# TIEA311
# Tietokonegrafiikan perusteet
kevät 2017

(“Principles of Computer Graphics” – Spring 2017)

# TIEA311 Tietokonegrafiikan perusteet – kevät 2017 (“Principles of Computer Graphics” – Spring 2017)

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 2017 at the Faculty of Information technology, University of Jyväskylä:
`http://users.jyu.fi/~nieminen/tgp17/`

# 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.
- (Leave it at yet another “cliffhanger” if need be. This is for Assignments 4 and 5 in any case)
- Have a break!

Part 2, After the break:

- React and adapt to observations in computer class, IRC, and face-to-face communications
- Recap (or try to learn for the first time:)) fundamental things from earlier lectures and assignments
- “C++ from MIT”, with Finnish commentary (started from this on lecture, meanwhile covering the above)

# 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:

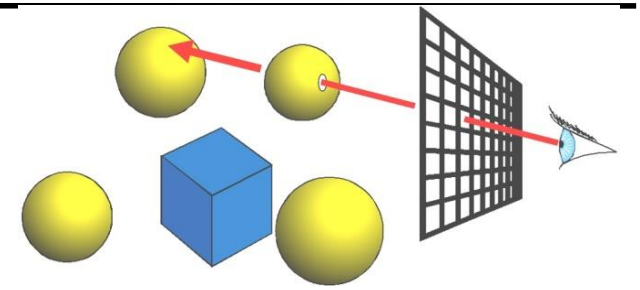
When you forget how some detail works, what do you do? You connect to the WWW and re-learn the detail.

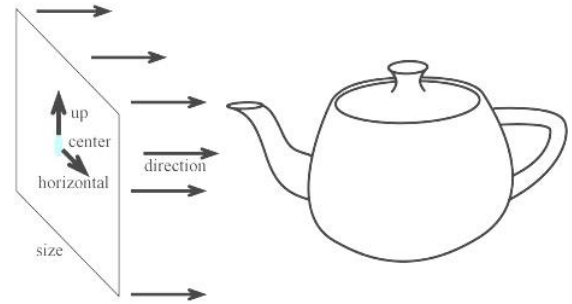
When you need to learn a new detail, what do you do? You connect to the WWW and learn the detail. You only need a keyword to find it.

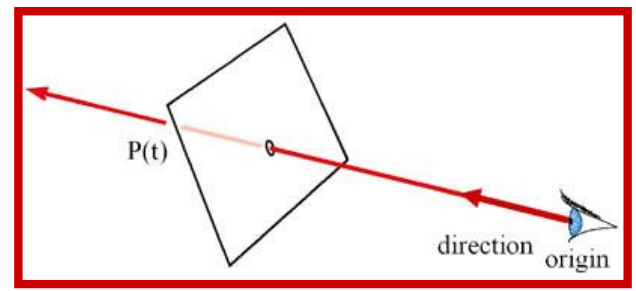
That much is simple. Things get interesting in putting the details to action.

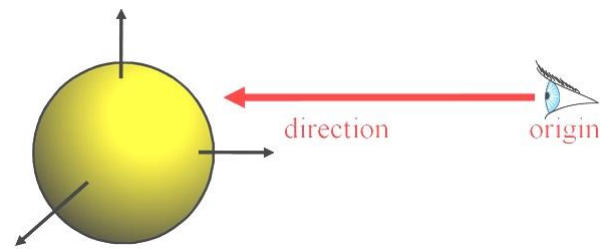
# Ray Casting

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





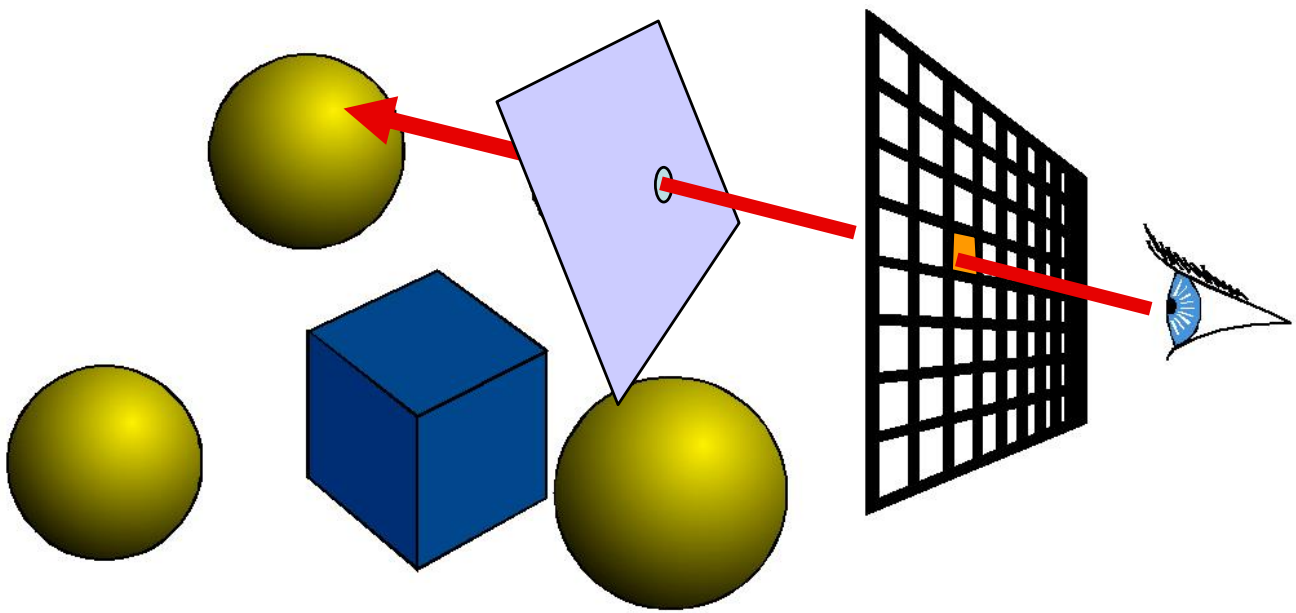



# Ray Casting

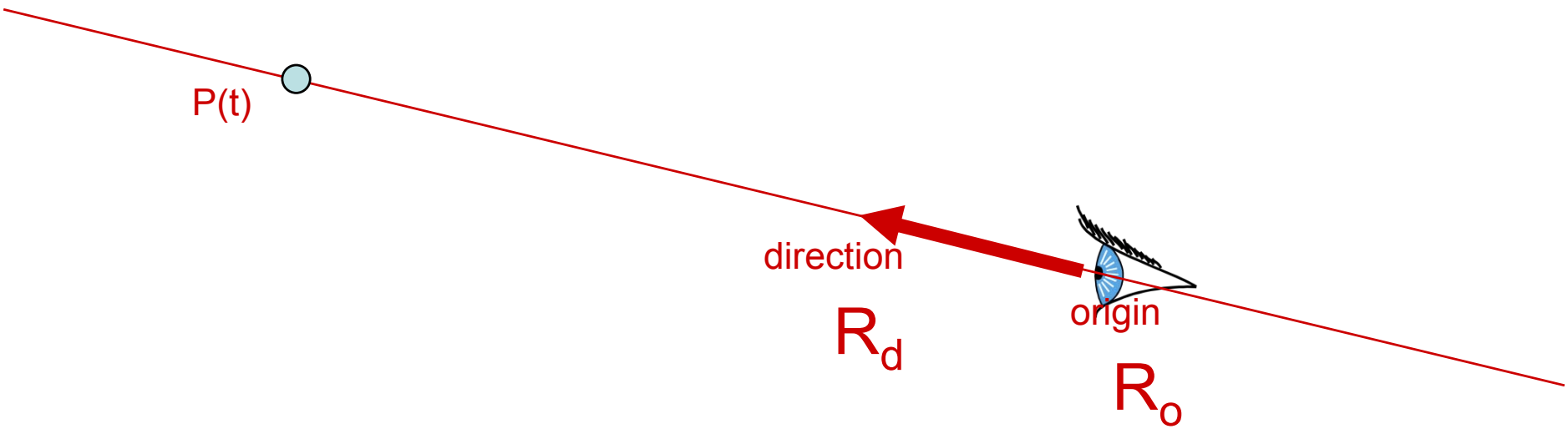
```
For every pixel
   Construct a ray from the eye
   For every object in the scene
       Find intersection with the ray
       Keep if closest
```

First we will study ray-plane intersection



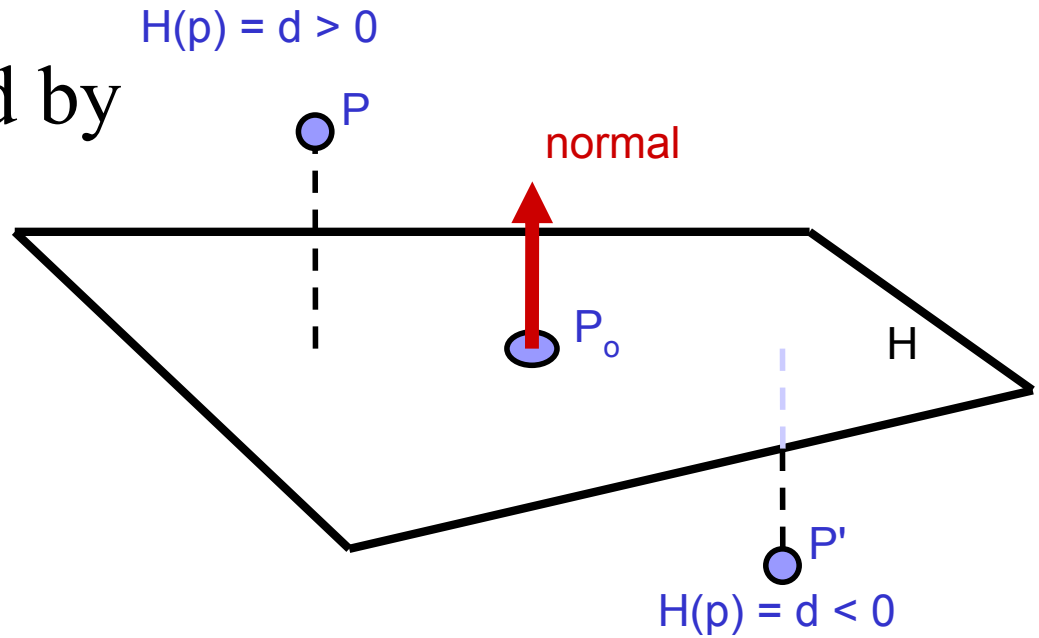
# Recall: Ray Representation

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



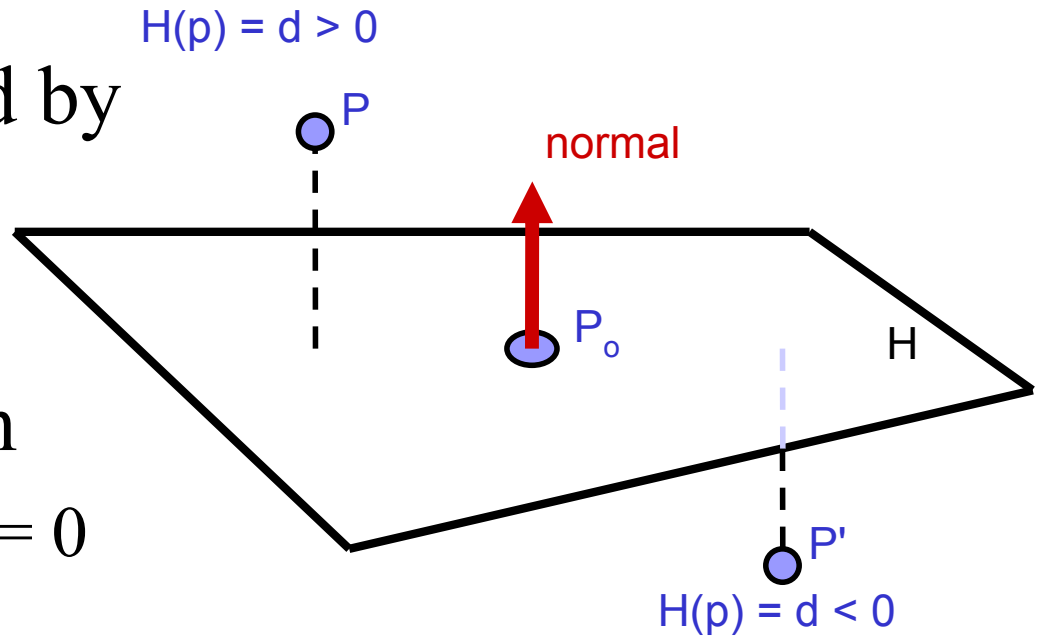
# 3D Plane Representation?

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



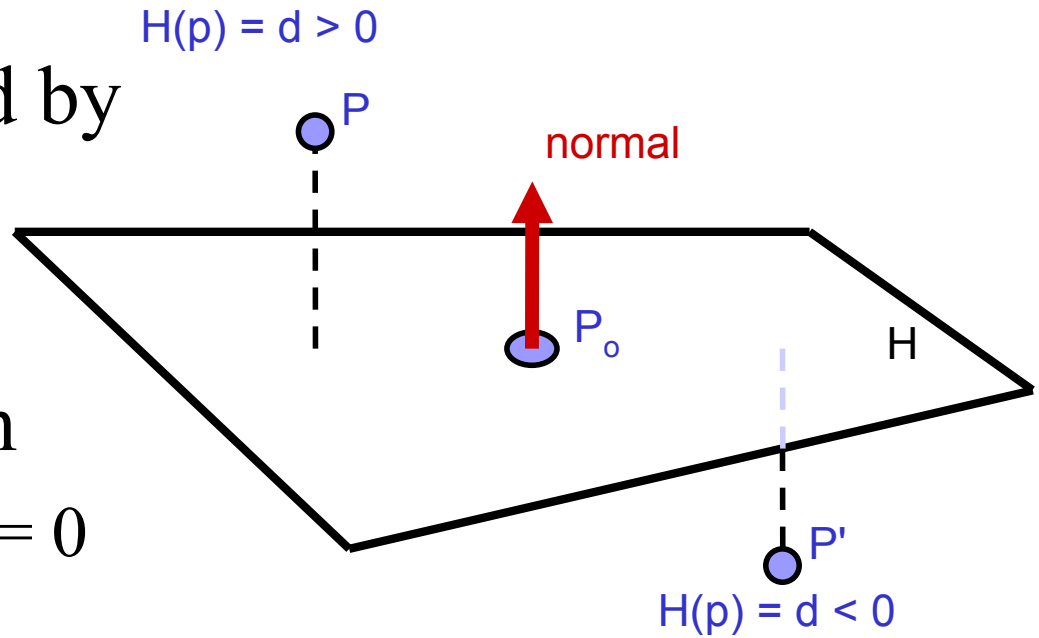
# 3D Plane Representation?

- (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?

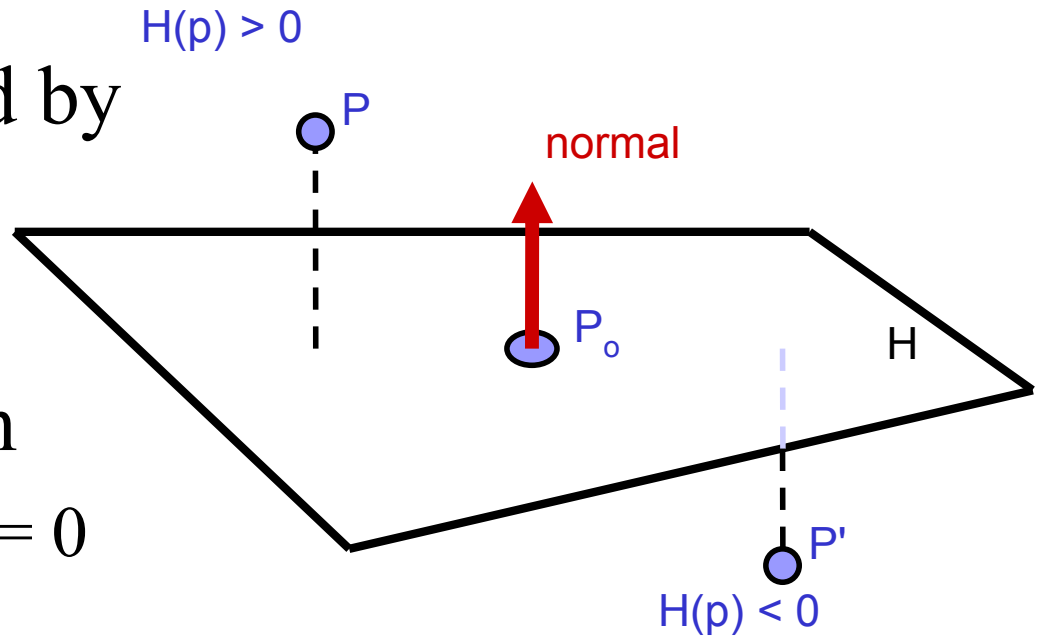



$$Ax_0 + By_0 + Cz_0 + D = 0$$ (Point $P_0$ must lie on plane)

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

# 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*!



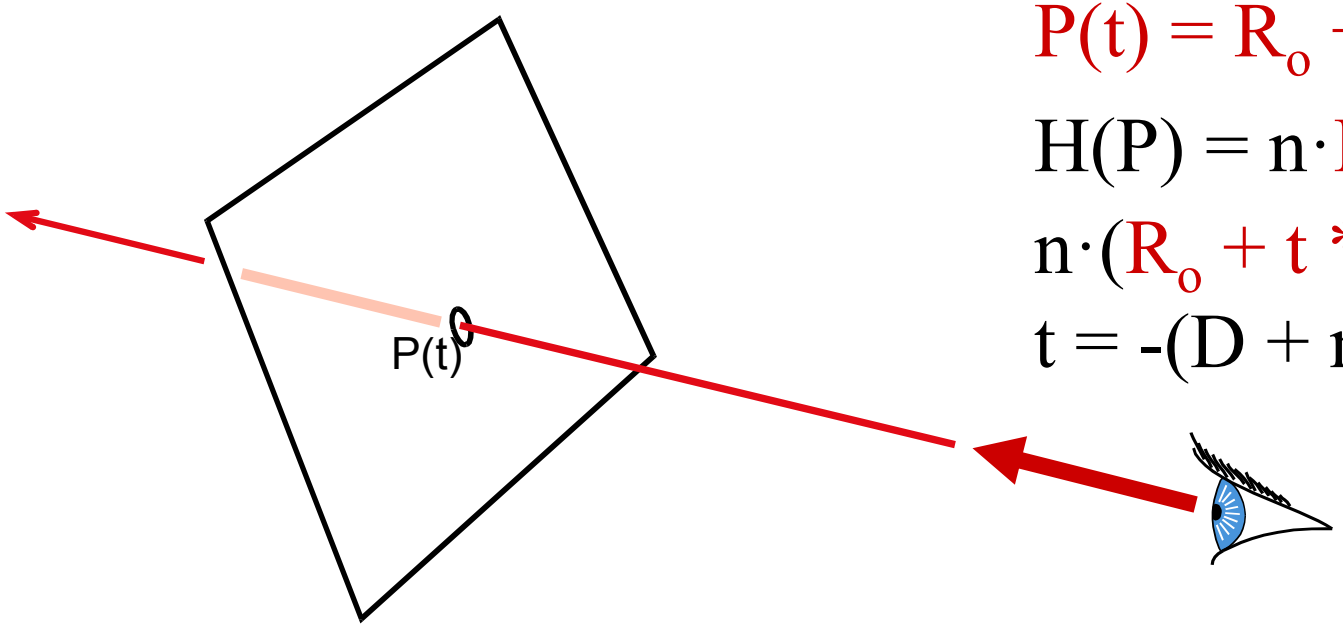
# Explicit vs. Implicit?

- 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**!

# 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.

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

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

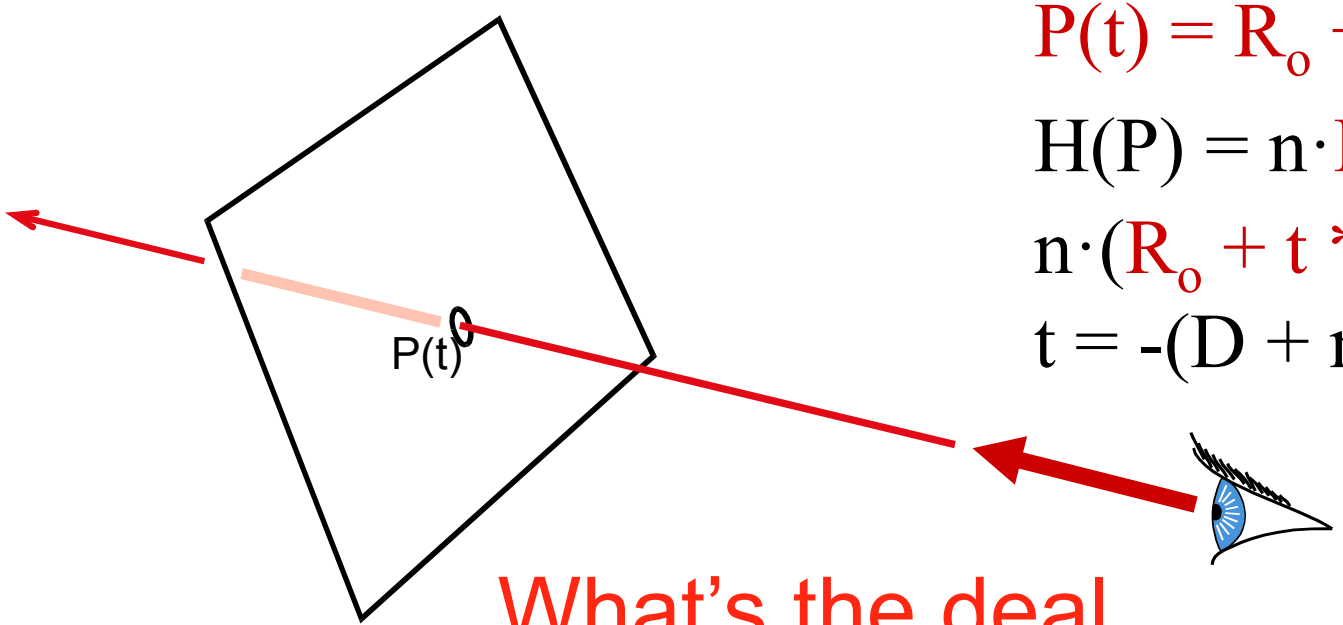
$$t = -\frac{D + \vec{n} \cdot \vec{R}_o}{\vec{n} \cdot \vec{R}_d}$$

“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!**

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

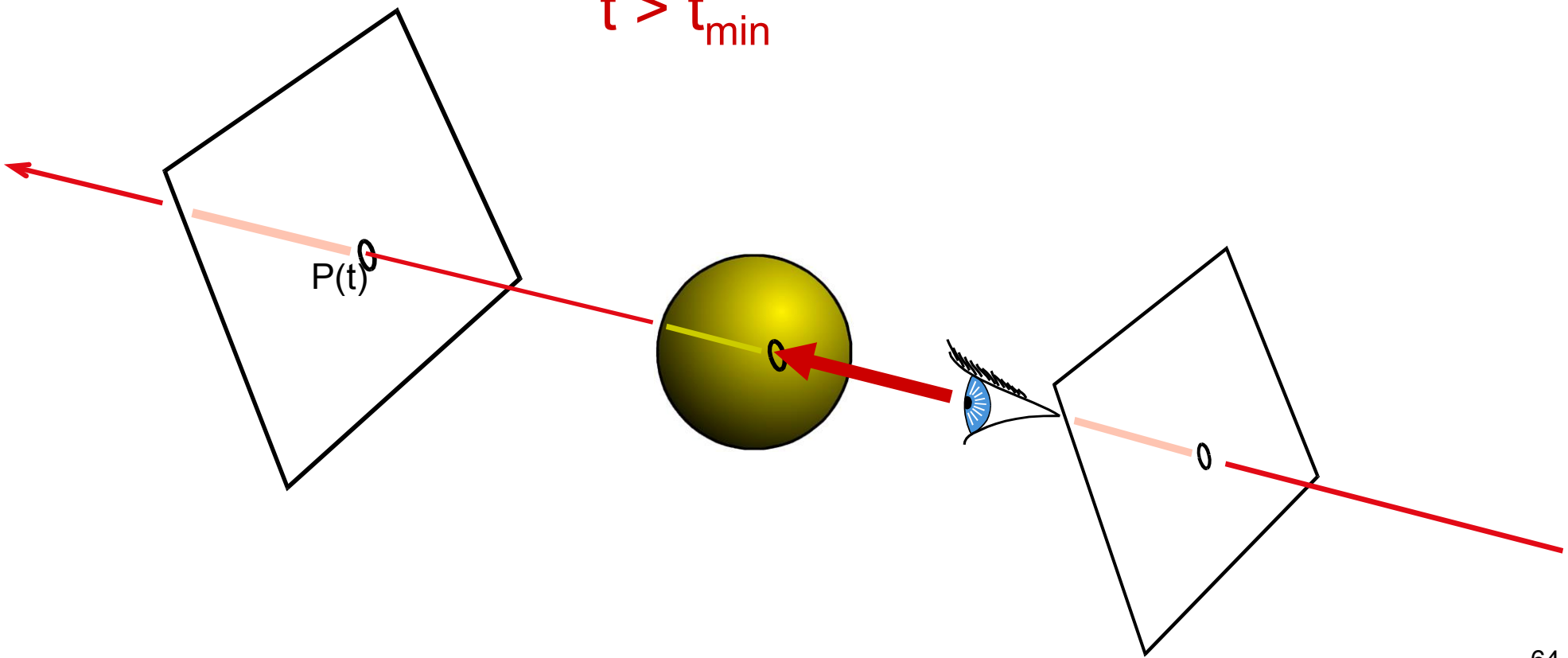
# Additional Bookkeeping

- Verify that intersection is closer than previous

$t < t_{current}$

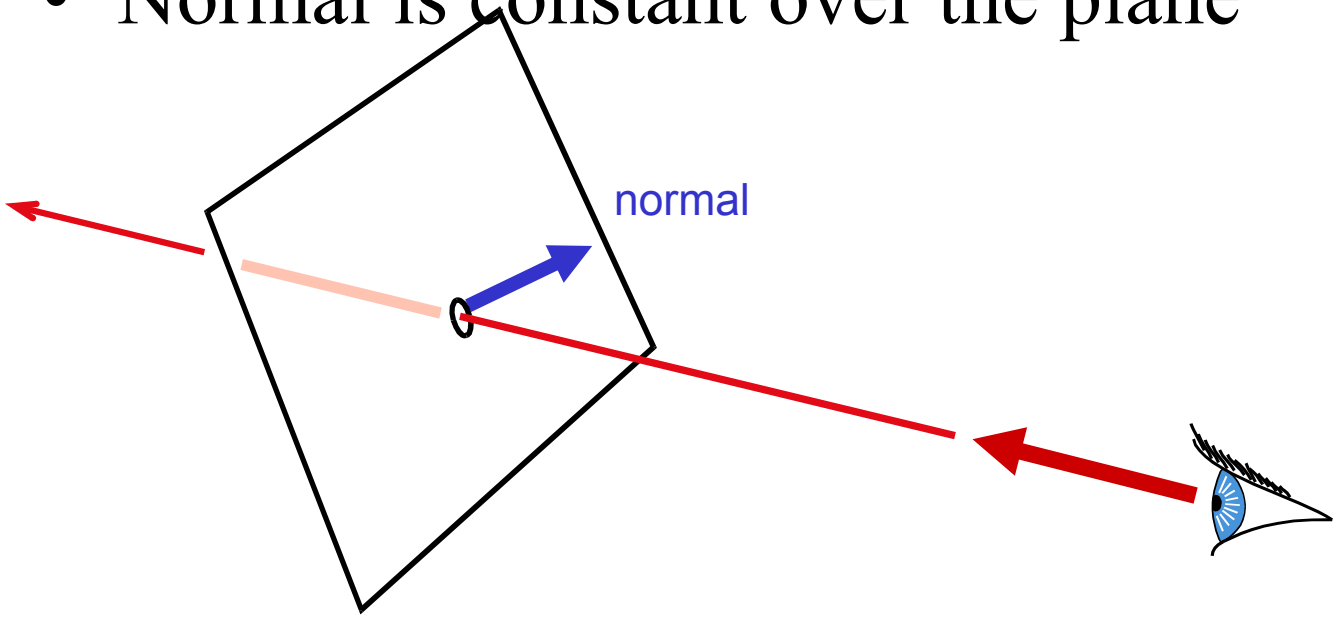
- Verify that it is not out of range (behind eye)

$t > t_{min}$

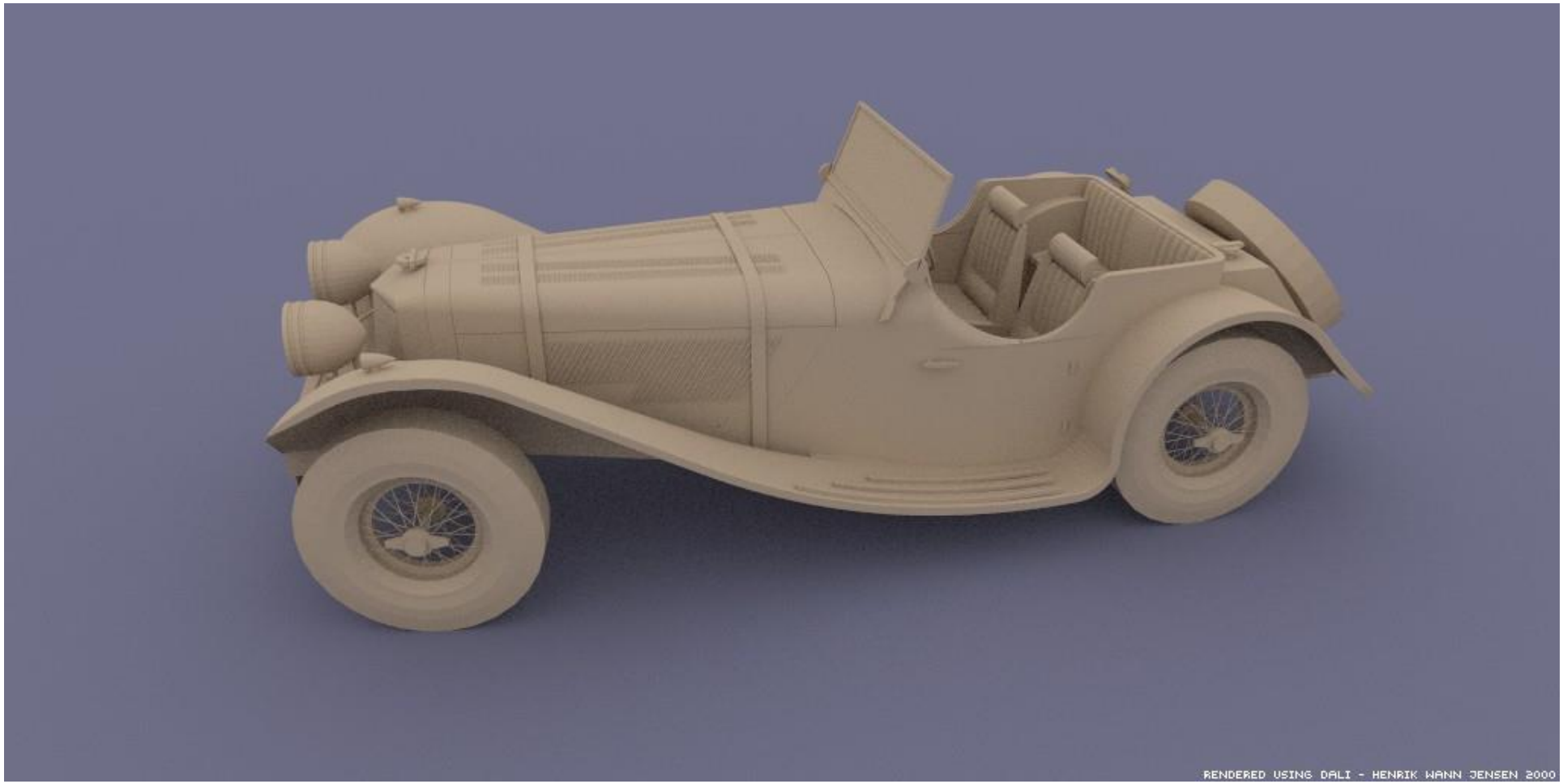


# Normal

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



# Questions?




Courtesy of Henrik Wann Jensen. Used with permission.

Image by Henrik Wann Jensen

# 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.
- (Leave it at yet another “cliffhanger” if need be. This is for Assignments 4 and 5 in any case)
- Have a break!

Part 2, After the break:

- React and adapt to observations in computer class, IRC, and face-to-face communications
- Recap (or try to learn for the first time:)) fundamental things from earlier lectures and assignments
- “C++ from MIT”, with Finnish commentary (started from this on lecture, meanwhile covering the above)

# C++

- 3 ways to pass arguments to a function
  - by value, e.g. float f(float x)
  - by reference, e.g. float f(float &x)
    - f can modify the value of x
  - by pointer, e.g. float f(float *x)
    - x here is a just a memory address
    - motivations:
      less memory than a full data structure if x has a complex type
      dirty hacks (pointer arithmetic),but just do not do it
    - clean languages do not use pointers
    - kind of redundant with reference
    - arrays are pointers

# Pointers

- Can get it from a variable using &
  - often a BAD idea. see next slide
- Can be dereferenced with *
  - float *px=new float; // px is a memory address to a float
  - *px=5.0; //modify the value at the address px
- Should be instantiated with new. See next slide

# Pointers, Heap, Stack

- Two ways to create objects
  - The BAD way, on the stack
    - myObject *f() {
      - myObject x;
      - ...
      - return &x
    - will crash because x is defined only locally and the memory gets de-allocated when you leave function f
  - The GOOD way, on the heap
    - myObject *f() {
      - myObject *x=new myObject;
      - ...
      - return x
    - but then you will probably eventually need to delete it

# Segmentation Fault

- When you read or, worse, write at an invalid address
- Easiest segmentation fault:
  - float *px; // px is a memory address to a float
  - *px=5.0; //modify the value at the address px
  - Not 100% guaranteed, but you haven't instantiated px, it could have any random memory address.
- 2nd easiest seg fault
  - Vector<float> vx(3);
  - vx[9]=0;

# Segmentation Fault

- TERRIBLE thing about segfault: the program does not necessarily crash where you caused the problem
- You might write at an address that is inappropriate but that exists
- You corrupt data or code at that location
- Next time you get there, crash

- When a segmentation fault occurs, always look for pointer or array operations before the crash, but not necessarily at the crash

# Debugging

- Display as much information as you can
  - image maps (e.g. per-pixel depth, normal)
  - OpenGL 3D display (e.g. vectors, etc.)
  - cerr<< or cout<< (with intermediate values, a message when you hit a given if statement, etc.)
- Doubt everything
  - Yes, you are sure this part of the code works, but test it nonetheless
- Use simple cases
  - e.g. plane z=0, ray with direction (1, 0, 0)
  - and display all intermediate computation

# Questions?

# TIEA311 - Today in Jyväskylä (in Finnish)

The “steps of Jarno” (Ajattelumallia tehtävien ratkaisuun):

1. Luentomateriaali
2. Tehtävänanto (muista mitä aiemmissa tehtävissä on tehty/annettu)
3. Hae lähdekoodi ja testaa sen toiminta
4. Yhdistä teoria tehtävään ja lähdekoodiin, ymmärrä kokonaisuus
5. Hahmottele kevyt ”speksi” esim. paperille UML, prosessikaavio, ...

   ---

6. Tee osatehtävä 1
7. Päivitä ”speksi”
8. Tee osatehtävä 2
9. Päivitä ”speksi”

   ...