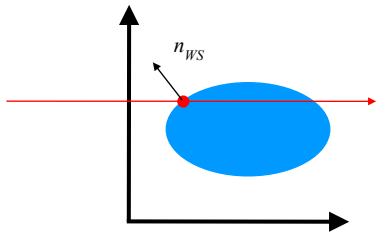
$$\mathbf{M}^{-1} = \mathbf{M}^{\top} \quad \text{i.e.} \quad (\mathbf{M}^{-1})^{\top} = \mathbf{M}$$

Connections

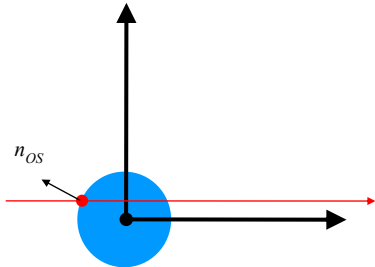
- Not part of class, but cool
 - “Covariant”: transformed by the matrix
 - e.g., tangent
 - “Contravariant”: transformed by the inverse transpose
 - e.g., the normal
 - a normal is a “co-vector”

- Google “differential geometry” to find out more

Recap: How to Transform Normals?



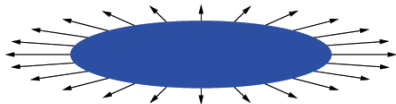
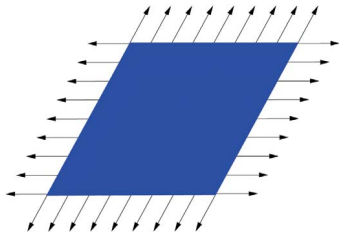
World Space



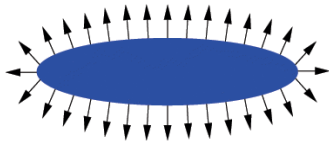
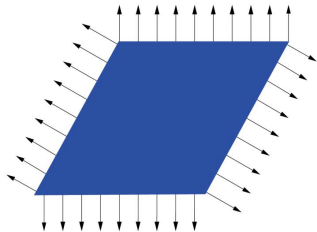
Object Space

Transformation for Shear and Scale

Incorrect
Normal
Transformation

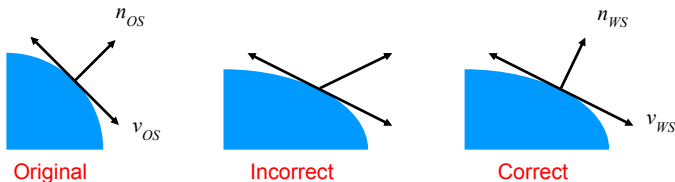


Correct
Normal
Transformation



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 :

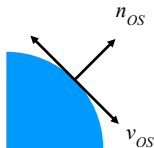
Dot product

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

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

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

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

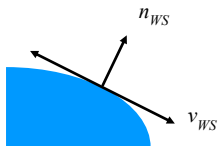


v_{WS} is perpendicular to normal n_{WS} :

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

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

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



Position, Direction, Normal

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

Ray Tracing



Henrik Wann Jensen

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

Ray Casting

For every pixel

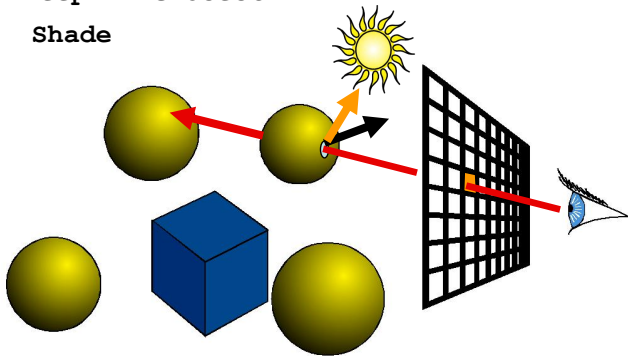
Construct a ray from the eye

For every object in the scene

Find intersection with the ray

Keep if closest

Shade



Today – Ray Tracing

~~(Indirect illumination)~~

Reflections

Refractions

Shadows

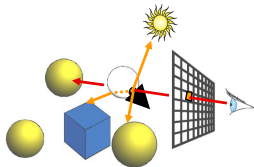
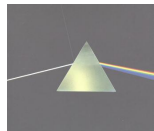
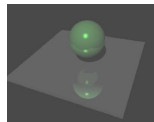
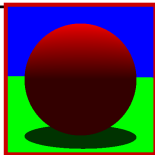
~~(Caustics)~~

HENRIK WANN JENSEN 2000

Courtesy of Henrik Wann Jensen. Used with permission.

Overview of Today

- Shadows
- Reflection
- Refraction
- Recursive Ray Tracing
 - “Hall of mirrors”



How Can We Add Shadows?

For every pixel

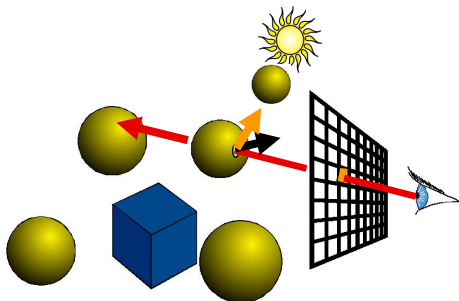
Construct a ray from the eye

For every object in the scene

Find intersection with the ray

Keep if closest

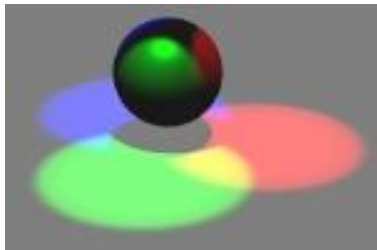
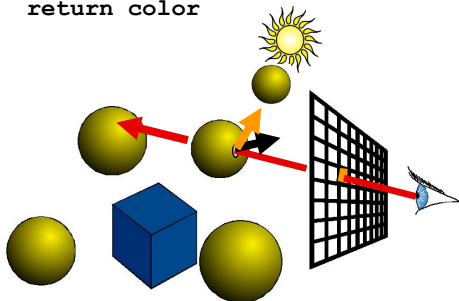
Shade



How Can We Add Shadows?

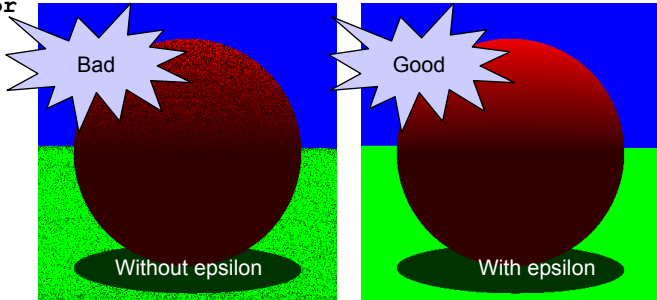
```
color = ambient*hit->getMaterial()->getDiffuseColor()
for every light
    Ray ray2(hitPoint, directionToLight)
    Hit hit2(distanceToLight, NULL, NULL)
    For every object
        object->intersect(ray2, hit2, 0)
    if (hit2->getT() = distanceToLight)
        color += hit->getMaterial()->Shade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

ambient = k_a
diffuseColor = k_d



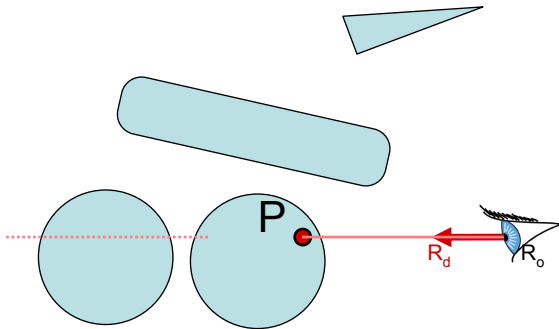
Problem: Self-Shadowing

```
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
```



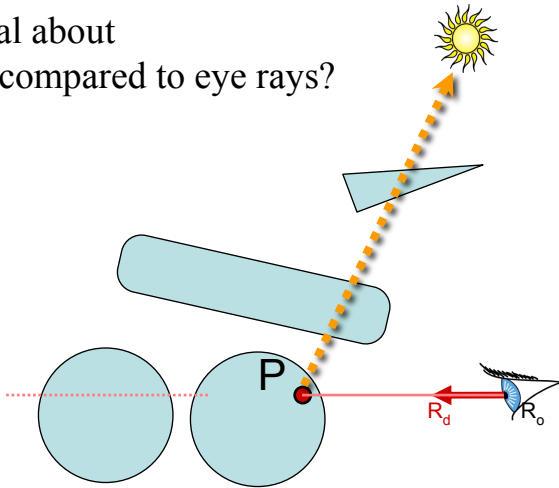
Let's Think About Shadow Rays

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



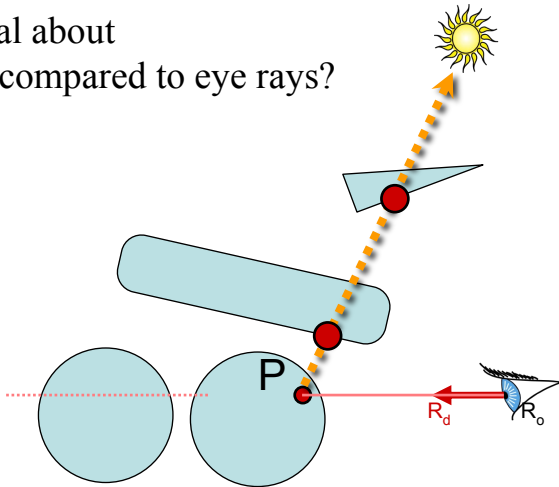
Let's Think About Shadow Rays

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



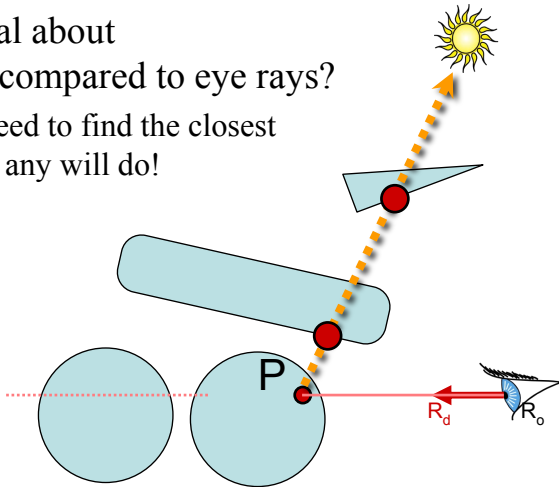
Let's Think About Shadow Rays

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



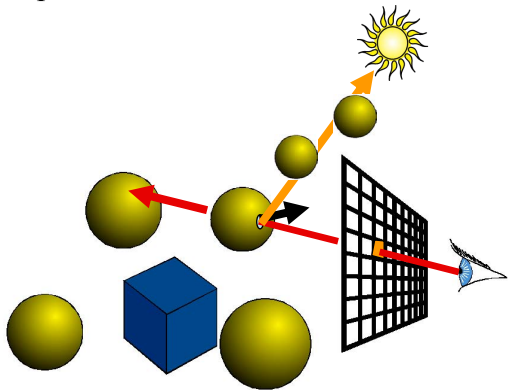
Let's Think About Shadow Rays

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



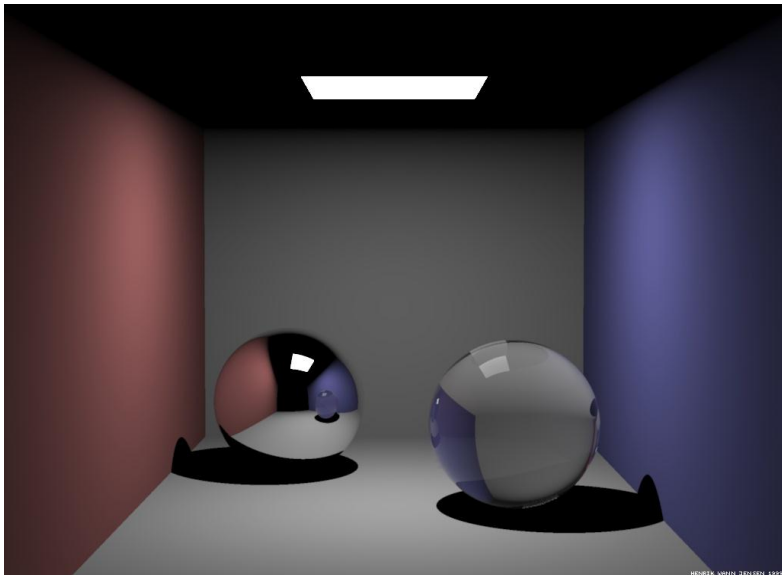
Shadow Optimization

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



Questions?

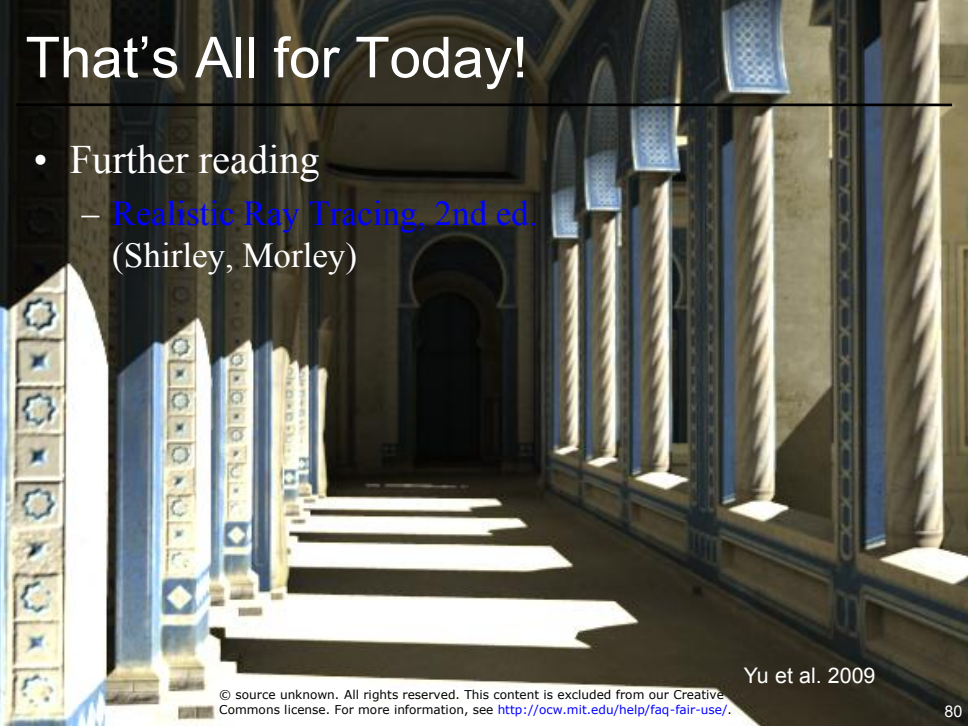
Henrik Wann Jensen



Courtesy of Henrik Wann Jensen. Used with permission.

That's All for Today!

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



Yu et al. 2009

TIEA311 - Today in Jyväskylä

The time allotted for this week's graphics lectures is now over.

Next lecture happens in 6 days and 4 hours.

The teacher will now tell his view about what **could be useful activities** for you during that time period.

→ see lecture video.

Make notes, if you have to.

Even if he forgets to say it, **remember to rest, too!**